

A method engineering approach to support dynamic evolution in composition-based distributed applications

Author:

Fung, Kam Hay

Publication Date:

2011

DOI:

<https://doi.org/10.26190/unsworks/23796>

License:

<https://creativecommons.org/licenses/by-nc-nd/3.0/au/>

Link to license to see what you are allowed to do with this resource.

Downloaded from <http://hdl.handle.net/1959.4/51233> in <https://unsworks.unsw.edu.au> on 2024-05-03

**A Method Engineering Approach
to Support Dynamic Evolution
in Composition-Based Distributed Applications**

by

Kam Hay Fung

March 2011



A thesis submitted in fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Information Systems, Technology and Management

Australian School of Business

The University of New South Wales, Australia

ORIGINALITY STATEMENT

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

Signed

Date

COPYRIGHT STATEMENT

'I hereby grant the University of New South Wales or its agents the right to archive and to make available my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known, subject to the provisions of the Copyright Act 1968. I retain all proprietary rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

I also authorise University Microfilms to use the 350 word abstract of my thesis in Dissertation Abstract International (this is applicable to doctoral theses only).

I have either used no substantial portions of copyright material in my thesis or I have obtained permission to use copyright material; where permission has not been granted I have applied/will apply for a partial restriction of the digital copy of my thesis or dissertation.'

Signed

Date

AUTHENTICITY STATEMENT

'I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.'

Signed

Date

ABSTRACT

Dynamic evolution is a phenomenon by which applications can be upgraded without requiring shutdown and restart. This capability is particularly important for critical distributed applications in which disruptions from shutdown for changes should be avoided.

Although a myriad of relevant techniques and tools exist, dynamic evolution is not well understood or adequately supported from a software development methodology perspective. This research aims to alleviate these shortcomings as a first step towards embracing dynamic evolution from this perspective. It leverages composition-based distributed applications - built of modules weaved into composite structures - for their ability in accommodating changes in their structures.

This research uses a three-phased design science research approach for its investigation. First, dynamic evolution requirements for methodological consideration in composition-based distributed applications were determined. Second, the methodological support for these requirements was developed. Third, this support was validated and enhanced by expert review and its application in a case study.

The first contribution of this research is a proposed set of dynamic evolution requirements: dynamic change requirements and dynamic evolution quality factors. The former characterise dynamic changes that a distributed application would accommodate while the latter are concerned with how well the application and its dynamic changes are designed to facilitate dynamic evolution. They provide areas of concerns for practitioners and researchers in developing, managing and testing distributed applications which can benefit from dynamic evolution.

The second major contribution is Continuum, a methodological extension comprising a suite of method fragments, specified with the International Standard ISO/IEC 24744: *Software Engineering - Metamodel for Development Methodologies*, to address the dynamic evolution requirements proposed from this research. Continuum centres on three key concerns: an application's lifecycle, transitional periods between successive application generations, and transformations which are modifications to the application. Via situational method engineering, the fragments can be incorporated into a fragment-structured methodology to extend its capability for dynamic evolution. These contributions are limited to the analysis and design aspects of software development and composition-based distributed applications.

PUBLICATIONS

A number of articles from earlier work of this research have been accepted and/or published in refereed journals, as well as presented to refereed conferences.

Refereed International Journal Articles

Fung, K.H., Low, G.C. and Ray, P.K. 2004, '**Embracing Dynamic Evolution in Distributed Systems**', *IEEE Software*, vol. 21, no. 2, pp. 49-55. DOI: 10.1109/MS.2004.1270762

- part of Chapter 2 in this thesis

Fung, K.H., Low, G. 2011, '**Quality factors for dynamic evolution in composition-based distributed applications**', *The DATA BASE for Advances in Information Systems*, vol. 42, no. 1, pp. 29-58. DOI: 10.1145/1952712.1952715

- part of Chapter 4 in this thesis

Fung, K.H. and Low, G. 2009, '**A methodology evaluation framework for dynamic evolution in composition-based distributed applications**', *Journal of Systems and Software*, vol. 82, no. 10, pp. 1950-1965. DOI: 10.1016/j.jss.2009.06.032

- part of Chapter 5 in this thesis

Refereed International Conference Papers

Fung, K.H. and Low, G. 2010, '**A process-oriented approach to support dynamic evolution in distributed applications**', a paper presented to *the 19th International Conference on Information Systems Development (ISD 2010)*, Prague, Czech Republic, 25-27 Aug 2010.

- part of Chapter 6 in this thesis

Fung, K.H. and Low, G.C. 2003, '**Design notation for dynamic evolution in component based distributed systems**', *Proceedings of the 7th International Conference on Enterprise Distributed Object Computing (EDOC 2003)*, Brisbane, Australia, 16-19 Sep 2003. DOI: 10.1109/EDOC.2003.1233859

- some of the concepts and notations from Appendix C.2 in this thesis

ACKNOWLEDGEMENTS

Doing a PhD study invariably involves other people and it is a great opportunity to recognise and acknowledge them here lest I forget their help in a few years' time.

First and foremost, I am most grateful to my supervisor Emeritus Professor Graham Low for his perseverance and interest in his commitment, encouragement, guidance, rigour and support in my PhD study. After searching for a candidate supervisor for a few months, I was fortunate to have found him at the School of Information Systems, Technology and Management, available and willing to take me on as a part-time student and to embark on such a long journey of part-time PhD study. He gradually mentored me to become a researcher by showing insightful knowledge in conducting research and innovative work. This also expanded my professional skill set which has otherwise been gained from work experience and built on my computer science and software engineering background. My thanks and bow to him.

I would also like to thank my co-supervisors at various stages of my PhD study: my current co-supervisor Associate Professor Aybüke Aurum for sourcing a sponsor in the case study for this research and her insightful comments on my thesis, and my past co-supervisors Associate Professors Pradeep Ray and Bob Edmundson for their involvement in my candidature reviews. It is also my pleasure to acknowledge Dr. Louise Fitzgerald at the Education Development Unit, the Australian School of Business, who provided constructive comments on earlier drafts of this thesis through a different lens to further improve its quality.

My gratitude is also extended to Associate Professor Fethi Rabhi who helped me in the surveys and the case study, to Patricia Hartley, Margaret Lo and Cathy Sharpley who have ensured the School's administrative work has run smoothly to support my PhD study, and to many fellow research students (past and present) for exchanging knowledge, news and gossip with me.

This research would not have been possible without the experienced practitioners from industry and researchers from academia who participated in various activities of this

PhD study, including surveys, meetings, documentation reviews and the case study. They have provided invaluable data and feedback which also improve the quality and credibility of this research's outcomes. Unfortunately, for reasons of anonymity and ethics, they are not individually acknowledged here.

Lastly, I am indebted to my father Kwok Woon Fung, my mother Ying Chan and my brother Timothy Fung for their unequivocal, tireless, loving support for and being tolerant of my PhD study.

TABLE OF CONTENTS

ABSTRACT.....	I
PUBLICATIONS	II
ACKNOWLEDGEMENTS.....	III
TABLE OF CONTENTS.....	V
LIST OF TABLES	X
LIST OF FIGURES.....	XII
ABBREVIATIONS AND ACRONYMS	XIV
CHAPTER 1. INTRODUCTION	1
1.1 BACKGROUND AND MOTIVATION	2
1.1.1 Dynamic Evolution.....	3
1.1.2 Role of Composition-Based Distributed Application	4
1.1.3 Role of Development Methodology	4
1.1.4 Role of Method Engineering	6
1.2 RESEARCH OBJECTIVE	7
1.3 RESEARCH DESIGN	8
1.4 CONTRIBUTIONS.....	9
1.5 RESEARCH DELIMITATIONS.....	11
1.6 CONCLUSION AND THESIS OUTLINE	12
CHAPTER 2. REVIEW OF DYNAMIC EVOLUTION AND METHODOLOGY.....	13
2.1 DEMYSTIFYING DYNAMIC EVOLUTION	14
2.1.1 System and Software Evolution.....	15
2.1.2 Types of Evolution in Applications	16
2.1.3 Release, Transformation and Dynamic Change.....	16
2.1.4 Evolution Space	18
2.1.5 Evolution Quality	19
2.2 COMPOSITION-BASED DISTRIBUTED APPLICATION	20
2.2.1 Component-Based Distributed Application	21
2.2.1.1 Software component	22
2.2.1.2 Connector	23
2.2.2 Service-Oriented Architecture Based Application.....	23
2.2.3 Embracing Dynamic Evolution.....	25
2.3 EVALUATION FRAMEWORKS.....	27
2.4 METHODOLOGIES SUPPORTING COMPOSITION-BASED DEVELOPMENT	31
2.4.1 ASG	34
2.4.2 CBDI-SAE	35
2.4.3 Erl	36
2.4.4 Oreizy et al. (AEM).....	38
2.4.5 SUPER.....	40
2.4.6 Rational Unified Process	41
2.4.7 EPIC	43
2.4.8 Papazoglou and van den Heuvel (P&H).....	44
2.4.9 Catalysis	46
2.4.10 OPEN Process Framework.....	47
2.4.11 Select Perspective.....	49
2.4.12 Kobra.....	50
2.4.13 SeCSE	52
2.4.14 Observations from Selected Methodologies.....	54
2.5 SITUATIONAL METHOD ENGINEERING	56
2.6 CONCLUSION	57
CHAPTER 3. RESEARCH DESIGN.....	59
3.1 DESIGN SCIENCE RESEARCH AND THE JUSTIFICATION OF THIS RESEARCH.....	59
3.2 RESEARCH ACTIVITIES AND PHASES.....	61
3.2.1 Phase 1: Determine important dynamic evolution requirements.....	62
3.2.2 Phase 2: Develop support for dynamic evolution	66
3.2.2.1 Task 2.1: Identify method fragments from relevant methodologies	67
3.2.2.2 Task 2.2: Develop method fragments	67
3.2.3 Phase 3: Evaluate and refine support for dynamic evolution	68
3.2.3.1 Task 3.1: Conduct an expert review of Continuum	68
3.2.3.2 Task 3.2: Apply Continuum to a case study	69
3.3 CONCLUSION	70

CHAPTER 4. DEVELOPMENT OF DYNAMIC EVOLUTION QUALITY FACTORS.....	71
4.1 STEP 1: SYNTHESIS OF DYNAMIC EVOLUTION QUALITY FACTORS	72
4.1.1 Soundness of Change	74
4.1.1.1 Completeness	74
4.1.1.2 Consistency	75
4.1.1.3 Correctness	76
4.1.2 Infusibility of Change	76
4.1.2.1 Efficiency	77
4.1.2.2 Locality.....	77
4.1.2.3 Maintainability	77
4.1.2.4 Transparency	78
4.1.3 Changeability of Application	78
4.1.3.1 Configurability	79
4.1.3.2 Coordination.....	79
4.1.3.3 Flexibility	79
4.1.3.4 Loose Coupling	80
4.1.3.5 Separation of Concerns.....	80
4.1.4 Robustness of Application	80
4.1.4.1 Fault Tolerance	81
4.1.4.2 Recoverability	81
4.1.4.3 Reliability	81
4.1.4.4 Safety.....	81
4.1.4.5 Security.....	82
4.1.5 Existing Evaluation Frameworks.....	82
4.1.6 Summary and Discussion.....	83
4.2 STEP 2: WEB SURVEY ASSESSMENT	88
4.2.1 Pilot Tests	90
4.2.2 Data Collection	90
4.2.3 Additional Dynamic Evolution Quality Factors/Attributes	91
4.2.4 Importance Rating of Dynamic Evolution Quality Factors.....	94
4.2.5 Software Practice and Project Information	98
4.3 STEP 3: DYNAMIC EVOLUTION QUALITY FACTORS EXTENSION FROM METHODOLOGICAL PERSPECTIVE	99
4.4 STEP 4: EVALUATION OF SUPPORT FOR DYNAMIC EVOLUTION QUALITY FACTORS IN METHODOLOGIES	103
4.5 RELATED WORK.....	108
4.5.1 Related Quality Standards.....	108
4.5.2 Related Quality Models	110
4.6 CONCLUSION	112
CHAPTER 5. DEVELOPMENT OF DYNAMIC CHANGE REQUIREMENTS.....	114
5.1 STEP 1: SYNTHESIS OF DYNAMIC CHANGE REQUIREMENTS	115
5.1.1 Modelling Related Dynamic Change Requirements	117
5.1.2 Work Related Dynamic Change Requirements.....	119
5.1.3 Existing Evaluation Frameworks.....	121
5.1.4 Summary and Discussion	122
5.2 STEP 2: SURVEY ASSESSMENT.....	125
5.2.1 Pilot Tests	126
5.2.2 Data Collection	126
5.2.3 Additional Dynamic Change Requirements.....	126
5.2.4 Importance Ratings of Dynamic Change Requirements.....	128
5.3 STEP 3: DYNAMIC CHANGE REQUIREMENTS EXTENSION FROM METHODOLOGICAL PERSPECTIVE	133
5.4 STEP 4: EVALUATION OF SUPPORT FOR DYNAMIC CHANGE REQUIREMENTS IN METHODOLOGIES	134
5.5 CONCLUSION	137
CHAPTER 6. DEVELOPMENT OF CONTINUUM.....	138
6.1 REQUIREMENTS FOR CONTINUUM	139
6.2 TASK 2.1: METHOD FRAGMENT IDENTIFICATION FROM RELEVANT METHODOLOGIES.....	142
6.3 TASK 2.2: METHOD FRAGMENT DEVELOPMENT	147
6.3.1 Electronic Product Catalogue Platform	148
6.3.2 Overview of Continuum	149
6.3.3 Dynamic Evolution Metamodel	152
6.3.4 Dynamic Change Method Fragments	157
6.3.4.1 Fragments for Application Lifecycle Analysis	157
6.3.4.2 Fragments for Transformation Identification	160
6.3.4.3 Fragments for Transformation Design	164
6.3.4.4 Fragments for Transformation Agent Design	170
6.3.5 Dynamic Evolution Quality Method Fragments	172
6.3.6 Producer Method Fragments	178
6.3.7 Usage Guidelines	178
6.4 TRACEABILITY OF REQUIREMENTS AND METHOD FRAGMENTS	181
6.5 CONCLUSION	188

CHAPTER 7. EVALUATION AND REFINEMENT	189
7.1 TASK 3.1 EXPERT REVIEW OF CONTINUUM	189
7.1.1 Selection of Experts	189
7.1.2 Procedure	190
7.1.3 Results	191
7.2 TASK 3.2 CASE STUDY APPLICATION OF CONTINUUM	192
7.2.1 Case Description	192
7.2.2 Selection of Participants	194
7.2.3 Procedure	195
7.2.3.1 Project Initiation	196
7.2.3.2 Applying Continuum	197
7.2.3.3 Evaluating Continuum	198
7.2.3.4 Refining Continuum	198
7.2.4 Results	199
7.2.4.1 Sub-Phase Configuration for Application of Continuum	199
7.2.4.2 Dealing with Dynamic Evolution Issues	203
7.2.4.3 Usefulness	204
7.2.4.4 Usability	206
7.2.5 Lessons Learned	211
7.3 CONCLUSION	212
CHAPTER 8. CONCLUSIONS	213
8.1 SUMMARY OF INVESTIGATIONS	213
8.2 CONTRIBUTIONS OF RESEARCH	216
8.3 VALIDITY AND RELIABILITY THREATS	218
8.3.1 Conclusion Validity	218
8.3.2 Construct Validity	218
8.3.3 Internal Validity	219
8.3.4 External Validity	220
8.3.5 Reliability	221
8.4 RECOMMENDATIONS FOR FUTURE WORK	221
8.4.1 Extension	221
8.4.2 Applying Continuum to a Variety of Applications and Domains	222
8.4.3 Tool Implementation	222
8.5 CONCLUDING REMARKS	223
BIBLIOGRAPHY	224
APPENDIX A. SYSTEMATIC LITERATURE REVIEW	246
APPENDIX B. FEATURE ANALYSIS RESULTS OF DEVELOPMENT METHODOLOGIES	251
B.1 EVALUATION RESULTS OF SUPPORT FOR DYNAMIC EVOLUTION QUALITY FACTORS	251
B.1.1 Soundness of Change	252
B.1.1.1 Completeness	252
B.1.1.2 Consistency	252
B.1.1.3 Correctness	254
B.1.2 Infusibility of Change	254
B.1.2.1 Efficiency	254
B.1.2.2 Locality	255
B.1.2.3 Maintainability	255
B.1.2.4 Transparency	256
B.1.3 Flexibility of Application	257
B.1.3.1 Autonomy	257
B.1.3.2 Configurability	257
B.1.3.3 Coordination	257
B.1.3.4 Extensibility	257
B.1.3.5 Flexibility	257
B.1.3.6 Loose Coupling	258
B.1.3.7 Separation of Concerns	258
B.1.4 Robustness of Application	259
B.1.4.1 Fault tolerance	259
B.1.4.2 Recoverability	260
B.1.4.3 Reliability	260
B.1.4.4 Safety	261
B.1.4.5 Security	262
B.2 EVALUATION RESULTS OF SUPPORT FOR MODELLING RELATED DYNAMIC CHANGE REQUIREMENTS	262
B.3 EVALUATION RESULTS OF SUPPORT FOR WORK RELATED DYNAMIC CHANGE REQUIREMENTS	265
APPENDIX C. DETAILED SPECIFICATIONS FOR CONTINUUM	268
C.1 PRODUCER METHOD FRAGMENTS	268
C.1.1 Dynamic Evolution Analyst	268
C.1.2 Dynamic Evolution Designer	269

C.1.3 Runtime Application Discovery Tool.....	269
C.2 WORK PRODUCT FRAGMENTS.....	270
C.2.1 Model Unit Fragments.....	270
C.2.1.1 Application.....	270
C.2.1.2 ApplicationLifecycle.....	271
C.2.1.3 ChangeCase.....	271
C.2.1.4 Generation.....	272
C.2.1.5 Impact.....	272
C.2.1.5.1 Custom attribute type ImpactType.....	273
C.2.1.5.2 Custom attribute type ImpactDisruptionLevel.....	273
C.2.1.6 OperationalProfile.....	273
C.2.1.7 Policy.....	274
C.2.1.8 PerformanceProfile.....	274
C.2.1.9 Resource.....	275
C.2.1.10 ResourceProfile.....	275
C.2.1.11 ServicingPolicy.....	275
C.2.1.11.1 Custom attribute type ServicingPolicyType.....	276
C.2.1.12 Stage.....	276
C.2.1.13 TransformableItem.....	276
C.2.1.13.1 Custom attribute type TransformableItemStateType.....	277
C.2.1.14 Transformation.....	278
C.2.1.15 TransformationAction.....	278
C.2.1.15.1 Custom attribute type TransformationActionType.....	279
C.2.1.16 TransformationAgent.....	279
C.2.1.17 TransformationException.....	280
C.2.1.18 TransformationExceptionResolution.....	280
C.2.1.19 TransitionalPeriod.....	280
C.2.1.20 Zone.....	281
C.2.1.21 ZoningPolicy.....	281
C.2.2 Diagram and Document Fragments.....	282
C.2.2.1 Application Lifecycle Diagram and Notation.....	283
C.2.2.2 Dynamic Application Change Document.....	285
C.2.2.3 Dynamic Evolution Quality Inspection Report.....	286
C.2.2.4 Dynamic Evolution Quality Problem Analysis Report.....	293
C.2.2.5 Dynamic Evolution Quality Profile Report.....	293
C.2.2.6 New and Replacement Transformable Item Catalogue.....	298
C.2.2.7 State Map.....	299
C.2.2.8 Structural Configuration - Notational Extensions.....	300
C.2.2.9 Transformation Diagram and Notation.....	302
C.2.2.10 Transformation Orchestration Diagram and Notation.....	304
C.2.2.11 Zone Change Document.....	306
C.3 WORK UNIT FRAGMENTS.....	307
C.3.1 Process and Task Fragments.....	307
C.3.1.1 Application Lifecycle Analysis.....	308
C.3.1.1.1 Identify As-Is Runtime Structure.....	309
C.3.1.1.2 Derive Change Cases.....	310
C.3.1.1.3 Extend Application Lifecycle.....	310
C.3.1.2 Dynamic Evolution Quality Management.....	311
C.3.1.2.1 Define Dynamic Evolution Quality Needs.....	312
C.3.1.2.2 Assess Dynamic Evolution Quality.....	313
C.3.1.2.3 Analyse Dynamic Evolution Quality Problems.....	314
C.3.1.2.4 Improve Dynamic Evolution Quality.....	315
C.3.1.3 Transformation Agent Design.....	317
C.3.1.3.1 Identify Transformation Agents.....	318
C.3.1.3.2 Define Transformation Orchestration.....	318
C.3.1.4 Transformation Design.....	319
C.3.1.4.1 Identify New and Replacement Transformable Items.....	319
C.3.1.4.2 Identify Changes to Zones.....	320
C.3.1.4.3 Define Servicing Policies.....	321
C.3.1.4.4 Develop Transformation.....	321
C.3.1.5 Transformation Identification.....	322
C.3.1.5.1 Define To-Be Runtime Structure.....	322
C.3.1.5.2 Refine Change Cases.....	323
C.3.1.5.3 Identify Transformations.....	323
C.3.2 Technique Fragments.....	324
C.3.2.1 Change Case Modelling.....	324
C.3.2.2 Change Case Partitioning and Ordering.....	326
C.3.2.3 Dynamic Change Impact Analysis.....	327
C.3.2.4 Dynamic Evolution Safety Risk Management.....	328
C.3.2.5 Dynamic Recomposition.....	332
C.3.2.6 Dynamic Refactoring.....	333
C.3.2.7 Dynamic Transformable Item Change.....	334
C.3.2.8 Dynamic Transformable Item (Re)binding.....	336

C.3.2.9 Resource Profile Modelling	337
C.3.2.10 Root Cause Analysis	338
C.3.2.11 Secure and Reliable Transformation Agent Coordination	340
C.3.2.12 Start-up State Configuration.....	341
C.3.2.13 Transformation Agent Disposition	342
C.3.2.14 Transformation Exception Management.....	343
C.3.2.15 Transformation Mining	345
C.3.2.16 Transformation Orchestration and Agent Coordination.....	347
C.3.2.17 Reused Technique Fragments.....	350
C.3.2.17.1 Dynamic Change Localisation	350
C.3.2.17.2 Dynamic Security Policy and Enforcement Management	351
C.3.2.17.3 Dynamic Transformable Item Adaptation	351
C.3.2.17.4 Dynamic Variation Management	351
C.3.2.17.5 Dynamic Workflow Change.....	352
C.3.2.17.6 Dynamic Wrapper.....	352
C.3.2.17.7 Inspections	354
C.3.2.17.8 Loose Coupling	355
C.3.2.17.9 Performance Profile Modelling.....	355
C.3.2.17.10 Recovery Blocks.....	356
C.3.2.17.11 Runtime Structure Recovery.....	356
C.3.2.17.12 Testability Analysis and Improvement.....	357
C.3.2.17.13 Transformable Item Autonomy.....	357
C.3.2.17.14 Transformable Item Mediation and Channelling	358
C.3.2.17.15 Transformable Item Regression Testing	358
APPENDIX D. CASE STUDY RESULTS OF APPLYING CONTINUUM.....	359
D.1 APPLICATION LIFECYCLE ANALYSIS OUTCOMES	359
D.1.1 Distributed Property Valuation - Generation V1	359
D.1.2 Change Cases.....	361
D.1.3 Application Lifecycle	367
D.2 TRANSFORMATION IDENTIFICATION OUTCOMES.....	368
D.3 TRANSFORMATION AGENT DESIGN OUTCOMES.....	375
D.4 TRANSFORMATION DESIGN OUTCOMES.....	379
D.4.1 New and Replacement Transformable Items.....	379
D.4.2 Zone Changes	382
D.4.3 Detailed Design of Transformations	384
D.5 DYNAMIC EVOLUTION QUALITY MANAGEMENT OUTCOMES	389
APPENDIX E. QUESTIONNAIRE FORMS.....	395
E.1 SURVEY ON DYNAMIC EVOLUTION QUALITY FACTORS.....	395
E.1.1 Participant Information Sheet	395
E.1.2 Survey Web Pages	396
E.1.2.1 Instructions	396
E.1.2.2 Terminology.....	397
E.1.2.3 Questionnaire overview	398
E.1.2.4 Rating questions - page 1	398
E.1.2.5 Rating questions - page 2	400
E.1.2.6 Rating questions - page 3	401
E.1.2.7 Respondent profile.....	401
E.1.2.8 Software practices	402
E.1.2.9 Project information	402
E.1.2.10 Software development activities.....	403
E.1.2.11 Questionnaire submission.....	404
E.2 SURVEY ON DYNAMIC CHANGE REQUIREMENTS	404
E.2.1 Participant Information Sheet	404
E.2.2 Questionnaire	405
E.2.2.1 Key terminology.....	405
E.2.2.2 Modelling features	406
E.2.2.3 Work related features.....	407
E.3 EXPERT REVIEW OF DEVELOPMENT METHODOLOGY EXTENSION TO SUPPORT DYNAMIC EVOLUTION.....	409
E.3.1 Participant Information Sheet	409
E.3.2 Questionnaire	410
E.3.2.1 Key terminology.....	410
E.3.2.2 Instructions for completing the questionnaire	410
E.3.2.3 Strengths of Continuum	411
E.3.2.4 Suggested improvements for Continuum	411
E.4 CASE STUDY EVALUATION ON DEVELOPING DYNAMIC EVOLUTION FOR A SOFTWARE PROTOTYPE.....	411
E.4.1 Participant Information Statement	411
E.4.2 Questionnaire	413
E.4.2.1 Instructions for completing the questionnaire	413
E.4.2.2 Usefulness.....	413
E.4.2.3 Dynamic evolution issues encountered	413

E.4.2.4 Completeness.....	414
E.4.2.5 Usability.....	414
E.4.2.5.1 Evaluation of metamodel	414
E.4.2.5.2 Evaluation of Application Lifecycle Analysis process and related fragments.....	415
E.4.2.5.3 Evaluation of Transformation Identification process and related fragments	416
E.4.2.5.4 Evaluation of Transformation Agent Design Process and related fragments.....	416
E.4.2.5.5 Evaluation of Transformation Design process and related fragments	417
E.4.2.5.6 Evaluation of Dynamic Evolution Quality Management process and related fragments	418
APPENDIX F. REFINEMENTS TO CONTINUUM.....	421

LIST OF TABLES

Table 1.1 Examples of distributed applications benefiting from dynamic evolution.....	3
Table 1.2 Example potential benefits of contributions to method users	10
Table 1.3 Potential benefits of contributions to method engineers.....	10
Table 2.1 Examples of specialisation of evolution studies	14
Table 2.2 Component-based vs. service-oriented elements	26
Table 2.3 Terminology mapping between SEMDM (ISO/IEC 24744) and reviewed methodologies	32
Table 2.4 Erl's strategies	37
Table 2.5 Support for composition and evolution in reviewed methodologies.....	54
Table 4.1 Source of literature examined for quality factor synthesis	73
Table 4.2 Potential quality factors/attributes selected from reviewed evaluation frameworks	83
Table 4.3 Origins of quality factors from literature and evaluation frameworks	83
Table 4.4 Analysis results of potential quality issues suggested by respondents	91
Table 4.5 Analysis results of additional quality attributes synthesised from the literature (cf. Table 4.3)	92
Table 4.6 Descriptive statistics and results of Wilcoxon one-sample signed-rank test on quality factors....	95
Table 4.7 Results of Wilcoxon signed-rank tests for matched pairs on quality factors	96
Table 4.8 Analysis of potential quality factors elicited from reviewed methodologies	100
Table 4.9 Evaluation results of methodological support for quality factors.....	104
Table 4.10 Correspondence between ISO/IEC 9126-1 and ISO/IEC 25010's definitions for maintainability and the dynamic evolution quality model	109
Table 4.11 Dynamic evolution quality factor requirements investigated in Task 1.1	113
Table 5.1 Source of literature examined for dynamic change requirement synthesis	115
Table 5.2 Potential dynamic change requirements selected from reviewed evaluation frameworks.....	121
Table 5.3 Modelling related dynamic change requirements from Step 1	122
Table 5.4 Work related dynamic change requirements from Step 1	123
Table 5.5 Analysis results of potential dynamic change requirements suggested by respondents	127
Table 5.6 Analysis results of additional dynamic change requirements synthesised from the literature (cf. Table 5.4)	127
Table 5.7 Descriptive statistics and Wilcoxon one-sample signed-rank test results for modelling related dynamic change requirements from Step 1 (cf. Table 5.3)	128
Table 5.8 Descriptive statistics and Wilcoxon one-sample signed-rank test results for work related dynamic change requirements from Step 1 (cf. Table 5.4).....	129
Table 5.9 Results of Wilcoxon signed-rank test for matched pairs on modelling related dynamic change requirements from Step 1 (cf. Table 5.3)	130
Table 5.10 Results of Wilcoxon signed-rank test for matched pairs on work related dynamic change requirements from Step 1 (cf. Table 5.4)	131
Table 5.11 Analysis of potential dynamic change requirements elicited from reviewed methodologies ...	133
Table 5.12 Feature analysis results of selected methodologies.....	135
Table 5.13 Dynamic change requirements investigated in Task 1.2	137
Table 6.1 Summary of dynamic evolution quality factor requirements for Continuum (cf. Table 4.11).....	140
Table 6.2 Summary of dynamic change requirements for Continuum (cf. Table 5.13)	141
Table 6.3 Features for reuse/enhancement for dynamic change requirements	144
Table 6.4 Features for reuse/enhancement for dynamic evolution quality factors	145
Table 6.5 Techniques used in Application Lifecycle Analysis	158
Table 6.6 EPCP: key change cases.....	159
Table 6.7 Techniques used in Transformation Identification.....	161
Table 6.8 EPCP: impact set for change case CC1	163
Table 6.9 EPCP: responsible transformations for refined change cases	164
Table 6.10 Techniques used in Transformation Design.....	165

Table 6.11 EPCP: changes to zones	167
Table 6.12 Techniques used in Transformation Agent Design	171
Table 6.13 Technique(s) used in task Assess Dynamic Evolution Quality	173
Table 6.14 Technique(s) used in task Analyse Dynamic Evolution Quality Problems	174
Table 6.15 Recommended tasks/techniques used in Task Improve Dynamic Evolution Quality	174
Table 6.16 Traceability between important dynamic change requirements (Table 6.2) and Continuum's method fragments	181
Table 6.17 Traceability between Continuum quality factor requirements (Table 6.1) and method fragments	184
Table 7.1 Expert review timeline	191
Table 7.2 Expert comments on strengths of Continuum	191
Table 7.3 Case study timeline	199
Table 7.4 Dynamic evolution issues encountered in case study and addressed with Continuum	203
Table 7.5 Comparison of Continuum and in-house methodology in tackling recurred dynamic evolution issues	205
Table 7.6 Understandability ratings for Continuum's metamodel	207
Table 7.7 Usability ratings for Continuum's work unit and work product fragments	207
Table 7.8 Understandability ratings for work products developed with Continuum	210
Table 8.1 Important dynamic evolution requirements	214
Table Appendix A.1 Selected Journal and conference proceedings for feature requirement synthesis	247
Table Appendix B.1 Scale points for scoring a methodology's feature	251
Table Appendix B.2 Evaluation of support for modelling related change requirements (see Sections 5.1.4 and 5.2.3 for definitions)	263
Table Appendix B.3 Evaluation of support for work related change requirements (see Sections 5.1.4 and 5.2.3 for definitions)	265
Table Appendix C.1 Enumerated values of ImpactType	273
Table Appendix C.2 Enumerated values of ImpactDisruptionLevel	273
Table Appendix C.3 Enumerated values of ServicingPolicyType	276
Table Appendix C.4 Enumerated values of TransformableItemStateType	277
Table Appendix C.5 Enumerated values of TransformationActionType	279
Table Appendix C.6 Model Unit fragments and their usage in model artefacts	282
Table Appendix C.7 Notations for Application Lifecycle Diagram	284
Table Appendix C.8 Example documentation of change case and impact	286
Table Appendix C.9 Example documentation of change case and transformation	286
Table Appendix C.10 Template for dynamic evolution quality inspection report	287
Table Appendix C.11 Example dynamic evolution quality problem analysis report	293
Table Appendix C.12 Template for dynamic evolution quality profile report	294
Table Appendix C.13 Example documentation of simple new and replacement transformable item catalogue	298
Table Appendix C.14 Example documentation of resource profile	299
Table Appendix C.15 Tabular representation of state map in Figure Appendix C.4	300
Table Appendix C.16 Dynamic evolution related notations for structural configurations	301
Table Appendix C.17 Notations for Transformation Diagram	303
Table Appendix C.18 Notations for Transformation Orchestration Diagram	304
Table Appendix C.19 Example zone change document	306
Table Appendix C.20 Summary of quality factors supported by Continuum	312
Table Appendix C.21 Summary of quality factors supported by Continuum	312
Table Appendix C.22 Work units for addressing particular aspects of quality factors	315
Table Appendix C.23 Gap operators and templates for expressing change case purposes	325
Table Appendix C.24 Areas of dynamic change impact analysis	328
Table Appendix C.25 Assets and Harms	329
Table Appendix C.26 Abnormal dynamic change events	330
Table Appendix C.27 Dynamic evolution safety design mechanisms	332
Table Appendix C.28 Suggested values for dynamic evolution quality defect/issue attributes	339
Table Appendix D.1 DPV: transformable items in generation V1	360
Table Appendix D.2 DPV: dynamic application change document	363
Table Appendix D.3 DPV: identified generations	368
Table Appendix D.4 DPV: change cases for progressing V1 to V1.1beta	370
Table Appendix D.5 DPV: change cases for progressing V1.1beta to V1.1	371
Table Appendix D.6 DPV: change cases for progressing V1.1 to V1.2	372
Table Appendix D.7 DPV: change cases for progressing V1.2 to V1.3	374
Table Appendix D.8 DPV: change cases for progressing V1.3 to V2	375
Table Appendix D.9 DPV: new and replacement transformable item catalogue	380
Table Appendix D.10 DPV: state map from JobAppointmentWS_V1 to its stub	382

Table Appendix D.11 DPV: zone change document	382
Table Appendix D.12 DPV: applying transformation patterns to transformation design	385
Table Appendix D.13 DPV: dynamic evolution quality profile	390
Table Appendix D.14 DPV: dynamic evolution quality inspection, assessment and improvement results for defects/issues	390
Table Appendix D.15 DPV: use of Continuum techniques in the Task Improve Dynamic Evolution Quality	392
Table Appendix F.1 Suggested improvements from expert review and subsequent refinements to Continuum	421
Table Appendix F.2 Suggested improvements from case study and actual refinements to Continuum	428

LIST OF FIGURES

Figure 1.1 Development phases for this research.....	8
Figure 2.1 Conceptual model for review sections in Chapter 2	13
Figure 2.2 Release-change-transformation relationship in dynamic evolution	17
Figure 2.3 Basic service-oriented model.....	24
Figure 2.4 State machine for using a service.....	24
Figure 2.5 Precedence relationships of reviewed methodologies	34
Figure 2.6 Erl's (2005) agile lifecycle model	38
Figure 2.7 Oreizy et al.'s. (1999) lifecycle model	39
Figure 2.8 SUPER's phases and layers	41
Figure 2.9 An example of RUP's evolution cycles	42
Figure 2.10 EPIC's trade space	43
Figure 2.11 Phases of Papazoglou and van den Heuvel's (2006) methodology	45
Figure 2.12 Application phases in OPF.....	48
Figure 2.13 An expanded view of Select Perspective's supply-manage-consume model.....	50
Figure 2.14 SeCSE composition methodology	53
Figure 3.1 Development phases adopted for this research (expanded from Figure 1.1).....	62
Figure 3.2 Research tasks, steps and techniques used in various phases	63
Figure 4.1 Information flow in Phase 1 for determining dynamic evolution quality factors	71
Figure 4.2 Steps in Task 1.1 of Phase 1	72
Figure 4.3 Dynamic evolution quality factors and categories synthesised from the literature	74
Figure 4.4 Quality factor importance rankings before and after expert review	97
Figure 4.5 Quality factor importance rankings before and after methodology extension.....	102
Figure 5.1 Information flow in Phase 1 for determining dynamic change requirements	114
Figure 5.2 Steps in Task 1.2 of Phase 1	115
Figure 5.3 Dynamic change requirements and categories synthesised from the literature	117
Figure 5.4 Example transitional forms	119
Figure 5.5 Importance rankings of dynamic change requirements after expert review	132
Figure 6.1 Information flow in Phase 2 for developing Continuum	138
Figure 6.2. EPCP: current structure.....	149
Figure 6.3. SEMDM and Continuum components.....	150
Figure 6.4. Application lifecycle, transitional periods and transformations	152
Figure 6.5 Dynamic evolution metamodel.....	153
Figure 6.6 Structural foundation classes.....	154
Figure 6.7 Application lifecycle related model unit fragments	155
Figure 6.8 Transitional period related model unit fragments	155
Figure 6.9 Transformation related model unit fragments	156
Figure 6.10 Policy related model unit fragments.....	157
Figure 6.11 Work units for Application Lifecycle Analysis.....	158
Figure 6.12. EPCP: application lifecycle diagram	160
Figure 6.13. Work units for Transformation Identification.....	160
Figure 6.14 EPCP: to-be generations after transitional periods a and b	163
Figure 6.15 EPCP: generation v2 after transitional period c	164
Figure 6.16. Work units for Transformation Design.....	165
Figure 6.17 EPCP: applying deployment transformation pattern to Catalogue Service2	168
Figure 6.18 EPCP: applying removal transformation pattern to Catalogue Service	168
Figure 6.19 EPCP: transformation design for "Catalogue Service2 Reconfiguration"	169

Figure 6.20 EPCP: delegating different search requests to WebUI and WebUI2.....	170
Figure 6.21. Work units for Transformation Agent Design	170
Figure 6.22 EPCP: orchestration designs for transformation agents	172
Figure 6.23. Work units for Dynamic Evolution Quality Management	173
Figure 6.24 EPCP: use of coordination agent during transitional period a	177
Figure 6.25. Example analysis and design project lifecycle with Continuum process fragments	180
Figure 6.26 Precedence relationships of reviewed methodologies and Continuum (extended from Figure 2.5)	188
Figure 7.1 Information flow in Phase 3 for evaluating and refining Continuum	189
Figure 7.2 Case Study's lifecycle diagram	195
Figure 7.3 Configuration for case study's "Application of Continuum" phase	197
Figure 7.4 Configuration for "Lifecycle Definition" sub-phase	200
Figure 7.5 DPV: generation overview	200
Figure 7.6 Configuration for "Transformation Identification", "Transformation Agent Definition" and "Transformation Design" sub-phases.....	201
Figure 7.7 Configuration for "Quality Improvement" sub-phase	203
Figure 8.1 Major relationships among research artefacts developed in this research.....	216
Figure Appendix C.1 Example hierarchical zones and illegal zones.....	281
Figure Appendix C.2 Example Petri nets in action	284
Figure Appendix C.3 Example application lifecycle diagrams.....	285
Figure Appendix C.4 Example state map for two UML state machines	300
Figure Appendix C.5 Example use of Structural Configuration - Notational Extensions	302
Figure Appendix C.6 Example transformation actions	304
Figure Appendix C.7 Example transformation orchestration diagram	306
Figure Appendix C.8 Example fault tree for replacement transformable item related transformation events	330
Figure Appendix C.9 Example fault tree for data related dynamic change events	331
Figure Appendix C.10 Example safety risk derived from hazard	331
Figure Appendix C.11 Addition, removal and property change transformation patterns	335
Figure Appendix C.12 Replacement using three transformation patterns.....	336
Figure Appendix C.13 (Re)binding transformation patterns	337
Figure Appendix C.14 Example uses of request and acknowledgement between two transformation agents	340
Figure Appendix C.15 Example of timing out a subordinate agent's transformation	341
Figure Appendix C.16 Example transformation agent hierarchy	343
Figure Appendix C.17 Example taxonomy of abnormal transformation events (exceptions labelled with *)	344
Figure Appendix C.18 Example transformation with exception declaration and rollback	345
Figure Appendix C.19 Example application used to illustrate Transformation Orchestration and Agent Coordination	347
Figure Appendix C.20 Example development of transformation orchestration	349
Figure Appendix D.1 DPV: generation V1.....	360
Figure Appendix D.2 DPV: application lifecycle diagram before process "Transformation Identification".	367
Figure Appendix D.3 DPV: updated application lifecycle diagram after new change cases identified in Process "Transformation Identification"	368
Figure Appendix D.4 DPV: generation V1.1beta.....	369
Figure Appendix D.5 DPV: generation V1.1.....	371
Figure Appendix D.6 DPV: generation V1.2.....	372
Figure Appendix D.7 DPV: generation V1.3.....	374
Figure Appendix D.8 DPV: generation V2.....	375
Figure Appendix D.9 DPV: disposition of transformation agents in different zones	376
Figure Appendix D.10 DPV: transformation orchestration diagram (V1 to V1.1beta).....	377
Figure Appendix D.11 DPV: transformation orchestration diagram (V1.1beta to V1.1).....	377
Figure Appendix D.12 DPV: transformation orchestration diagram (V1.1 to V1.2).....	378
Figure Appendix D.13 DPV: transformation orchestration diagram (V1.2 to V1.3).....	378
Figure Appendix D.14 DPV: transformation orchestration diagram (V1.3 to V2).....	379
Figure Appendix D.15 DPV: transformation pattern for "tomcat addition"	385
Figure Appendix D.16 DPV: transformation design for "ds_v1: to ds_v2: data replication"	387
Figure Appendix D.17 DPV: transformation design for "ds_v1: to ds_v2: data sync"	387
Figure Appendix D.18 DPV: transformation design for "ja_v1:JobAppointmentWS_V1 recovery"	388

ABBREVIATIONS AND ACRONYMS

<i>Abbreviation / Acronym</i>	<i>Description</i>
BPMN	Business Process Modelling Notation
COTS	commercial-off-the-shelf
IS	information systems
IT	information technology
N/A	not applicable
OO	object-oriented
PDA	personal digital assistant
QoS	quality of service
RM-ODP	Reference Model of Open Distributed Processing, an international standard developed by ISO and ITU-T for large distributed applications
SEMDM	International Standard ISO/IEC 24744:2007 Software Engineering - Metamodel for Development Methodologies
SISTM	The School of Information Systems, Technology and Management, UNSW
SLA	service level agreement
SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
UML	OMG's Unified Modelling Language
UNSW	The University of New South Wales, Australia
UI	user interface
XML	eXtensible Markup Language

Chapter 1. INTRODUCTION

“Change is the law of life. And those who look only to the past or present are certain to miss the future.” - John F. Kennedy, US President

Dynamic evolution is a phenomenon by which applications can be upgraded without the need for shutdown and restart. Their ability to offer their services while minimising the loss of business revenue due to downtime, for instance, is particularly important for critical distributed applications running around the clock. A platform well suited for realising dynamic evolution is the notion of a composition-based distributed application - built of modules weaved into composite structures (Schuster 2008). Dynamically accommodating changes in such an application is accomplished via the assembly and disassembly of modules into and from this kind of application.

Studies have called for evolution to be addressed from the methodological perspective (Bennett & Rajlich 2000; Coyle et al. 2010) and, although there has been a myriad of techniques and tools developed for dynamic evolution, the methodological aspect has been largely ignored. Dynamic evolution is not only poorly understood but also inadequately supported in existing methodologies encompassing composition-based distributed application development (cf. Section 1.1.3).

The objective of this research was thus to develop support for dynamic evolution in composition-based distributed applications from the methodological perspective. It encompassed the identification of features that a software development methodology should possess to tackle dynamic evolution and the development of these features. These features have the flexibility of being incorporated into and thus enhancing existing software development methodologies in which dynamic evolution is not well supported.

The rest of this Chapter is organised as follows. The background and motivation for this research is presented in Section 1.1. Then, the objective of this research is stated in Section 1.2. Afterwards, the research programme undertaken, as determined by the decisions made during the development of this research, is presented in Section 1.3. The contributions and the delimitations of this research are then summarised in Sections 1.4 and 1.5 respectively. Finally, a conclusion of this Chapter and an outline of the rest of this thesis are shown in Section 1.6.

1.1 BACKGROUND AND MOTIVATION

Distributed application technologies are gaining widespread use in organisations to develop distributed applications with architectures spreading from two-tiered (e.g. client-server) to multi-tiered models. This trend is driven largely by, for instance, the need for efficiently utilising geographically dispersed computing resources, making services of distributed applications more readily available to other users and applications on a network, and integrating existing legacy applications to provide enterprise-level services. As modern distributed applications grow in size and complexity integrating a barrage of components and legacy applications, there tend to be more changes (upgrades, fixes, reconfigurations etc.) which become increasingly difficult and costly (Fragopoulou et al. 2010). For instance, as well as reasoning about a change in affected parts of an application, one must analyse if the change potentially interferes with other parts of the application and requires further analysis and changes (Yu et al. 2010). This is because faults caused by improper changes to one part of the application can potentially be catastrophic as the faults may spread to other parts of the application (Yau et al. 1993).

Many distributed applications, such as credit card payment gateways, provide critical services with little or no interruption (Clitherow et al. 2008; Gupta et al. 1996; Oreizy et al. 1999). Shutting them down for changes is costly and not always a desirable option (e.g. average loss of US\$6.5M per hour for financial brokerage systems due to downtime (Jayaswal 2005)). Instead, it is preferable to apply changes to these applications with minimal interruption.

Koskinen (2004) studied a range of software development projects between 1979 and 2000. He observed that software maintenance and evolution costs account for 50% to 90% portion of the overall software development effort. The significance of such costs is consistent with other studies (e.g. Petrenko et al. (2007):65%; Kluth (2004):70-80%; Erlikh (2000):85-90%; and Lientz and Swanson (1980):70%). Lehman and Ramil (2001) further postulate that real world systems that address a problem or an activity are E-type, meaning that they must undergo continual evolution to remain satisfactory to end users and stakeholders. On a different note, the information technology (IT) industry is moving away from the traditional waterfall approach of software development towards iterative and incremental development approaches in which software changes are regularly and predictably released. This is in part driven by the need for greater agility and flexibility with respect to requirement changes (Kruchten

2003). Another driving force behind regular changes is that project risks can be reduced by implementing smaller changes regularly (Gilb 1988).

In summary, the tendency for changes to become frequent, regular and predictable substantiates the need for an application to accommodate changes as required. Making changes to distributed applications gets increasingly harder as they grow in size and complexity with more parts distributed in a network, and as they are unlikely to be able to be shut down for changes. To address these issues, this research argues for accommodating changes in this kind of application without the need for shutdown and restart - dynamic evolution.

1.1.1 Dynamic Evolution

The kinds of changes described in Section 1.1 can be handled with *dynamic* or *runtime evolution*. From a business perspective, dynamic evolution may also increase the flexibility of an application in implementing changes in response to unforeseen demands. Such demands can arise from changing user and business needs, business competitiveness and regulatory compliance (Papazoglou 2008).

Many specialised distributed applications, including the types listed in Table 1.1, will benefit from factoring dynamic evolution into their designs.

Table 1.1 Examples of distributed applications benefiting from dynamic evolution

<i>Application Type</i>	<i>Characteristics</i>	<i>Example</i>
Self-adaptive (Oreizy et al. 1999)	Responsive to environmental changes and demands for new functionality by adjusting behaviour autonomously	Intelligent agents (Morandini et al. 2008)
Long-lived (Morrison et al. 2007)	Services provided continuously	Banking system (Hillman & Warren 2004)
Autonomous (Kramer & Magee 2007)	Human interventions erroneous or not quick enough to react	Autonomous underwater vehicle (Kramer & Magee 2007)
Critical (Wang et al. 2006)	Shut-down prohibitive and costly	Telecommunication system (Buckley et al. 2005)
Time-to-market (Bennett & Rajlich 2000)	Minimum features released initially and early to the market, with frequent upgrades and add-ons	(Online) financial product (Bennett & Rajlich 2000)

Note: All cited references have a theme towards dynamic evolution.

Dynamic evolution can be tackled with hardware- and/or software-based approaches (Segal 2002; Segal & Frieder 1993). In the former, albeit costly and complex, redundant or secondary hardware is firstly loaded with new code. Then, the state and data of the programmes running on the original hardware are transferred to the redundant hardware, followed by rerouting of the network traffic to the secondary hardware. In

contrast, a software-based approach is relatively cost effective and flexible (Wang et al. 2006), involving the alternation of the structure, behaviour and/or states of an executing application to achieve a similar effect. In this research, the term “dynamic evolution” is used synonymously with “dynamic software evolution” since this research undertakes a software-based approach to evolution. Furthermore, the term “distributed applications” is used in preference to “distributed systems” to emphasise the software focus of this research’s treatment of dynamic evolution. One way to embrace dynamic evolution in distributed applications is to leverage the notion of the composition-based approach which is described next. Afterwards, an argument for supporting dynamic evolution from the methodological perspective is made in Section 1.1.3.

1.1.2 Role of Composition-Based Distributed Application

Large distributed applications have been focused on being “assembled from independent and reusable collections of functionality” (Brown et al. 2002). More specifically, “composition-based” software development refers to a paradigm of “weaving” existing or custom-developed applications, (commercial-off-the-shelf) software components, leased or rented applications and services into composite applications (Schuster 2008). Common examples include component-based applications (of components and connectors) (Evans & Dickman 1999) and service-oriented applications (of services and bindings) (Papazoglou & Georgakopoulos 2003). Composition-based applications are *well suited to dynamic evolution* because their structures are fabricated from loosely coupled *parts* - units of functionality - and their *bindings* as elementary building blocks for applications (e.g. Szyperski 2003). Ongoing changes can be accommodated into these applications via elementary operations such as addition, replacement and removal of parts with varying functionality as needed (Zhang et al. 2009). On the other hand, the notion of distribution in which application parts are scattered and interact over a network further facilitates this kind of change by connecting (new) parts to and disconnecting (existing) parts from an application.

1.1.3 Role of Development Methodology

Studies recommend that evolution in applications should be regular, anticipated (Oreizy et al. 1999) and considered from the methodological perspective (Bennett & Rajlich 2000; Coyle et al. 2010). The International Standard ISO/IEC 24744: Software Engineering - Metamodel for Development Methodologies defines a development methodology as the “specification of the process to follow together with the work products to be used and generated, plus the consideration of the people and tools involved...” (ISO/IEC

2007)¹. An organisation with a mature systematic software development approach does not always produce quality software (Pfleeger 1999, p. 34) but without one it would be hard to trace the cause of errors and correct them when things go wrong (Firesmith et al. 1997). Such a mature development approach is even more important to a live application as, once undetected errors are manifested in the application through dynamic evolution, they may immediately affect the application. Furthermore, a mature development approach helps an application to evolve (Abran & Moore 2004) and improves an organisation's productivity and effectiveness in software development (Paulk et al. 1993; SEI 2006).

While there have been many studies into techniques and tools that could be useful in dynamic evolution, the methodological aspect has been largely ignored. For example, in a more general context, Breivold and Crnkovic (2010) performed a systematic literature review of fifty-eight studies addressing software evolution (i.e. without regard to the dynamism of evolution) but found no development process support for software evolution. The lack of understanding and support for dynamic evolution from the methodological perspective is further evident from two evaluations, conducted in this research, of a number of development methodologies supporting composition-based distributed application development (detailed in Sections 4.4 and 5.4). The results suggest a lack of maturity in these methodologies with respect to dynamic evolution. For example, Oreizy et al. (1999) use two process lifecycles to deal with dynamic self-adaptation in software: one to anticipate changes and produce their descriptions, and the other to realise the changes. Their approach emphasises particular architectural styles - which are design patterns for organising parts in an architecture's structure (Garlan et al. 1994) - to embrace dynamic changes but lacks details on implementing changes. In work related to software change, Bohner (1996) defines a process for change impact analysis to ensure all parts of an application are considered when proposing changes. The process involves determining changes required for an application, analysing their impacts on the application, designing and implementing the changes, and retesting the changed application. It makes no provision for analysing and evolving a running application to accommodate dynamic changes. The "staged

¹ In this research, the term "methodology" is synonymous with "method" (ISO/IEC 2007; Jayaratna 1994). However, the term "method" is reserved for use in conventional terms "method engineer", "method engineering" and "method fragment", instead of using "methodology engineer", "methodology engineering" and "methodology fragment". Furthermore, as the context of this research is in software development, both "methodology" and "method" imply "development methodology".

lifecycle” model (Bennett & Rajlich 2000) represents an application’s lifecycle as going through several stages from initial development to close-down and suffers the same drawback as Bohner’s (1996).

There have also been no comprehensive studies identifying what kinds of dynamic changes should be addressed in software development (see Sections 2.4.13, 2.3, 4.3 and 5.3 for evidence). Take the case of an evaluation framework which comprises a “checklist” (Siau & Rossi 2007) of ideal methodology feature requirements or criteria used to evaluate the extent to which alternative methodologies address the feature requirements. A feature requirement for a methodology defines a problem or an outcome that methodology or method users² (Nuseibeh et al. 1996) intend to address for a particular activity, task or objective (Kitchenham 1996; Wasserman et al. 1983). A number of evaluation frameworks have been proposed but none specifically cover dynamic evolution (see Section 5.1.3 for review).

Quality is another important consideration in dynamic evolution since dynamic evolution poses a number of particular challenges to product quality. *Low-quality* changes may not only compromise important features of a running application (e.g. security), but also make it harder to evolve in the future. In dynamic evolution, such a compromise materialises in the application as soon as dynamic changes to the application has been made. This is in contrast to static changes in which an application can be taken offline for repair and tested for errors before being placed into production, hopefully, error free. While quality plays an important role in information systems (IS) and their development, little attention has been paid in the literature to quality in dynamic evolution.

1.1.4 Role of Method Engineering

To embrace support for dynamic evolution in methodologies, one can envisage *method engineering* and *method fragment* development. Method engineering is a discipline for the design, construction and adaptation of methodologies for software development, and can be achieved through the use of method fragments (Brinkkemper 1996; Henderson-Sellers 2003). Method fragments³ - a.k.a. methodology elements (Henderson-Sellers & Ralyté 2010) - are essentially building blocks (e.g. requirements modelling) for constructing a methodology. Each method fragment can be process-

² A “method user” is a role that can be played by people in software development, such as business analysts, architects and software developers.

³ Strictly speaking, “method fragments” should be termed “method fragment kinds”. For convenience and ease of reading, the “kind” suffix has been dropped.

(e.g. procedure) or product-related (e.g. diagram) (Brinkkemper et al. 1998). Method fragments can be extracted from existing methodologies (e.g. for reuse) or developed from scratch to meet the needs of practitioners. They can be stored in a methodbase or repository (Harmsen et al. 1994). Via *situational method engineering*, fragments are selected from a repository and assembled into a methodology to adapt it to the development needs of a particular kind of application in a specific project situation (Brinkkemper 1996; Brinkkemper et al. 1998; Harmsen et al. 1994; Ralyté & Rolland 2001). In this research, developing support for dynamic evolution using a method engineering approach has advantages over developing a single methodology for dynamic evolution. Apart from the ability to be assembled into a methodology to adapt to a variety of (project) situations, method fragments facilitate incorporation into existing fragment-structured methodologies supporting composition-based distributed application development, to enhance their capability to support dynamic evolution.

1.2 RESEARCH OBJECTIVE

Motivated by the benefits of dynamic evolution to a distributed application and the lack of methodological support for dynamic evolution as discussed in the last Section, this research was carried out to:

via method engineering, develop a set of method fragments as enhancements to existing software development methodologies, to support the analysis and design aspects of dynamic evolution for composition-based distributed applications, to accommodate dynamic changes over time without the need for shutdown and restart.

To achieve the above goal, this research investigated two research questions. The first research question dealt with the concern highlighted earlier in Section 1.1.3 that there is a lack of feature requirements that should be considered from the methodological perspective:

RQ1: *What are the important requirements for consideration in composition-based distributed application development to support dynamic evolution?*

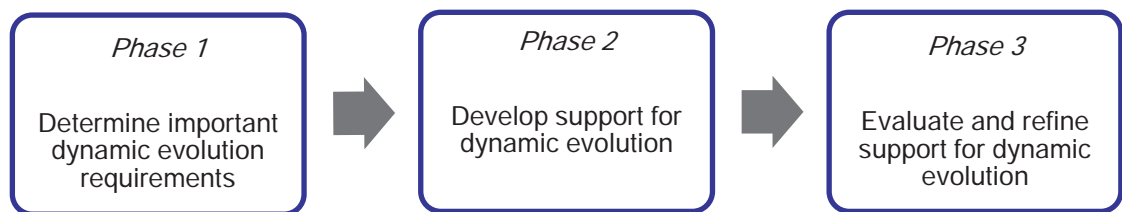
The second research question concerned the methodological support, exploiting the flexibility of both the composition-based approach and method engineering, to fulfil the important requirements that were inquired with the first research question. Again, the argument for (the lack of) methodological support was made earlier in Section 1.1.3:

RQ2: *How can these important requirements be addressed with method fragments*

used in composition-based distributed application development?

1.3 RESEARCH DESIGN

Since this research aimed to *define* a set of requirements to be addressed from the methodological perspective (i.e. research question RQ1) and to *build* a set of methodological features to meet these requirements (i.e. research question RQ2), it falls into a *prescriptive* inquiry (Gregor 2006) and is best achieved with design science research - a.k.a. design research (Hevner et al. 2004; March & Smith 1995; Vaishnavi & Kuechler 2004). Thus, the programme for this research was conducted, using design science research as the overall research framework, in three phases as shown in Figure 1.1:



source: developed for this research

Figure 1.1 Development phases for this research

Phase 1: Determine important dynamic evolution requirements.

The first phase responded to research question RQ1 by determining important requirements that should be considered when addressing dynamic evolution in composition-based distributed applications. The requirements were used later as inputs to the next phase for the development of methodological support for dynamic evolution. Tasks performed during this phase include:

- Synthesise dynamic evolution requirements from the literature and relevant evaluation frameworks;
- Assess the importance of and extend dynamic evolution requirements using a survey of experienced practitioners and researchers; and
- Extend the set of important dynamic evolution requirements with those already considered in relevant methodologies.

Phase 2: Develop support for dynamic evolution.

This phase acted on research question RQ1 by developing the structure and content of Continuum to address the important dynamic evolution requirements determined from Phase 1. Continuum is a methodology extension comprising a set

of International Standard ISO/IEC 24744 (ISO/IEC 2007) based method fragments, to be incorporated into an existing methodology, to specifically support the analysis and design aspects of dynamic evolution during software development. Suitable fragments were reused from existing methodologies, enhanced from existing methodologies, or otherwise developed from scratch where appropriate. The development of Continuum involved the execution of the following two tasks:

- Identify method fragments from relevant methodologies suitable for reuse or requiring small enhancement to address the requirements determined in the last phase.
- Develop method fragments by enhancing those candidates identified in the last task if necessary, and defining new ones to fulfil the dynamic evolution requirements that are not addressed satisfactorily by those fragments identified in the last task.

Phase 3: Evaluate and refine support for dynamic evolution.

During this phase, Continuum was progressively evaluated and refined in two tasks:

- Conduct an expert review of Continuum with involves subject matter experts reviewing the documentation for Continuum. Refine Continuum based on the experts' feedback.
- Apply Continuum to a case study, involving the analysis and design of dynamic evolution for an application. Again, refine Continuum based on the case study participants' feedback.

1.4 CONTRIBUTIONS

In response to the poor understanding and support for dynamic evolution in the literature concerning methodology development and research (as noted in Section 1.1.3), this research fills this gap by making two major contributions as follows. They are intended to be sufficiently generic to a variety of composition-based distributed applications, including component-based and service-oriented applications on which this research focuses:

A proposed set of important dynamic evolution requirements. The novelty lies in their coverage for both dynamic changes and quality aspects of dynamic changes and applications, for explicit consideration in methodologies supporting composition-based distributed application development. These requirements can also be applied to evaluate a methodology for its extent of support for

dynamic evolution.

Continuum, a set of method fragments to specifically address the analysis and design aspects of dynamic evolution as a separate concern. Continuum has the flexibility of being orchestrated and sequenced to run alongside conventional software development activities to extend their capability for dynamic evolution.

These contributions target method users and method engineers (Nuseibeh et al. 1996). (Note that "method users" and "method engineers" are role names adopted from the context of method engineering; they can be played by but are not limited to researchers from academia and practitioners from industry.) First of all, these contributions hopefully help method users (cf. Section 1.1.3) to leverage the advantages of dynamic evolution in applications for their business in several ways such as those illustrated in Table 1.2:

Table 1.2 Example potential benefits of contributions to method users

<i>Contribution</i>	<i>Potential Benefits</i>
Dynamic evolution requirements	<ul style="list-style-type: none"> • Become more familiar with the characteristics and quality aspects of dynamic evolution for an application. • Evaluate a methodology using the proposed dynamic evolution requirements to ascertain the methodology's support for dynamic evolution. • Articulate dynamic evolution requirements to method engineers (Nuseibeh et al. 1996) when the engineers configure and assemble a methodology.
Continuum	<ul style="list-style-type: none"> • Become more familiar with the characteristics and quality aspects of dynamic evolution for an application. • Handle dynamic evolution as a separate concern during development, along with conventional analysis and design activities.

Second, these contributions offer methodological foundations for method engineers who create method fragments and define methodologies from these fragments for use by method users (Gonzalez-Perez & Henderson-Sellers 2006a) to better handle dynamic evolution. Example benefits to method engineers are given in Table 1.3:

Table 1.3 Potential benefits of contributions to method engineers

<i>Contribution</i>	<i>Potential Benefits</i>
Dynamic evolution requirements	<ul style="list-style-type: none"> • Compare support of various methodologies for dynamic evolution. • Select a methodology of best fit to meet particular dynamic evolution objectives. • Identify limitations in a methodology for improvement to better support dynamic evolution. • Use the dynamic evolution requirements as a baseline for future extension to better describe dynamic evolution needs for specific types of composition-based distributed applications (e.g. SOA-based ones).

<i>Contribution</i>	<i>Potential Benefits</i>
Continuum	<ul style="list-style-type: none"> • Separate the dynamic evolution concern from conventional analysis and design concerns in a methodology. • Assemble a new methodology or extend existing methodologies to suit a development environment or a project in which dynamic evolution is desirable or required for a composition-based distributed application. • Use Continuum as a baseline for future development to better support specific types of composition-based distributed applications (e.g. SOA-based ones).

1.5 RESEARCH DELIMITATIONS

The following areas are deemed out of the scope for this research, to confine the focus and the boundary of this research:

- *Conventional aspects of software development other than analysis and design* (configuration management, requirements engineering, testing and change management etc., which are expected to be taken care of by existing methodologies);
- *Technological and tool related support for dynamic evolution*, which can be implementation-oriented;
- *Application contexts*, which may influence the analysis and design for dynamic evolution (e.g. online transaction applications which may be more vulnerable to financial loss because of faulty dynamic changes, leading to the need for better error handling in the design to deal with dynamic changes);
- *Environmental issues*, which can be related to non-software issues;
- *Organisational, business, people, project, programme management and other issues falling outside of application development*, which may influence how dynamic evolution is used in applications;
- *Data and schema evolution*, a branch of specialised studies for evolution (e.g. Roddick 1992);
- *Architectural support for dynamic evolution* (e.g. certain architectural designs potentially facilitating dynamic evolution more easily than others); and
- *Incorporation of Continuum into non-fragment based methodologies*. (Note: Continuum will also be useful for non-fragment based methodologies. However, the effort of incorporating its fragments into this kind of methodology for an endeavour will vary widely, depending on the structure and customisability of individual methodologies.)

1.6 CONCLUSION AND THESIS OUTLINE

This Chapter argued how important it is for this research to support dynamic evolution in the context of development methodology, leveraging the capability of a composition-based distributed application. In particular, this Chapter presented the research background and motivation, the research objective derived from the motivation, a research programme to accomplish this objective, key contributions of this research, and areas excluded from this research. The rest of this thesis is organised as follows:

Chapter 2 presents a review of dynamic evolution and methodology. The dynamic evolution part covers concepts of dynamic evolution, composition-based distributed applications and how they can support dynamic evolution. The methodology part covers evaluation frameworks to assess methodologies, relevant and existing software development methodologies supporting composition-based development, and situational method engineering for the development of methodologies.

Chapter 3 overviews design science research, presents the rationale for selecting design science research for this research, and the research plan tailored from the selected paradigm in terms of three phases and associated tasks carried out to accomplish the plan.

Chapter 4 and Chapter 5 describe the execution of Phase 1 and its results, which are a set of important dynamic evolution requirements to be considered for the development of Continuum.

Chapter 6 documents the content of Continuum. This includes the identification of features from existing methodologies suitable for reuse/enhancement in Continuum, and method fragments specifically developed in Phase 2 to address the requirements selected for Continuum. For convenience, the documentation also incorporates refinements based on the evaluation results in Phase 3.

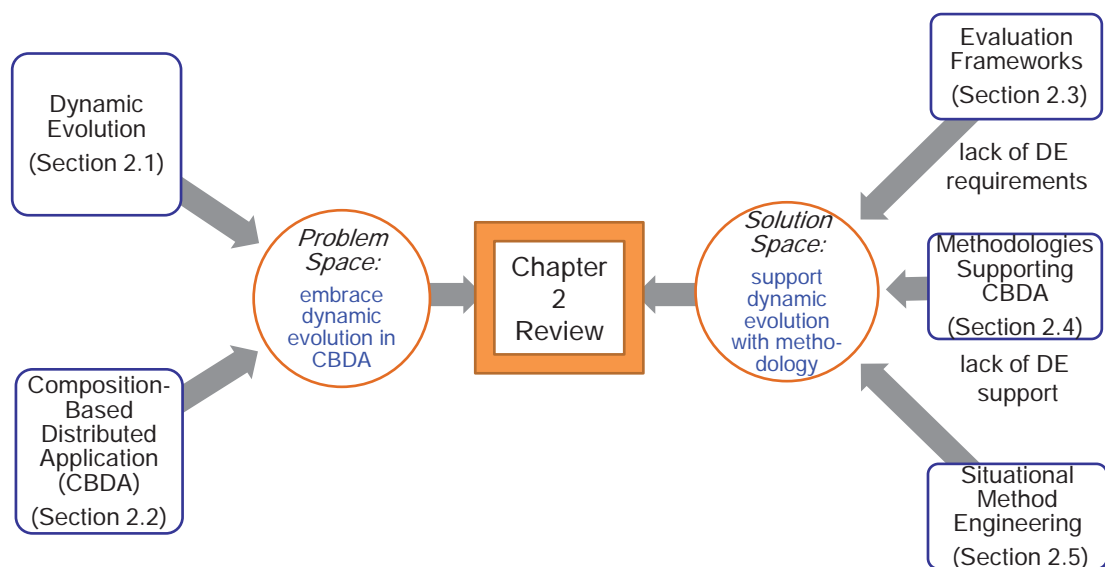
Chapter 7 presents the outcomes of Phase 3, comprising the evaluation results from an expert review of and a case study application of Continuum, plus refinements made to Continuum based on these results.

Chapter 8 discusses the main conclusions with respect to the research questions investigated, the contributions of this research, threats to the validity and reliability of the findings of this research, and recommendations for future work.

Chapter 2. REVIEW OF DYNAMIC EVOLUTION AND METHODOLOGY

“Study the past if you would define the future.” - Confucius

The previous Chapter introduced the rationale for and the context of this research, and presented both the research problem and questions to be explored. In turn, this Chapter reviews five relevant topics, conceptually divided into two groups as shown in Figure 2.1.



source: developed for this research

Figure 2.1 Conceptual model for review sections in Chapter 2

- *Problem space (for dynamic evolution)*: This part reviews the literature concerning the problem aspects of this research, being dynamic evolution in the context of composition-based distributed applications. First, the definitions of dynamic evolution are discussed in detail, covering areas from evolution in general to specific characteristics of dynamic evolution (Section 2.1). Then, the concept of composition-based distributed application is reviewed and argued as a suitable platform for dynamic evolution (Section 2.2).
- *Solution space (for dynamic evolution)*: This part reviews the literature concerning the solution aspects of this research, being the use of methodologies in tackling dynamic evolution. First, it discusses the “state of the art” in two methodological areas and points out their gaps in dealing with dynamic evolution. The first area concerns evaluation frameworks which define

ideal requirements that a methodology should support and points out that existing frameworks lack requirements specific to dynamic evolution (Section 2.3). The second area concerns existing development methodologies that support composition-based software development and argues that dynamic evolution is poorly understood and supported in these methodologies (Section 2.4). The review then turns to the topic of situational method engineering in the development of methodologies and its ability to construct a methodology to a particular problem domain (e.g. dynamic evolution) in a project situation (Section 2.5).

2.1 DEMYSTIFYING DYNAMIC EVOLUTION

The definition of evolution finds its roots in the natural progression of animal and plant species, referring to "the process of developing from a rudimentary to a mature or complex state" (Oxford English Dictionary), or a gradual development from a simple to a complex form over generations to survive in a continuously changing environment (Darwin 1859). Evolution is an essential and necessary capability for species to adapt to a dynamic environment.

Evolution is specialised into temporal and spatial evolution (e.g. Aassine & El Jaï 2002). Temporal evolution specialises in the time-dependent or time-series aspect of evolution (i.e. how species change over time). In contrast, spatial evolution concerns the location dynamics of species (i.e. how the distribution of species changes). These specialisations are also commonly found in sciences that study characteristics of a phenomenon in space and/or time, as exemplified in Table 2.1. Although temporal and spatial evolution are treated as two orthogonal constructs, their effects may influence each other. Therefore, spatio-temporal evolution studies examine their combined effect (e.g. Martínez-Beneito et al. 2008).

Table 2.1 Examples of specialisation of evolution studies

<i>Specialisation</i>	<i>Temporal</i>	<i>Spatial</i>
Distributed applications	Gradual upgrade at runtime to improve performance to cope with increasing loads	Migration and replication of system parts to appropriate network nodes to distribute loads
Demography	Effect of population ageing over time on population growth	Effect of geographical preferences on the distribution of population
Meteorology	Depletion of the ozone layer over time	Spread and size of the ozone layer
Astronomy	Formulation of black holes over time	Location and spread of stars

2.1.1 System and Software Evolution

In system engineering, Rowe et al. (1998) refer evolution to changes of a system over its lifespan. Furthermore, they elicit that the origins of changes are attributed to functional (maintenance, enhancement etc.) and non-functional needs (performance tuning, change in environment and technology etc.), resulting in changes in the requirements of the system: “[the nature of] a system to accommodate change in its requirements throughout the system’s life span with the least possible cost while maintaining architectural integrity” (Rowe et al. 1998).

While this definition is in line with this research, further justification in the context of a software system or an application is needed. Evans and Dickman (1999) clarify that computational changes do not constitute software evolution. By definition, computation refers to changes in data which are supported as part of the functionality of an application. This definition includes data transactions in applications containing database sub-systems where changes to the data are coordinated. Unlike computation, software evolution results in the alternation of the structure (e.g. by addition/removal of parts) and semantics (e.g. by replacement of existing parts) in the software so that it behaves differently (Zhang et al. 2009). Certain requirement driven changes also contribute to software evolution to a lesser extent. For example, in dynamic document-based applications (de Lara et al. 2005) and the user-interface tier of a multi-tiered architecture (Alur et al. 2003) capable of dynamic content provisioning (Challenger et al. 2005), changes to the information presented to end users and the way it is presented may trigger software evolution. For instance, adding hyperlinks on a web page to link it to other pages extends the navigation model which may also require update to the workflow of the underlying application, whereas colour adjustment on web pages does not lead to changes to the application’s backend components.

In application development, evolution is related to but distinguished from maintenance. Kemerer and Slaughter (1999) explain that evolution examines how an application changes over time whereas maintenance activities focus on modification and error correction after the development of a new application. With regard to evolution, multi-year longitudinal studies, for instance, were conducted to examine factors causing changes to applications (see examples in Kemerer & Slaughter 1999, p. 496). Lientz and Swanson (1980) categorise maintenance into three activities: adaptive, corrective and perfective. Adaptive maintenance mandates that an application must change to adapt to a dynamically changing operating environment. Corrective maintenance is triggered by discovered errors which must be removed. Perfective maintenance

concerns the desire to add, modify and enhance functionality to obtain better applications. In addition, Pressman (2005) and *the Guide to the Software Engineering Body of Knowledge* (Abran & Moore 2004) propose an extra activity called preventive maintenance which concerns the modification of an application after delivery to detect and correct latent errors before they surface in its operating environment.

2.1.2 Types of Evolution in Applications

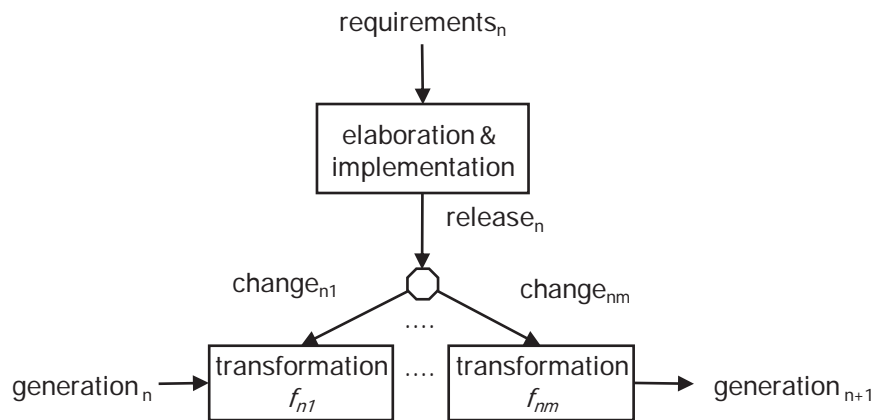
The nature of evolution depends on when changes are incorporated into an application. Buckley et al. (2005) describe three categories of evolution: static, dynamic and load-time. In static evolution, commonly known as software maintenance, changes are made to an application's source code which is then recompiled into a new executable application. In contrast, dynamic evolution (a.k.a. runtime evolution) concerns changes that are made and activated in an application without its being shutdown. This includes removing existing binary code from an application, hot-swapping existing binary code with a newer version, and adding new code to an application. Load-time evolution lies between static and dynamic evolution; it occurs when some binary code is not loaded into an executable application until needed (Buckley et al. 2005).

Dynamic evolution in applications is studied under different guises, including online software evolution (Wang et al. 2006), online upgrades (OMG 2003b), dynamic reconfiguration (Agnew et al. 1994a), dynamic programme updating (Segal & Frieder 1993), runtime evolution, runtime reconfiguration, dynamic upgrading, and dynamic adaptation (Mens 2008). Dynamic adaptation has a particular goal for evolution; make changes to an application dynamically *in response to changes in its environment* (McKinley et al. 2004). Furthermore, studies regard dynamic evolution as a precursor and a facilitator to successful dynamic adaptation (e.g. Andrikopoulos et al. 2008; Oreizy et al. 2008). Dynamic evolution, however, is differentiated from other disciplines of software evolution, including "requirements evolution", "architecture evolution", "data evolution" and "language evolution" (Mens 2008). In data evolution (a.k.a. "schema evolution" (Roddick 1992)), schema definitions of a relational database are evolved to adapt to changing business data needs (e.g. adding a secondary contact phone number for staff). This also requires migrating the underlying relational data to conform to a new schema (e.g. adding the "2nd_phone_no" attribute to all staff records).

2.1.3 Release, Transformation and Dynamic Change

As an application evolves from its initial release to its retirement, it can be thought of as

progressing through a number of generations (Kruchten 2003). Each generation succeeds its predecessor by accommodating a particular release of changes to reach its next desirable form. A release is the result of requirement elaboration and change implementation. In dynamic evolution, a release is broken down into dynamic changes (Papazoglou 2008; Yu et al. 2005), referring to *what* are to be modified in an application at runtime. Each dynamic change is kept small enough to be applied to the application without causing significant disruptions. As opposed to dynamic change, a transformation⁴ refers to an *act* of performing dynamic modifications to the application (Mens & Demeyer 2008). A simple solution is to define a transformation for each dynamic change to realise it in the application, as shown in Figure 2.2.



source: adapted from Fung et al. (2004)

Figure 2.2 Release-change-transformation relationship in dynamic evolution

The term "transformation" is used by Blaha and Premerlani (1996) to describe a function to map a source object model into a target object model. It is also used in the discipline of *transformational software engineering* to model a chain of transformations for the production and evolution of software (Mens & Demeyer 2008, p. 305). A transformation is distinguished from a *transaction*, a common term associated with changes in databases or distributed applications (Date 2003). A transaction refers to a coordinated manipulation or modification on data held within an application while preserving the ACID (i.e. atomicity, consistency, isolation and durability) property (Haerder & Reuter 1983), or a coordinated execution of a piece of work (Papazoglou & Kratz 2007). The subtle difference between transformation and transaction is similar to that of software evolution vs. computational changes (in data) as discussed in Section 2.1.1.

⁴ In this research, the term "transformation" is confined to refer to "dynamic transformation" which refers to a transformation that is acted on an application at runtime.

2.1.4 Evolution Space

The evolution of an application lifecycle can be described by an evolution space, which is an extension of a version model over time. The version model represents the static version information of software parts, the organisation of the software parts making up the whole application, and how individual versions of the application can be constructed from the software parts. Conradi and Wesfechtel (1998) describe a version model consisting of a version space, a product space, and a configuration language. The version space represents the organisation of versions - a.k.a. "revisions" and "variants" - for individual parts and the application (e.g. versioning support for Web Services (Leitner et al. 2008)). The revisions of an artefact can be described by an ordered list of subsequent changes applied to the first version of the artefact, each identified by a unique label (e.g. "Windows Vista™" vs. "Windows XP™"). Variants determine different forms of each part. In this case, a variant of a particular release of a calendar software specifies in which environment the software can operate (e.g. mobile phone vs. desktop computer). The product space characterises the relationships among individual parts to construct the application, without considering any version information. Typical relationships are composition (e.g. a transaction system composed of clients and servers) and dependency (e.g. certain software requires the Microsoft .NET Framework™). Although product and version spaces are different but related concepts, approaches exist to model both concepts together to visualise evolution (e.g. evolution graph, Atkinson et al. 2002).

A particular release of an application can be configured by selecting correct entities within the product and version spaces (e.g. a word processing software packaged with version 5.0 of the UK English spell checking function). Thus, the number of possible configurations grows exponentially with respect to the number of entities available to choose from. Some rules or a configuration language must therefore be applied to make the selection process manageable and produce a consistent application⁵ to address this combinability problem (Kon & Campbell 2000). Such a language will offer expressions for constraints in selecting and configuring a particular configuration, as well as defining preferences of the configured application. With these expressions, any particular application can be retrieved from a configuration management system. For example, the query, "Fetch me the latest version of XYZ word processor software that runs on

5 A consistent application does not exhibit errors due to incompatibility among selected parts within the application.

the Linux operating system and has the German spell checking function", will cause a particular XYZ configuration to be built and released.

2.1.5 Evolution Quality

Eriksson and Törn (1991) characterise IS quality in three perspectives: IS cost effectiveness, IS use and IS work quality. IS cost effectiveness addresses the benefit and cost objectives of an information system, while IS use quality considers how well an information system serves its users. IS work quality relates to the work performed by practitioners in providing an information system. Anderson and von Hellens (1997) expanded on the Eriksson and Törn's model by incorporating the management aspects required to address IS work quality and to assure the quality of systems for evolution and use. The quality of IS management is considered from two viewpoints: evolution and user support quality. Evolution quality is concerned with how well the evolution of a system is managed and its documents are updated without compromising quality, as opposed to user support quality which is concerned with how well user support is managed. Evolution quality would appear to be the most appropriate quality dimension for dynamic evolution.

Quality is an important consideration of both the software development process and product. In the case of dynamic evolution, the urgency to repair a running application after dynamic evolution is high since it is unacceptable to leave an erroneous application running indefinitely. It is hence critical that changes, their transformations, and the change process are of sufficient quality to ensure that the evolving application is free from errors. On the other hand, *low-quality* changes may not only compromise core features of a running application (e.g. security), but also make the application harder to evolve in the future. Thus, investing in quality for dynamic evolution ensures the running application can effectively evolve when required and its operation can be effectively sustained in the future, and makes the benefits of dynamic evolution more appealing to practitioners, end users and the business. Since evolution in applications should be considered in a development methodology (Bennett & Rajlich 2000; Coyle et al. 2010) and quality is a key driver for cost and effort reduction in IS maintenance (and thus evolution) (Eriksson & Törn 1991), a comprehensive methodology should include quality as one of its capabilities (e.g. quality engineering (OPFRO 2009)).

The literature lacks a comprehensive discussion of *quality factors* (ISO/IEC 2008) relevant to dynamic evolution (see also related work in Section 4.1.6). For example, Segal (2002) identifies several relevant challenges for online upgrades in a distributed

environment including the ability of multiple hosts to coordinate upgrades, upgrading at an appropriate time, securing the upgrading functions from attackers, and the ability to address failed upgrades. Feiler and Li (1998) discuss fault-tolerant software changes and define what consistency and completeness mean to software changes. Evans and Dickman (1999) focus on managing the complexity of dynamic changes by using application partitioning. Unfortunately the literature is not always clear about the meaning of the various factors. For example, the terms “correctness” and “correctly” are associated with changes to a system without clarifications (e.g. Cook & Dage 1999; Kon et al. 2001; Zhang et al. 2005). Inconsistent terminology has also been used to epitomise similar concepts, such as “efficient ... in time use” (Agnew et al. 1994a), “quickly” (Bennett & Rajlich 2000) and “time taken ... minimised” (Gupta et al. 1996) for characterising transformation efficiency. Breivold and Crnkovic (2010) conducted a systematic literature review of quality considerations for architecture design to support evolution. A few quality terms and synonyms (e.g. evolvability and changeability) were used as keywords to search for articles for the review but no definitions were given to those keywords. A more comprehensive analysis and definition of quality factors should thus be sought to better tackle dynamic evolution quality.

2.2 COMPOSITION-BASED DISTRIBUTED APPLICATION

Software architecture is defined as an abstraction of the structure of an application, consisting of elements, their externally visible properties, and the relationships among these elements (Bass et al. 2003). Properties of an element refer to assumptions about the element, such as its Quality of Service (QoS) profile. Typical relationships include dependency and interactions, just as an element depends on and interacts with others to fulfil its role (i.e. perform its computation). Furthermore, a proper architecture shows the correspondence between system requirements and its elements (Shaw et al. 1995). In recent years, large applications are increasingly assembled from “independent and reusable collections of functionality” (Brown et al. 2002). More specifically, “composition-based” software development refers to a paradigm of “weaving” existing or custom-developed applications, (commercial-off-the-shelf) software components, leased or rented applications and services into composite applications (Schuster 2008). In a more abstract view, this kind of application is fabricated from loosely coupled parts bound together (e.g. Szyperski 2003). Bindings associate or connect parts to form a composite which becomes yet another part, and to mediate the interaction among parts (Bruneton et al. 2002; Magee & Kramer 1996) in accordance with interaction specifications. Parts are units of functionality which can work independently or with

others. Parts expose themselves to others as complex composition points or interfaces, usually specified as some kind of contract (Stojanovic et al. 2004a), while hiding their internal implementations. For certain composition-based applications, it is common to have their constituent parts scattered over a network and hence the name composition-based *distributed* applications.

Architectural paradigms that fit the description of a composition-based distributed model include component-based distributed applications (comprising components and connectors) (Evans & Dickman 1999) and service-oriented applications (comprising services and bindings) (Papazoglou & Georgakopoulos 2003). In this research, the latter is also referred to as service-oriented architecture (SOA) based applications for the peculiar architectural style with services. Various kinds of composition topologies exist, such as hierarchical composition in component-based applications (Peltz 1999; Velasco Elizondo & Lau 2010; Wienberg et al. 1999) and conversational composition in SOA-based applications (Khalaf et al. 2003).

2.2.1 Component-Based Distributed Application

In component-based distributed applications, components and connectors form the elementary building blocks (Shaw et al. 1995). Components are units of computation (Shaw et al. 1995; Shaw & Garlan 1996) and distribution (Szyperski 2003), scattered within the system boundary according to the design and operating choices. Connectors mediate component interaction by gluing or wiring components together to provide reliable communications among components over the network. Abstracting interaction and communication concerns from components to connectors permit components to be either oriented on business or on computation (Wang et al. 1999). Because of their idiosyncrasy, connectors deserve to be treated as first-class entities in architectures (Lopes et al. 2003). Approaches to building a component-based system to yield the same result - a composite structure of components and connectors - include:

- *Starting from components* (i.e. bottom-up)

Suitable components are acquired first and then glued together by connectors, such as “port-n-links” (Wang et al. 1999) or “protocol specifications” (Yellin & Strom 1997) to progressively assemble an architecture. Syntactical incompatibilities among components can usually be overcome if their specifications are sufficiently precise (Yellin & Strom 1997). Constructing a component-based system this way is analogous to composing a model with

ready-to-use Lego™ cubes of different shapes (Szyperski 2003; Unhelkar 1997).

- *Starting from architecture and connectors* (i.e. top-down (D'Souza & Wills 1998))

The skeleton of an architecture is formed by defining and refining component specifications and their interactions (i.e. connectors), whilst system functionality is decomposed and allocated to components and connectors. Components are then built and plugged into the architecture.

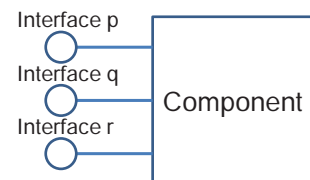
- *Combining both the above approaches*

An architecture is formed by iterating over the design of existing and new components as well as connectors, such as Unicon (Shaw et al. 1995).

2.2.1.1 Software component

A software component is commonly visualised as a black box with one or more well-defined interfaces (shown as lollipops) attached to its boundary, such as the one shown on the right.

Szyperski (2003) formally defines a software component as “a unit of composition with contractually specified interfaces and explicit context dependencies only” and it “can be deployed independently and is subject to composition by third parties”.



A software component specifies its behaviour via its “provided” and “required” interfaces (OMG 2010b). A provided interface is an interaction point for other components to access the component's operations (Wang et al. 1999; Yellin & Strom 1997). A required interface specifies what its component needs from others in order to perform its functions or fulfil its obligations (OMG 2010b).

Components are designed to support composition (D'Souza & Wills 1998); more complex components can be *assembled* from simpler components by linking the simpler ones with connectors, forming a hierarchy of components and connectors. Atomic components are those that are not assembled from others. At runtime, components are instantiated as needed (Brown et al. 2002).

Software components can be distribution-capable; they are not confined to a single address space (i.e. or the same execution environment) but rather connected and scattered over a network. This, however, does not mean that they are equivalent to operating-system processes. A process provides components with its address space in which they operate. Components do not have to be designed and implemented using object-oriented (OO) technologies (Brown & Wallnau 1998), but they seem to be the

best choice (Szyperski 2003).

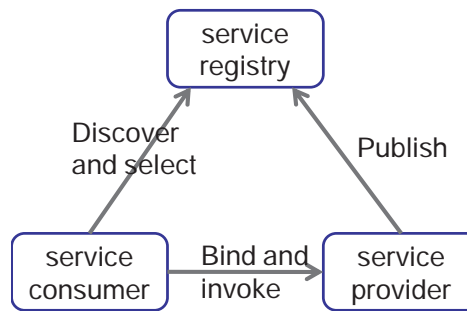
2.2.1.2 Connector

A connector is specified by “roles” and “interactions” (or more formally “interaction protocols”) among these roles (Allen & Garlan 1997). A role represents the expected behaviour of a component participating in an interaction. A variety of interaction styles exist for designers to choose from, covering the means of communication used, such as events/messages, remote procedure calls and shared memory (Shaw & Garlan 1996). Interaction styles also cover the multiplicity of components (i.e. how many components are allowed to interact via one connector) (OMG 2010a) and synchronisation among connected components. In the event-based style, a component can choose to synchronously send an event to another component and wait until an acknowledgement is replied, or asynchronously broadcast it to a few other subscribing components.

A connector is distribution aware; it can link components distributed on different hosts. A connector thus spans multiple nodes to provide the link for those distributed components. The “liveness” (Shatz 1993) of a connector (e.g. how long it remains idle before dropping out) depends on the interaction style used.

2.2.2 Service-Oriented Architecture Based Application

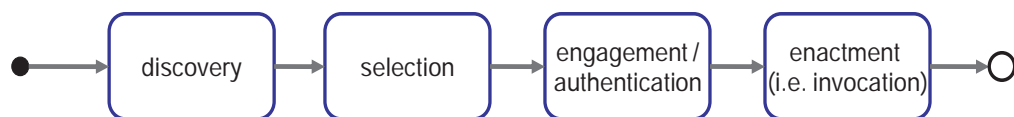
A service-oriented architecture (SOA) based approach centres on utilising services as the fundamental elements for developing applications (Papazoglou & Georgakopoulos 2003). A *service* is a self-contained, self-describing, discoverable, coarse-grained software entity that operates as a single instance interacting with applications and other services (Brown et al. 2002; Papazoglou & Georgakopoulos 2003; Stojanovic et al. 2004b). A service provides an effective way of exposing discrete business functions (Brown et al. 2002); it encapsulates business logic into a single cohesive conceptual module (Yang 2003). In an *open* networked environment, services are provided by multiple organisations and communities. This openness facilitates the reuse of existing services available from various parties so as to gain productivity in software development and to improve the quality of software produced (Huhns & Singh 2005).



source: Huhns and Singh (2005) and Yu et al. (2005)

Figure 2.3 Basic service-oriented model

A *basic* service-oriented model has three main entity types (Figure 2.3): a service provider, a service consumer and a service registry (Huhns & Singh 2005). A provider publishes its offered services in a registry, similar to a yellow pages directory. A consumer simply looks up the registry for potential providers for a service of interest (i.e. “discovery”), selects one provider of best fit (i.e. “selection”), binds to the selected provider (i.e. “engagement”) and invokes its service accordingly (i.e. “enactment”). During the engagement phase, the consumer and the provider may mutually authenticate each other to establish a trust relationship before the use of services commences. These steps are summarised by a state machine as shown in Figure 2.4.



source: derived from Huhns and Singh (2005)

Figure 2.4 State machine for using a service

In circumstances where only one provider is available and well known to the consumer, a service registry is redundant and optional in the basic service-oriented model (Figure 2.3). Similar to a component in a component-based system, a service’s descriptions are independent from the service’s internal implementation and developers are free to choose technologies to their liking for the actual implementation (Huhns & Singh 2005).

Services are the basic building blocks from which to create service-based applications (Curbera et al. 2003). Services are independent and self-contained (Brown et al. 2002) but can be composed or aggregated together according to some business rules to serve a business goal or to provide new functionality (Curbera et al. 2003). For instance, a one-stop travel booking service can be offered to customers online to arrange transport booking and hotel reservation using the services provided by a particular airline and a hotel chain. A service composition model can be expressed

using a hierarchical “goal graph” or a “workflow” (a.k.a. business process) (Huhns & Singh 2005). Services should be *stateless* (Milanovic & Malek 2004; Pasley 2005; Stal 2006) to improve loose coupling and scalability of an architecture (Stal 2006). The state information can otherwise be kept in the running (instances of the) business processes or saved in a data store (Pasley 2005).

In an *extended* service-oriented model, additional entities support the use and management of services (Papazoglou & Georgakopoulos 2003). For instance, a “composite services layer” governs the consolidation of multiple services into a single composite service (Papazoglou et al. 2008), such as coordinating the execution of services and data flow among them. At the next higher layer, utilities are available to manage the operations of services, to monitor their activities, and to support specialised marketplaces for conducting business electronically with services.

Web services is a realization of a service using the Internet as the means of communication (Brown et al. 2002) and it is made possible by the proliferation and maturity of Web service technologies and standards in use today.

2.2.3 Embracing Dynamic Evolution

Composition-based development is well suited to dynamic evolution because the essence of a composition-based application lies in its structure which is fabricated from loosely coupled parts bound together (e.g. Szyperski 2003). In response to the need for changes, parts can be integrated to such an existing application to add new functionality (Chaudet et al. 2000), upgraded to change their offered functionality and removed from the application (i.e. to rid functionality of it) once the parts are no longer required. On the other hand, in a distributed application where parts are scattered and interact over a network, these changes can be further facilitated by connecting (new) parts to and disconnecting (existing) parts from the application.

The ability of composition-based distributed applications in embracing dynamic evolution is exemplified by component-based and SOA-based distributed applications. In the component-based paradigm, a system built with components and connectors embodies the concepts of distribution units (i.e. components), composition, explicit dependency and loose coupling in its architecture. The resulting structure facilitates simple changes abstracted as addition, replacement and removal of components and connectors (Brown & Wallnau 1996; Oreizy et al. 1999; Zhang et al. 2009), whilst connectors play the role of *dynamic binding* and *adapters*. The former describes the wiring and rewiring components at runtime (Rasche & Polze 2003) and the latter is

responsible for wrapping and plugging components into an application, plus resolving functional, interface, information and protocol mismatches of components (Brown & Wallnau 1996; Jiao & Mei 2005). Connectors also abstract the *evolution of the protocol* among connected components from the components per se (Ryan & Wolf 2004).

SOA-based applications can be changed at runtime to meet new software requirements (Tsai et al. 2006) because many runtime features in an SOA environment appear to support dynamic evolution. For instance, service composition is by nature *dynamic* using the descriptions of services at runtime such that services can be dynamically connected and composite services dynamically reconfigured for use (Li et al. 2006; Yen et al. 2008). Likewise, dynamic binding (Blake 2007; Curbera et al. 2003) occurs naturally just before a service is used and the binding reference can be dropped after service invocation. In other words, a server consumer is usually decoupled from any service provider; it does not and is not required to retain any reference to a particular provider. Before binding is established, a service consumer often searches for and discovers a list of up-to-date potential providers offering services of interest (i.e. service discovery, Cervantes & Hall 2005), and then picks a provider (i.e. service selection, Huhns & Singh 2005) most suitable for its needs at runtime. Dynamic evolution is also regarded as a necessity in an SOA environment to adapt SOA-based applications to an open environment such as the Internet and changing user requirements (Liu et al. 2009).

Similarities have been drawn between component- and SOA-based applications. Yu et al. (2005) proposed a mapping between elements in component-based applications and SOA-based applications that use Web services (see Table 2.2). Thus, dynamic evolution principles and guidelines for component-based applications are likely to be also applicable to SOA-based applications.

Table 2.2 Component-based vs. service-oriented elements

<i>Component-Based</i>	<i>Service-Oriented</i>
Component	Service
Port	Interface: Web service Description Language (Christensen et al. 2001)
Connector	Protocol: Simple Object Access Protocol (SOAP) (W3C 2003) and WS-Coordination (OASIS 2007)
Role	Invocation relationship between services
Configuration	Service composition structure
Style	Composition pattern
Constraints	Composition constraints

source: Yu et al. (2005)

Composite services can be dynamically recomposed for use in service orientation (Li et al. 2006) to meet new software requirements (Tsai et al. 2006). In this respect an SOA-based approach is better suited for dynamic evolution than a component-based one. Elfatraty (2007) further supports this argument with the following observations:

- Late-binding of service consumers and providers allows providers to be flexible with changes;
- Services are specified as a right level of granularity close to the real-world business activities and are more adaptable to changing needs of the business;
- Services are packaged to deliver functionality and are better aligned to changes in functionality as business requirements evolve; and
- Service consumers are decoupled from providers by brokers.

The review of topics in this Chapter so far has established the argument for composition-based distributed applications as a suitable platform for realising dynamic evolution, which also concludes the review of the problem space of this research (cf. Figure 2.1). Beginning from the next section, this Chapter turns to the solution space of this research (cf. Figure 2.1), dealing with dynamic evolution in composition-based distributed applications from a methodology's viewpoint. The first topic to review is the notion of an evaluation framework which is a systematic way to determine and assess particular capabilities for a methodology, such as of designing dynamic evolution in composition-based distributed applications.

2.3 EVALUATION FRAMEWORKS

An evaluation framework comprises a “checklist” of ideal methodology feature requirements (Siau & Rossi 1998). A feature requirement defines a problem or an outcome that methodology users intend to address for a particular activity, task or objective (Kitchenham 1996; Wasserman et al. 1983). Feature requirements can be thought of as problems or outcomes that a methodology is expected to address when dealing with the particular activity, task or objective, and as criteria for evaluating the extent to which alternative methodologies address the feature requirements. In contrast, a “feature” is a characteristic possessed by a methodology aimed at solving a problem or fulfilling an outcome (Kitchenham 1996). Note that although neither a feature nor a feature requirement has any bearing on the structure and composition of a methodology, a distinction between them is made clear with the following arguments. Two completely different features (e.g. object-oriented analysis and structured analysis) can fulfil the same feature requirement (e.g. development of an information domain

model). Likewise, one feature can fulfil two different feature requirements.

For each feature requirement, a methodology is assessed in two stages: identifying one or more candidate features from the methodology supporting the requirement, and assigning a scale point to each feature reflecting its level of support for the requirement (Kitchenham & Jones 1997). Scale points are defined and tailored for different evaluation goals. Applying an evaluation framework to assess one or more methodologies can:

- provide an understanding of particular aspects of a methodology (e.g. problem solving, Jayaratna 1994);
- assist in selecting and/or discarding a methodology for achieving a particular goal (Siau & Rossi 1998);
- assist in methodology enhancement (e.g. Tran & Low 2008); and
- provide a comparison of methodologies (e.g. Grimán et al. 2006).

Use of an evaluation framework allows a large number of methodologies to be assessed in a relatively short period of time (Kitchenham et al. 1997). However, subjectivity may be an issue since evaluations are usually performed by the same person. To reduce this effect, Iivari and Kerola (1983) suggest using a kind of Delphi method (Okoli & Pawlowski 2004) where an evaluation is done by framework authors and subject matter experts. Significant differences in the ratings are then resolved via post-evaluation discussions to reach a consensus. Another option would be to invite experts and practitioners to independently evaluate each methodology (e.g. Stojanovic et al. 2004a). In the latter case, it could be challenging to recruit evaluators when the number of methodologies to be evaluated and the requirement set become large, both implying a time-consuming evaluation effort.

Depending on the objectives and the context of an evaluation, alternative techniques may be more appropriate. Siau and Rossi (1998) divide evaluation techniques into two categories. *Non-empirical* techniques (e.g. feature analysis) assess methodologies and the contexts of the problems they intend to solve. *Empirical* techniques (e.g. experiments) derive the evaluation results by observing the experience gained through the use of methodologies. On the other hand, Kitchenham et al.'s (1997) DESMET evaluation methodology divides evaluation techniques into nine categories, covering both qualitative and quantitative evaluations. DESMET also provides criteria which will help evaluators in selecting a suitable technique.

A number of relevant evaluation frameworks are now briefly reviewed, covering

frameworks intended for composition-based applications and those that are not specific to this kind of application. The inclusion of the latter frameworks is because reviewing the latter might potentially cover certain aspects of dynamic evolution that are overlooked by the former:

- *Composition-based*: Boertien et al.'s (2005) proposed a framework for component-based application development in the context of e-business. Their framework is founded on Sol's (1992) analytical framework for understanding IS development along five dimensions: *way of thinking* (the philosophy behind an approach to software development, including its assumptions and viewpoints), *way of working* (e.g. processes to follow), *way of modelling* (e.g. concepts), *way of controlling* (e.g. people management), and *way of supporting* (e.g. tools). Dahanayake et al.'s (2003) framework for evaluating component-based applications is also based on the Sol's analytical framework but their framework pays particular attention to component concepts and structures. Stojanovic et al. (2004a) extended Dahanayake et al.'s framework to accommodate both component-based and service-oriented applications. With respect to evolution, both Dahanayake et al.'s and Stojanovic et al.'s frameworks recognise maintenance as a key phase in the full lifecycle of a component or service.
- *Non-composition-based*: Wood et al. (1988) proposed an in-depth multidimensional framework suitable for evaluating real-time system development. The framework comprises a five-step evaluation process and a set of evaluative questions ranging from representation and implementation issues to management characteristics. Asadi and Ramsin (2008) defined a set of feature requirements for identifying strengths and weaknesses of model-driven-architecture based methodologies.

There are a number of relevant evaluation approaches for feature analysis in the object-oriented (OO) arena. HP Labs (Arnold et al. 1991) proposed four categories of feature requirements to compare OO methodologies: concepts, models, process and pragmatics. Object Agency Inc. (1995) refined HP Labs' feature requirements by adding two new categories of requirements and defining a 0-6 scoring system to rate methodology features. Monarchi and Puhr (1992) suggested four different categories of feature requirements; OO analysis process, OO design process, representations, and complexity management. By comparison, Hong et al. (1993) mapped the analysis and design steps, concepts and techniques of a methodology into a methodology metamodel

before comparing them with a set of predefined feature requirements. Beyond low-level feature requirements, Henderson-Sellers et al. (2001) used a list of characteristics at the methodology level (e.g. project management) to compare two methodologies. Similarly, Ramsin and Paige (2008) proposed a process-centred template to define areas of investigation (e.g. steps) where features identified from methodologies are slotted into their respective areas of the template. Other OO-based evaluation frameworks focus on the OO modelling aspects (e.g. Barbier & Henderson-Sellers 2000; Bobkowska 2005; Paige et al. 2000), and some almost entirely on OO analysis (e.g. de Champeaux & Faure 1992; Iivari 1995; Liang 2000). None of the OO frameworks noted above consider dynamic evolution as a characteristic of a methodological capability. They are only specific to their respective paradigms (e.g. inheritance in OO concepts).

At a broader level, Jayarathna (1994) proposed NIMSAD, a generic framework for evaluating any kind of system methodology, including IS development methodologies. NIMSAD is used for assessing and understanding not only a methodology, but also the person who uses the methodology and the context in which the methodology is applied. It is however ineffective in offering solutions for problems in many contexts (e.g. dynamic evolution), and in choosing methodologies for solving a particular problem (Adman 1997). Martinlassi (2004) derived an evaluation framework from NIMSAD for product line architecture development (van Gurp et al. 2001). There are also several reports of early work in analysing and comparing conventional system analysis and design methodologies (Olle et al. 1983; Olle et al. 1986).

Although the evaluation frameworks discussed above offer extensive feature requirements in their respective domains, none of them specially cover dynamic evolution aspects (more details in Sections 4.1.5 and 5.1.3). Thus, a comprehensive set of dynamic evolution feature requirements that a methodology should fulfil cannot be solely determined from these frameworks. Furthermore, without the definition of a comprehensive set of dynamic evolution feature requirements to fulfil, a methodology is unlikely to adequately support dynamic evolution. Accordingly, the development of dynamic evolution requirements formed part of this research (cf. Section 3.2.1). Later on in Sections 4.1.5 and 5.1.3, these evaluation frameworks are examined in depth, as part of this development effort, to extract relevant feature requirements, albeit a short list, for use in this research.

A review of methodologies supporting composition-based development and the extent to which they support dynamic evolution is presented next. The latter part is discussed at a high level since there lacks a comprehensive and detailed set of dynamic evolution feature requirements.

2.4 METHODOLOGIES SUPPORTING COMPOSITION-BASED DEVELOPMENT

As a vast number of methodologies exist today, it is prudent to have a means of selecting relevant and suitable methodologies of interest for this research. To facilitate this, the following selection criteria were used:

- Composition-based development is a main focus in the methodology, even though it may also be suitable for other paradigm(s) (e.g. OO);
- The methodology supports both analysis and design aspects of software development;
- The methodology offers dedicated support that may facilitate changes, maintenance and/or evolution of an application;
- The methodology has been applied in a real world project and/or a case study; and
- Documentation of the methodology is available and accessible in the public domain to ensure it could be independently reviewed.

Development methodologies meeting the selection criteria and thus included in the evaluation were: ASG, Catalysis, CBDI-SAE, Erl's (2005), EPIC, Kobra, OPEN Process Framework, Oreizy et al.'s (1999) (referred to as "AEM"), Papazoglou and van den Heuvel's (2006), Rational Unified Process, SeCSE, Select Perspective and SUPER. Because of the extensiveness of a number of these methodologies and space constraints, it is not possible to discuss each methodology in detail. In contrast, the purpose is to review them in the context of support for dynamic evolution. The reader is referred to their respective documentation for further details. Several methodologies and approaches not satisfying all the criteria were also considered but excluded from this research:

- Methodologies: Bennett and Rajlich (2000), Bohner (1996), Business Component Factory (Herzum & Sims 2000), Chang and Kim (2007), COMET (SINTEF 2007), COMO (Cho et al. 2004), Change Management process of Information Technology Infrastructure Library (ITIL) (Office of Government

Commerce 2005a), Freedom Service-Oriented Methodology (LDJ Trust 2003), IBM's Service-Oriented Modelling Architecture (SOMA) (Arsanjani 2004), Incremental Change (Rajlich & Gosavi 2004), Jones and Morris's (2005) submission to OASIS as an SOA Adoption Blueprint, MaRMI-III (Ham et al. 2004), PLASTIC (Autili et al. 2007), SCIPIO (Veryard 1998), Service-Oriented Architectural Framework (SOAF) (Erradi et al. 2006a), Service Oriented Unified Process (SOUP) (Mittal 2006), the Software Change process model (Petrenko et al. 2007), UML Components (Cheesman & Daniels 2001), Web Services Implementation Methodology (WSIM) (Lee et al. 2006); and

- Approaches: Altunel and Tolun (2007), Baglietto et al. (2005), Bohmann et al. (2003), CADA (Hubbers & Verhoef 2000), Castek Inc.'s CBD/e (Sparling 2000), CBD96 (Cho et al. 2004), Gu and Lago (2007), IBM's Service-Oriented Analysis and Design (SOAD) (Zimmermann et al. 2004), M4SOD (Kim & Yun 2006), O2BC (Ganesan & Sengupta 2001), Rapid Service Development (RSD) (GigaTS 2001), Uniface (Howard 2006), Zhang et al. (2007), Zhang et al. (2009).

A challenge to evaluating and/or comparing several methodologies is that they may use key methodological terms to mean different things, and different terms to mean the same thing. To counter this effect, a cross reference between terms used in the reviewed methodologies and those defined by the International Standard ISO/IEC 24744 (ISO/IEC 2007) was developed as documented in Table 2.3, to consistently guide the study/evaluation/comparison (e.g. mapping "deliverable" and "artefact" to "work product" in the Standard).

Table 2.3 Terminology mapping between SEMDM (ISO/IEC 24744) and reviewed methodologies

Methodology	Producer	Work Unit			Work Product	Stage With Duration		
		Task	Guideline, Technique	Process		Time Cycle	Phase	Build
ASG	role	activity and task	technique	process	artefact	lifecycle	phase	
Catalysis		process pattern	process pattern	activity, route	deliverable (including model, framework, component)		phase	cycle
CBDI-SAE	role	process unit	guideline, template	discipline	deliverable	lifecycle		cycle
EPIC	worker	task and step	guideline, tool-mentor, template	activity or workflow	artefact	process, lifecycle	phase	iteration
Erl (2005)		step	principle, guideline	process	model, document	lifecycle	phase	
KobrA		activity		process	artefact	product line lifecycle	engineering process	

<i>Methodology</i>	<i>Producer</i>	<i>Work Unit</i>			<i>Work Product</i>	<i>Stage With Duration</i>		
		<i>Task</i>	<i>Guideline, Technique</i>	<i>Process</i>		<i>Time Cycle</i>	<i>Phase</i>	<i>Build</i>
OPEN Process Framework	rôle, direct producer	task and step	technique	activity, work flow	work product	lifecycle process	phase	
Oreizy et al. (1999)		task	technique	process		lifecycle		
Papazoglou and van den Heuvel (2006)		activity and task/step	guideline	phase	artefact	lifecycle	phase	iteration
RUP 2003 and v7.2.0.1	role	activity and step (2003) task and step (v7.2.0.1)	guideline, tool-mentor, template	workflow (2003), discipline (v7.2.0.1)	artefact	process	phase	
SeCSE	role	process/sub-process and task	guideline, technique	process and phase	work product	sequence of processes		
Select Perspective	role	activity or step	technique	workflow	artefact	lifecycle		iteration
SUPER	role	activity or step	technique, method	phase	deliverable	lifecycle	phase	

Notes about International Standard ISO/IEC 24744 terms:

1. A “Producer” is a person, a role, a team or a tool responsible for executing a particular work unit according to his/her/its areas of expertise.
2. A “Work Unit” prescribes steps to follow (i.e. process), what is to be done (a task) or how to achieve its objectives (guideline or technique).
3. A “Work Product” is an artefact produced and/or used in software development, such as a document or a diagram.
4. A “Stage With Duration” is a managed interval of time to achieve a particular outcome. It can be producing a newer version of an existing work product (designated with “Build”), the delivery of a final product or service (designated with “Time Cycle”) or a period of a specific cognitive activity such as requirements analysis (designated with “Phase”).

It is not unusual for a methodology to be created as an extension, specialisation and/or improvement of another. Thus, when studying a methodology, it is useful to also look at its precedents to improve the understanding of the methodology. Figure 2.5 describes the precedence relationships for the reviewed methodologies, with each line indicating that a methodology is built on another (e.g. Catalysis precedes Kobra).

Methodological aspects that fall outside of the scope of this research (Section 1.3) are not included for review. Other than that, a review of each selected methodology covers:

- a brief description about the methodology and its background information;
- support for composition;
- the process and/or lifecycle aspect (a.k.a. time cycle); and
- the extent to which (dynamic) evolution is supported.

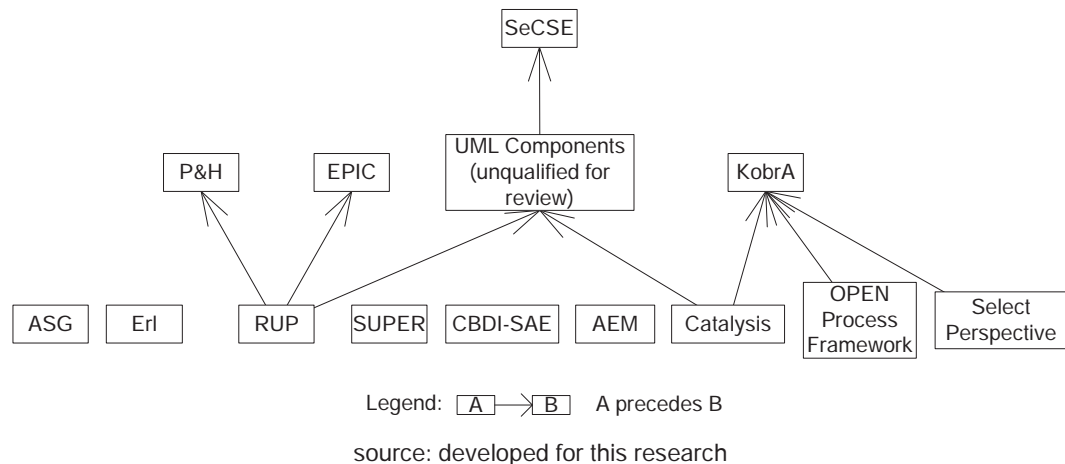


Figure 2.5 Precedence relationships of reviewed methodologies

2.4.1 ASG

The Adaptive Service Grid (ASG) was a project sponsored by the European Commission, (Lehner et al. 2006). Its goal is to develop a prototype platform (called the “ASG platform”) to support automatic adaptive applications in an SOA environment. More specifically, ASG incorporates semantics into services of applications running on the ASG platform to improve the applications’ agility and adaptability. Semantics is also used in ASG to automate the planning, enacting and monitoring of service compositions in applications running on the ASG platform in the following fashion. At runtime, an ASG application and the ASG platform respond to service changes and availability. When an ASG application receives a user initiated request, it converts the request into a *user goal*. (i.e. *planning* started). The goal is then forwarded to the ASG platform which searches for and selects a service (or a composition thereof) from the platform’s internal database. Next, an agreement on the quality of service is made with the provider offering the selected service (or with a set of providers offering services in the selected composition) for fulfilling the goal. Afterwards, an *enactment* takes place whereby the user goal is delivered to the selected provider (or a set of them) to fulfil the goal. During *monitoring*, a failed enactment (i.e. without meeting the agreement) triggers replanning and re-composition of the services and subsequently a reiteration of the whole lifecycle.

support for composition

An ASG Service is a service in an SOA environment and annotated with semantic specifications. An ASG application is composed of ASG services based on their semantics, and these services can also be composed of smaller ASG services.

process

ASG consists of three development processes, servicing three levels of abstraction in the ASG environment:

- *Application Engineering* process, concerning the development of service-oriented applications using the ASG platform and its provided services. Requirements are elicited from which user interfaces, workflows and services are identified. Design, implementation and testing follow;
- *Service Engineering* process, concerning the development and registration of services for the required functionality; and
- *Platform Engineering* process, for developing the ASG platform itself which supports SOA-based applications and their services.

support for (dynamic) evolution

ASG uses its own database to store its domain ontologies and to register services and applications as they are updated and released. ASG applications, services and the platform undergo three phases in their lifespan: from initial *development*, ongoing *maintenance and evolution*, to eventual *retirement*. During the *maintenance and evolution* phase, four kinds of work are noted - correct defects, add new functionality, reengineer or redesign artefacts, and migrate artefacts to a new environment (e.g. new standards) but no task or activity is defined. Apart from that, an *ASG Platform Feedback* document is produced to capture new requirements for the current release of the platform, and to drive new development.

2.4.2 CBDI-SAE

CBDI Service Architecture and Engineering (CBDI-SAE) is a proprietary framework built from experience from real-world project and feedback from users (Allen 2007; Allen & Brown 2007). CBDI-SAE proposes a metamodel for SOA concepts, process definitions, a reference architecture and SOA deliverable templates. CBDI-SAE maintains a proprietary subscription-based forum to conduct online discussions on the experience, best practices and guidelines of using CBDI-SAE. Restricted access to this forum limits the review of CBDI-SAE.

support for composition

CBDI-SAE offers its own metamodel for the SOA paradigm based on experience from projects using relevant SOA standards and technologies (Dodd et al. 2007). A “service”

is described by one or more “service specification”, each implemented by one or more “automation unit”. Smaller automation units can be composed into one unit.

process

CBDI-SAE defines four “disciplines” (i.e. processes) and collectively called the *Service-Oriented Process* to construct SOA solutions:

- *Manage*, for assessing and planning the adoption of the SOA paradigm, and defining SOA governance capabilities to manage services;
- *Consume*, for iteratively identifying business requirements and business improvements, and specifying services and service-oriented solutions assembled from those services;
- *Provide*, for realising services and a service-oriented solution with suitable technologies to meet the business needs of service consumers; and
- *Enable*, for designing a technology infrastructure to run and manage services.

support for (dynamic) evolution

Once an SOA solution is in operation, CBDI-SAE’s Service-Oriented Process conducts continuous monitoring of the solution, and refines the design and implementation of services as required. Two artefacts are under version control: service specifications and application specifications (Dodd et al. 2007). Support for evolution is not evident in CBI-SAE’s documentation.

2.4.3 Erl

Erl (2005) proposed a comprehensive development methodology for SOA-based solutions (hereafter referred to as “Erl”). It defines standard- and technology-based service-oriented principles and development activities for designing SOA-based solutions. Erl distinguishes the design of individual services from the design of their composition to build agility in an SOA-based solution. It models an SOA platform on three layers of abstraction:

1. *Orchestration service layer*, consisting of business processes composed of services from the business service layer;
2. *Business service layer*, comprising two types of business-oriented services: a *task-centric service* encapsulating business logic specific to a business process or task (e.g. order fulfilment) and an *entity-centric service* representing a reusable business entity (e.g. purchase order); and

3. *Application service layer*, consisting of reusable application services which represent technology-specific logic, including utility services (e.g. for database access) and wrapper services (for legacy systems).

This layered approach brings agility to an SOA-based solution and facilitates adaptation to evolution. For instance, technological changes are confined to the application service layer.

support for composition

Erl adopts common definitions of service and hence composition from relevant SOA standards and technologies.

process

Erl features three processes called “strategies” to suit different objectives (Table 2.4).

Table 2.4 Erl's strategies

<i>Strategy</i>	<i>Description</i>	<i>Benefit(s)</i>	<i>Drawback(s)</i>
Top-down	Start from the enterprise business models, progressively identify and design services from higher to lower layers.	Aligned to the business models, high quality SOA	Significant up-front analysis work
Bottom-up	Construct ad hoc application services on demand, and perform integration with other services to build services in the higher layers as needed.	Quick reuse of existing applications, quick delivery of implementations	Difficulty in attaining high quality SOA
Agile (a.k.a. meet-in-the-middle) (see note)	Combine both top-down and bottom-up approaches.	Aligned to the business models, quick delivery of implementation	Overhead in rework, redesign and refinement

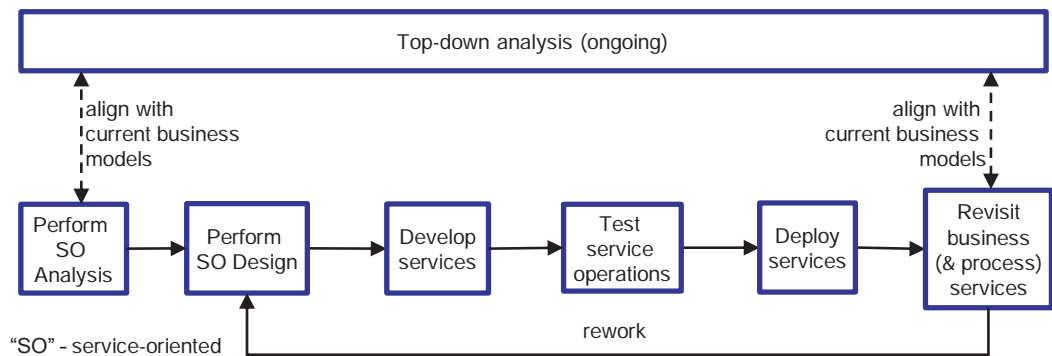
source: Erl (2005)

Note: Erl (2005) defines the term “agile” in a peculiar way that it does not accord with the term “agile” in a agile software development approach such as Extreme Programming (Beck & Andres 2005).

support for (dynamic) evolution

Erl's agile strategy is the most relevant to evolution and maintenance, consisting of two parallel streams of activities: top-down analysis and development. The top-down analysis stream conducts on-going analysis of business goals at an enterprise level to derive (new) business objectives (top of Figure 2.6). Once sufficient and new objectives from the analysis have been identified, the development stream (bottom of Figure 2.6) commences in parallel, producing service-oriented artefacts that fit into the three layers. At the end of this stream, the design is compared with the current business models at hand which may have evolved to identify discrepancies (in the “Revisit business and process services” phase). Rework is performed to build extensions, redesign, redevelop, retest and redeploy new artefacts. Apart from the above description, Erl

does not define specific tasks and activities to handle evolution and maintenance.



source: redrawn from Erl (2005)

Figure 2.6 Erl's (2005) agile lifecycle model

2.4.4 Oreizy et al. (AEM)

Oreizy et al. (1999) prescribe a methodology to support self-adaptive software systems. Such systems modify their own behaviour in response to changes in their operating environment observable by the systems. Their methodology centres on the notion of a software architecture in planning, coordinating, monitoring, evaluating and implementing adaptation, by means of both adaptation and evolution management (hereafter the methodology is referred to as "AEM"). These management aspects are described later in this section.

support for composition

To embrace dynamic changes at the architectural level, AEM advocates the notion of a "C2-style" architecture (Medvidovic et al. 1999). In such an architectural style, components are distributed in layers. Components in one layer can only use and communicate with components in the layer above it. A connector links components between two layers and provides an abstraction for component communication using asynchronous message exchanges.

process

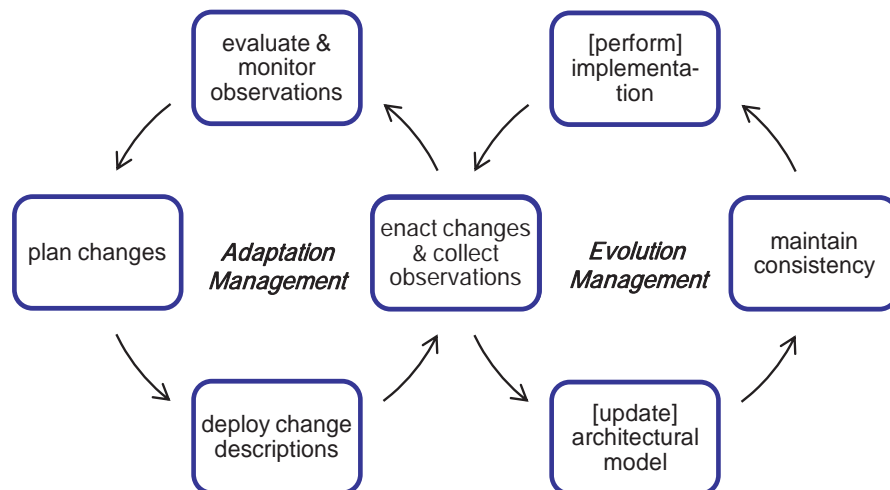
AEM's lifecycle model consists of two interlocked processes which can be performed by humans or fully automated. The *adaptation management* process (left of Figure 2.7) consists of the following four tasks:

- *Evaluate and monitor observations*, of a software system's execution and its environment;

- *Plan changes*, which involves accepting evaluations and defining an appropriate adaptation and its blueprint for execution;
- *Deploy change descriptions*, to the software system's implementation environment according to the defined adaptation; and
- *Enact changes and collect observations*, to carry out change implementation.

The *evolution management* process (right of Figure 2.7) focuses on the mechanisms employed to modify the software system. It continues after Task “enact changes and collect observations” of the adaptation management process with the following tasks:

- *[Update] architectural models*, to accommodate the changes specified;
- *Maintain consistency*, between the architectural model and the implementation as a result of the accommodated changes;
- *[Perform] implementation*, which reflects changes in the architectural models in the software system's implementation; and
- *Collect observations*, as part of the task “enact changes and collect observations” of the adaptation management process, to provide feedback to the adaptation management process for further adaptation needs.



source: redrawn from Oreizy et al. (1999)

Figure 2.7 Oreizy et al.'s. (1999) lifecycle model

support for (dynamic) evolution

AEM recognises additions, removals and replacements of components and connectors, changes to their parameters, and modifications to the topology of an architecture. AEM's evolution management process is the most relevant to dynamic evolution, although it is brief as to the details of tasks described in its literature.

2.4.5 SUPER

The Semantics Utilised for Process Management within and between Enterprises (SUPER) project aims at shifting business process management (BPM) away from the information technology (IT) level where BPM is mostly and commonly addressed, to the business level (SUPER 2007). To achieve this, SUPER brings the notions of semantics and ontology into BPM and creates several deliverables to support BPM. The SUPER methodology guides semantic business process (SBP) development in a lifecycle.

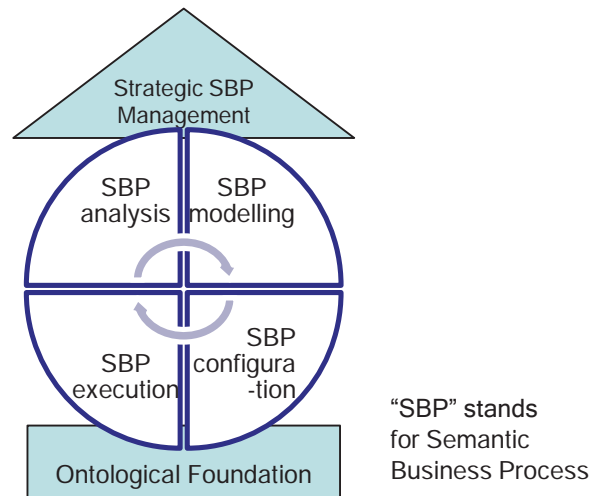
support for composition

The notion of composition is manifested in a business process model within which process elements (e.g. a business activity) can be added from a repository or library, updated and removed. Existing process elements can also be discovered from a repository or library before being added to the model.

process

The SUPER methodology's lifecycle addresses two layers of concern. The *Ontological Foundation* layer defines a set of ontologies for BPM while the *Strategic Semantic Business Process Management* layer defines the business goals from a strategic and organisational perspective and initiates the lifecycle to create "deliverables" (i.e. work products) to accomplish the goals. The two layers of concern are addressed with four ordered "phases" (i.e. processes) in the methodology's lifecycle (Figure 2.8):

- *SBP modelling*, for defining a business process model with ontological annotations.
- *SBP configuration*, which defines semi-automated mapping of the business process model to an IT-oriented executable form.
- *SBP execution*, for runtime discovery of semantic web services, composition of them into business processes, and execution of the business processes in respective IT systems.
- *SBP analysis*, for monitoring and analysing business processes in execution against their intended objectives from which change and optimisation recommendations are made.



source: redrawn from SUPER (2007)

Figure 2.8 SUPER's phases and layers

support for (dynamic) evolution

SUPER notes that continuous process improvement (i.e. evolution) to adapt a business to internal and external changes is achieved through repeating the lifecycle. For example, reports produced in the analysis phase are used in the SBP analysis and modelling phases to drive enhancement of existing business process models and development of new ones. There is no support for identifying and accommodating changes into business processes to handle evolution.

2.4.6 Rational Unified Process

The Rational Unified Process (RUP) is a commercial process framework developed at Rational Inc., now part of IBM Inc. It consists of a variety of pre-tailored process forms to suit various types of applications (Kruchten 2003). Earlier editions of RUP were originally published as books (e.g. Kruchten 2003). RUP is bundled into IBM's Rational Method Composer (RMC), a tool for RUP. A variant of RUP is OpenUP, an open-sourced and subset version of RUP (The Eclipse Foundation 2009).

support for composition

RUP adopts the common definitions of component and service from relevant SOA standards and technologies to define composition (UML component, business process etc.). To provide support for component-based and SOA development, RMC includes specialised plug-ins: "COTS Package Delivery" (for COTS based systems), "Asset Based Development" (for product lines), "SOA" and "J2EE" (a commercial component technology).

process

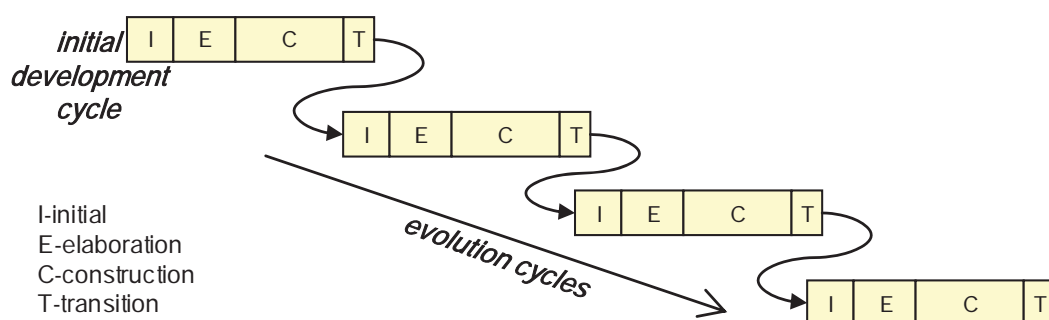
RUP's process metamodel, known as the Unified Method Architecture (UMA), unifies three methodological approaches from IBM: RUP (2003 version), Global Services Method (GS Method) and Rational Summit Ascendant. A RUP "process" (i.e. time cycle) is described by "disciplines" (i.e. processes) and phases. RUP predefines nine disciplines, each a collection of tasks relevant to a major area of concern within a project. A phase marks a significant period in a project and ends with a key milestone and/or deliverables. RUP's development cycle consists of four phases:

- *Inception*, for defining a software product's vision and the scope for a project;
- *Elaboration*, for defining the overall architecture for the product plus resource and activity planning;
- *Construction*, for gradual refinement of the architecture and construction of the product to meet the vision; and
- *Transition*, for rolling out the product, viz. a particular "software generation", to end users.

Each phase emphasises the involvement of each discipline differently (e.g. more effort for the Business Modelling discipline during the Inception phase but less during the Elaboration phase). A phase is sometimes subdivided into "iterations" to make its objectives smaller to manage and its progress easier to track.

support for (dynamic) evolution

RUP supports evolution by way of repeating development cycles as "evolution-cycles" (Figure 2.9), to progress a software product into successive releases.



source: redrawn from Kruchten (2003)

Figure 2.9 An example of RUP's evolution cycles

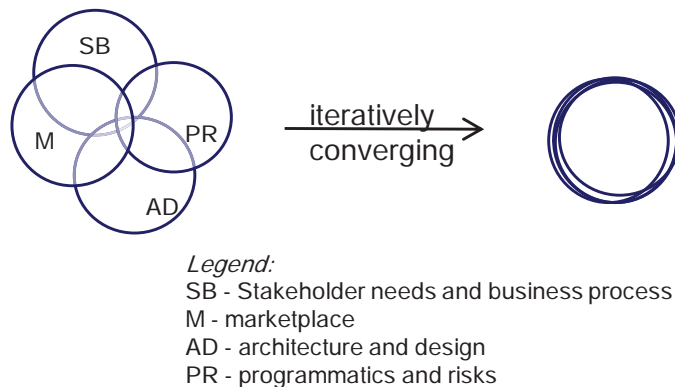
Recognising the scope and importance of product roll-out, RUP offers the *Deployment* discipline to separately deal with release management. However, neither RUP nor its Deployment discipline deals with how software changes will be accommodated into a

live application and how the application is promoted to its new version.

2.4.7 EPIC

The Evolutionary Process for Integrating Commercial-off-the-Shelf (COTS) Based Systems (EPIC) methodology is an extension to RUP (Section 2.4.6) and published by Software Engineering Institute, Carnegie Mellon University for supporting system development with COTS and pre-existing components (Albert & Brownsword 2002).

When building a solution, EPIC balances “four spheres of influence” - *stakeholder needs and business process*, *marketplace*, *architecture and design*, and *programmatics and risks* - each denoting a set of criteria for the solution to be built. A project kick-off defines the criteria for the solution in each of these four spheres. On the left of Figure 2.10 is an example configuration of these spheres. The area where all the spheres overlap represents criteria for the solution agreed among the spheres whereas the rest of the area bounded by the spheres highlights their disparate and contradictory needs. As the project progresses, new information is gathered (e.g. what the actual needs are and what can be built from the marketplace at a reasonable cost) to drive stakeholder buy-in to resolve and reduce disparate and contradictory stakeholder needs. This is reflected by an increase in the overlapping area of the spheres such as the one shown on the right of Figure 2.10.



source: adapted from Albert and Brownsword (2002)

Figure 2.10 EPIC's trade space

support for composition

In EPIC, a component can be a software component, a hardware unit or a legacy system. A composition is formed by integrating components.

process

To converge the trade space, EPIC iterates five “activities” (i.e. processes) repeatedly.

Each activity is a collection of tasks relevant to a major area of concern within a project.

The activities are:

- *Plan the iteration*, for analysing and refining the information from the four spheres, and defining the objectives of the iteration (e.g. specific artefacts to be developed);
- *Gather information*, which is needed to meet the iteration objectives (e.g. information about COTS components from the market);
- *Refine the understanding of the solution*, which combines the information gathered from the last activity to develop candidate solutions, and understands their similarities and differences;
- *Assemble an executable representation*, which builds and tests the candidate solutions; and
- *Assess the iteration*, which validates the candidate solutions against the iteration objectives and selects a solution from the candidates.

In addition to the activities above, EPIC refines RUP's four "phases" (i.e. time cycles) in a project timeline. A phase marks a significant period in a project and may end itself with a key milestone and/or deliverable. A phase defines what aspects of each process should be emphasised during the phase. For instance, at the beginning of a project, the Gather Information activity focuses on identifying suitable COTS components from the market whereas towards the end of the project, this activity focuses on monitoring releases of new component versions from the market so as to contemplate possible upgrades.

support for (dynamic) evolution

Ongoing minor changes to a solution (e.g. new component versions) occur in the final of the four phases until the solution is retired or replaced by a new one. Significant changes will require an initiation of a new project going through all the phases. None of the activities specifically discuss how maintenance or evolution is handled.

2.4.8 Papazoglou and van den Heuvel (P&H)

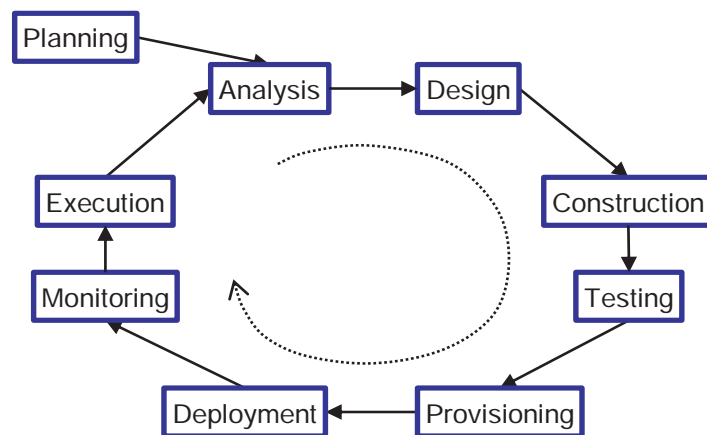
Papazoglou and van den Heuvel (2006) feature a design and development methodology, based on RUP (Section 2.4.6), for developing SOA implementations (hereafter referred to as "P&H") suitable for use by both service producers and consumers.

support for composition

P&H adopts the common definitions of service and hence composition from relevant SOA standards and technologies.

process

P&H defines a preparatory and eight iterative “phases” (i.e. processes) in its lifecycle (see Figure 2.11). The preparatory phase called *Planning* kicks off subsequent phases of development by defining the requirements for a service-oriented solution, its scope and nature. This phase also aims at understanding the feasibility of the solution, and how it will fit into an organisation’s business environment.



source: redrawn from Papazoglou and van den Heuvel (2006)

Figure 2.11 Phases of Papazoglou and van den Heuvel's (2006) methodology

The eight iterative phases drive “artefacts” (i.e. work products) towards completion taking into account both technical and business concerns. Each iteration starts with the *Analysis* phase during which requirements are investigated in accordance with business goals and objectives. A “gap analysis” is conducted between the current (as-is) business process model and the future (to-be) process model to analyse potential changes to current applications and processes required, and to identify opportunities for redesign and reuse of existing applications and processes. Then, various realisation options are examined and from which one is chosen. During the *Design* phase abstract business process models are transformed into technical business processes and Web services according to P&H’s service design principles. During the *Construction* phase, Web services are implemented by packaging existing applications as Web services, by composition of existing Web services and/or applications, or by coding new Web services. During the *Testing* phase, Web services and business process implementations are verified as meeting specified requirements.

The *Provisioning* phase follows, during which the policing of service provisioning in compliance within an organisation's or others' business models is defined. This covers service governance, certification, metering and billing (for service usage). Afterwards, the solution is rolled out to and used in its prospective environment in the *Deployment* phase. Measurements, monitoring and reporting are conducted regularly during the *Execution* and *Monitoring* phases to ensure business services meet their service level agreements and improve their quality of service.

support for (dynamic) evolution

P&H does not explicitly consider maintenance or evolution, although continuous monitoring and improvement of services and applications are noted in the *Monitoring* phase.

2.4.9 Catalysis

Catalysis (D'Souza & Wills 1998) is a development methodology for the design and construction of component-based software using objects, frameworks and components. Catalysis has its roots in many well known OO techniques and principles, and incorporates an earlier version of Unified Modelling Language (UML, v1.0) as its core modelling notation.

support for composition

Catalysis centres on three modelling concepts in composition - "type, collaboration and refinement" - plus frameworks to describe the recurring patterns of these concepts. A type specifies an external and observable behaviour of an object or a component. Collaboration defines the schemes of interaction among objects/components together with the concept of "actions". An action characterises the effect of a particular behaviour or state change of a type. Through iterative refinement, types and collaborations are evolved from abstract and simple to concrete and detailed architectural design.

process

Catalysis assumes that "there is no single process that fits every project" (D'Souza & Wills 1998, pp31). Therefore, it does not prescribe a formal lifecycle process for software development. Instead, it offers several patterns (both modelling and process oriented), design principles, how-to's, and guidelines for use during development. Catalysis emphasises that artefacts are almost never produced in sequential tasks and that a development lifecycle is largely iterative and incremental. It proposed two composition "routes" (i.e. processes) as starting points for planning a project:

- The *build* route for constructing a system from scratch using four steps: construction of requirements, construction of system specification, architectural design and component implementation; and
- The *assembly* route suitable for developing systems with existing components: construction of requirements (same as the build route), construction of system specification (same as the build route), construction of component type models, construction of component type mapping, refinement of system specification and refinement of domain model.

support for (dynamic) evolution

Catalysis identifies components and systems as the unit of maintenance and notes that certain running systems require upgrade without shutdown. More generally, upgrade is handled during the deployment phase of a project. Catalysis also prescribes the notion of a “snapshot” diagram to model the effect of a change on a structure over two time instants (pre- and post-change). Beyond that, it does not elaborate on maintenance and evolution in its features.

2.4.10 OPEN Process Framework

The OPEN Process Framework (OPF) is a comprehensive and fragment-structured methodology for creating and tuning a process for developing a variety of applications - OO, web, component-based etc. - (Firesmith & Henderson-Sellers 2002). OPF is available on its public web site (OPFRO 2009), with ongoing enhancements. To start a project with OPF, one constructs a “process instance” by selecting from OPF’s framework suitable elements called “process components”. They are then assembled to form a lifecycle model, with “phases” (i.e. time intervals placed in a sequence for iteration) and “activities” (i.e. processes).

support for composition

A component is regarded as a software package (Haire et al. 2001). Components can be custom built or COTS based (Henderson-Sellers et al. 2005). They are used for assembling software systems (Haire et al. 2001).

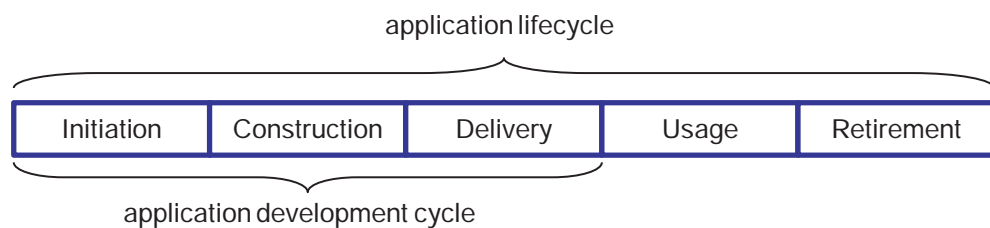
process

OPF offers process support for component-based development involving COTS and/or custom-built components, with the *Component Selection* “activity” (i.e. process) being the most specific to component-based development, drives the development effort to

establish policies on component acquisition, to evaluate and select components, and to adapt/integrate components into an architectural design. To a lesser extent, the *Integration*, *Evaluation* and *Reuse Engineering* activities are also pivotal to successful component-based development.

OPF defines five technical phases⁶ to describe the temporal existence of an application during its full “application lifecycle”, with the first three phases collectively called the “application development cycle” during which an application is developed (Figure 2.12):

1. *Initiation*, for development kick-off;
2. *Construction*, for artefacts development;
3. *Delivery*, for releasing artefacts to an operating environment;
4. *Usage*, for placing artefacts into service; and
5. *Retirement*, phasing out and decommissioning artefacts from being used.



source: redrawn from Firesmith & Henderson-Sellers (2002)

Figure 2.12 Application phases in OPF

Throughout a development cycle, activities are identified and performed to create the intended work products. Each activity is calibrated with a different weighting in each phase according to its necessity and importance during that phase. For example, the Initiation phase places more emphasis on requirement analysis which tapers off as a project nears completion.

support for (dynamic) evolution

OPF has an activity devoted to maintenance which aims to incorporate minor fixes and enhancements after the initial deployment of an application for use. It addresses four types of maintenance - adaptive, corrective, preventative and predictive - covering data, software, hardware and documentation artefacts. Apart from that, it does not deal with evolution or dynamic aspects of evolution.

⁶ The online live version of OPF has two additional phases after the Construction phase: *Initial Production* and *Full Scale Production* (OPFRO 2009).

2.4.11 Select Perspective

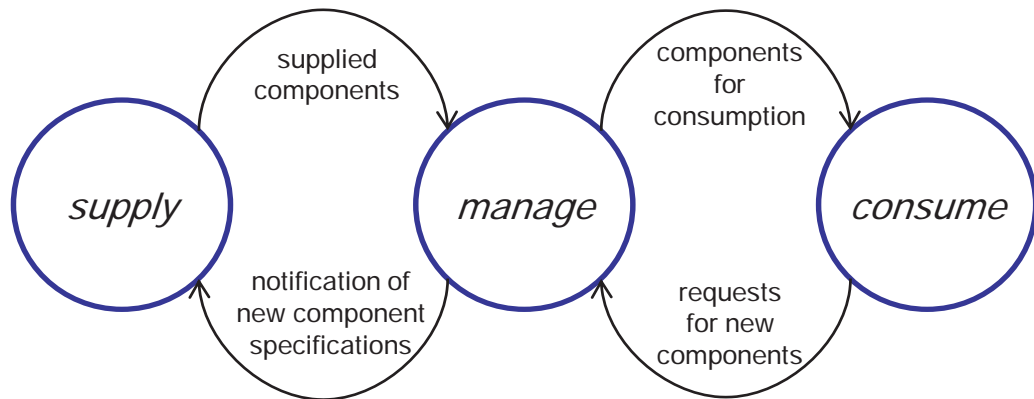
Select Perspective, as part of the commercial tool called Select Process Director Plus, was developed by Select Business Solutions, Inc. with a view to leveraging parallel development activities to reduce the time-to-market for service- and component-based applications (Apperly et al. 2003).

support for composition

Unlike some methodologies that adopt public SOA and component standards, Select Perspective defines its own concepts for services and components. A service is regarded as a “publicly exposed capability (functionality)” of an application for others to use the service (e.g. at a business level) and is more like a service end-point than a service in a service-oriented architecture (Papazoglou & Georgakopoulos 2003). A component is seen as a package of related services and conforms to a component model in that it is a unit of composition and deployment. Definitions for service and component concepts tend to overlap and “there is little difference between using a component and a Web service” (Apperly et al. 2003, p. 20).

process

In Select Perspective, applications are constructed based on the “supply-manage-consume” model (Figure 2.13). Component suppliers construct, deliver and maintain components for use. Component managers source candidate components from suppliers, perform certification, classify the components and then publish them in their own component repositories. System developers (i.e. consumers) search repositories for suitable components and apply them to their system development. If no components are found, developers either build their own, or register new component specifications with component managers. Managers notify potential suppliers to provide components built for the new specifications.



source: redrawn from Apperly et al. (2003)

Figure 2.13 An expanded view of Select Perspective's supply-manage-consume model

Select Perspective's development lifecycle is based on the supply-manage-consume model and correspondingly three basic "workflows" (i.e. processes). In the *Supply* workflow, suppliers drive the work required for the construction and provisioning of components according to the needs of consumers. The workflow also handles the extraction of components (termed "componentisation") from existing legacy systems. In the *Consume* workflow, consumers conduct delivery, maintenance and support of application composition from acquired components. Lastly, managers perform the *Manage* workflow covering component acquisition, quality assurance (i.e. certification), publication and maintenance of components.

support for (dynamic) evolution

Select Perspective has limited support for maintenance but no specific support for evolution. In its Supply workflow, component defects and change requests are resolved to ensure a business's to continually meet its needs. In its Manage workflow, components undergo configuration control to track their deployments as they are changed. In its Consume workflow, ongoing work on an operational application is carried out to resolve errors and meet new business demands.

2.4.12 Kobra

Kobra (Atkinson et al. 2002) was developed as part of a project funded by the German Ministry of Education and Research. It integrates three paradigms - component-based development, product line engineering and model driven architecture (OMG 2003a) - to lower the time-to-market and cost of application development via high levels of software reuse. Component-based development regards components as the unit of reuse. Product line engineering to software development exploits the fact that a family of

similar products is developed by many organisations. It consolidates common parts (termed “commonalities”) into a single high-quality core, and facilitates the customisation and adaptation of variable parts (termed “variabilities”). The model driven architecture paradigm separates the development of models and architectures from their implementation with specific technologies so that designers and developers make the best of both domains. While Kobra has a product line focus, it can also be applied to single application development by skipping the commonality/variability analysis in its development lifecycle. Kobra is influenced by Catalysis (Section 2.4.9), OPF (Section 2.4.10) and Select Perspective (Section 2.4.11).

support for composition

Kobra defines its notion of components - “Komponent”. A Komponent is prescribed by a “specification” (what it does), a “realisation” (into smaller Komponenten or particular OO designs), and optionally an “implementation” (for translation into programming code). A Komponent model forms a hierarchy. Komponenten at a higher level of abstraction are refined into smaller Komponenten at a lower level of abstraction. Reuse is achieved by mapping the specification of an existing Komponent or a third-party component to the Komponent specification required for an architecture.

process

Kobra has two processes in its lifecycle model:

- *Framework engineering*, for constructing and maintaining a framework for a variety of applications. This process begins with the context realisation activity, analysing the business problems a product line is intended to solve, plus commonalities and variabilities for its framework and its applications. Komponent modelling tasks then follow, with variants for each application instantiated from the framework factored into the framework’s design.
- *Application engineering*, for constructing an application for a particular business need. This process starts with instantiating a skeleton application from the framework, instantiating a concrete context in which the application operates, and then applying a “decision modelling” task to incorporate suitable variable parts into the application (e.g. adding a data back-up service for a business transaction system). Custom Komponenten are modelled iteratively and added to the application where needed. The application is then translated to binary code and packaged into a deliverable.

support for (dynamic) evolution

KobrA incorporates maintenance into its processes, in recognition of the complexity and size of managing changes and evolution of a product line, its framework and applications. Maintenance work is managed by two supplementary processes: “configuration management” (versioning, updating and configuring artefacts) and “change management” (registering, evaluating, applying and propagating changes on static artefacts originating from development only). KobrA regards evolution as part of maintenance (Atkinson et al. 2002, pp40) and prescribes an “evolution graph” to track new releases, updates and retirements of assets - including applications and their parts - across a product line.

2.4.13 SeCSE

The Service Centric System Engineering consortium (SeCSE) aims to provide free and open source instruments for engineering of service-centric applications (SeCSE 2007). It offers deliverable tools, methods and techniques supporting cost-effective development and use of service-centric applications. SeCSE is co-funded by the European Commission under the “Information Society Technologies” programme. SeCSE is based on UML Components (not reviewed) which is built on RUP (Section 2.4.6) and Catalysis (Section 2.4.9).

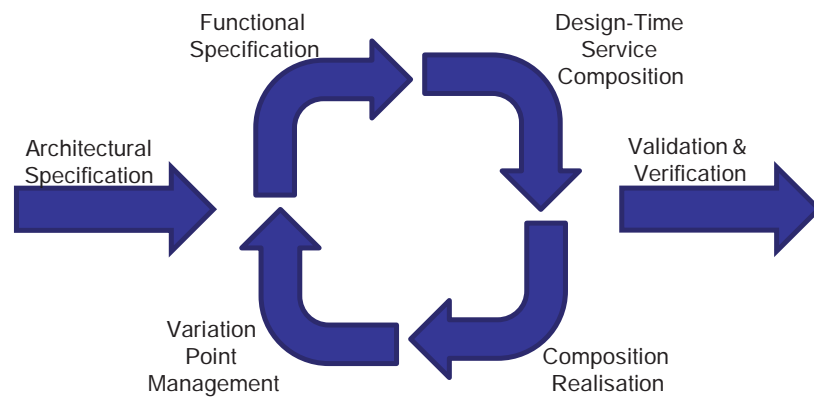
support for composition

SeCSE defines a service as a unit of composition with abstract and concrete aspects (Bastida 2008), respectively specifying its functionality and an appropriate implementation of the functionality. A composition or workflow is defined by combining a set of abstract services to fulfil the requirements for the composition as a whole. To realise a composition, potential concrete services that can play the roles of the abstract services in the composition are identified, evaluated, selected and then bound to the composition.

process

SeCSE offers processes in four areas: *Service-Centric System Engineering* for service-based application development, *Service Engineering* for individual service development, *Service Acquisition/Provisioning* for sourcing services from external suppliers or third parties, and *Validation/Verification*. Within Service-Centric System Engineering is the SeCSE composition methodology comprising an ordered execution of the following processes in a lifecycle (Figure 2.14):

- *Architectural Specification*, for defining and maintaining domain- and organisation-wide architecture models;
- *Functional Specification*, for specifying the functionality of an intended composition;
- *Design-Time Service Composition*, for designing appropriate service descriptions in a composition for the identified functionality;
- *Variation Point Management*, for defining areas in a composition that might change and design variability into the composition;
- *Composition Realisation*, for discovering and selecting existing services, or development of new services; and
- *Validation and Verification*, of a composition against the functionality.



source: redrawn from Bastida (2008)

Figure 2.14 SeCSE composition methodology

support for (dynamic) evolution

SeCSE features three work units to add dynamism to a service composition (SeCSE 2006), addressing dynamic evolution to a certain extent by accommodating *anticipated* changes (Almeida et al. 2001; Karsai et al. 2010) in a composition:

- *Binding and re-binding* of services, for replacing a service at runtime with similar functionality and sometimes better performances;
- Composition *re-planning*, for modifying a composition at runtime to recover a service's lost functionality that cannot be offered by any existing service; and
- *Variation points* in a composition, for defining places in a composition's design where there will be different alternatives (e.g. currencies) to execute an abstract function (e.g. bill payment). (At runtime, services from different alternatives are chosen and executed in the composition.)

Apart from the above, SeCSE does not offer support for handling *unanticipated* changes (Almeida et al. 2001; Karsai et al. 2010), such as those arising from new business needs.

2.4.14 Observations from Selected Methodologies

A summary of the reviewed methodologies, in terms of their support for composition and evolution is documented in Table 2.5. All the selected methodologies support the notion of composition. Whilst the selected development methodologies offer a variety of capabilities and approaches for handling conventional development activities such as requirement analysis, design and testing, they do not comprehensively support maintenance or evolution as part of their mainstream activities. For instance, although the term evolution is noted in Catalysis, it does not elaborate on challenges and issues relevant to evolution. AEM dedicates a separate process for dynamic evolution but lacks details on its tasks, techniques and what kinds of work products are created. SeCSE and ASG offer some support for dynamic evolution, by way of controlled runtime substitution of services to handle fault tolerance and small variations in functionality. More generally, Bennett and Rajlich (2000) argue that at a higher level of specialisation, dealing with evolution is little known or non-uniform in engineering research and practice. This lack of support for evolution is also attributed to the ambiguity and poor understanding of system evolvability (Rowe et al. 1998). A more detailed analysis of the extent of support for dynamic evolution, using the set of dynamic evolution feature requirements developed during this research, is presented later in Sections 4.4 and 5.4.

Table 2.5 Support for composition and evolution in reviewed methodologies

Methodology	Composition Paradigm ¹		Unit of Composition	Process Support for Evolution	Evolution Mode	Evolution Target
	C	S				
ASG		✓	ASG service (with semantics)	maintenance and evolution only noted	undefined, but appearing to be dynamic	ontology, services, application
Catalysis	✓		collaboration, type [of component]	upgrade only noted in deployment phase	static - maintenance dynamic - upgrade without shutdown briefly noted but with no details	component, system
CBDI-SAE		✓	automation unit (which is equivalent to a service implementation)	solution monitoring and refinement only noted	static	reference framework, security architecture,

Methodology	Composition Paradigm ¹		Unit of Composition	Process Support for Evolution	Evolution Mode	Evolution Target
	C	S				
EPIC	✓		component: software component, hardware unit, legacy system	minor changes handled in the final phase of a project, significant changes handled with a new project - no further details provided in EPIC activities (i.e. processes)	static - * minor change to a solution or a component * component upgrade as new ones are released from the market * significant change	component
Erl (2005)		✓	business process, business service, application service, web services	ongoing rework, redesign and refinement noted in the agile process, but no further details	static	business process, business service, application service, web services
KobrA	✓		Komponent	dedicated configuration management and change management processes for development time artefacts	static - evolution considered as part of maintenance	product line, product line framework, application
OPEN Process Framework	✓		component	maintenance activity for minor fixes and enhancement	static	component, application
Oreizy et al. (1999)	✓		component	evolution management	dynamic	component, connector, parameters, topology
Papazoglou and van den Heuvel (2006)		✓	service, business process	continuous monitoring and improvement of services and applications noted in monitoring phase	undefined	service, application
RUP 2003 and v7.2.0.1	✓	✓	component, service, business process	reiterating development cycle to handle evolution, deployment discipline to support change releases - no further details	static	software product [i.e. application]
SeCSE		✓	service	binding and re-binding, re-planning, and variation points management processes	dynamic - for anticipated changes in composition	service, service composition

Methodology	Composition Paradigm ¹		Unit of Composition	Process Support for Evolution	Evolution Mode	Evolution Target
	C	S				
Select Perspective	✓		own definitions of service and component	changes to components and applications noted in Supply and Consume workflows, tracking of component changes noted in Manage workflow	static - limited support for maintenance	component, application
SUPER		✓	business process element, business process	reiterating development cycle to handle changes noted but no further details	static	business process

Note 1: "Composition Paradigm" - "C":supporting component-based, "S":supporting SOA-based

2.5 SITUATIONAL METHOD ENGINEERING

A one-size-fits-all methodology which can suit any project from a variety of domains and sizes is unlikely to be attainable (Henderson-Sellers 2006). Thus, an organisation could be supplied with a plethora of different methodologies (off-the-shelf, pre-built and the like) to choose from for each project situation at hand. However, the possibility of an organisation to have a number of methodologies available is likely to be low (Bajec et al. 2007). This is compounded by the fact that it is not always possible to find practitioners knowledgeable in all methodologies of the organisation to be able to choose a suitable methodology (Bajec et al. 2007). Alternatively, a methodology can be tailored to adapt to the changing nature and actual needs of a project (Firesmith & Henderson-Sellers 2002; Fitzgerald et al. 2003). In this spirit, method engineering (introduced earlier in Section 1.1.4) strives to design, construct and adapt methodologies for software development (Brinkkemper 1996). More specifically, *situational method engineering* seeks to develop a purpose-built methodology to suit a specific organisational and/or project situation (ter Hofstede & Verhoef 1997). A situation has two aspects: context and project type (Bucher et al. 2007). A project type characterises the transformation of a design artefact from one state (i.e. version) to its next. A context is an environmental factor that has a significant impact on the effectiveness and efficiency of applying a methodology (e.g. organisation size). Both contexts and project types can be analysed when constructing a methodology for use in a particular situation (Bucher et al. 2007).

There are a few ways to apply situational method engineering to construct a methodology: *assembly-based*, *extension-based*, *paradigm-based* and *ad hoc* (Bajec et al. 2007; Henderson-Sellers & Ralyté 2010; Ralyté et al. 2004). In the assembly-based

approach, method components are assembled into a methodology of best fit for a particular situation (Brinkkemper 1996; Ralyté & Rolland 2001). Variants of method components include method chunks and method fragments (Henderson-Sellers & Ralyté 2010). For example, Brinkkemper (1996) presents a process workflow to configure methodologies, starting from characterising a project environment, to the selection and assembly of fragments. Brinkkemper et al. (1998) use a fragment classification model to assemble fragments and define rules and guidelines to govern the method assembly. Ralyté and Rolland (2001) offer a process model to guide the retrieval and assembly of method fragments using different strategies depending on the type of situation. The assembly-based approach offers opportunities for method fragment reuse, especially when fragments are sourced from existing methodologies for methodology construction.

In the extension-based approach, typical extension opportunities are first identified from a skeleton methodology. Extensions are then performed using a set of “extension patterns” as guidelines. Ralyté et al. (2003) describe a few ways to perform extensions using extension patterns and meta-patterns for extensions. Some studies contend that this approach overlooks the situation where certain features in a methodology are not applicable and should be discarded (Karlsson & Ågerfalk 2004). Hence, feature reduction should be also performed to complement extension.

In the paradigm-based (a.k.a. “evolution-based” (Bajec et al. 2007)) approach, a new methodology is constructed by either abstracting from an existing methodology concerning a particular paradigm, or instantiating from a methodology metamodel, according to the objectives for the new methodology. Ralyté et al. (2003) suggest that when creating a methodology this way, a product model is constructed before the process model. They offer several strategies for constructing these models.

In an ad hoc approach, a methodology is constructed from scratch, i.e. without regard to existing fragments (Ralyté et al. 2004). Fragments of the newly constructed methodology can be imported into a methodbase. A rationale for this approach is that the problem domain to be addressed is not yet supported by an existing methodology. Nevertheless, the ad hoc approach can be used to augment the assembly-, extension- and paradigm-based approaches to construct a more suitable methodology (Henderson-Sellers & Ralyté 2010).

2.6 CONCLUSION

In this Chapter, the notion of dynamic evolution in distributed applications has been

discussed along the topics of relevant dynamic evolution concepts (cf. Sections 2.1), composition-based distributed applications, and how dynamic evolution could be embraced with this kind of application, such as SOA- and component-based ones (cf. Section 2.2). To assess a methodology for its extent of support for a particular domain at a more detailed level, an evaluation framework could also be utilised. A review of existing evaluation frameworks presented in this Chapter reveals that they do not specially cover dynamic evolution aspects (cf. Section 2.3). This Chapter has also reviewed a number of relevant methodologies supporting composition-based development, highlighting their lack of comprehensive support for dynamic evolution (cf. Section 2.4). Lastly, the notion of situational method engineering has been discussed as a promising platform for incorporating different features, packaged as method components, method fragments, method chunks, extension patterns etc., into a methodology and adapting it to the needs of a project (cf. Section 2.5). Acknowledging the lack of definitive feature requirements for dynamic evolution and methodological support for dynamic evolution, the next Chapter presents the design of a programme undertaken in this research to tackle these limitations.

Chapter 3. RESEARCH DESIGN

“It is a bad plan that admits of no modification.” - Publilius Syrus

Chapter 2 provided a review of dynamic evolution and development methodology as the background information to this research. It argued that even though dynamic evolution can be realised in composition-based distributed applications, the problem of dynamic evolution is not only poorly defined in the literature and evaluation frameworks but also inadequately supported by development methodologies. This Chapter describes a research programme to fill these gaps.

This Chapter begins with a brief introduction to *design science research* (March & Smith 1995), or *design research* (Vaishnavi & Kuechler 2004), being selected as the overall framework for this research. A justification for the appropriateness of design science research for this research is given in Section 3.1. The research design for this research, which is a “plan and structure of investigation so conceived as to obtain answers to research questions” (Kerlinger 1986, p. 279), is presented in Section 3.2. More specifically, the research activities and tasks in the research design to determine dynamic evolution requirements to be addressed by a methodology and to develop methodological support to satisfy these requirements are outlined.

3.1 DESIGN SCIENCE RESEARCH AND THE JUSTIFICATION OF THIS RESEARCH

Chapter 1 argued that dynamic evolution is beneficial to certain kinds of distributed applications and should be supported from the methodological perspective (cf. Section 1.1.1). However, as discussed earlier in Sections 2.3 and 2.4.13, the notion of dynamic evolution is poorly understood and supported in existing evaluation frameworks and methodologies (see also Sections 4.1.5, 4.4, 5.1.3 and 5.4). For that reason, this research aimed to *define* dynamic evolution requirements that should be addressed by a methodology, and to *build* method fragments to address these requirements from the methodological perspective, both a form of *prescriptive* inquiry (Gregor 2006).

To carry out effective information systems (IS) research of this genre, it is beneficial to follow a particular and defined research paradigm suitable for this kind of research. Generally speaking, a paradigm is a model or pattern accepted by a scientific and/or research community to nurture a “particular coherent tradition of scientific research” (Kuhn 1996, p. 10) and for one to seek particular research outcomes in an established way

(e.g. analysis and description, explanation, prediction, and prescription (Gregor 2006)). The IS research discipline, in particular, fits into a multi-paradigmatic regime (Vaishnavi & Kuechler 2004). It advocates a variety of research methods and paradigms for the investigation of the use, development and understanding of information systems. This diversity stems from the characteristics of information systems and IS research at the “confluence of people, organizations and technology” (Hevner et al. 2004). This diversity also makes IS research creative and broadens the foundations from which its knowledge is claimed (Robey 1996).

Since the outcomes of this research are a proposed set of important dynamic evolution requirements and associated method fragments to fulfil them, this research called for an inquiry of prescribing and innovating IS artefacts (e.g. a new methodology). Such an endeavour fits into the word “design” in its noun form which is a product as sensed by the world (Hevner et al. 2004). Among a variety of IS research paradigms⁷, this research adopts design science research as the overall research framework because design science research focuses on the problem solving aspects by *creating and prescribing artificial innovations* for the effective development, management and use of information systems (Hevner et al. 2004). Design science research is practised under different guises, including *constructive research* (e.g. Iivari et al. 1998), *systems development research* (e.g. Burstein & Gregor 1999), and *software engineering research* (e.g. Morrison & George 1995). Indeed, its importance has been recognised in IS research (Gregor 2002) because of its prescriptive nature in creating advances in vast domains in which problems are inadequately solved (Hevner et al. 2004). Example advances produced from design science research include constructs, models, methods, instantiations (March & Smith 1995), better theories (Vaishnavi & Kuechler 2004), and IS development methodologies (Vaishnavi & Kuechler 2004; Venable & Travis 1999).

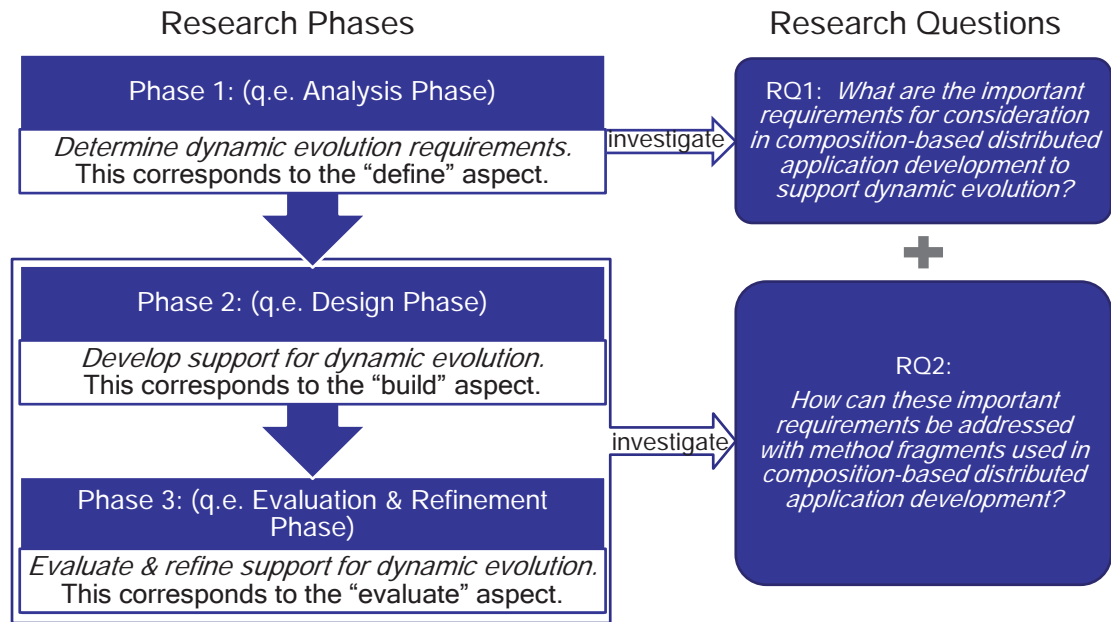
Given that design science research has been argued as an appropriate and hence selected approach for this research, the details of a design science research based programme tailored for this research are presented next.

⁷ Examples of different IS research paradigms (Niehaves & Stahl 2006) include design science research (Hevner et al. 2004), behavioural science research (Hevner et al. 2004; March & Smith 1995), positivism (Chen & Hirschheim 2004; Myers 2004), interpretivism (Chen & Hirschheim 2004; Orlikowski & Baroudi 1991), critical research (Ngwenyama 1991) and non-critical research (Orlikowski & Baroudi 1991).

3.2 RESEARCH ACTIVITIES AND PHASES

A basic design science research effort involves two activities (March & Smith 1995). “Build”, the first activity, concerns the construction of artefacts for a specific purpose - the methodological support for dynamic evolution in this research. “Evaluate”, the second activity, is the process of assessing how well these artefacts address a problem for which they were built. Although a variety of evaluation approaches exist specifically for demonstrating the validity of methodologies (e.g. Zelkowitz & Wallace 1998), the build part of design science is not well understood (March & Smith 1995). Vaishnavi et al. (2004) thus refined the build activity into three phases: identification of a new research effort (“awareness of a problem”); prototyping a tentative solution (“suggestion”); and implementation of the tentative solution into a full solution (“development”). Their approach though suffers a drawback in that it does not identify a class of goals - a.k.a. *meta-requirements* (Walls et al. 1992) - that a designed artefact is aimed at addressing for a particular problem.

Acknowledging the importance of meta-requirements, Peffers et al. (2008) established an activity called “define objectives of a solution” to be performed ahead of the build activity which they referred to as “design and development”. “Define objectives of a solution” determines the meta-requirements for the artefact to build whilst “design and development” constructs the artefact to satisfy the meta-requirements. Thus, a comprehensive methodology development endeavour should at least cover the *define* (feature requirements) aspect, in addition to the *build* (features) and *evaluate* (features) aspects of a basic design science research programme originally suggested by March et al. (1995). The lack of the “define” aspect is also in agreement with the observation that there have been no comprehensive studies identifying what kinds of dynamic evolution capabilities should be addressed in software development methodologies (cf. Section 2.3). Correspondingly, this research programme was organised into three phases to take into account the define, build and evaluate aspects, as depicted in Figure 3.1 below. The research question that each phase be intended to address is also indicated in Figure 3.1 (cf. Section 1.2).



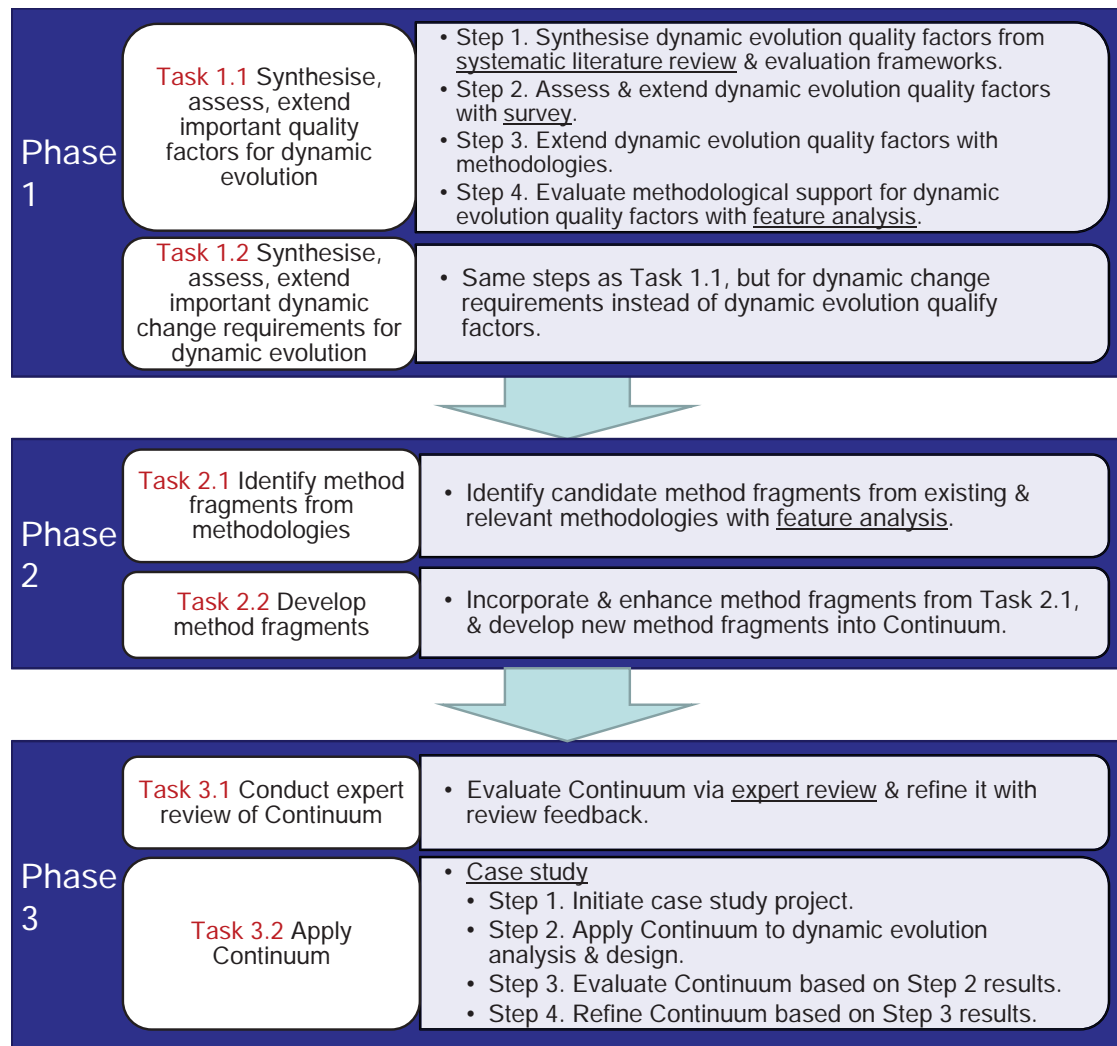
source: developed for this research

Figure 3.1 Development phases adopted for this research (expanded from Figure 1.1)

The three phases in this research comprised a number of tasks, each associated with one or more steps and research techniques applied to accomplish the objectives of the task. Figure 3.2 gives a bird's-eye view of these activities, with key research techniques applied in the steps shown in underlined text. Due to visual cluttering, the task and step descriptions are shortened in Figure 3.2. The three phases in terms of the tasks and steps performed, together with the research techniques applied, are presented next.

3.2.1 Phase 1: Determine important dynamic evolution requirements

Phase 1 determined important dynamic evolution requirements that should be addressed by a methodology, and are perceived as important by methodology users. These requirements were then used for the development of Continuum in Phase 2. Tasks in this phase were carried out in response to research question RQ1 stated in Section 1.2, “What are the important requirements for consideration in composition-based distributed application development to support dynamic evolution?” The *determined* requirements also give some indication of the scope of work during later phases.



source: developed for this research

Figure 3.2 Research tasks, steps and techniques used in various phases

To manage the complexity of dynamic evolution, this investigation distinguished between two types of requirements: *dynamic evolution quality factor* and *dynamic change requirements*. Dynamic evolution quality factor requirements (or “dynamic evolution quality factors” for short) are concerned with how well a distributed application and dynamic changes to it are designed to facilitate dynamic evolution. Dynamic change requirements account for the characteristics of dynamic changes to a distributed application including various kinds of changes that a distributed application would accommodate; who enacts such changes; and the notion of errors arisen from such changes. Accordingly, two tasks were performed to develop each requirement type:

- Synthesise, assess and extend important dynamic evolution quality factors (hereafter referred to as Task 1.1).

- Synthesise, assess and extend important dynamic change requirements (hereafter referred to as Task 1.2).

Since the structure and the approach of two tasks above are similar, a generic four-step process as described below was defined to suit both of these tasks. In the step descriptions below, “dynamic evolution requirements” refer to either dynamic evolution quality factors or dynamic change requirements. Differences in the treatment of the two requirement types are noted in the step descriptions where applicable. The full details and outcomes of these steps are documented in Chapter 4 (for dynamic evolution quality factors) and Chapter 5 (for dynamic change requirements).

Step 1: Synthesise (i.e. identify and categorise) an initial set of dynamic evolution requirements from both the literature and evaluation frameworks.

Potential dynamic evolution requirements were *identified* from a systematic literature review and from evaluation frameworks. Identified requirements were then *categorised* using one of the schemes described below, and ready for use in Step 2:

- *For dynamic evolution quality factors*

Each identified requirement is referred to as a quality attribute. Quality attributes characterising the same quality aspect (e.g. security) were *grouped* to form the definitions for a quality factor. Quality factors were then *categorised* according to their similarities and contexts. The category-factor-attribute hierarchy corresponds to the quality levels (i.e. *characteristic*, *sub-characteristic* and *attribute*) in the quality model of the International Standard ISO/IEC 9126-1 (ISO/IEC 2001).

- *For dynamic change requirements*

Each identified requirement is referred to as a dynamic change requirement. Dynamic change requirements were *categorised* along two dimensions: methodology and application. The methodology dimension determines whether a dynamic change requirement is a modelling related (concept, notation and model) or work related concern (what must be done and how to do it) that would be addressed by a methodology to support dynamic evolution. The application dimension classifies dynamic change requirements in line with their areas of concern: for individual parts, for an application as a whole, and for all other situations.

Step 2: Assess and extend the set of dynamic evolution requirements synthesised from the literature and evaluation frameworks using a survey of experienced practitioners and researchers.

A survey was conducted to assess the completeness and perceived importance of the dynamic evolution requirements synthesised in the last step. The survey fulfilled these objectives:

- *Assessment of the importance of dynamic evolution requirements*

Each respondent was asked to rate the level of importance of each dynamic evolution requirement. The rating scores of the dynamic evolution requirements were used to prioritise subsequent work in this research: whether dynamic evolution requirements were desirable (i.e. more important) and should be considered for Continuum, or they were less important and should be discarded.

- *Identification of additional dynamic evolution requirements*

Respondents were asked to suggest potential dynamic evolution requirements for consideration. Two experts experienced in dynamic evolution then reviewed these dynamic evolution requirements via a Delphi-type method (Okoli & Pawlowski 2004). Suggested dynamic evolution requirements that were also recommended by both experts were incorporated into the set of important dynamic evolution requirements.

The survey set-up for dynamic evolution quality factors was slightly different from that of dynamic change requirements. In the former, a web site was prepared for respondents to complete a survey form online whereas in the latter, respondents were emailed a survey form to complete and return.

Step 3: Extend the dynamic evolution requirements from relevant methodologies.

Dynamic evolution requirements were identified from relevant methodologies independently from the survey (Step 2). Each requirement identified was then checked to see if it mapped to one or more requirements from the extended set of dynamic evolution requirements from Step 2. If not, it represented a new requirement and was therefore incorporated into the extended set.

Step 4: Evaluate relevant methodologies for their extent of support for dynamic evolution requirements.

To evaluate the dynamic evolution requirements synthesised in this task, the methodologies reviewed in Step 3 were evaluated for their level of support for each requirement using the feature analysis technique (Kitchenham & Jones 1997). The evaluation applied four scale points - *High* for full support, *Medium* for partial support requiring small enhancement, *Low* for inadequate support requiring significant enhancement, and *None* for no support - to determine what features from existing methodologies could be reused, what features could be enhanced for use and any additional support required. Features identified for potential reuse or requiring small enhancement were then considered in the development of Continuum in Phase 2.

3.2.2 Phase 2: Develop support for dynamic evolution

Given the definitions of the requirements from the last phase, the development of an initial version of Continuum occurred in this phase, with respect to research question RQ2 stated in Section 1.2, “How can these important requirements be addressed with method fragments used in composition-based distributed application development?” The development of Continuum opted for a method engineering approach (Section 1.1.4) that produces method fragments to address a problem domain (which is dynamic evolution in this case). Through assembly-based situational method engineering (Section 2.5), they can be adapted to a particular situation to provide support for dynamic evolution. The developed Continuum comprises the following method fragments⁸ to satisfy these requirements:

- *Metamodel* (i.e. a model of models (Gonzalez-Perez & Henderson-Sellers 2007)), which prescribes key concepts and their relationships for dynamic evolution;
- *Work product fragments* to be created, updated and used;
- *Work unit fragments* to be performed (ISO/IEC 2007), which are:
 - process fragments, each representing a large-grained work unit within a given area of expertise;
 - task fragments specifying what is achieved by those process fragments;

⁸ Strictly speaking, according to 24744, “method fragments” (including “work product fragments”, “work unit fragments”, “process fragments”, “task fragments”, “technique fragments” and “producer fragments”) should be termed “method fragment kinds” (henceforth “work product fragment kinds”, “work unit fragment kinds”, “process fragment kinds”, “task fragment kinds”, “technique fragment kinds” and “producer fragment kinds” respectively). For convenience and ease of reading, the “kind” suffix is dropped.

- technique fragments specifying how to achieve the given purposes within the task fragments; and
- *Producer fragments*, specifying who does which of the above work unit fragments.

Continuum's method fragments were reused from existing methodologies, enhanced from existing methodologies, or developed from scratch where appropriate. The development of Continuum involved the execution of the following two tasks: Identify method fragments from relevant methodologies (Section 3.2.2.1) and Develop method fragments (Section 3.2.2.2).

3.2.2.1 Task 2.1: Identify method fragments from relevant methodologies

Task 2.1 aimed to identify candidate method fragments from existing and relevant methodologies suitable for addressing particular dynamic evolution requirements defined in Phase 2 (Section 3.2.2), and for reuse and enhancement consideration in the next task. This was because some methodologies may already have features that fully satisfy certain dynamic evolution requirements and/or other features that partially address some other dynamic evolution requirements. This avoided redeveloping features and reduced the effort in the next task. The identification involved analysing the results from the methodology evaluation conducted in Phase 1 (i.e. Step 4 of both Tasks 1.1 and 1.2) and locating features meeting the following criteria:

- A feature is rated as *High*, indicating that it fulfils a particular dynamic evolution requirement and is suitable for *reuse* in Continuum; or
- A feature is rated as *Medium*, indicating that it appears to support a particular dynamic evolution requirement to some extent and needs a *small enhancement* to be incorporated into Continuum to fulfil the requirement.

Next, for each requirement the highest rated feature from among those rated as *High* or *Medium* was selected for consideration in the next task. In situations where more than one feature from different methodologies satisfies the same requirement to a similar extent, an analysis was performed to choose the most appropriate feature. Each decision for choosing a particular feature was justified and recorded. Selected features were then extracted from their respective methodologies as method fragments for the next task. The full details of this task are documented in Section 6.2.

3.2.2.2 Task 2.2: Develop method fragments

Following the identification of suitable method fragments from existing methodologies in

Task 2.1, this task carried out the development of Continuum in the following areas:

- a. Incorporation of *selected* existing method fragments from the evaluated methodologies into Continuum as per Task 2.1;
- b. Enhancement of *selected* existing method fragments from the evaluated methodologies into Continuum as per Task 2.1; and
- c. Development of new method fragments for requirements that are not accounted for by method fragments from (a) and (b) above. Each requirement was addressed by firstly reviewing the literature not concerning methodologies for potential approaches that might fulfil the requirement. The review ensured approaches from outside the methodology domain were also investigated since methodologies were only considered in Task 2.1. If no suitable approach was identified, a new fragment was then developed to satisfy the requirement.

After the development of individual method fragments, they were structured and linked to form a coherent set of method fragments. Continuum's method fragments were documented in accordance with the International Standard ISO/IEC 24744 (ISO/IEC 2007). The full details and outcomes of this task can be found in Section 6.3.

3.2.3 Phase 3: Evaluate and refine support for dynamic evolution

Once the initial version of Continuum was developed, this phase turned to the incremental evaluation and refinement of Continuum, as a further investigation with respect to research question RQ2 stated in Section 1.2, "How can these important requirements be addressed with method fragments used in composition-based distributed application development?" To increase the credibility of Continuum, two evaluation and refinement tasks (i.e. Tasks 3.1 and 3.2) were performed in a sequential order to produce the final version of Continuum.

3.2.3.1 Task 3.1: Conduct an expert review of Continuum

An expert review of Continuum was conducted as an initial means of evaluating and refining Continuum. More specifically, two subject matter experts, both experienced in the field of dynamic evolution, were given an initial version of Continuum developed from the last phase, and asked to critique its strengths, weaknesses and completeness. For weaknesses and completeness, experts were asked to provide:

- any suggestion for improvement of existing features; and
- any suggestion for new features to complement the existing ones.

The review was orchestrated to ensure all changes to Continuum satisfied both experts. To begin with, one expert performed the review after which changes were made to Continuum in response to the review results and discussed with this expert. The second expert then performed the review of the updated version of Continuum. Additional changes as per the second expert's suggestions were then checked with him/her, and subsequently with the first expert before being incorporated into Continuum. The full details and outcomes of this task are documented in Section 7.1.

3.2.3.2 Task 3.2: Apply Continuum to a case study

In this task, a case study (Yin 2003) was applied to evaluate Continuum and the evaluation results from which were used to further improve it. A case study is appropriate for Continuum since case studies are suitable for *new* methodologies in their early stages of development when the theoretical basis is being established (Murphy et al. 1999). The case study involved two participants from a sponsoring organisation jointly developing dynamic evolution capability for a real-world application (i.e. industrial case study) using Continuum. It was divided into the following steps (full details and outcomes in Section 7.2):

Step 1: Initiate a case study project.

A kick-off meeting was held with the participants to disseminate the purpose, expectations and potential benefits of the case study. A project plan was also developed in the meeting. This was followed by induction training for the participants in both Continuum and method engineering.

Step 2: Apply Continuum to the analysis and design for dynamic evolution.

The participants assembled a full methodology lifecycle from Continuum's method fragments. Then, they applied the lifecycle to analyse and design dynamic evolution for the application from one version to the next at runtime. A set of dynamic evolution artefacts was developed using Continuum's work products as templates.

Step 3: Evaluate Continuum based on outcomes and experience from Step 1.

After completing the analysis and design for dynamic evolution, the participants evaluated the *usefulness* and *usability* (Murphy et al. 1999) of Continuum, in addition to evaluating its weaknesses and completeness as described in the expert review method in Task 3.1, with respect to how well it addressed the problem situation in the case study. For usefulness, the participants were asked

to rate how well Continuum addressed the relevant issues encountered during the case study, and compare it with how well the methodology adopted in their organisation would address the same issues. For usability, the participants rated Continuum's method fragments in terms of *understandability* (e.g. "Is 'X' easy to understand?") and *ease of use* (e.g. "Is 'X' easy to use?").

Step 4: Refine Continuum based on evaluation results and feedback.

Finally, the evaluation results from the last step were collected from the participants and confirmed with them in a follow-up meeting. Changes - such as new features and extensions to existing features - were then made to Continuum in consultation with the suggestions from the participants, which can be found from the evaluation results. Afterwards, the participants reviewed the changes and confirmed whether they improved Continuum. The review was then repeated with the experts who took part in the expert review in Task 3.1.

3.3 CONCLUSION

This Chapter briefly described design science research and provided arguments for adopting it as the overall programme for this research. It also discussed the detailed design for the research programme, being composed of research tasks and their associated steps undertaken in three phases:

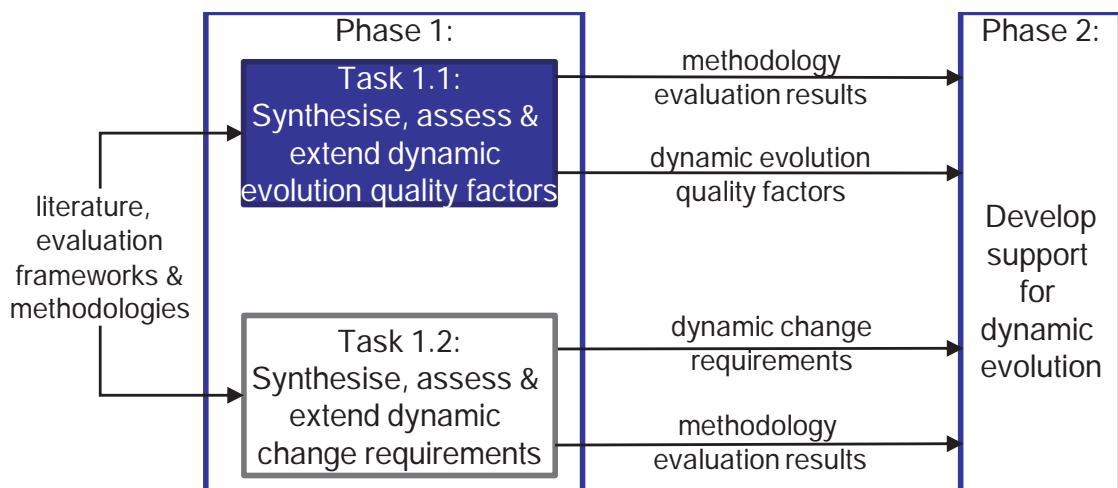
- Phase 1 "Determine important dynamic evolution requirements" (Section 3.2.1)
- Phase 2 "Develop support for dynamic evolution" (Section 3.2.2)
- Phase 3 "Evaluate and refine support for dynamic evolution" (Section 3.2.3)

The next four Chapters document the execution of tasks in and outcomes from these phases.

Chapter 4. DEVELOPMENT OF DYNAMIC EVOLUTION QUALITY FACTORS

“Quality is everyone's responsibility.” - W. Edwards Deming

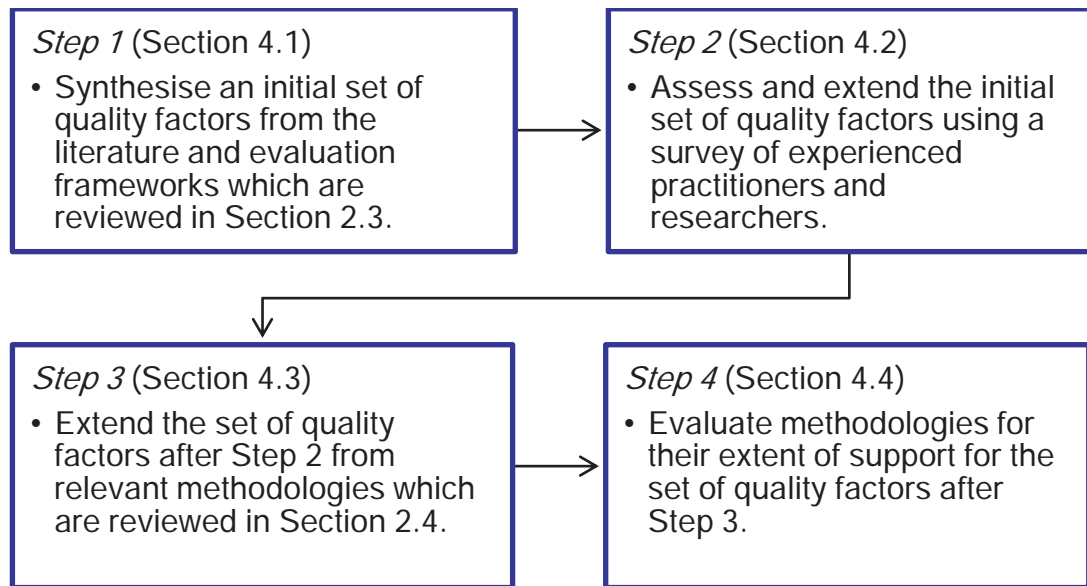
Following the description of the research design in Chapter 3, this Chapter reports the execution and outcomes from Task 1.1, which aimed to “Synthesise, assess and extend important dynamic evolution quality factors”. The outcomes of Task 1.1 comprise a generic set of important dynamic evolution quality factors suitable for composition-based distributed applications and the results of evaluating methodologies for their extent of support for these factors. These outcomes form part of the deliverables to the Phase 2 during which the content of Continuum was developed to address these quality factors. The relationships between Task 1.1, and other tasks and phases, are summarised in Figure 4.1.



source: developed for this research

Figure 4.1 Information flow in Phase 1 for determining dynamic evolution quality factors

As discussed in Section 3.2.1, Task 1.1 has four steps as illustrated in Figure 4.2. The first three steps concern the incremental development and assessment of the dynamic evolution quality factors. In Step 4, methodologies were evaluated for their support of the factors. Respectively, Sections 4.1 to 4.4 describe the execution and outcomes of these four steps (see Figure 4.2). Afterwards, the developed set of quality factors are compared with related work in Section 4.5. Section 4.6 concludes this Chapter.



source: developed for this research

Figure 4.2 Steps in Task 1.1 of Phase 1

4.1 STEP 1: SYNTHESIS OF DYNAMIC EVOLUTION QUALITY FACTORS

This step synthesised an initial set of dynamic evolution quality factor requirements (or “quality factors” for short) from the literature and evaluation frameworks. A systematic literature review (Kitchenham et al. 2009; Kitchenham 2004) was undertaken to guide the synthesis. The objective of the review was to answer the following research question:

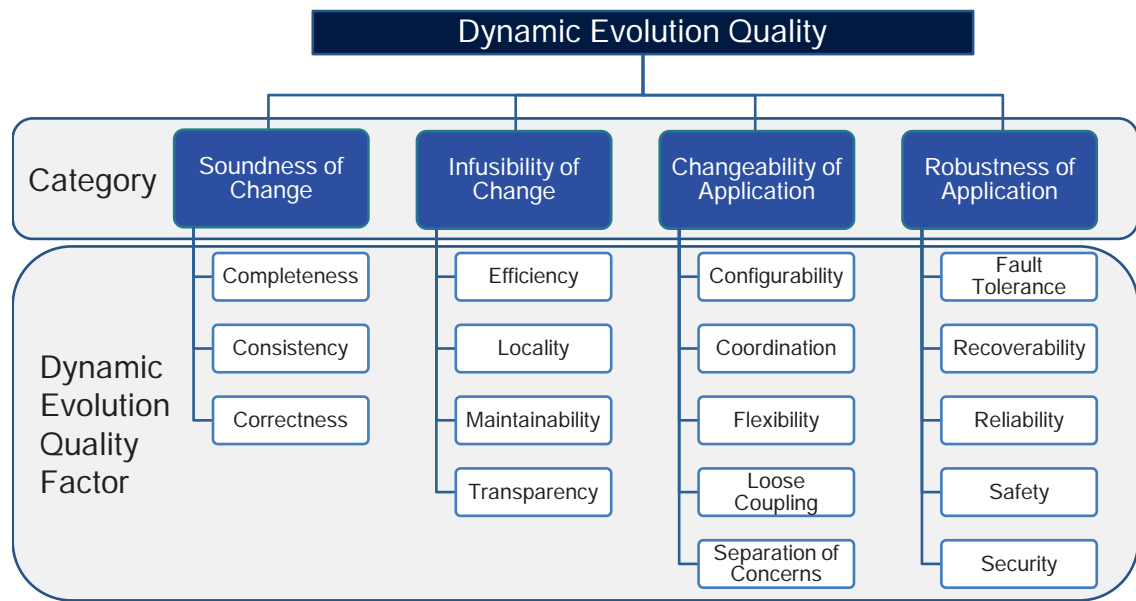
What dynamic evolution quality factor requirements are considered in the literature and should be addressed during software development for composition-based distributed applications?

The corresponding review steps are documented in Appendix A. Table 4.1 shows the results of the search defined in the review steps. The search covered two parts. The first part covered a number of journals and conference proceedings published between 1994 and 2010 and identified two hundred and seventeen articles for quality factor synthesis, and quality attributes for the synthesis were found from sixty-five articles. The “others” category, at the bottom of Table 4.1, refers to articles examined from the bibliographies of the articles that were from the first part, for additional quality factors. This was to account for important and cited articles not covered by the search scope for the first part.

Table 4.1 Source of literature examined for quality factor synthesis

<i>Source</i>	<i>Title</i>	<i>Abbreviation</i>	<i>Articles examined</i>	<i>Articles with quality attributes</i>
Journals (1994-2010)	Communications of the ACM	CACM	8	2
	ACM Transactions on Computer Systems	TOCS	1	1
	ACM Transactions on Software Engineering and Methodology	TOSEM	3	1
	European Journal of Information Systems	EJIS	0	0
	IEEE Computer		11	5
	IEEE Software		6	0
	IEEE Transactions on Software Engineering	TSE	10	3
	IET Software (formerly "IEE Proceedings Software" and "Software Engineering Journal")	IETS	12	4
	Information and Software Technology	IST	9	2
	Information Systems Journal	ISJ	0	0
	Information Systems Research	ISR	0	0
	Journal of Information Technology	JIT	0	0
	Journal of Software Maintenance and Evolution: Research and Practice (formerly "Journal of Software Maintenance")	JSME	8	6
	Journal of Systems and Software	JSS	7	1
	MIS Quarterly	MISQ	0	0
	Requirements Engineering		2	0
Conference Proceedings (1994-2010)	International Symposium on Component-Based Software Engineering (formerly "ICSE Workshop on Component-Based Software Engineering")	CBSE	14	3
	International Conference on Configurable Distributed Systems	IWCDS, ICCDS	26	13
	International Conference on Distributed Computing Systems	ICDCS	7	2
	International Conference on Service-Oriented Computing	ICSOC	22	0
	International Conference on Software Engineering	ICSE	16	3
	International Conference on Software Maintenance	ICSM	13	1
Others	Articles not in any of the publications and years above		42	18
Total			217	65

Figure 4.3 summarises the dynamic evolution quality factors synthesised from the literature and their categories. The textual definitions of the quality factors and their categories (i.e. Soundness of Change, Infusibility of Change, Changeability of Application, and Robustness of Application) are described next (Sections 4.1.1 to 4.1.4). A review of existing evaluation frameworks (Section 2.3) was used to extend the quality factors (Section 4.1.5). Section 4.1.6 summarises the definitions of the factors and includes a discussion of quality factors considered but not included in the set.



source: developed for this research

Figure 4.3 Dynamic evolution quality factors and categories synthesised from the literature

4.1.1 Soundness of Change

Soundness refers to the extent of that changes to an application and associated transformations are free from defects or flaws, and do not make an application harder to evolve. For instance, it is undesirable to apply a change to an application if the change makes the application more difficult to maintain. Soundness is characterised by Completeness, Consistency and Correctness.

4.1.1.1 Completeness

Completeness is a condition that a transformation and its change(s) should not cause missing, broken or illegal parts, bindings or functions in an updated application. Incompleteness could be due to unfinished transformations (e.g. additions, removals and/or updates of parts), or design flaws in transformations and changes. Incompleteness may cause the application to fail. Completeness requires that parts that are defined in an application and its interaction specifications are all present after a transformation (Agnew et al. 1994b; Allen & Garlan 1997). Furthermore, all functions required by a part are provided by one or more parts of the updated application (Allen & Garlan 1997; Medvidovic et al. 1999). After a transformation there should be no missing, illegal, or broken bindings among parts in the application (Feiler & Li 1998; Hillman & Warren 2004) which may affect communication among parts (Aksit & Choukair 2003). The binding issue is less of a concern in service-oriented computing

because service bindings tend to be dynamic: established before a service invocation and released afterwards (Curbera et al. 2003). Completeness also mandates that dynamic changes to an application will meet the assumptions and properties of the application and its parts (Allen & Garlan 1997; Feiler & Li 1998; Mens & D'Hondt 2000).

4.1.1.2 Consistency

Consistency implies that a transformation and its change(s) should not result in an error state for an application (Lee & Chang 2005; Zimmermann & Drobniak 1994) such as a deadlock (Gregersen & Jørgensen 2009; Sun & Jiang 2009) or deviating it from a particular architectural style (Loulou et al. 2010). Rather, the application continues processing (Kramer & Magee 1990) from a reachable state, which can be progressively transitioned from its start-up state(s), after the transformation has occurred (Gupta et al. 1996). Maintaining consistency requires that all parts involved in a dynamic change are identified before starting a transformation (Warren & Sommerville 1996). Moreover, parts and bindings, both new and replacement, should be allocated adequate resources (e.g. CPU time) and support to operate correctly after the transformation (Ben-Shaul et al. 2001; Feiler & Li 1998). It is also desirable that no critical functions are being executed by parts that will be affected by the transformation (Warren & Sommerville 1996). There should also be no unprocessed messages, unfinished interactions or transactions being performed by an application at the time of transformation (Adamek & Plasil 2005; Bidan et al. 1998; Chen 2002; Kramer & Magee 1990; Li 2009; Wang et al. 2006; Warren & Sommerville 1996). Maintaining consistency also means that, similar to completeness, any dynamic changes to an application should meet the assumptions and properties of the application and its parts (Allen & Garlan 1997; Feiler & Li 1998; Mens & D'Hondt 2000).

To preserve consistency after a transformation, the types and directions of connected component ports for communication (OMG 2010b) should match (Feiler & Li 1998). In the service-oriented paradigm, this means the type of service consumer is compatible with the type of service provider (Meredith & Bjorg 2003). For instance, it does not make sense to link two output ports in terms of data flow.

The communications protocol used by parts should also be compatible (Allen & Garlan 1997). Newly added parts should be *state-synchronised* with their hosting application for it to continue to execute as expected and do so in a timely fashion (Aksit & Choukair 2003; Oreizy et al. 1998; Warren & Sommerville 1996; Zieba & van Sinderen 2006). For example, when adding a database connection pool to an application, it should be

preloaded with database connection objects so that it is ready for use by the rest of the application after the transformation. In service-oriented computing (Huhns & Singh 2005), an interaction among services, such as a business process, can be long-running (Dustdar & Schreiner 2005). One way to track the state for a long-running interaction, is to embed the state information into the fields of messages exchanged among the services (Curbera et al. 2003). At a broader level, system invariants (e.g. no read access allowed on a deleted file) should be unaffected by a transformation (Kramer & Magee 1998; Léger et al. 2010; Oueichek & Rousset de Pina 1996).

4.1.1.3 Correctness

Correctness is the degree to which a transformation correctly and effectively applies the associated and feasible dynamic change(s) to an application. A dynamic change is feasible (Bennett & Rajlich 2000) when it is non-arbitrary and can be implemented (Etzkorn 1992; Gupta et al. 1996; Wermelinger 1998). During and after a transformation, the application should not behave incorrectly or unintentionally (Aksit & Choukair 2003; Buckley et al. 2005; Coyle et al. 2010; Gregersen & Jørgensen 2009; Mens & D'Hondt 2000; Segal & Frieder 1993; Zhang & Cheng 2006; Zieba & van Sinderen 2006).

Correctness is also affected by temporality, referring to the time-related aspects of models and expressions, such as temporal ordering of messages sent and received by objects in object-oriented applications (Arapis 1995). In dynamic evolution, temporality characterises the degree to which changes and transformations satisfy two conditions: appropriate time and correct order. A transformation should be executed at an appropriate time (Gupta et al. 1996; Segal & Frieder 1993) to minimise downtime during critical business hours in case of transformation failure. A transformation should also perform its steps (such as inserting, updating and removing parts and bindings) in a correct order (Lim 1996; Segal 2002; Segal & Frieder 1993). Consider F and B, two new components that are to be inserted into an online application. B provides business services over the Internet while F regulates access to B. If B is inserted into the application first, there is a small window during which B is potentially exposed to Internet worms, for instance, without the protection of F. A better way is to insert F into the application first, and B afterwards.

4.1.2 Infusibility of Change

Infusibility refers to the ease with which a change can be accommodated (i.e. infused) into an application at runtime. High Infusibility reduces the complexity and the effort of

accommodating the change into an application. Efficiency, Locality, Maintainability and Transparency characterise Infusibility.

4.1.2.1 Efficiency

Efficiency refers to the ease with which changes can be incorporated into a distributed application in a timely and resource efficient manner while minimising disruptions to services provided by the application. This means, a transformation can be executed in a distributed application easily (Bennett & Rajlich 2000; Hicks & Nettles 2005; Wang et al. 2002) and quickly (Agnew et al. 1994b; Bennett & Rajlich 2000; Gupta et al. 1996). The transformation should also make efficient use of resources (e.g. available network bandwidth) that it needs to execute (Agnew et al. 1994b). At the same time, the transformation should cause minimal disruptions to services provided by the application and to the parts using these services (Bloom & Day 1993; Chen et al. 2007; Chen 2002; Goudarzi & Kramer 1996; Hauptmann & Wasel 1996; Janssens et al. 2005; Karamanolis & Magee 1996; Kramer & Magee 1990; Milazzo et al. 2005; Pfleeger & Böhner 1990; Taentzer et al. 2000; Zimmermann & Drobniak 1994), and minimal degradation to the performance of the application (Bidan et al. 1998; Böhner 1996; Chen 2002; Gregersen & Jørgensen 2009; Hauptmann & Wasel 1996; Hicks & Nettles 2005; Oueichek & Rousset de Pina 1996; Segal & Frieder 1993).

4.1.2.2 Locality

Locality concerns the extent to which a change is explicitly confined within a logical boundary of an application to reduce the effort of managing and implementing the change and its associated transformation(s). One way to attain Locality is by partitioning an application into regions and confining a dynamic change to a region (Sun & Jiang 2009) (a.k.a. localisation (Medvidovic & Taylor 1997)). This means the effect of alternation is kept in one region without affecting parts in other regions of the distributed application (Evans & Dickman 1999; Fayad & Cline 1996; Gregersen & Jørgensen 2009; Oreizy & Taylor 1998; Plášil et al. 1998). There are other benefits with localisation. It makes the change more manageable in software development (Evans & Dickman 1999), and easier to understand (Oreizy et al. 1998) and apply to a distributed application (Bennett & Rajlich 2000).

4.1.2.3 Maintainability

In software and system engineering, maintainability is defined as “the ease with which a software system or component can be modified to change or add capabilities, correct faults or

defects, improve performance or other attributes, or adapt to a changed environment” (ISO/IEC 2008). As such, a change should not make an application more difficult and costly to modify (Bohner 1996; Hicks & Nettles 2005), or harder to test (Pfleeger & Bohner 1990). Proper and adequate specification of interactions among parts aids maintainability which should not be compromised by changes. For example, clear definitions of parts, or roles that parts play, in an interaction (or a workflow) help in understanding the architectural composition of an application and the data flow among its parts (Allen & Garlan 1997). Likewise, a clear and detailed interaction (e.g. in diagrams or formal specifications) makes it easier to apply changes to the interaction. For example, consider a common practice of drawing components and their connections as boxes and lines joining the boxes (Allen & Garlan 1997). These lines merely show which components are connected but say little about how they interact and communicate; they could well be using an event-based messaging mechanism, remote procedure calls, or something else. Without this detail, it is difficult to apply changes to these interactions and accordingly to their application (Shaw et al. 1996).

4.1.2.4 Transparency

Transparency concerns the extent to which transformations, when being executed, are unnoticeable to entities internal and external to a distributed application. Internal entities include parts and bindings of the application unaffected by a transformation (Etzkorn 1992; Swaminathan & Goldman 1996). Transparency ensures that the “liveness” property (Shatz 1993) of the application is maintained or the “availability” of its services is continuous (Cao et al. 2005). External entities include end users who should not notice any transformation is taking place (Agnew et al. 1994b; Segal & Frieder 1993) and the application's operating environment which should be unaware of the presence of transformation agents (McKinley et al. 2004; Segal & Frieder 1993). Furthermore, transformation design and implementation should be transparent to the application programmers (Chen 2002; Goudarzi & Kramer 1996; Gregersen & Jørgensen 2009; Segal & Frieder 1993) so that they can focus on the structure and functionality of the application without being concerned about dynamic evolution.

4.1.3 Changeability of Application

Changeability refers to the ease with which a distributed application can accommodate dynamic changes (Cook et al. 2001). Changeability is embodied in the design of the application to support dynamic evolution. Factors contributing to Changeability include Configurability, Coordination, Flexibility and Separation of Concerns.

4.1.3.1 Configurability

Configurability concerns the degree to which a distributed application is configured to suit different change policies when implementing changes with transformations (Gregersen & Jørgensen 2009). A change policy governs how dynamic changes are accommodated into an application (Oreizy et al. 1998; Oreizy & Taylor 1998). A simple policy for upgrading a component handling financial transactions would be to permit all existing transactions to be terminated before the upgrade. A more graceful policy would let this component continue processing existing transactions while the component that is replacing it handles new transactions. Upon processing all its existing transactions, the old component is removed. A policy in effect constrains the design for dynamic evolution. However, the decision on using particular change policies is largely determined by business requirements (Oreizy et al. 1998).

4.1.3.2 Coordination

Coordination concerns the ability of different nodes of a distributed application to interact to deploy and install dynamic changes to remote nodes or logical partitions to achieve the overall change effect (Chen et al. 2001; McKinley et al. 2004; Segal 2002; Segal & Frieder 1993). This involves, for example, an orderly execution of transformation commands: transmitting change units to appropriate regions; activating, synchronising and deactivating changes; and recovery from transformation failure at a remote node. In addition, the distributed application needs to be able to handle unreliability in the communications network during a transformation (Segal & Frieder 1993) since a transformation command may not reach a target node for execution because of intermittent loss of communication.

4.1.3.3 Flexibility

From the International Standard ISO/IEC 24765, Flexibility refers to “the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed” (cf. ISO/IEC 2008). Applied to dynamic evolution, Flexibility concerns the ability of a distributed application to easily accommodate a variety of dynamic changes (Fayad & Cline 1996; Gregersen & Jørgensen 2009; Pahl 2004; Segal & Frieder 1993), including addition, removal, update, merging and splitting (elaborated in Section 5.1.2). Furthermore, any part of a flexible application should be changeable without requiring shutdown and restart (Hicks & Nettles 2005).

4.1.3.4 Loose Coupling

Loose coupling means parts composing an application have their own lifecycles and runtime environments (Wang et al. 2006). Such an application facilitates dynamic evolution; updating one part in such a structure has less effect on other parts (Wang et al. 2006). To support dynamic evolution further, parts should be kept independent of each other, such as parts not having knowledge of or containing static references to each other as they evolve (Redmond & Cahill 2006).

4.1.3.5 Separation of Concerns

An intent of Separation of Concerns is to break a complex problem into distinct concerns or aspects with as few overlapping features as possible, to let one address particular aspects in a comprehensible manner (Dijkstra 1976). Applied to composition-based applications, Separation of Concerns refers to the extent to which functionality is separated from dynamic change, communication and security concerns to reduce the complexity in managing dynamic evolution. For instance, the separation of functional behaviour from structural reconfiguration (e.g. via transformations) makes it easier to alter one without affecting the other (Andrade et al. 2002; Bidan et al. 1998; Etzkorn 1992; Kramer & Magee 1990; Milazzo et al. 2005; Oreizy et al. 1998; Oreizy & Taylor 1998; Zhang et al. 2005). This also enables designers to concentrate on designing an application's business logic without worrying about dynamic evolution issues during application development. Likewise, the separation of functionality and communication concerns allows each to evolve independently (Andrade et al. 2002; Goldman et al. 1995; Oreizy et al. 1999; Wang et al. 1999). This way, the communications protocol used by parts can be replaced without affecting their functionality. Moreover, security concerns should be separated from normal business logic. This allows customisation of the security rules governing the use and behaviour of new and replacement parts independent of their functionality (Grimm & Bershad 2001).

4.1.4 Robustness of Application

Robustness (a.k.a. Defensibility (OPFRO 2009)) is concerned with the degree to which an application can withstand or reject invalid dynamic changes (cf. ISO/IEC 2008). A robust application is less likely to stop working or crash in these circumstances. Fault Tolerance, Recoverability, Reliability, Safety and Security contribute to Robustness of an application.

4.1.4.1 Fault Tolerance

Fault tolerance refers to the extent to which an application can tolerate faulty new and/or replacement parts (Feiler & Li 1998; Hicks & Nettles 2005) since they can inject errors into the application. Other times, the reliability of parts is not known before use and the application must ignore their errors as they are added to the application (Voas 1998). An application may establish barriers to contain faulty new and replacement parts to minimise their impact on itself (Gama & Donsez 2010).

4.1.4.2 Recoverability

The International Standard ISO/IEC 24765 defines **Recoverability** as “the restoration of a system, program, database, or other system resource to a state in which it can perform required functions” (ISO/IEC 2008). In the context of dynamic evolution, Recoverability concerns the ability of an application to be restored to a state in which it can continue to perform its functionality, after a failure caused by a transformation and/or its dynamic change(s) (Robertson & Williams 2006). For example, if one part fails after a transformation, the part should be replaced (Bianculli et al. 2007). Recovery was discussed earlier as an instance of Coordination (cf. Section 4.1.3.2) but not considered as an explicit quality factor. This definition makes Recoverability explicit and distinct from Coordination.

4.1.4.3 Reliability

Reliability refers to “the extent to which an application performs its intended function with the required precision” (Pressman 2005). Applied to dynamic evolution, reliability refers to the ability of an application to keep its intended functions from being compromised by ongoing changes and transformations such that it behaves in an unexpected manner (Cook & Dage 1999). Use of high quality parts can still make an application unreliable (Voas 1998) if these parts are incompatible with the rest of the application, or they demand a quality of service that cannot be met by the application. In addition, during a transformation, a part to be replaced should not be removed until its replacement part fully satisfies its roles (Cook & Dage 1999; Desnos et al. 2007; Gregersen & Jørgensen 2009).

4.1.4.4 Safety

Safety is defined as the ability of a distributed application and its parts to continue operating in a safe manner during and after a transformation (McKinley et al. 2004). In distributed computing, a safe system should prohibit its execution from violating any safety property in a predefined set, such as preventing an intrusion detection system

from failing (Gärtner 1999). Any identifiable discrete event (e.g. a transformation) that prohibits the application from executing safely needs to be detected and corrected (Gärtner 1999). For safety-critical computing in particular, an unsafe system means that its failure could endanger human life, lead to property damage, or cause environmental damage (ISO/IEC 1998; Knight 2002). A component that offers a critical function should remain unused if it is to be removed during a transformation (Kon & Campbell 2000) since removing it might cause the function during execution to abort and the application to crash.

4.1.4.5 Security

Security impacts dynamic evolution in two ways. Firstly, it affects the degree to which a distributed application is protected from security threats as it undergoes dynamic evolution. This involves confining new and replacement parts within a distributed application such that they can only perform permitted operations without compromising security (Lindqvist & Jonsson 1998). For instance, when a non-trusted component is inserted into an application, the component must be prohibited from retrieving user passwords. Conversely, the new and replacement parts secure themselves by imposing restrictions on how the rest of the application can access their functionality (Ben-Shaul et al. 2001). Notwithstanding the new and replacement parts, maintaining security requires that transformation agents are protected from unauthorised access (Horie et al. 1998; McKinley et al. 2004) in order to avoid unexpected and illegal modifications.

Secondly, the security model of an application should be able to easily adapt to the changing needs of the application. In this case, a security model should be divided into security policies, which specify the security rules to be followed, and the associated mechanism to govern and enforce the policies (Grimm & Bershad 2001). Furthermore, updates to security policies for parts should be performed as parts are added to and removed from the application (Grimm & Bershad 2001).

4.1.5 Existing Evaluation Frameworks

In this part of the quality factor identification, existing evaluation frameworks reviewed in Section 2.3 were examined to determine if any quality factor from these frameworks could be identified as meeting the selection criteria for quality factors (cf. steps 5 and 6, Appendix A). Of the evaluation frameworks reviewed, only one quality factor meeting the criteria was identified from Kung (1983). This factor corresponds to an existing quality factor (i.e. Locality) identified from the literature, and hence there was no need

to extend the initial set of quality factors identified from the literature. The original description of this factor is documented in Table 4.2.

Table 4.2 Potential quality factors/attributes selected from reviewed evaluation frameworks

<i>Framework and Type</i>	<i>Original Requirement</i>	<i>Relevant Quality Factor</i>
Kung (1983)	A model can achieve high changeability if it is localised . It is localised if a few pieces of the model needs to be modified when a change is required.	Locality

4.1.6 Summary and Discussion

Table 4.3 summarises the resulting dynamic evolution quality factors and their associated quality attributes, with reference to the literature and evaluation frameworks from which they were derived. A quality factor is expressed with one or more quality attributes. Each quality attribute can be verified (e.g. “no missing functionality”). Note that in Table 4.3 the attribute “assumptions and properties of a distributed application and its parts met by a dynamic change” is associated with two quality factors: Consistency and Completeness. Feiler and Li (1998) associate assumptions and properties with Consistency whereas Allen and Galan (1997) link assumptions to Completeness.

Table 4.3 Origins of quality factors from literature and evaluation frameworks

<i>Quality Categories, Factors and Attributes</i>	<i>Relevant Literature and Evaluation Framework</i>
<i>Soundness of Change</i>	
<i>Completeness</i>	
No missing functionality after a transformation	Allen and Garlan (1997), Medvidovic et al. (1999)
No missing parts after a transformation	Agnew et al. (1994b), Allen and Garlan (1997)
No missing, illegal or broken bindings after a transformation	Aksit and Choukair (2003), Feiler and Li (1998), Hillman and Warren (2004)
(Also in Consistency) assumptions and properties of a distributed application and its parts met by a dynamic change	Allen and Garlan (1997), Feiler and Li (1998), Mens and D'Hondt (2000)
<i>Consistency</i>	
Compatible bindings	Feiler and Li (1998), Meredith and Bjorg (2003)
Compatible communications protocol among parts	Allen and Garlan (1997)
All parts involved in a dynamic change identified before a transformation	Warren and Sommerville (1996)
No progression towards an error state after a transformation	Kramer and Magee (1990), Lee and Chang (2005), Gregersen and Jørgensen (2009), Sun and Jiang (2009), Zimmermann and Drobnik (1994)
Synchronisation of application's and parts' states after a transformation	Aksit and Choukair (2003), Oreizy et al. (1998), Warren and Sommerville (1996), Zieba and van Sinderen (2006)
A reachable state attained after a transformation	Gupta et al. (1996)
No critical procedures executed before a transformation	Warren and Sommerville (1996)

<i>Quality Categories, Factors and Attributes</i>	<i>Relevant Literature and Evaluation Framework</i>
No pending messages, interactions or transactions before a transformation	Adamek and Plasil (2005) , Bidan et al.(1998), Chen (2002), Kramer and Magee (1990), Warren and Sommerville (1996), Wang et al. (2006), Li (2009)
System invariants preserved from a transformation	Kramer and Magee (1998), Léger et al. (2010), Oueichek and Rousset de Pina (1996)
Adequate resources and support for new and replacement parts	Ben-Shaul et al. (2001), Feiler and Li (1998)
(Also in Completeness) assumptions and properties of a distributed application and its parts met by a dynamic change	Allen and Garlan (1997), Feiler and Li (1998), Mens and D'Hondt (2000)
<i>Correctness</i>	
Non-arbitrary and admissible dynamic changes	Bennett and Rajlich (2000), Gupta et al. (1996), Etzkorn (1992), Wermelinger (1998)
No unintentional behaviour during and after a transformation	Aksit and Choukair (2003), Buckley et al. (2005), Coyle et al. (2010), Gregersen and Jørgensen (2009), Mens and D'Hondt (2000), Segal and Frieder (1993), Zhang and Cheng (2006), Zieba and van Sinderen (2006)
Correct ordering of transformations	Segal (2002), Lim (1996), Segal and Frieder (1993)
Transformations at a right time	Gupta et al. (1996), Segal and Frieder (1993)
<i>Infusibility of Change</i>	
<i>Efficiency</i>	
Easily executed transformations	Bennett and Rajlich (2000), Hicks and Nettles (2005), Wang et al. (2002)
Quickly executed transformations	Agnew et al. (1994b), Bennett and Rajlich (2000), Gupta et al. (1996)
Resource efficient transformations	Agnew et al. (1994b),
Minimal disruptions to application functions and their users during a transformation	Bloom and Day (1993), Chen (2002), Chen et al. (2007), Janssens et al. (2005), Goudarzi and Kramer (1996), Hauptmann and Wasel (1996), Karamanolis and Magee (1996), Kramer and Magee (1990), Milazzo et al.(2005), Pfleeger and Bohner (1990), Taentzer et al. (2000), Zimmermann and Drobnik (1994)

<i>Quality Categories, Factors and Attributes</i>	<i>Relevant Literature and Evaluation Framework</i>
Minimal degradation to application performance during and after a transformation	Bidan et al. (1998), Bohner (1996), Chen (2002), Gregersen and Jørgensen (2009), Hauptmann and Wasel (1996), Hicks and Nettles (2005), Oueichek and Rousset de Pina (1996), Segal and Frieder (1993)
<i>Locality</i>	
Application partitioning and change localisation to partitions	Evans and Dickman (1999), Fayad and Cline (1996), Gregersen and Jørgensen (2009), Kung (1983) (framework), Oreizy and Taylor (1998), Plášil et al. (1998), Sun & Jiang (2009)
<i>Maintainability</i>	
All parts clearly defined in interaction (or workflow) specifications	Allen and Garlan (1997)
No degradation in cost and ease of modifications	Bohner (1996), Hicks and Nettles (2005)
No reduction in testability	Pfleeger and Bohner (1990)
Clear and detailed interactions	Shaw et al. (1996)
<i>Transparency</i>	
Transformations hidden from end users	Agnew et al. (1994b), Segal and Frieder (1993)
Transformation design and implementation hidden from application programmers	Chen (2002), Goudarzi and Kramer (1996), Segal and Frieder (1993), Gregersen and Jørgensen (2009)
Transformations hidden from parts unaffected by the transformations	Etz Korn (1992), Swaminathan and Goldman (1996)
Transformation agents hidden from operating environment	McKinley et al. (2004), Segal and Frieder (1993)
<i>Changeability of Application</i>	
<i>+ Configurability</i>	
+ Distributed application configurable to different policies for dynamic changes and transformations	Gregersen and Jørgensen (2009)
<i>Coordination</i>	
Transformations coordinated among multiple nodes/organisations	Chen et al. (2001), McKinley et al. (2004), Segal (2002), Segal and Frieder (1993)
Transformation agents tolerant of network unreliability during a transformation	Segal and Frieder (1993)
<i>+ Flexibility</i>	
+ Distributed application accommodating a variety of runtime changes (e.g. add, remove, update, merge, split)	Fayad and Cline (1996), Gregersen and Jørgensen (2009), Pahl (2004), Segal and Frieder (1993)
+ Any part of a distributed application to be changeable at runtime	Hicks and Nettles (2005)
<i>+ Loose Coupling</i>	
+ High level of independence between parts	Redmond and Cahill (2006)
+ Parts having their own lifecycles and runtime environments	Wang et al. (2006)

<i>Quality Categories, Factors and Attributes</i>	<i>Relevant Literature and Evaluation Framework</i>
<i>Separation of Concerns</i>	
Separating dynamic change concerns from functionality concerns	Andrade et al. (2002), Bidan et al. (1998), Etzkorn (1992), Kramer and Magee (1990), Milazzo et al. (2005), Oreizy et al. (1998), Oreizy and Taylor (1998), Zhang et al. (2005)
Separating communication concerns from functionality concerns	Andrade et al. (2002), Goldman et al. (1995), Oreizy et al. (1999), Wang et al. (1999)
Separating security support from functionality concerns	Grimm and Bershad (2001)
<i>Robustness of Application</i>	
<i>Fault Tolerance</i>	
High tolerance of faulty new and/or replacement parts	Feiler and Li (1998), Hicks and Nettles (2005), Voas (1998)
+ Barriers established to contain potentially faulty new and replacement parts	Gama and Donsez (2010)
<i>+ Recoverability</i>	
+ Restoration of an application to a state to continue to perform its functionality, after a failure caused by a transformation and/or its dynamic change(s)	Robertson and Williams (2006), Bianculli et al. (2007)
<i>Reliability</i>	
No compromise on intended functionality after a transformation	Cook and Dage (1999)
Replacement parts fully satisfying their roles	Cook and Dage (1999), Desnos et al. (2007), Gregersen and Jørgensen (2009)
<i>Safety</i>	
Distributed application and its parts operating safely during and after a transformation	McKinley et al. (2004)
<i>Security</i>	
Transformation agents secured from unauthorised access	Horie et al. (1998), McKinley et al. (2004)
No security compromise by new and replacement parts after a transformation	Lindqvist and Jonsson (1998)
Access to new and replacement parts restricted after a transformation	Ben-Shaul et al. (2001)
Dynamically updated security policy	Grimm and Bershad (2001)
Separating security policy from security enforcement	Grimm and Bershad (2001)

Notes:

1. All references to evaluation frameworks are explicitly labelled with "framework".
2. Quality factors/attributes labelled with "+" were not included in the survey which was commenced in 2006 (cf. Section 4.2).

The quality factors and their attribute descriptions were subsequently reviewed for completeness and clarity, and refined using the pilot test for a survey and feedback from survey respondents (see Section 4.2).

Despite their orientation towards composition-based applications, the proposed factors should also be useful for other (emerging) application types that increasingly need changes to be performed without application shutdown. For instance, dynamic evolution is a precursor to successful dynamic adaptation (Andrikopoulos et al. 2008).

Adaptive software development seeks to improve an application's fit to its environment by changing its parameters, components and composition (Aksit & Choukair 2003; McKinley et al. 2004). However, there are likely to be environment specific quality factors suitable for adaptive software but not covered in this research. For instance, Oreizy et al. (1999) suggest that an autonomous and self-adaptive system's ability to monitor environmental fluctuations and control its adaptation should be considered a quality factor for dynamic adaptation.

Quality factors specific to a particular type of composition-based distributed application were encountered but excluded from consideration. For instance, in the component-based approach, Dependency concerns the extent to which relationships among components are clearly defined with sufficient detail to facilitate applying changes to the application. A reason for the explicitness of the identification and management of dependencies (Dellarocas 1997) is that changes often result in adjustments to dependencies or the relationship among modules and nodes of a distributed application, as well as its operating environment. With explicit dependency information between a component and its peers, the component can configure itself or be configured by transformations to adapt to the evolving distributed application and its environment (Kon & Campbell 2000). In contrast, the SOA paradigm contends that services should be *independent* and *self-contained* modules, offering business functionality and reusability in isolation from other services (Mak et al. 2005; Pasley 2005; Yu et al. 2005). Accordingly, Dependency is not as important in SOA-based applications. Another example is Interoperability, an important characteristic in an SOA environment. The International Standard ISO/IEC 24765 defines Interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" (ISO/IEC 2008). More specifically, as elements in an SOA-based application evolve, they should not compromise the application's capability for Interoperability among its services (Ponnekanti & Fox 2004). Interoperability in SOA implementations is greatly facilitated by existing standards and guidelines such as those offered by the Web-Service Interoperability Organisation (WS-I 2006). Comparatively speaking, there is less demand for interoperability in component-based than in SOA-based applications because of the vendor lock-in (e.g. Sun's Enterprise Java Beans, CORBA⁹ components and Microsoft .NET frameworkTM) and proprietary

⁹ "CORBATM" stands for Common Object Request Broker Architecture, a platform-independent specification and model for building distributed object applications in a distributed environment.

technologies used in component implementations (Elfatraty 2007). In component-based applications, adapters or wrappers are often used to plug different components into an application to facilitate interoperability at the expense of higher maintenance and evolution costs (Brereton & Budgen 2000). Alternatively, the types of components used in an application can be restricted which may reduce the potential for component reuse (Brereton & Budgen 2000). On a different note, composition-based distributed applications are increasingly built using third party products, and different parts of applications may increasingly be developed and serviced by different vendors/teams. An analysis of the quality of these types of parts in influencing the ability of an application to evolve is beyond the scope of this research.

Two other factors were considered: Backwards-compatibility and Portability. Studies advocate that, as a part is upgraded, its replacement should be backwards-compatible with the original (e.g. Kaminski et al. 2006; Tsai et al. 2006; Wang et al. 2006). In this research, Backwards Compatibility is regarded as a functional concern rather than a quality factor in dynamic evolution, since Papazoglou and van den Huevel (2006) observe that both backwards-compatible and non-backwards-compatible changes are valid in dynamic evolution. Moreover, as an application evolves over an extended period of time, old functionality tends to change and lessens the need for Backwards-Compatibility. The International Standard ISO/IEC 24765 defines Portability as “the ease with which a system or component can be transferred from one hardware or software environment to another” (ISO/IEC 2008). Portability is a concern to a variety of application types and especially important for those that are intended to run on multiple platforms and environments, whether or not they are required to evolve. Portability is thus not included in the set of quality factors specific to dynamic evolution.

4.2 STEP 2: WEB SURVEY ASSESSMENT

This step aimed to assess and extend the set of quality factors synthesised in Step 1 using a web-based survey (Yun & Trumbo 2000) of experienced software development practitioners and researchers. Specifically, the survey asked each respondent to provide the following information:

- *Rating on quality attributes*

Each respondent was asked to rate the level of importance of the attributes of each quality factor on a 0-100 point quasi-continuous scale (0 for not-important-at-all, 100 for extremely important) with a 50 point score representing a neutral

rating (i.e. neither important nor not-important)¹⁰. As the survey was commenced in 2006, only factors identified up to this point were included in the survey. The ratings were used to calculate the perceived importance of the quality factors (see Section 4.2.4).

- *Additional comments*

Each respondent was asked to provide feedback/comments on the survey, and suggestions for additional quality issues for consideration. The latter assessed the completeness of the quality factors/attributes. Any additional quality issues were then subject to an expert review (see Section 4.2.3).

Moreover, the survey asked respondents about the following to gain an insight into application development and existing methodologies used in their organisations:

- *General information about the software development practices in each respondent's organisation*

Each respondent was asked about the software development methodologies used and any relevant certification/accreditation(s) held (e.g. ISO 9001) in his/her organisation. The inclusion of the former was to survey candidate methodologies in use for consideration in Step 3 (Section 4.3), and for evaluation of methodological support for the quality factors (Section 4.4).

- *Information about projects on distributed applications in each respondent's organisation* (see Section 4.2.5).

Follow-up interviews via email correspondence and/or face-to-face meetings were conducted with consenting respondents to clarify response data and any feedback/comment raised. To improve the consistency and reduce the vagueness of the survey, the definition of terms specific to dynamic evolution and component-based applications (transformation, component etc.) was provided in the first section of the

¹⁰ A quasi-continuous rating scale (0-100) is an improvement over a Likert scale in that it provides data of a finer level of granularity than a Likert scale (Chimi & Russell 2009). It increases the likelihood of spreading the collected data more evenly than a Likert scale, reducing errors in data analysis (Ladd 2009; Wu et al. 2005). It is usually implemented as a visual slide bar on a graphical user interface, although it can also be implemented on paper instruments (Chimi & Russell 2009). In this survey, a slide bar was implemented on the survey web site. To assign a rating to a quality attribute, a respondent simply drags the bar's handle to a desired position instead of typing a value for it. The bar positions also provide visual cues to a respondent on the relative importance of their respective quality attributes (cf. Appendix E.1.2.1).

survey. This information helped respondents to comprehend and scope the meaning of the rating questions. Abstract composition-based terms (i.e. parts and bindings) were substituted with component-based terms (i.e. components and connectors) for reasons of increased tangibility and relevance to real-world technologies in use.

4.2.1 Pilot Tests

The paper version of the survey was pilot tested by two research peers and four experienced practitioners. Discussions on the content, structure, wording and clarity of the rating questions were held with reviewers afterwards. Feedback and comments were reviewed and the content was revised accordingly to improve the quality of the survey and the definitions of the quality attributes. The revised paper survey was then implemented on a web site where a final on-line trial run was conducted by two research peers. Revisions reduced the average time to complete the survey from 40-50 minutes to 30-40 minutes. The final version can be found in Appendix E.1.

4.2.2 Data Collection

Candidates were recruited via email invitations to voluntarily participate in the web survey. The recruitment targeted people with adequate experience and/or knowledge in component-based technologies. In particular, the invitations were emailed to contacts from the industry and academia, some known to the researcher, plus public email alias groups that covered related topics (e.g. distributed system development). The email alias groups were included since their subscribers should have a genuine interest and familiarity in areas relevant to dynamic evolution. On the other hand, obtaining responses from a targeted group of people, a.k.a. *purposeful sampling* (Patton 2002), improves the credibility and authenticity of the responses collected at the expense of bringing selection bias into the responses. A friendly email reminder was sent to candidates who had not completed the web survey, close to the end-date of the survey period. All responses were further screened with respect to the respondent profile data to ensure respondents had sufficient experience in software development methodologies (minimum of two years), application development (minimum of three years), and distributed application development (minimum of one year) to contribute to the survey.

Invitations were sent to over one hundred and sixty email addresses and eight email aliases, of which forty valid responses were collected. With respect to demography, most respondents lived in Oceania (70%), whilst the others were from Europe (7.5%),

North America (15%) and Asia (7.5%). Of the responses collected, thirty-eight (95%) were recruited via direct email invitations (vs. two via invitations sent to email aliases) of which 74% were known to the researcher.

4.2.3 Additional Dynamic Evolution Quality Factors/Attributes

In addition to the quality attributes presented in the survey, the respondents suggested several issues for consideration. To improve credibility of the issues suggested by respondents and determine whether they should be included in the set, a two-step examination was performed. First, issues not satisfying the criteria (Appendix A) were eliminated from further analysis, leaving a total of six potential issues (left of Table 4.4).

Table 4.4 Analysis results of potential quality issues suggested by respondents

<i>Suggested Issue</i>	<i>Analysis</i>	<i>Expert 1's Comment</i>	<i>Expert 2's Comment</i>	<i>Inclusion to the Set</i>
Integrity (1)	The International Standard ISO/IEC 24765 defines integrity as “the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data”. With respect to dynamic evolution, the computer programme aspect is covered by Security. The data aspect requires expert review.	<i>should-be-considered</i> (Very important)	<i>should-be-excluded</i> (Too complex and costly to address)	No
Robustness (1)	Already included as a quality factor which was not shown to survey respondents	N/A	N/A	Already included
Error handling and reporting (1)	Out of scope - considered as a solution to Recoverability and Fault Tolerance and thus discarded.	N/A	N/A	No
No unintentional behaviour when faulty parts are present (1)	Unintentional behaviour of an application may be caused by faulty parts during and after a transformation.	<i>should-be-excluded</i> (Improbable to achieve)	<i>should-be-excluded</i> (Too complex to detect and implement)	No
Deterministic and repeatable transformations (1)	Closely related to two of Correctness's attributes “non-arbitrary and admissible dynamic changes” and “no unintentional behaviour during and after a transformation”.	<i>should-be-considered</i> (Very important, essential for transformations to have known outcomes)	<i>should-be-excluded</i> (Too complex and costly to address)	No
Recovery (2)	Recovering an individual part to continue to function after a failure [caused by a transformation and/or its dynamic change(s)]. (see also “Recoverability”, Table 4.5)	<i>should-be-considered</i> (Very important and significance of recovery analogous to rolling back a transaction in a database system)	<i>should-be-considered</i> (Error recovery an extremely important area)	Yes

Note: The number enclosed in parenthesis following each issue represents the number of respondent(s) who raised the issue:

In the second step, a panel of two experts, one from academia and one from the industry, reviewed the potential issues using a Delphi-type method. The experts firstly independently assessed if each potential issue should be considered for dynamic evolution, and provided a justification for his/her recommendation. Then, one expert at a time was given an opportunity to revise his/her responses on potential issues that had been assessed differently by the other expert. For inclusion in the set of quality factors after Step 1, a potential issue needed to be assessed by both experts as “should-be-considered” for dynamic evolution. Expert comments and the outcomes on selecting which issues to be incorporated into the set of quality factors are shown in Table 4.4.

The same two-step approach was repeated for additional quality attributes found from articles and evaluation frameworks that were published after the survey commenced in 2006 (cf. Section 4.1.6). The analysis results of these new quality factors/attributes are documented in Table 4.5.

Table 4.5 Analysis results of additional quality attributes synthesised from the literature (cf. Table 4.3)

<i>Quality Attribute</i>	<i>Expert 1's Comment</i>	<i>Expert 2's Comment</i>	<i>Inclusion to the Set</i>
Distributed application configurable to different policies for dynamic changes and transformations, e.g. “abort all business transactions” vs. “wait until all business transactions finish” (cf. Configurability, Section 4.1.3.1)	<i>should-be-excluded</i> (Moderately important, making an application much more powerful and flexible in supporting dynamic evolution)	<i>should-be-considered</i> (Common and very important. With respect to the policy examples, it needs to be very clear on how to handle existing transactions during a transformation. It is good to have both policies available to suit different needs if such a transformation is often performed.)	No
Barriers established to contain potentially faulty new and replacement parts (cf. Fault Tolerance, Section 4.1.4.1)	<i>should-be-considered</i> (Very important since overall application integrity could be compromised because of faults)	<i>should-be-considered</i> (Extremely important and common in runtime adaptation)	Yes
Any part of a distributed application to be changeable at runtime (cf. Flexibility, Section 4.1.3.3)	<i>should-be-excluded</i> (Dependent on requirements but adding unnecessary complexity into the design)	<i>should-be-excluded</i> (Not important at all, often impossible to design such an application without incurring prohibitive complexity and performance cost. A more common practice is to predict the likely changed parts and optimise the design to cater just for adapting those specific parts at runtime.)	No

<i>Quality Attribute</i>	<i>Expert 1's Comment</i>	<i>Expert 2's Comment</i>	<i>Inclusion to the Set</i>
Distributed application accommodating a variety of runtime changes (e.g. add, remove, update, merge, split) (cf. Flexibility, Section 4.1.3.3)	<i>should-be-excluded</i> (Same as the above)	<i>should-be-excluded</i> (Again, the design should anticipate potential needs rather than "over engineer" to support all types of changes, even those with small chance of being needed. There is often a trade-off between cost/complexity/performance and flexibility.)	No
Parts having their own lifecycles and runtime environments (cf. Loose Coupling, Section 4.1.3.4)	<i>should-be-considered</i> (Very important, enforcing loose coupling and robustness between components and services)	<i>should-be-considered</i> (Common, very important and desired property for components/services in components-based software engineering)	Yes
High level of independence between parts (cf. Loose Coupling, Section 4.1.3.4)	<i>should-be-considered</i> (Same as above)	<i>should-be-considered</i> (Common, very important to decouple components/services from one another)	Yes
Restoration of an application to a state to continue to perform its functionality, after a failure caused by a transformation and/or its dynamic change(s) (cf. Recoverability, Section 4.1.4.2) (see also "Recovery" in Table 4.4)	<i>should-be-considered</i> (Very important and significance of recovery analogous to rolling back a transaction in a database system)	<i>should-be-considered</i> (Error recovery an extremely important area)	Yes

The expert review led to the addition of the following quality attributes/issues (in *italic* font) as incorporated in the revised set of quality factors (all summarised in Table 4.3):

- *Barriers established to contain potentially faulty new and replacement parts* (from Table 4.5)

This attribute originates from the literature published after the survey commenced in 2006 and is incorporated into Fault Tolerance as a new attribute (cf. Table 4.3).

- *Parts having their own lifecycles and runtime environments* (from Table 4.5)

This attribute originates from the literature published after the survey commenced in 2006 (cf. Table 4.3). A new quality factor called Loose Coupling is defined for it according to the original literature (cf. Section 4.1.3.4).

- *Restoration of an application and its parts to a state to continue to perform their functionality, after a failure caused by a transformation and/or dynamic change(s)*

This attribute is combined from two closely related statements “recovering an individual part to continue to function after a failure [caused by a transformation and/or its dynamic change(s)]” (from “recovery” in Table 4.4) and “restoration of an application to a state to continue to perform its functionality, after a failure caused by a transformation and/or dynamic change(s)” (from Table 4.5) The first statement was suggested by two respondents in the survey whereas the second one originates from the literature published after the survey commenced in 2006 (cf. Section 4.1.4.2). This attribute defines a new quality factor called Recoverability.

Quality issues and attributes rejected by experts in this step are excluded from further consideration in the revised set even if they are identified at a later stage in the analysis (i.e. extension of quality factors from methodologies in Step 3).

4.2.4 Importance Rating of Dynamic Evolution Quality Factors

The importance rating for each quality factor was derived from the scores of its associated attribute rating questions using Partial Least Squares¹¹. A Wilcoxon one-sample signed-rank test¹² was applied to the data to check if they were from a distribution away from a known median of 50 being the mid-point between “not-important-at-all” (0) and “extremely-important” (100). Note that the analysis was limited to the quality attributes synthesised from articles and evaluation frameworks published *before 2006* (cf. Table 4.3). As reported in Table 4.3, there are in fact new quality attributes found from articles and evaluation frameworks published between 2006 and 2010. Thus, no data could be collected for these new attributes in the survey and they were excluded from the importance analysis in this section. However, they were

¹¹ Partial Least Squares (PLS) is a multivariate analysis method for modelling the relationships between a latent variable (e.g. quality factor) and its indicators (e.g. quality attributes) (Chin et al. 1996; Wold et al. 2001). It allows each indicator to vary in how much it contributes to the composite score of the latent variable, rather than assuming that all indicators have equal weights on the latter. The quality factor importance rating was calculated as follows. The weight of each attribute on the factor was first estimated with the scores using PLS and then normalised. Next, the importance rating was calculated by multiplying the weight with each associated attribute score and adding all the weighted attribute scores for the factor.

¹² The Wilcoxon one-sample signed-rank test is a non-parametric method which tests whether the median of a population is significantly different from a specified value (e.g. midpoint) (Arnold 1965; Wilcoxon 1945). The test is based on rankings drawn from population samples.

assessed in an expert review, to be discussed in Section 4.2.3.

The signed-rank test results together with median, minimum, maximum and Cronbach's alpha (α) scores¹³ are shown in Table 4.6. It indicates that all the quality factors appeared to be significantly different ($p_{wsr}=0.000$) from the mid-point. Note also that for each quality factor, the sum of positive signed-ranks (s^+) was much greater than the sum of negative signed-ranks (s^-), implying that the factor was perceived as important.

Table 4.6 Descriptive statistics and results of Wilcoxon one-sample signed-rank test on quality factors

Category	Quality Factor	Median	Min	Max	s^+	s^-	Z_{wsr}	p_{wsr}	Cronbach's alpha (α)
Soundness of Change	Completeness	85.7	41.6	100.0	815	5	5.44	0.000	0.816
	Consistency	81.0	58.1	100.0	820	0	5.51	0.000	0.665
	Correctness	78.5	44.3	100.0	817	3	5.47	0.000	0.660
Infusibility of Change	Efficiency	66.3	18.0	94.1	726	94	4.25	0.000	0.785
	Locality	81.0	40.0	100.0	809	11	5.36	0.000	N/A ¹
	Maintainability	73.5	25.0	97.8	765	55	4.77	0.000	0.761
	Transparency	77.0	30.4	98.1	795	25	5.17	0.000	0.706
Changeability of Application	Coordination	75.8	21.0	100.0	799	21	5.23	0.000	0.703
	Separation of Concerns	78.4	43.1	100.0	816	4	5.46	0.000	0.803
Robustness of Application	Fault Tolerance	83.0	25.0	100.0	807	13	5.34	0.000	N/A ¹
	Reliability	90.0	45.0	100.0	817	3	5.47	0.000	0.605
	Safety	90.0	50.0	100.0	820	0	5.51	0.000	N/A ¹
	Security	77.5	49.6	100.0	817	3	5.47	0.000	0.786

Note:

1. Cronbach's alpha is not applicable to quality factors having only one attribute (i.e. one rating question).

Difference in and grouping of perceived level of importance

Which quality factors were perceived to be more important were determined by examining the ranking order of the quality factors and the statistical grouping of similarly ranked factors which were not different among themselves. The ranking order is indicative of how important one factor is compared with another, and was determined by calculating and ordering the mean rank scores of the quality factors.

The statistical grouping of quality factors was determined by an analysis of difference among the factors in two steps. In the first step, a Friedman test¹⁴ was performed to

¹³ Cronbach's alpha checks whether independent variables (e.g. quality attributes) are closely related when a dependent variable (e.g. quality factor) is derived from them (Cronbach 1951).

¹⁴ The Friedman test is a non-parametric method to check whether there is any difference among a set of independent variables based on their samples (Friedman 1937).

detect if there was any difference between the various factors ($n=40$, $\chi^2=102$, $df=12$, $p=0.000$). Given the positive result, the second step involving the use of the Wilcoxon matched-pair signed-rank test¹⁵ was carried out to identify pairs of factors with significant differences. The results of the test, together with the mean rank scores of the perceived importance of the quality factors, are shown in Table 4.7. For convenience, pairs with significant difference ($p<0.05$) are highlighted in their corresponding cells in Table 4.7 (e.g. Completeness vs. Safety).

Table 4.7 Results of Wilcoxon signed-rank tests for matched pairs on quality factors

		Reliability	Safety	Completeness	Coordination	Consistency	Fault Tolerance	Correctness	Security	Locality	Separation of Concerns	Transparency	Maintainability	Efficiency
	mean rank	significance level (p)												
Reliability	9.99		0.75	0.03	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Safety	9.93	0.75		0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Completeness	7.96	0.03	0.01		0.66	0.82	0.77	0.06	0.13	0.08	0.13	0.09	0.00	0.00
Coordination	7.39	0.00	0.00	0.66		0.45	0.44	0.72	0.53	0.43	0.75	0.28	0.01	0.00
Consistency	7.34	0.00	0.00	0.82	0.45		0.90	0.24	0.12	0.16	0.07	0.05	0.00	0.00
Fault Tolerance	7.28	0.03	0.01	0.77	0.44	0.90		0.54	0.32	0.22	0.31	0.09	0.02	0.00
Correctness	7.09	0.00	0.00	0.06	0.72	0.24	0.54		0.28	0.26	0.26	0.30	0.00	0.00
Security	6.73	0.00	0.00	0.13	0.53	0.12	0.32	0.28		0.79	0.79	0.21	0.01	0.00
Locality	6.70	0.00	0.00	0.08	0.43	0.16	0.22	0.26	0.79		0.79	0.92	0.17	0.00
Separation of Concerns	6.49	0.00	0.00	0.13	0.75	0.07	0.31	0.26	0.79	0.79		0.32	0.00	0.00
Transparency	6.00	0.00	0.00	0.09	0.28	0.05	0.09	0.30	0.21	0.92	0.32		0.11	0.00
Maintainability	4.68	0.00	0.00	0.00	0.01	0.00	0.02	0.00	0.01	0.17	0.00	0.11		0.06
Efficiency	3.45	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	

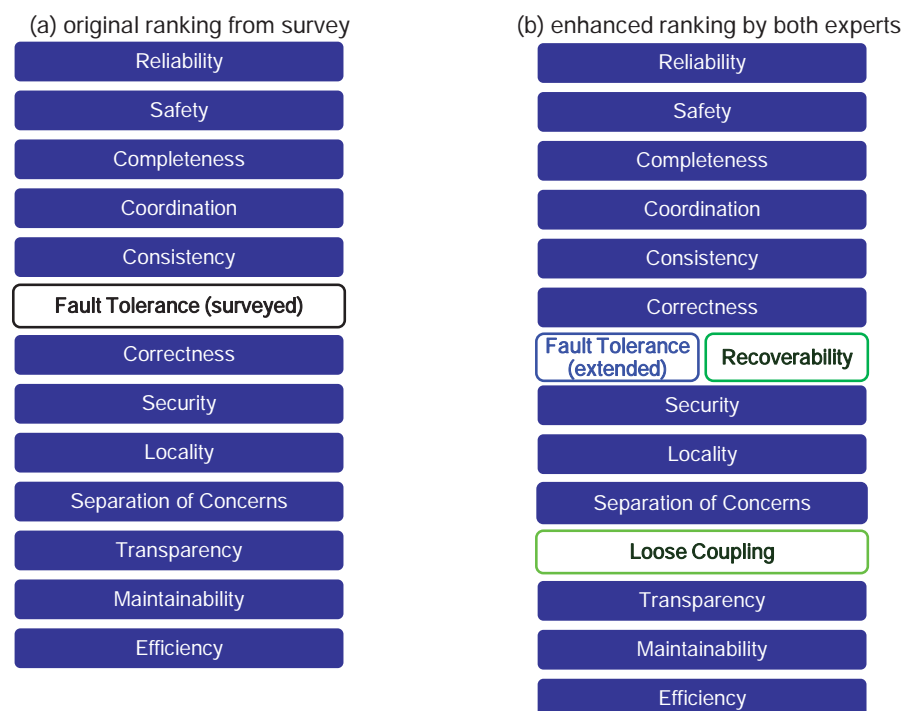
Note: Each highlighted cell represents a significant difference ($p<0.05$) between the two quality factors in its corresponding row and column.

Of particular interest to this research is the grouping formed by *Reliability* and *Safety*, which were perceived to be significantly different from all other quality factors except between themselves ($p=0.75$) (cf. Table 4.7). Furthermore, they had the highest mean

¹⁵ The Wilcoxon matched-pair signed-rank test is a non-parametric method to test if two populations of the same size have different medians based on their ranked data (Wilcoxon 1945).

rank scores (9.99 and 9.93). This suggests that they were perceived as the most important quality factors (both under Robustness of Application). No distinct sub-grouping can be identified from the rest of the factors. Nevertheless, *Efficiency* was perceived to be significantly different from all other quality factors except Maintainability ($p=0.06$) and had the lowest mean rank score.

As reported in Section 4.2.3, there are quality attributes in addition to those rated by respondents in the survey, leading to an enhancement of the existing factor Fault Tolerance and two new factors: Loose Coupling and Recoverability. To assess their importance relative to the factors already analysed in Table 4.7, the experts were asked to update the ranking order (cf. Table 4.7) by incorporating these three factors into the ranking order in two steps. First, the experts independently incorporated the new quality factors into the ranking order. Then, their *differences* were discussed with each expert and progressively reconciled in a new ranking order which was agreed by both experts. Figure 4.4 shows (a) the original ranking order calculated from the survey results, and (b) the enhanced ranking order with these three factors as jointly agreed by the experts.



source: developed for this research

Figure 4.4 Quality factor importance rankings before and after expert review

While the experts individually assessed the three factors, they rated Loose Coupling similarly. For Fault Tolerance, however, the experts agreed to lower its rating slightly, from above Correctness (Figure 4.4(a)) to below Correctness (Figure 4.4(b)). One

expert explained that, with support from the other expert, if transformations and dynamic changes appear to be *correct*, they would lessen the need for Fault Tolerance. Hence Fault Tolerance could be treated as less important than Correctness. Lastly, the experts contended that since both Recoverability and Fault Tolerance concern fault treatment, they rated Recoverability as important as Fault Tolerance.

In the survey feedback, two respondents commented that contextual issues would influence their perception on dynamic evolution quality factors and hence their importance rankings would be affected (e.g. Safety being highly important to safety-critical applications). While this research acknowledges contextual issues, they are not in scope for this research (Section 1.5). Nonetheless, this research aims to be generic rather than being specific to a particular context.

4.2.5 Software Practice and Project Information

All respondents reported the use of software development methodologies within their organisations. The majority used in-house methodologies (55%) while one quarter used IBM Rational Unified Process (RUP) (cf. Section 2.4.5) as-is or with some extensions. One organisation adopting RUP supplemented it with the Extreme Programming methodology (Beck & Andres 2005) and another with PRINCE2 for project management (Office of Government Commerce 2005b). Another organisation supplemented RUP with the Reference Model of Open Distributed Processing (RM-ODP) (ISO/IEC 1995), for specifying architectural representations using a reference model, and with significant in-house enhancements. In addition, half the respondents reported their organisations had gained an internationally recognised accreditation. International Standard Organisation accreditations dominated, particularly ISO 9001 (60%). Five organisations earned the CMMI (Capability Maturity Model for Integration) levelled certification, with one attaining level 5.

About one hundred distributed application development projects, in eighteen industries, were reported by respondents. The dominant industry was telecommunication (27%) with other major segments being finance (14%), government (10%) and IT product/services (10%). There were some interesting comments about these projects. Thirteen percent of the applications utilised dynamic evolution in which changes were activated on-the-fly without requiring application restart. The other eighty-seven percent of the applications used the manual restart option to activate new changes. With regard to the frequency of releases, many applications required subsequent releases after their initial/first deployment with the number of releases being clustered in the 2-to-6

band. Furthermore, a minority (18%) of the applications having subsequent releases did not require application restart when releases were deployed. In one extreme case, a value of 99 was reported by a respondent, who mentioned at a follow-up meeting, that the application was indeed undergoing a new release every week! The interval between releases ranged from less than one month to three years.

4.3 STEP 3: DYNAMIC EVOLUTION QUALITY FACTORS EXTENSION FROM METHODOLOGICAL PERSPECTIVE

This step was to extend (if necessary) the set of dynamic evolution quality factors after Step 2 by examining relevant software development methodologies. Any additional quality factors were included in the set.

Of the methodologies used by the organisations for which respondents surveyed worked, the documentation for RUP, XP, RM-ODP and PRINCE2 was available to the researcher and thus checked for incorporation into the examination. RUP was included in the examination since it meets the selected criteria stated in Section 2.4. XP was rejected as its activities are independent of dynamic evolution and the kind of application being developed. So was PRINCE2 which is project management specific. Although RM-ODP observes small aspects of dynamic changes (e.g. the presence of dynamic selection of objects and runtime bindings for objects), it was not included in the examination since it is not a methodology per se and its limited support of composition and evolution is restricted to objects at design time. In addition to RUP, a number of development methodologies (reviewed in Section 2.4) met the selection criteria and were included in the evaluation.

Based on the same selection criteria for quality factors specified in steps 5 and 6 of Appendix A, potential and relevant quality factors were identified from the selected methodologies (Table 4.8). Factors rejected by the experts in Step 2 were excluded. The identification was conducted without regard to the classification of factors adopted previously in this research to ensure that the results of this step were not unduly influenced by prior thinking. Each factor identified was then assessed to determine if it mapped to one or more factors from the extended set of quality factors from Step 2. If it did, the equivalent factor(s) from the extended set was documented in the rightmost column of Table 4.8. For instance, Select Perspective's notion of Maintainability maps to the attributes of Efficiency and Maintainability from the extended set.

Table 4.8 Analysis of potential quality factors elicited from reviewed methodologies

Methodology				Mapping to quality factors as revised from Step 2
Name	Quality Factor	Quality Attribute	Relevant Literature	
AEM		An adaptation might require <u>coordination among multiple sites</u> when the application is physically distributed and adaptation requires changes at several sites simultaneously.	Oreizy et al. (1999)	Coordination: “transformations coordinated among multiple nodes/organisations”
		<u>Strictly separating computation from communication</u> lets a system’s computation and communication relationships evolve independently of one another, including rearranging and replacing the components and connectors of an application while the application executes.		Separation of Concerns: “separating communication concerns from functionality concerns”
		A C2 component is unaware of components below itself, so it is <u>oblivious to runtime changes that involve these components</u> .		Transparency: “transformations hidden from parts unaffected by the transformations”
Catalysis	Upgradability	Can the system at runtime be <u>upgraded</u> with new features or versions of software without bringing operations to a halt?	D'Souza et al. (1998)	none: new quality factor Extensibility
CBDI-SAE	Separation of Concerns	Provide-Consume Separation: Consumers can compose and <u>evolve</u> solutions independent of providers’ technologies and business contexts.	Allen (2007)	none: new attribute for Separation of Concerns
		Flexible Provision: separating service specification from implementation ... allowing implementations to be <u>replaced</u> .		Separation of Concerns: “separating part specification from realisation concerns”
ERL	Service Loose Coupling	The degree of independence between services. Loose coupling increases the agility of an application to respond to <u>unforeseen changes and evolution</u> .	Erl (2005)	Loose Coupling: “high level of independence between parts”
	Service Autonomy	The degree of a service to have control and governance of its processing. Low autonomy inhibits service deployment and <u>evolution</u> .		none: new quality factor Autonomy
	Extensibility	The ability of a service to be <u>extended with new</u> functionality. The ability of an application to be <u>extended with new services</u> .		none: new quality factor Extensibility
KobrA	Encapsulation	The description of what a component does (i.e. Component specification) is separated from the description of how it does it (i.e. component realization). This allows <u>new versions of a component to be interchanged with old versions</u> .	Atkinson et al. (2002)	none: new attribute for Separation of Concerns

<i>Methodology</i>				<i>Mapping to quality factors as revised from Step 2</i>
<i>Name</i>	<i>Quality Factor</i>	<i>Quality Attribute</i>	<i>Relevant Literature</i>	
OPF	Modifiability - Extensibility (encompassing Scalability)	The degree of easiness of <u>enhancing</u> a business enterprise, system, application, or component and increasing its capabilities to meet future factors and significantly changing requirements.	OPFRO (2009)	none: new quality factor Extensibility
	Modifiability - Maintainability	The degree of easiness with which a business enterprise, system, application, or component can be <u>modified</u> between major releases when not required by changes to requirements.		Maintainability : “no degradation in cost and ease of modification ”
P&H	Service Coupling - Representational Coupling	The ability of existing services to be <u>swapped with new service</u> implementations.	Papazoglou and van den Heuvel (2006)	none: new quality factor Extensibility
RUP	Run-time Replaceability	The ability of a component to be (re)deployed into a running software system to enable the system to be <u>upgraded</u> without loss of availability.	online tool version of Kruchten (2003)	none: new quality factor Extensibility
Select Perspective	Maintainability	The application should be designed in such a way that <u>upgrades and other changes</u> can be made with minimum disruption.	Apperly et al. (2003)	Efficiency : “minimal disruptions to application functions and their users during a transformation” and “minimal degradation to application performance during and after a transformation”
		Reduce the time and cost of <u>repairs</u> and making <u>enhancements</u> .		Maintainability : “no degradation in cost and ease of modification”

Notes:

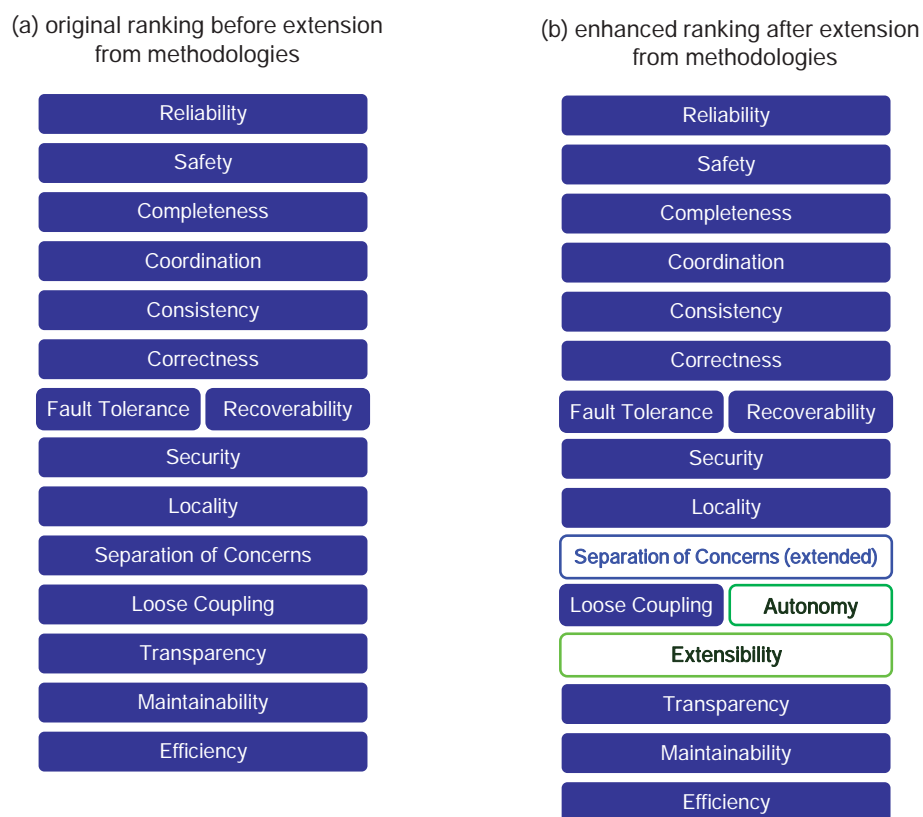
1. Underlined text highlights the relevance of quality attributes with changes, maintenance and/or evolution as argued by respective methodologies.
2. ASG's notions of Availability, Fault Tolerance and Recovery correspond to Transparency, Fault Tolerance and Recoverability respectively. However, since ASG does not relate these factors to changes/maintenance/evolution, ASG is excluded from Table 4.8.

Other factors (e.g. Autonomy from ERL) not covered in the revised set of quality factors after Step 2 (Section 4.2) represent potential additional quality factors. Analysis results for their inclusion into the set are shown in the rightmost column of Table 4.8. The revisions to the set of quality factors following this step are shown in *italic* font below:

- *Autonomy* (new factor)
 - *the ability of a part to have control and governance of its own processing*
- *Extensibility* (new factor)

- *runtime extension/upgrade of an application with new functionality*
- *runtime extension/upgrade of parts in an application with new functionality*
- *runtime extension/upgrade of an application with new parts*
- Separation of Concerns (existing factor)
 - (new quality attribute) *separating part specification from realisation concerns*
 - (new quality attribute) *separating realisations of parts* (e.g. service providers) *from those of part clients* (e.g. service consumers)

Because of the above changes, experts were asked again to update the importance ranking order similarly to Step 2 (Section 4.2.4). The results are shown in Figure 4.5. The ranking for Separation of Concerns was unchanged. Autonomy was ranked the same as Loose Coupling and one expert commented that it was because they are closely related concepts. Extensibility was less important than Loose Coupling because given Loose Coupling, an architecture would provide support for Extensibility to a certain extent.



source: developed for this research

Figure 4.5 Quality factor importance rankings before and after methodology extension

4.4 STEP 4: EVALUATION OF SUPPORT FOR DYNAMIC EVOLUTION QUALITY FACTORS IN METHODOLOGIES

The evaluation objective was to determine what features from existing methodologies could be reused, what features could be enhanced for use and any additional support required. Accordingly, four scale points - “H(high)” for full support, “M(medium)” for partial support requiring small enhancement, “L(low)” for inadequate support requiring significant enhancement, and “None” for no support - were applied to grade the level of support provided by the methodologies for each of the quality attributes. These scale points also provide some indication as to the potential for reuse of the methodology feature in its support of the related quality attributes. The initial version of the scale points were reviewed by two research peers whose comments were incorporated into the final version of the scale points (cf. Table Appendix B.1).

The methodology evaluation was performed by the researcher on the analysis and design aspects of the reviewed methodologies which fall within the scope of this research. The limitation of having one person to perform the evaluation is discussed in Section 8.3. Table 4.9 summarises the evaluation results. The actual features from the reviewed methodologies offering the support are detailed in Appendix B.1. At the attribute level, only eleven of the attributes are sufficiently supported by the methodologies to suggest models/tasks/techniques for reuse (i.e. “H”), while seven have a level of support that requires small enhancement (i.e. “M”). New methodological support is needed for the remaining attributes (over thirty).

At the quality factor level, Autonomy appears to be fully supported (i.e. all attributes graded with “H”), while Loose Coupling and Reliability are the next best supported (i.e. with at least half of each factor’s attributes graded with “H”). The worst cases are Configurability, Coordination, Efficiency, Extensibility, Flexibility, Recoverability and Transparency; no methodology evaluated offers explicit support that can be enhanced or reused. The rest of the quality factors are partially supported, with some elements suitable for reuse (i.e. graded with “H”) as well as some others requiring enhancement (i.e. graded with “M”). Of the quality factors (Safety and Reliability) perceived as most important by respondents in Section 4.2.4, their level of support from the methodologies assessed is partial (scoring one “H” out of three attributes). The fact that the methodologies do not provide adequate support for these quality factors is a concern, suggesting areas to which priority should be given when developing methodological support for dynamic evolution.

From the methodological viewpoint, none of the methodologies offer adequate support for all the factors in Table 4.9. At best, SeCSE offers the most number of reusable elements (i.e. three H's) for Consistency, Reliability and Separation of Concerns but not all of their attributes. The next best methodologies (with two H's) split evenly between those that support component-based development (i.e. Catalyst, Select Perspective and RUP) and those that support SOA-based development (i.e. RUP, ERL and P&H). None of the other methodologies adequately support more than one attribute or factor. The argument is made that a more comprehensive approach from a methodological perspective is needed to broadly cover the quality factors as a whole to tackle dynamic evolution.

Table 4.9 Evaluation results of methodological support for quality factors

Quality Categories, Factors and Attributes	Methodology (C:supporting component-based, S:supporting SOA-based)												
	C	C	C	C	C	C	C, S	S	S	S	S	S	S
	AEM	Catalysis	EPIC	KobrA	OPF	Select Perspective	RUP	ASG	CBDI-SAE	ERL	P&H	SeCSE	SUPER
Soundness of Change													
Completeness													
No missing functionality after a transformation					L		H						
No missing parts after a transformation							L						
No missing, illegal or broken bindings after a transformation							L						
(Also in consistency) assumptions and properties of a distributed application and its parts met by a dynamic change				L			L					L	

Quality Categories, Factors and Attributes	Methodology (C:supporting component-based, S:supporting SOA-based)												
	C	C	C	C	C	C	C _S	S	S	S	S	S	S
	AEM	Catalysis	EPIC	KobrA	OPF	Select Perspective	RUP	ASG	CBDI-SAE	ERL	P&H	SeCSE	SUPER
Consistency													
Compatible bindings													
Compatible communications protocol among parts													
All parts involved in a dynamic change identified before a transformation							L						
No progression towards an error state after a transformation													
Synchronisation of application's and parts' states after a transformation													
A reachable state attained after a transformation													
No critical procedures executed before a transformation													
No pending messages, interactions or transactions before a transformation													
System invariants preserved from a transformation												H	
Adequate resources and support for new and replacement parts													
(Also in completeness) assumptions and properties of a distributed application and its parts met by a dynamic change				L			L					L	
Correctness													
Non-arbitrary and admissible dynamic changes	L				L		L						
No unintentional behaviour during and after a transformation													
Correct ordering of transformations													
Transformations at a right time													
Infusibility of Change													
Efficiency													
Easily executed transformations													
Quickly executed transformations													
Resource efficient transformations													
Minimal disruptions to application functions and their users during a transformation													
Minimal degradation to application performance during and after a transformation													
Locality													
Application partitioning and change localisation to partitions		M			L	L							

Quality Categories, Factors and Attributes	Methodology (C:supporting component-based, S:supporting SOA-based)												
	C	C	C	C	C	C	C _S	S	S	S	S	S	S
	AEM	Catalysis	EPIC	KobrA	OPF	Select Perspective	RUP	ASG	CBDI-SAE	ERL	P&H	SeCSE	SUPER
Maintainability													
All parts clearly defined in interaction (or workflow) specifications													
No degradation in cost and ease of modifications			M		M	M							
No reduction in testability							L						
Clear and detailed interactions													
Transparency													
Transformations hidden from end users													
Transformation design and implementation hidden from application programmers													
Transformations hidden from parts unaffected by the transformations	L												
Transformation agents hidden from operating environment													
Changeability of Application													
Autonomy (after Step 3)													
Self-control and self-governance of parts' own processing										H			
Configurability													
Distributed application configurable to different policies for dynamic changes and transformations (after Step 2)													
Coordination													
Transformations coordinated among multiple nodes/organisations													
Transformation agents tolerant of network unreliability during a transformation													
Extensibility (after Step 3)													
Runtime extension/upgrade of an application with new functionality										L			
Runtime extension/upgrade of parts in an application with new functionality													
Runtime extension/upgrade of an application with new parts										L			
Flexibility (after Step 2)													
Any part of a distributed application to be changeable at runtime	L												
Distributed application accommodating a variety of runtime changes													
Loose Coupling													
High level of independence between parts		H		L	L		M			M	H		

Quality Categories, Factors and Attributes	Methodology (C:supporting component-based, S:supporting SOA-based)												
	C	C	C	C	C	C	C _S	S	S	S	S	S	S
	AEM	Catalysis	EPIC	KobrA	OPF	Select Perspective	RUP	ASG	CBDI-SAE	ERL	P&H	SeCSE	SUPER
Parts having their own lifecycles and runtime environments (after Step 2)													
Separation of Concerns													
Separating dynamic change concerns from functionality concerns													
Separating communication concerns from functionality concerns													
Separating security support from functionality concerns													
Separating realisations of parts from those of part clients (after Step 3)						H			L				
Separating part specification from realisation concerns (after Step 3)		H		H		H	H	H	L	H	H	H	
Robustness of Application													
Fault Tolerance													
High tolerance of faulty new and/or changed parts					L		L					M	
Barriers established to contain potentially faulty new and replacement parts (after Step 2)					L								
Recoverability (after Step 2)													
Restoration of an application and its parts to a state to continue to perform their functionality, after a failure caused by a transformation and/or its dynamic change(s)					L							L	
Reliability													
No compromise on intended functionality after a transformation			L				L						
Replacement parts fully satisfying their roles			L				L					H	
Safety													
Distributed application and its parts operating safely during and after a transformation	L				M								
Security													
Transformation agents secured from unauthorised access			L		M		L						
No security compromise by new and replacement parts after a transformation			L		M		L						
Access to new and replacement parts restricted after a transformation			L		M		L						
Dynamically updated security policy													
Separating security policy from security enforcement													

Notes:

1. The evaluation score indicates the extent of support as per Table Appendix B.1.
2. Quality factors/attributes labelled with “Step 2” mean they originate from survey respondents and/or the literature and evaluation frameworks published after the survey, and were subsequently reviewed and recommended by the experts in Step 2.
3. Quality factors/attributes labelled with “Step 3” refer to quality factors and attributes sourced from methodologies reviewed in Step 3.
4. A blank cell means there is no support from a methodology (identified by the column) for a particular quality attribute (identified by the row).

4.5 RELATED WORK

This section reviews a number of quality standards and quality models, and compares them with the set of dynamic evolution quality factors developed in Task 1.1.

4.5.1 Related Quality Standards

A number of standards prescribe quality models suitable for software systems: IEEE 830:1998, ISO/IEC 9126-1 and ISO/IEC 25010. As argued below, dynamic evolution quality is vague and inadequate in these standards. The IEEE 830-1998 Standard, “IEEE Recommended Practice for Software Requirements Specifications” (IEEE Computer Society 1998), defines guidelines for documenting software requirements specifications. It provides a list of example software quality characteristics that can serve as requirements for an application. The only characteristic relevant to evolution is Maintainability which relates to the ease of maintenance of the application itself. This Standard, however, does not elaborate on what attributes characterise Maintainability.

The International Standard ISO/IEC 9126-1, “Software engineering -- Product quality -- Part 1: Quality model” (ISO/IEC 2001), defines a quality model for external and internal quality of software as a product. It categorises software quality attributes into six characteristics (functionality, reliability, usability, efficiency, maintainability and portability) which are further subdivided into thirty-four sub-characteristics.

A succession of ISO/IEC 9126-1 is the ISO/IEC 25000 series of standards, collectively titled “Software product Quality Requirements and Evaluation (SQuaRE)”, for the specification, measurement and evaluation of quality requirements for software products (ISO/IEC 2005). They are revised and restructured from two international standards: ISO/IEC 9126-1 and ISO/IEC 14598 (for software product evaluation). The revised quality model can be found under the International Standard ISO/IEC 25010 (ISO/IEC 2011) which defines eight quality characteristics (functional suitability, reliability, performance efficiency, operability, security, compatibility, maintainability and portability) and thirty-eight sub-characteristics.

To assess ISO/IEC 9126-1 and ISO/IEC 25010's extent of support for dynamic evolution, one way is to compare their quality characteristics with the dynamic evolution quality factors synthesised in this research. The comparison focuses on maintainability since it is the only characteristic in both these standards that is relevant to evolution. Table 4.10 shows the correspondence between sub-characteristics of maintainability and the dynamic evolution quality factors. Note that there are no dynamic evolution quality factors associated with Reusability, Analysability and Maintainability Compliance. An explanation for Reusability is that it is a design-time concern taken care of during conventional application development whereas dynamic evolution is a runtime concern for an application. Analysability does not directly map to any of the dynamic evolution quality factors. However, both "Locality" (i.e. keeping changes in one region to make them easier to understand and apply, cf. Section 4.1.2.2) and "Loose Coupling" (i.e. having less effect on other parts when updating one part, cf. 4.1.3.4) should improve Analysability. The case for Maintainability Compliance can be partially explained by the fact that no standards or conventions exist for dynamic evolution quality.

Table 4.10 Correspondence between ISO/IEC 9126-1 and ISO/IEC 25010's definitions for maintainability and the dynamic evolution quality model

<i>Maintainability Sub-Characteristic</i>	<i>ISO/IEC 9126-1 Description</i>	<i>ISO/IEC 25010 Description</i>	<i>Relevant Dynamic Evolution Quality Factor(s)</i>
Modularity	Undefined	The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.	Locality (Section 4.1.2.2) and Loose Coupling (Section 4.1.3.4)
Reusability	Undefined	The degree to which an asset can be used in more than one software system, or in building other assets.	None
Analysability	The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.	The ease with which the impact of an intended change on the rest of the software can be assessed, or the software product can be diagnosed for deficiencies or causes of failures in the software, or the parts to be modified to be identified.	Partially linked to Locality (Section 4.1.2.2) and Loose Coupling (Section 4.1.3.4)
Changeability	The capability of the software product to enable a specified modification to be implemented.	The degree to which the product enables a specified modification to be implemented. The ease with which a software product can be modified.	All quality factors under the category "Changeability of Application": Autonomy, Coordination, Extensibility, Loose Coupling and Separation of Concerns (Sections 4.1.3 and 4.3)

<i>Maintainability Sub-Characteristic</i>	<i>ISO/IEC 9126-1 Description</i>	<i>ISO/IEC 25010 Description</i>	<i>Relevant Dynamic Evolution Quality Factor(s)</i>
(Modification) Stability	The capability of the software product to avoid unexpected effects from modifications of the software.	The degree to which the product can avoid unexpected effects from modifications of the software.	Reliability (Section 4.1.4.3)
Testability	The capability of the software product to enable modified software to be validated.	The degree to which an objective and feasible test can be designed to determine whether a requirement is met.	Attribute “no reduction in testability” of “Maintainability” (Section 4.1.2.3)
Maintainability Compliance	The capability of the software product to adhere to standards or conventions relating to maintainability.	The degree to which the product adheres to standards or conventions relating to maintainability.	None

source: ISO/IEC (2001; 2011)

4.5.2 Related Quality Models

In this section, a number of software quality models are reviewed. In work related to evolution quality, Rowe et al. (1998) describe four kinds of change (changing existing design/implementation, component replacement, adding new components to an architecture and adding new functionality to an architecture) and correspondingly four architectural quality attributes (generality, adaptability, scalability and extensibility) to accommodate these changes. In contrast, the set of quality factors synthesised in this research emphasise the dynamic aspects of evolution and are independent of the types of changes.

For component-based applications, Mari and Eila (2003) distinguish execution from evolution quality attributes. The former expresses runtime observable quality characteristics of an application in execution while the latter expresses the quality of the static structure of an application to support static evolution. There is no analysis of whether these attributes relate to the dynamic aspect of evolution.

Deprez et al. (2007) investigated evolvability in a project context using free and open source software (F/OSS). They proposed a set of quality factors for evolvability from their surveys of practitioners, the scientific literature, the International Standard ISO/IEC 9126-1 and several quality assessment techniques for F/OSS. Their factors cover evolvability of the software, the team involved, the development processes followed and the tools used. With respect to software, their quality factors are maintainability, testability, adherence to standards, interoperability, readability and portability. Maintainability and testability are incorporated into “Maintainability” synthesised in this research (cf. Table 4.3). Readability (of code and documentation) is

a general characteristic of well-documented applications, not necessarily specific to dynamic evolution which is the focus of this research, and as such is not considered in this research. The argument for excluding interoperability, portability and adherence to standards from consideration in this research has been made in Sections 4.1.6 and 4.5.1 (adherence to standards discussed under “maintenance compliance” in Section 4.5.1).

Breivold and Crnkovic (2010) undertook a systematic literature review of when and where evolvability is dealt with in a software architecture during software development. They associate evolvability with adaptability, analysability, changeability, extensibility, flexibility, maintainability and modifiability. All but flexibility and modifiability are adopted from the International Standard ISO/IEC 9126-1 and Rowe et al.’s (1998) (discussed earlier in this section). The authors do not provide definitions for flexibility and modifiability.

In work towards software quality, Cavano and McCall (1978) proposed a framework, a.k.a. the McCall model, for measuring the quality of a software product on three aspects: product revision, product transition and product operations. Maintainability and flexibility, both under product revision, are the quality factors relevant to evolution. Maintainability focuses on fixing defects whereas “Maintainability” synthesised in this research (Section 4.1.2.3) focuses on changes in general. Flexibility relates to the “Changeability of Application” category synthesised in this research which is characterised by “Configurability”, “Coordination”, “Flexibility”, “Loose Coupling”, and “Separation of Concerns” (Section 4.1.3).

Dromey (1995) proposed a set of quality-carrying properties of software (e.g. variables initialised) which are linked to the quality characteristics from the ISO-9126-1 Standard. In this respect, four types of properties for variables and expressions in the source code are defined: correctness, structural, modularity and descriptive. This model targets quality analysis at the source code level and is not relevant to dynamic evolution.

The NFR framework lists over one hundred and sixty non-functional requirements but does not provide their definitions (Chung et al. 1999, pp. 159-160). An exception though is a small subset of requirements which deal with accuracy, security and performance requirements for software. Despite that, a small number of requirements, judged by their names alone, appear to be relevant to dynamic evolution: adaptability, enhanceability, extensibility, modifiability, reconfigurability, additivity, evolvability.

The FURPS+ model stands for “f”unctionality, “u”sability, “r”eliability, “p”erformance and

“supportability” (Grady 1992, p. 32). A number of quality factors in the supportability category, judged by their names alone, appear to relate to maintenance and/or evolution: testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, localizability (internationalization). FURPS+ suffers the same drawback as the NFR framework; no definitions are provided for its quality factors.

In summary, while the reviewed quality standards and models offer quality factors that could be useful to dynamic evolution, they do not adequately and specifically deal with dynamic evolution. They do not suggest any additional factors that should be considered for dynamic evolution, nor do they describe any tailoring approaches to make them suitable for dynamic evolution. For instance, they do not account for any of the synthesised dynamic evolution quality factors under the categories “Soundness of Change” (Completeness, Consistency, Correctness, Section 4.1.1) and “Robustness of Application” (Fault Tolerance, Recoverability, Reliability, Safety Security, Section 4.1.4).

4.6 CONCLUSION

In this Chapter, the outcomes of Task 1.1 of this research - viz. Synthesise, assess and extend important dynamic evolution quality factors - have been reported. This task adopted a multi-step approach to the incremental development of an extended set of quality factors with associated attributes suitable for dynamic evolution in composition-based distributed applications.

The initial set of factors was elicited from the literature and relevant evaluation frameworks, and subsequently assessed in a web survey. Survey feedback was used to extend the initial set with new factors. Then, the extended set was further extended via a review of selected development methodologies, as inputs for Phase 2. To demonstrate the use of the dynamic change requirements, the selected methodologies were evaluated with respect to their extent of support for these requirements. Correspondingly, features from these methodologies suitable for reuse or enhancement were identified for the development of Continuum in Phase 2. The set of quality factor requirements identified in this Task is summarised in Table 4.11.

Table 4.11 Dynamic evolution quality factor requirements investigated in Task 1.1

<i>Category</i>	<i>Quality factors</i>
Soundness of Change	Completeness, Consistency, Correctness
Infusibility of Change	Efficiency, Locality, Maintainability, Transparency
Changeability of Application	Autonomy, Coordination, Extensibility, Loose Coupling, Separation of Concerns
Robustness of Application	Fault Tolerance, Recoverability, Reliability, Safety, Security

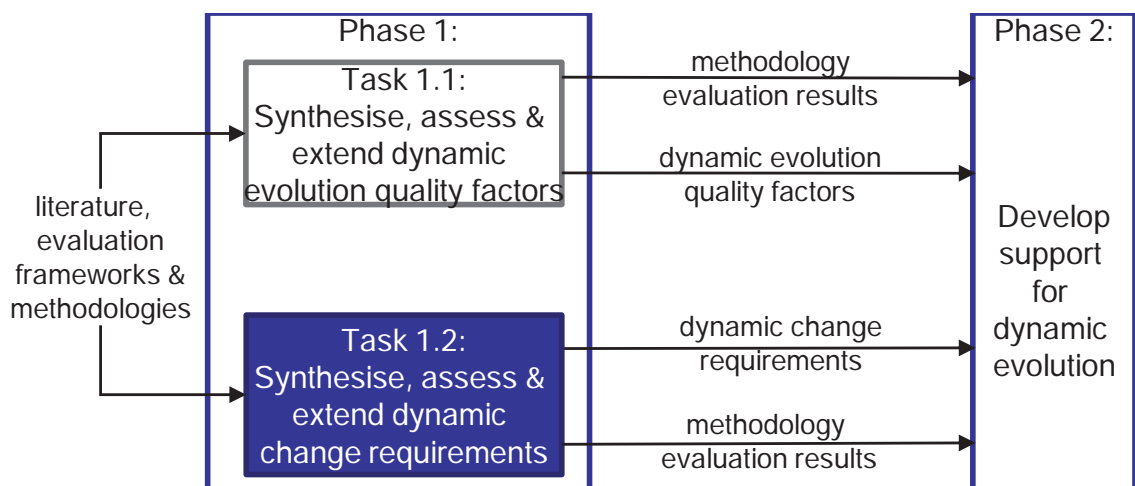
Notes:

1. Reliability and Safety were perceived by survey respondents as the most important in Step 2 (cf. Section 4.2.3).
2. Configurability and Flexibility are not included in the table above since they have been discarded in the expert review (cf. Section 4.2.3).

Chapter 5. DEVELOPMENT OF DYNAMIC CHANGE REQUIREMENTS

“Throughout the world, change is the order of the day.” - Mahatma Gandhi

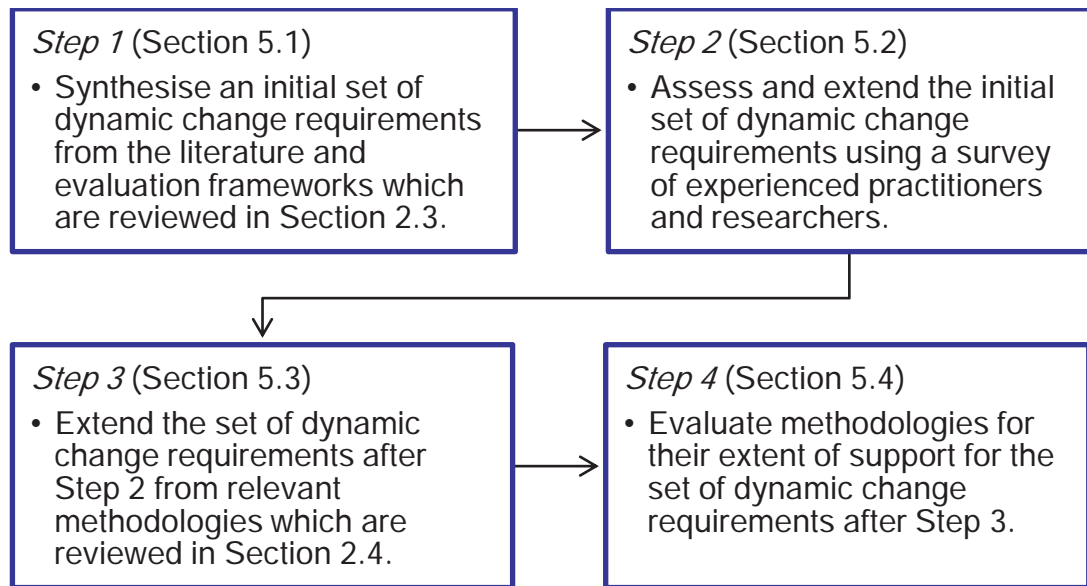
Chapter 4 reported Task 1.1 of Phase 1 from which a set of dynamic evolution quality factors was determined. This Chapter reports Task 1.2 of Phase 1, which aimed to develop dynamic change requirements suitable for composition-based distributed applications in the same way as Task 1.1. The outcomes comprise a generic set of dynamic change requirements and the results of evaluating methodologies for their extent of support for these requirements. The outcomes from Tasks 1.1 and 1.2 were used for the development of Continuum in Phase 2 (Chapter 6). Figure 5.1 shows the relationship between Task 1.2 and other tasks/phases in this research:



source: developed for this research

Figure 5.1 Information flow in Phase 1 for determining dynamic change requirements

As shown in Figure 5.2 Task 1.2 repeats the four steps in Task 1.1, but is tailored for dynamic change requirements (cf. Section 3.2.1). The first three steps concern the incremental development and assessment of the dynamic change requirements. Relevant methodologies were then evaluated in Step 4 for their support of the requirements. Respectively, Sections 5.1 to 5.4 describe the execution and outcomes of these four steps (see Figure 5.2). Section 5.5 concludes this Chapter.



source: developed for this research

Figure 5.2 Steps in Task 1.2 of Phase 1

5.1 STEP 1: SYNTHESIS OF DYNAMIC CHANGE REQUIREMENTS

The development of the initial set of dynamic change requirements followed the same systematic literature review approach as the one used for dynamic evolution quality factors in Task 1.1 (Section 4.1). More specifically, the objective of the review was to answer the following question:

What dynamic change requirements are considered in the literature and should be addressed during software development for composition-based distributed applications?

The corresponding review steps can be found in Appendix A. Table 5.1 shows the results of the search defined in the review steps. The search covered the same set of journals and conference proceedings published between 1994 and 2010 as those for the dynamic evolution quality factors. Of the two hundred and seventeen articles matching the search criteria, dynamic change requirements were identified and synthesised from sixty-eight of them.

Table 5.1 Source of literature examined for dynamic change requirement synthesis

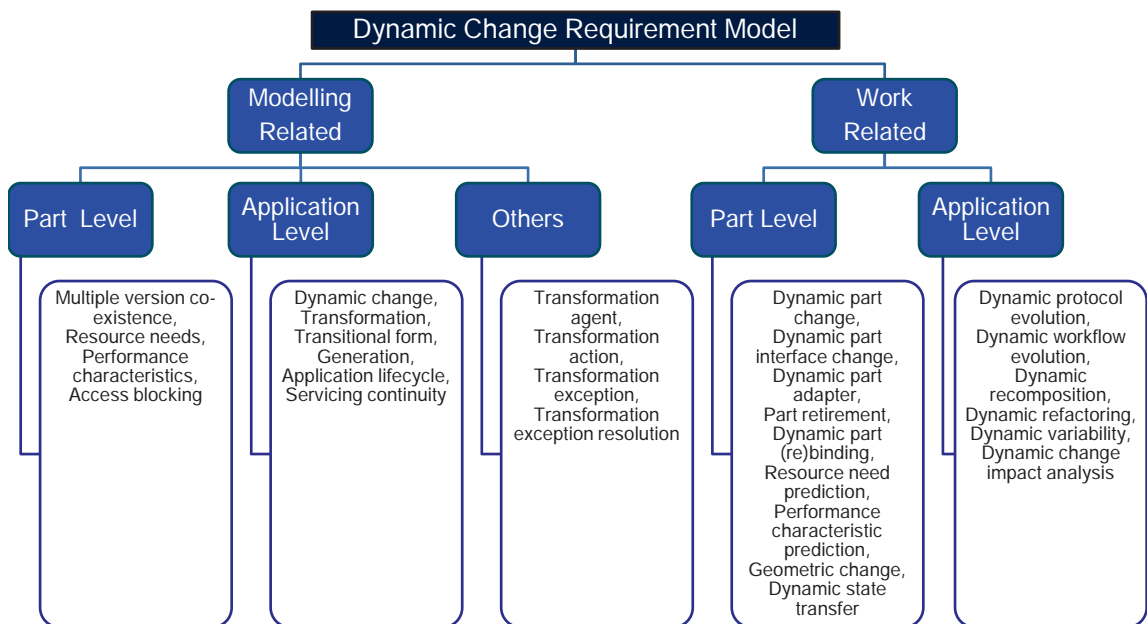
<i>Source</i>	<i>Title</i>	<i>Abbreviation</i>	<i>Articles examined</i>	<i>Articles with dynamic change requirement</i>
Journals (1994-2010)	Communications of the ACM	CACM	8	1
	ACM Transactions on Computer Systems	TOCS	1	0
	ACM Transactions on Software Engineering and Methodology	TOSEM	3	0
	European Journal of Information Systems	EJIS	0	0

<i>Source</i>	<i>Title</i>	<i>Abbreviation</i>	<i>Articles examined</i>	<i>Articles with dynamic change requirement</i>
	IEEE Computer		11	4
	IEEE Software		6	1
	IEEE Transactions on Software Engineering	TSE	10	4
	IET Software (formerly "IEE Proceedings Software" and "Software Engineering Journal")	IETS	12	7
	Information and Software Technology	IST	9	4
	Information Systems Journal	ISJ	0	0
	Information Systems Research	ISR	0	0
	Journal of Information Technology	JIT	0	0
	Journal of Software Maintenance and Evolution: Research and Practice (formerly "Journal of Software Maintenance")	JSME	8	5
	Journal of Systems and Software	JSS	7	3
	MIS Quarterly	MISQ	0	0
	Requirements Engineering		2	0
Conference Proceedings (1994-2010)	International Symposium on Component-Based Software Engineering (formerly "ICSE Workshop on Component-Based Software Engineering")	CBSE	14	2
	International Conference on Configurable Distributed Systems	IWCDS, ICCDS	26	3
	International Conference on Distributed Computing Systems	ICDCS	7	2
	International Conference on Service-Oriented Computing	ICSOC	22	3
	International Conference on Software Engineering	ICSE	16	5
	International Conference on Software Maintenance	ICSM	13	3
Others	Articles not in any of the publications and years above		42	21
Total			217	68

Note: The "others" category refers to articles examined from the bibliographies of the articles that were found to potentially offer dynamic change requirements.

A preview of the initial set of dynamic change requirements produced from the systematic literature review is given in Figure 5.3. The requirement hierarchy reflects the result of categorising the requirements as per the categorisation scheme specified in the research design (cf. Section 3.2.1). More specifically, the scheme categorises each requirement along two dimensions: methodology and application. The *methodology* dimension classifies dynamic change requirements into modelling and work related requirements from the methodological perspective. "Modelling related" dynamic change requirements concern the modelling concepts, notations and models for dynamic evolution used/produced during analysis and design. "Work related" dynamic change requirements concern what must be done to achieve given purposes specific to dynamic evolution during analysis and design. A piece of work can be a process to follow, a task to complete, or a technique to use. The *application* dimension further classifies dynamic change requirements with respect to their areas of concern in an application: "part level" for requirements related to individual parts in an application, "application level" for requirements related to an application as a whole, and "others" for

all other situations.



Source: developed from this research

Figure 5.3 Dynamic change requirements and categories synthesised from the literature

This set of dynamic change requirements thus synthesised is described next (Section 5.1.1 for modelling and Section 5.1.2 for work related requirements), with keywords in the requirements italicised. A review of existing evaluation frameworks (Section 2.3) was used to extend the set and the results are reported in Section 5.1.3. Section 5.1.4 summarises the initial set of dynamic change requirements thus synthesised.

5.1.1 Modelling Related Dynamic Change Requirements

Dynamic evolution can be described as a series of changes and transformations throughout the *lifecycle* of an application. A *dynamic change* (a.k.a. runtime change) (Kramer & Magee 1990; Oreizy et al. 1998) refers to an intended result of a modification to the application, whereas a *transformation* (a.k.a. dynamic configuration (Tsai et al. 2004)) refers to an act of performing dynamic modifications to the application (Mens & Demeyer 2008; Yen et al. 2008). A change can originate from the business needs (Brown & Wallnau 1998) or the environment in which the application operates (Chen et al. 2001). In a lifecycle, a series of *generations* exist, each referring to an application running a particular version of its code (Kruchten 2003).

Transformations are coordinated and performed by some agents (Almeida et al. 2001; Hall et al. 1999; Lovrek et al. 2003), referred to as *transformation agents* in this

research (e.g. Upgrade Manager, OMG 2003b), in small units of execution called *transformation actions* on a software architecture (Erradi et al. 2006b; Mens et al. 2010; Zhang et al. 2005) (cf. a subset of "Actions" in UML, OMG 2010b). During a transformation, certain parts affected by it may become unavailable temporarily and access to them should be *blocked* or *queued* (Hauptmann & Wasel 1996; Hillman & Warren 2004; Oreizy et al. 1998; Wang et al. 2006). Occasionally, a transformation *exception* (Jones et al. 2002; Segal 2002) might occur because an application may become unstable or erroneous after a change has been applied. A transformation exception *resolution*, such as a rollback to an older version to recover the application to a known state (Segal 2002), is sometimes performed to process and resolve errors caused by a transformation exception.

In reality, deploying a new release of an application almost never completes instantly but occurs in a finite time (Oreizy et al. 2008), during which one or more temporary forms of the application may occur. The term *transitional form*, originally used by Darwin (1859) for species evolution, is adopted to describe this kind of short and temporary form in the application (cf. "transition architecture", Zhang et al. 2009). For instance, when a web server is being upgraded, a temporary server is added to an architecture to intercept and service incoming web requests (Hillman & Warren 2004). As another example, when a service is being upgraded, both its old and new versions may co-exist temporarily (Li 2009). Figure 5.4 shows how a hypothetical application can be progressed from one generation to its successive one, applicable to both a small upgrade and a major release. In the former case, C is a newer revision of A. In the latter case, a release could be the replacement of a dial-up gateway (A) with an ISDN gateway (C) for an information system (B).

When a release is being deployed, a phenomenon referred to as *servicing continuity* in this research occurs (a.k.a. continuous availability (Oreizy et al. 1998; Oriol & Serugendo 2004)). It is an interesting state of applications which deterministically continue to provide their services in some capacity in the presence of transformations (Oreizy et al. 2008). Note that more than one transitional form is possible between two generations. Without specifying the exact transitional form that takes place during a release, an application's behaviour can become arbitrary and erratic. In the previous example, if the transitional form A-B-C in the top middle of Figure 5.4 occurs, the dial-up gateway is still available to the information system which can gradually redirect all its accesses to the dial-up gateway onto the ISDN gateway. In contrast, transitional form B means that the information system is no longer permitted to access the dial-up

gateway and the information system must treat this as a faulty condition.

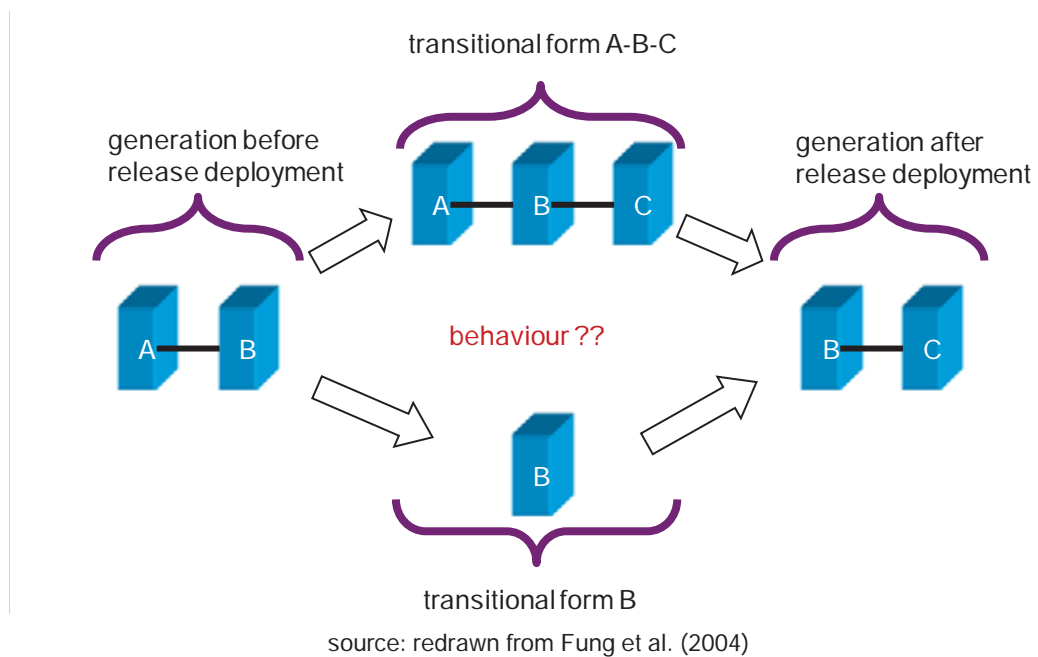


Figure 5.4 Example transitional forms

As an application evolves and with it its parts also evolve, many versions of its parts are built. Over time, multiple versions of the same artefact may *co-exist* in the application (Li 2009; McKinley et al. 2004), just as different parts of the application can use different versions of the same artefact (Evans & Dickman 1999). In another case, an older version can be used as a fallback option if a newer version fails to operate (Cook & Dage 1999).

5.1.2 Work Related Dynamic Change Requirements

This section describes the kinds of dynamic changes that occur in a composition-based distributed application, indicative of the changes that could be analysed and designed with a methodology. Dynamic evolution can occur at various levels of abstraction in a composition-based distributed application. At the fine-grained level, one can *add*, *replace* and *remove* individual parts (Aksit & Choukair 2003; de Paula et al. 2000; Heider et al. 2010; Hofmeister 1993; Karastoyanova et al. 2005; Koning et al. 2009; Kramer & Magee 1990; Kulkarni & Biyani 2004; Loulou et al. 2010; Parzyjegla et al. 2006; Taentzer et al. 2000) to change functionality (Bucchiarone et al. 2010; Kniesel 1999), as well as *modifying their parameters* (Chen et al. 2001). Fine-grained dynamic evolution also occurs at the interface level, such as *adding new interfaces* to an existing part, *removing interfaces* from an existing part (Jones et al. 2002), as well as *adding new operations* to (Wang et al. 2006) and *modifying existing operations* of an existing

part's interfaces (Aksit & Choukair 2003; Jones et al. 2002; Mens et al. 2010).

When adding a new part, an *adapter* may be required to accommodate the part into an application (Mens & Demeyer 2008; Yellin & Strom 1997) to resolve mismatches between the part and the application (Canal et al. 1999; Canal et al. 2008; Motahari Nezhad et al. 2007). Mismatches include functional, interface, interaction, protocol, semantic, deadlock, and those relating to quality-of-service (Canal et al. 2008; Motahari Nezhad et al. 2007; Pelliccione et al. 2008; Yellin & Strom 1997). When replacing an existing part with a new version, the state of the old version is sometimes *transferred* to the instance of the new version (Adamek & Plasil 2005; Ben-Shaul et al. 2001; Bloom & Day 1993; Gregersen & Jørgensen 2009; Gupta et al. 1996; Hauptmann & Wasel 1996; Lee & Chang 2005; Li 2009; Pelliccione et al. 2008; Plášil et al. 1998; Sun & Jiang 2009; Vandewoude et al. 2007). Other times, as a part is removed from an application, *retirement* tasks, such as relinquishing resources held by the part, might be performed to gracefully remove it from the application (Blake 2007; Chapin et al. 2001; Li 2009).

To support part addition, replacement and removal, *dynamic binding* is performed to (re)wire those parts in their application, such as to use the new and/or replacement parts, at runtime (Aksit & Choukair 2003; 2006; Karastoyanova et al. 2005; Kramer & Magee 1990; Lee & Chang 2005; Oreizy et al. 1998; Taentzer et al. 2000). In an SOA environment, dynamic binding between a service consumer and a service provider (Blake 2007; Curbera et al. 2003; Hu & Grefen 2003; Koning et al. 2009) occurs naturally just before a service is invoked and the binding reference can be dropped after service invocation. As an application evolves, its parts may relocate or move to different locations (Aksit & Choukair 2003; Ben-Shaul et al. 2001; Cugola et al. 2004; de Paula et al. 2000; Holder et al. 1999; Parzyjegla et al. 2006). Hofmeister (1993) refers to this as *geometric change*.

At the application level, an architecture can be designed with *variability* which means customisation points are defined in an architecture to facilitate a variety of parts with slight variations in their functionality (e.g. algorithms) to be customised and plugged into the architecture (Andersson & Bosch 2005; Heider et al. 2010; Kim et al. 2007; Koning et al. 2009; Siljee et al. 2005). This supports evolution to a certain degree by plugging parts with different/new functionality into the application.

Sometimes an application's structure and topology is *recomposed* (Bucchiarone et al. 2010; Chaudet et al. 2000; Koning et al. 2009; McKinley et al. 2004), say, to alter the application's behaviour or to recover it from errors. In the Unix operating system, pipes

are constructs that connect programs to create an executing process behaving differently from original programs. Merging and splitting (Kramer & Magee 1990) are special cases of recomposition in which smaller applications are combined into a larger one, and a subset of an application contributing to a particular functionality is sliced out and used elsewhere. Compared with recomposition on structural aspects, *workflow evolution* concerns alternating the sequence of interactions and dataflow among collaborating parts (Gil et al. 2007; Shrivastava & Wheeler 1998; Sun & Jiang 2009). In an SOA environment, a business process built by orchestrating services into a workflow at runtime can also evolve to satisfy changing business offerings (Ammon et al. 2010). Another form of interaction evolution is that the *protocol* used by parts to interact may also evolve, such as changing the messages exchanged, to suit new requirements (Hauptmann & Wasel 1996; Jones et al. 2002; Ryan & Wolf 2004). A complex form of application change is *refactoring* (Kataoka et al. 2001; Pelliccione et al. 2008) which involves changes that cross-cut many parts of its structure and topology.

At all levels of abstraction, *change impact analysis* can be performed to determine all the parts in an architecture impacted by a proposed change and actions to be taken to synchronise with the change (Bohner 1996; Zhao et al. 2002).

5.1.3 Existing Evaluation Frameworks

In this part of the requirement synthesis, existing evaluation frameworks (reviewed in Section 2.3) were examined to identify if any dynamic change requirements from these frameworks meet the same criteria outlined in the systematic literature review (cf. Appendix A), and can be incorporated into the initial set.

Of the evaluation frameworks reviewed, seven relevant and potential dynamic change requirements meeting the criteria were identified from Wood et al. (1988), HP Labs (Arnold et al. 1991) and Martinlassi (2004). Their original descriptions are documented in Table 5.2 with text relevant to changes underlined.

Table 5.2 Potential dynamic change requirements selected from reviewed evaluation frameworks

Framework and Type	Original Requirement
Martinlassi (2004), component product-line	How does the method support <u>variability expression</u> ?
Wood et al. (1988), real-time	Do the representations help maintainers determine the <u>scope of effect of a proposed change</u> to a particular module or set of modules?
	Does the method provide techniques for organizing its representations to support the evolution of the system into <u>multiple versions</u> over time?
	Can designers use the method to <u>predict the resource requirements and performance characteristics</u> of an evolving design?

Framework and Type	Original Requirement
	Do the method's design representations <u>capture resource estimates</u> for individual components?
	Do the same representations <u>capture performance estimates</u> for these components?
Arnold et al. (1991), object-oriented	Does the process provide support for <u>adding functionality</u> to existing systems ...? [This is considered a special case of dynamic change.]

5.1.4 Summary and Discussion

Table 5.3 and Table 5.4 document the classified modelling and work related set of dynamic change requirements respectively, citing reference(s) to the literature and/or evaluation framework(s) from which each requirement was derived. Note that in these tables the three sub-columns ("generic", "component-based" and "SOA-based") under the "Literature" column classify the genres of the cited references from which the corresponding requirements were derived.

Table 5.3 Modelling related dynamic change requirements from Step 1

Requirement	Description	Literature			Evaluation Framework
		Generic	Component-based	SOA-based	
Part Level					
Multiple version coexistence	Model the ability of a service to have <i>multiple versions</i> present in various regions of an application as the service evolves over time.	Evans and Dickman (1999)	Cook and Dage (1999), Li (2009), McKinley et al. (2004)		Wood et al. (1988).
Resource needs	Model the changing <i>resource needs</i> of a service as it evolves.				Wood et al. (1988).
Performance characteristics	Model the changing <i>performance characteristics</i> of a service as it evolves.				Wood et al. (1988).
Access blocking	Model the ability to <i>block access</i> to a service while it is being updated or replaced.	Hauptmann and Wasel (1996)	Hillman and Warren (2004), Oreizy et al. (1998), Wang et al. (2006)		
Application Level					
Dynamic change	Model the notion of a <i>runtime change</i> which is the <i>intended result</i> of a modification to an application at runtime.	Kramer and Magee (1990)	Oreizy et al. (1998)		HP Labs (Arnold et al. 1991).
Transformation	Model the notion of a <i>transformation</i> which is an act of performing modifications to an application at runtime.	Mens and Demeyer (2008)		Tsai et al. (2004), Yen et al. (2008)	
Transitional form	Model the presence of a <i>transitional form</i> between two generations as an application progressively evolves from one generation to the next in a finite amount of time.		Hillman and Warren (2004), Zhang et al. (2009)		
Generation	Model the notion of a <i>generation</i> which represents a stable version of an application at a particular time.				Wood et al. (1988).

Requirement	Description	Literature			Evaluation Framework
		Generic	Component-based	SOA-based	
Application lifecycle	Model the notion of an <i>application lifecycle</i> which organises application evolution as a series of generations over time.				Wood et al. (1988).
Servicing continuity	Model the ability of an application to <i>continuously offer some functionality</i> during a transformation.	Oreizy et al. (2008)	Oreizy et al. (1998), Oriol and Serugendo (2004)		
<i>Others</i>					
Transformation agent	Model the notion of a <i>transformation agent</i> responsible for performing transformations.	Hall et al. (1999), Lovrek et al. (2003)	Almeida et al. (2001)		
Transformation action	Model the <i>steps</i> undertaken in a transformation, e.g. adding a new service, followed by configuring the service.		Mens et al. (2010), Zhang et al. (2005)	Erradi et al. (2006b)	
Transformation exception	Model the notion of an <i>error condition</i> occurred during a transformation, e.g. because of faulty changes.	Jones et al. (2002), Segal (2002)			
Transformation exception resolution	Model the notion of an <i>exception resolution</i> - e.g. a rollback - to revert errors caused by aborted or faulty changes.	Segal (2002)			

Table 5.4 Work related dynamic change requirements from Step 1

Requirement	Description	Literature			Evaluation Framework
		Generic	Component-based	SOA-based	
Part Level					
Dynamic part change	Define support to <i>add, replace, remove individual parts</i> , and <i>modify their parameters</i> at runtime.	Hofmeister (1993), Kramer and Magee (1990), Parzyjegl et al. (2006), Taentzer et al. (2000)	Aksit and Choukair (2003), Chen et al. (2001), de Paula et al. (2000), Heider et al. (2010), Kniesel (1999), Kulkarni and Biyani (2004), Loulou et al. (2010)	Bucchiarone et al. (2010), Koning et al. (2009), Karastoyanova et al. (2005)	
+ Dynamic part interface change	Define support to <i>add, remove and modify part interfaces</i> at runtime ("Modify part interfaces" means add, remove and modify operations of interfaces).	Jones et al. (2002)	Aksit and Choukair (2003), Mens et al. (2010), Wang et al. (2006)		
Dynamic part adapter	Define <i>adapters</i> for wrapping and plugging parts into an architecture at runtime, and resolving their mismatches.	Mens and Demeyer (2008)	Canal et al. (1999), Canal et al. (2008), Yellin and Strom (1997), Pelliccione et al. (2008)	Motahari Nezhad et al. (2007)	
Part retirement	Define housekeeping support to <i>retire parts</i> after they are no longer needed and have been removed from an application.	Chapin et al. (2001)	Li (2009)	Blake (2007)	

Requirement	Description	Literature			Evaluation Framework
		Generic	Component-based	SOA-based	
Dynamic part (re)binding	Define support to <i>(re)bind one part to a new part at runtime</i> as the original part is replaced or upgraded by the new part.	Kramer and Magee (1990), Taentzer et al. (2000)	Aksit and Choukair (2003), Oreizy et al. (1998), Lee and Chang (2005)	Blake (2007), Curbera et al. (2003), Hu and Grefen (2003), Karastoyanova et al. (2005), Koning et al. (2009)	
Resource need prediction	Define support to <i>predict the resource needs</i> for a new part.				Wood et al. (1988).
Performance characteristic prediction	Define support to <i>predict the performance characteristics</i> for a new part.				Wood et al. (1988).
Geometric change	Define support to <i>relocate parts</i> to a different hosting environment as needed.	Cugola et al. (2004), Hofmeister (1993), Parzyjegl et al. (2006)	Aksit and Choukair (2003), Ben-Shaul et al. (2001), de Paula et al. (2000), Holder et al. (1999)		
Dynamic state transfer	Define support to <i>transfer the state</i> from an instance of an old implementation of a part/workflow to an instance of the new implementation of the part/workflow as it evolves.	Bloom and Day (1993), Gupta et al. (1996), Sun & Jiang (2009), Vandewoude et al. (2007)	Adamek and Plasil (2005), Ben-Shaul et al. (2001), Gregersen and Jørgensen (2009), Hauptmann and Wasel (1996), Lee and Chang (2005), Li (2009), Pelliccione et al. (2008), Plášil et al. (1998)		
<i>Application Level</i>					
Dynamic protocol evolution	Define support to <i>evolve a protocol definition</i> on which parts base their interactions while the protocol is being used.	Jones et al. (2002), Ryan and Wolf (2004)	Hauptmann and Wasel (1996)		
Dynamic workflow evolution	Define support to <i>evolve a workflow definition</i> while the workflow is operational.	Gil et al. (2007), Sun & Jiang (2009)	Shrivastava and Wheeler (1998)	Ammon et al. (2010)	
Dynamic recomposition	Define support to <i>combine</i> several parts/workflows into a larger unit, <i>split</i> a larger part/workflow into smaller units or <i>reconfigure</i> the architectural structure at runtime.	Kramer and Magee (1990)	Chaudet et al. (2000), McKinley et al. (2004)	Bucchiarone et al. (2010), Koning et al. (2009),	
Dynamic refactoring	Define support to <i>refactor</i> an application structure without functional changes at runtime, say, to reduce its complexity and improve its performance.	Kataoka et al. (2001)	Pelliccione et al. (2008)		
Dynamic variability	Define customisation points in an architecture to plug in or swap different parts at runtime to support limited <i>variations</i> in functionality (e.g. using credit card vs. account debit for payment).	Andersson and Bosch (2005)	Heider et al. (2010), Kim et al. (2007)	Siljee et al. (2005), Koning et al. (2009)	Matinlassi (2004)

Requirement	Description	Literature			Evaluation Framework
		Generic	Component-based	SOA-based	
Dynamic change impact analysis	Analyse the <i>impact of dynamic changes</i> to an application to determine which parts/workflows of the application will need to be affected and updated to accommodate the changes.	Bohner (1996)	Zhao et al. (2002)		Wood et al. (1988).

Note: Any requirements labelled with “+” were not included in the survey since they were identified after the survey commented in late 2006 (cf. Section 5.2). Instead, they were reviewed by experts (cf. Section 5.2.3).

Where a dynamic change requirement was specific to a particular type of composition-based application, it was regarded as not sufficiently generic for inclusion since the intention was that the *generic* set of dynamic change requirements should be applicable to a variety of composition-based applications. Examples of dynamic change requirements excluded are service discovery (Cervantes & Hall 2005) and service selection (Huhns & Singh 2005) in the SOA paradigm. They allow service consumers to look up a variety of suitable service providers on demand, and then choose a service provider most suitable for their needs at runtime as the providers evolve (Akram et al. 2003). In another case, service policies governing the intended use of services may need to change to suit the business needs or as services are updated (Gu & Lago 2007; Tsai et al. 2005).

5.2 STEP 2: SURVEY ASSESSMENT

This step aimed to assess and extend the set of dynamic change requirements synthesised in Step 1 using a questionnaire survey of experienced software development practitioners and researchers. The questionnaire asked each respondent to provide the following information:

- *Ratings on modelling and work related requirements*

Each respondent was asked to rate the level of importance of each requirement on a Likert scale (1 for “not-important-at-all”, 5 for “extremely-important”). As the survey was commenced from late 2006, only requirements identified up to this point were included in the survey. The ratings were used to assess the perceived importance of the requirements (see Section 5.2.4).

- *Additional comments*

Each respondent was also asked to provide feedback/comments on the rating questions in the survey, such as suggestions for additional requirements for

consideration.

Follow-up interviews via email correspondence and/or face-to-face meetings were conducted with consenting respondents to clarify response data and any feedback/comment raised. Any additional requirements were then subject to an expert review (see Section 5.2.3).

5.2.1 Pilot Tests

The questionnaire was pilot tested by two research peers, one experienced in distributed system development and the other in SOA development. The completion time was about 15-20 minutes which was noted in the final version of the questionnaire. Feedback and comments on content, structure, wording and clarity were reviewed and the content was revised accordingly to improve the quality of the questionnaire. The final version can be found in Appendix E.2.

5.2.2 Data Collection

The questionnaire as an electronic document was emailed to practitioners from the IT industry and research scientists from academia to complete. This recruitment approach invited contacts known to the researcher to ensure that candidates have adequate experience and/or knowledge in SOA related technologies. This approach, a.k.a. *purposeful sampling* (Patton 2002), improves the credibility and authenticity of the questionnaire responses collected at the expense of introducing selection bias into the responses. Towards the end of the survey, a friendly reminder was emailed to invited candidates who had not completed and returned the questionnaire.

Over fifty-five candidates were invited to the survey and a total of thirty-six valid responses were returned. All respondents worked in Australia. Nine respondents were conducting SOA related research in academia whilst the rest worked in the IT industry using SOA related technologies. All responses were screened to ensure they had sufficient experience in software development methodologies (minimum of one year), application development (minimum of three years), and distributed application development/research in relation to Web services (minimum of two years) to contribute to this survey.

5.2.3 Additional Dynamic Change Requirements

Respondents were asked to suggest additional dynamic change requirements that should be included in the set. The same two-step approach used for selecting quality

factors/attributes (Section 4.2.3) was repeated for these requirements. The first step qualified four potential requirements, shown in Table 5.5, using the same criteria as those defined in the systematic literature review (cf. Appendix A). The expert review in the second step reduced the set to three requirements as shown in Table 5.5.

Table 5.5 Analysis results of potential dynamic change requirements suggested by respondents

<i>Suggested dynamic change requirement</i>	<i>Expert 1's Comment</i>	<i>Expert 2's Comment</i>	<i>Inclusion to requirement set</i>
Modelling the notion of an expected dynamic change impact (e.g. scope and extent).	<i>should-be-considered</i> (Very nice to have)	<i>should-be-considered</i> (Very important. Some analysis will identify characteristics of impact)	Yes
Modelling the notion of deprecation in a part's lifecycle (i.e. one part being superseded by another part or version). A rationale is that deprecated parts may still be present in a composition-based distributed application after they have been updated.	<i>should-be-considered</i> (Very important, similar to deprecation in Java programming language)	<i>should-be-excluded</i> (Deprecation important but not a key issue in dynamic evolution, better addressed in versioning)	No
Define support for expected impact of dynamic changes on other applications.	<i>should-be-considered</i> (Very important with respect to external impact)	<i>should-be-considered</i> (Extremely important to parts/application in proximity)	Yes
Contract management: updating of contracts between parts as the parts evolve.	<i>should-be-considered</i> (It must be supported for dynamic evolution)	<i>should-be-considered</i> (Contracts between parts require update as changes to parts happen)	Yes

The same two-step review approach was repeated for additional dynamic change requirements found from articles and evaluation frameworks that were published after the survey commenced in late 2006 (cf. Table 5.3 and Table 5.4). The results are documented in Table 5.6.

Table 5.6 Analysis results of additional dynamic change requirements synthesised from the literature (cf. Table 5.4)

<i>New dynamic change requirement</i>	<i>Expert 1's Comment</i>	<i>Expert 2's Comment</i>	<i>Inclusion to the requirement set</i>
Dynamic part interface change - Define support to add, remove and modify part interfaces at runtime ("modify part interfaces" means add, remove and modify operations of interfaces).	<i>should-be-excluded</i> (Only a nice to have. But it should be accompanied by support for notification of changes to the parts clients.)	<i>should-be-considered</i> (Common to allow service end point evolution)	No

The expert review led to the following revisions to the set of dynamic change requirements after Step 1 (in *italic* font). These changes originated from the respondents' suggestions (cf. Table 5.5). The revised set of requirements is

summarised in Table 5.12:

- *Expected dynamic change impact* (new modelling related requirement)
Model the impact of dynamic changes (scope and extent) expected on parts within the application and other applications affected by the changes.
- *Dynamic change impact analysis* (revision to existing work related requirement)
Analyse the impact of dynamic changes to an application to determine which parts/workflows of the application, *those parts/workflows outside the application, and other applications*, are to be affected and updated to accommodate the changes.
- *Dynamic contract update* (new work related requirement)
Support the update of contracts among parts as the parts/contracts evolve.

Dynamic change requirements rejected by experts in this step are excluded from further consideration in the revised set even if they are identified at a later stage in the analysis (i.e. extension of requirements from methodologies in Step 3, Section 5.3).

5.2.4 Importance Ratings of Dynamic Change Requirements

A Wilcoxon one-sample signed-rank test (Arnold 1965; Wilcoxon 1945) was applied to the data to check if the importance ratings of the requirements were from a distribution away from a known median of 3 being the mid-point between “not-important-at-all” (1) and “extremely-important” (5). Note that the analysis was limited to the requirements synthesised from articles and evaluation frameworks published *before late 2006* (cf. Table 5.3 and Table 5.4). The signed-rank test results together with median and range scores are shown in Table 5.7 and Table 5.8. Of the modelling related dynamic change requirements (Table 5.7), “transitional form” was not perceived to be important ($p_{wsr}=0.19$, highlighted). Of the work related dynamic change requirements, neither “dynamic protocol evolution” nor “part retirement” was regarded as significant ($p_{wsr}=0.38$, 0.11, highlighted).

Table 5.7 Descriptive statistics and Wilcoxon one-sample signed-rank test results for modelling related dynamic change requirements from Step 1 (cf. Table 5.3)

Requirement	Median	Range	S+	S-	p_{wsr}
<i>Part level</i>					
Multiple version coexistence	4	1-5	358	107	0.00
Resource needs	4	2-5	265	60	0.00
Performance characteristics	4	2-5	288	63	0.00
Access blocking	3.5	1-5	208	45	0.00

<i>Requirement</i>	<i>Median</i>	<i>Range</i>	<i>S+</i>	<i>S-</i>	<i>p_{wsr}</i>
<i>Application level</i>					
Dynamic change	4	2-5	338	40	0.00
Transformation	4	2-5	331	48	0.00
<i>Transitional form</i>	3	1-5	208	143	0.19
Generation	4	2-5	319	32	0.00
Application lifecycle	4	2-5	301	50	0.00
Servicing continuity	3	1-5	262	90	0.01
<i>Others</i>					
Transformation agent	3	1-5	233	93	0.02
Transformation action	4	1-5	329	77	0.00
Transformation exception	4	1-5	457	40	0.00
Transformation exception resolution	4	1-5	537	25	0.00

Table 5.8 Descriptive statistics and Wilcoxon one-sample signed-rank test results for work related dynamic change requirements from Step 1 (cf. Table 5.4)

<i>Requirement</i>	<i>Median</i>	<i>Range</i>	<i>S+</i>	<i>S-</i>	<i>p_{wsr}</i>
<i>Part level</i>					
Dynamic part change	4	2-5	471	25	0.00
Dynamic part adapter	4	2-5	328	50	0.00
<i>Part retirement</i>	3	1-5	149	83	0.11
Dynamic part (re)binding	4	3-5	406	0	0.00
Resource need prediction	3	1-5	160	50	0.02
Performance characteristic prediction	3	1-5	183	48	0.01
Geometric change	3	2-5	133	57	0.04
Dynamic state transfer	4	2-5	332	46	0.00
<i>Application level</i>					
<i>Dynamic protocol evolution</i>	3	1-5	148	129	0.38
Dynamic workflow evolution	4	2-5	244	32	0.00
Dynamic recomposition	4	1-5	361	75	0.00
Dynamic refactoring	3	1-5	191	41	0.00
Dynamic variability	4	2-5	332	19	0.00
Dynamic change impact analysis	4	2-5	458	7	0.00

Requirements perceived to be more important were identified by examining the ranking order of the requirements and the grouping of similarly ranked requirements that were not different among themselves. The order is indicative of how important one requirement is compared with another, and was determined by calculating and ordering the mean rank scores of the requirements. The grouping of requirements was then determined by an analysis of difference among the requirements. A Friedman test (Friedman 1937) was performed to check if there was a difference between any two requirements ($p < 0.05$). If a difference was detected, a Wilcoxon matched-pair signed-rank test (Wilcoxon 1945) was performed to identify pairs of requirements with significant differences. The Wilcoxon test results, together with the mean rank scores of the perceived importance of the requirements, are shown in Table 5.9 and Table 5.10. For convenience, pairs with significant difference ($p < 0.05$) are highlighted in their corresponding cells (e.g. “transitional form” vs. “transformation exception resolution” in Table 5.9). For modelling related dynamic change requirements, the analysis did not suggest any grouping of dynamic change requirements as being significantly different from the remaining requirements. Nevertheless, based on the mean rank scores alone,

the requirements that relate to the error condition of a transformation - viz. “transformation exception resolution” and “transformation exception” - have the highest ranks (Table 5.9) and the highest mean values (Table 5.7). The relatively high ranking of these requirements may suggest that respondents were most concerned about the success of transformations, with one respondent commenting in a follow-up interview, “dynamic evolution must work or, at least, recover when it doesn’t”. In contrast, “transitional form” is the lowest ranked requirement (Table 5.9) and was found to be not significant based on the Wilcoxon one-sample signed-rank test results ($p_{\text{wsr}}=0.19$, Table 5.7). The fact that tools can be used to compute and take care of the intermediate transitional forms, making them transparent to end-users, may help explain this low ranking.

Table 5.9 Results of Wilcoxon signed-rank test for matched pairs on modelling related dynamic change requirements from Step 1 (cf. Table 5.3)

		Transformation exception resolution	Transformation exception	Dynamic change	Transformation	Generation	Application lifecycle	Transformation action	Performance characteristics	Access blocking	Multiple version coexistence	Servicing continuity	Resource needs	Transformation agent	Transitional form
	Mean Rank	Significance Level (p)													
Transformation exception resolution	10.4		0.07	0.01	0.01	0.02	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Transformation exception	9.4	0.07		0.13	0.06	0.23	0.10	0.02	0.01	0.00	0.03	0.02	0.01	0.00	0.00
Dynamic change	8.1	0.01	0.13		0.85	0.81	0.54	0.45	0.30	0.27	0.44	0.19	0.19	0.06	0.00
Transformation	8.0	0.01	0.06	0.85		0.78	0.64	0.63	0.55	0.39	0.51	0.20	0.21	0.09	0.00
Generation	7.9	0.02	0.23	0.81	0.78		0.47	0.45	0.18	0.25	0.27	0.10	0.12	0.07	0.00
Application lifecycle	7.8	0.01	0.10	0.54	0.64	0.47		0.92	0.76	0.61	0.73	0.52	0.56	0.24	0.05
Transformation action	7.4	0.00	0.02	0.45	0.63	0.45	0.92		0.66	0.49	0.76	0.44	0.55	0.10	0.04
Performance characteristics	7.0	0.00	0.01	0.30	0.55	0.18	0.76	0.66		0.72	0.95	0.63	0.99	0.21	0.04
Access blocking	7.0	0.00	0.00	0.27	0.39	0.25	0.61	0.49	0.72		0.83	0.78	0.94	0.56	0.11
Multiple version coexistence	6.9	0.00	0.03	0.44	0.51	0.27	0.73	0.76	0.95	0.83		0.78	0.81	0.47	0.11
Servicing continuity	6.8	0.00	0.02	0.19	0.20	0.10	0.52	0.44	0.63	0.78	0.78		0.60	0.76	0.24
Resource needs	6.8	0.00	0.01	0.19	0.21	0.12	0.56	0.55	0.99	0.94	0.81	0.60		0.48	0.10
Transformation agent	6.2	0.00	0.00	0.06	0.09	0.07	0.24	0.10	0.21	0.56	0.47	0.76	0.48		0.26
Transitional form	5.5	0.00	0.00	0.00	0.00	0.00	0.05	0.04	0.04	0.11	0.11	0.24	0.10	0.26	

In a similar analysis with work related dynamic change requirements, no distinct grouping of requirements was identified from Table 5.10. Based on the mean rank scores from Table 5.10, however, the top ranked requirements are “dynamic change impact analysis”, “dynamic part change” and “dynamic part (re)binding”. These requirements tend to require considerable attention in composition-based application development.

For instance, “dynamic change impact analysis” is essential in development as the impact of a dynamic change may ripple through many parts of a composition-based application, leading to unexpected results if additional parts also affected by the proposed change are identified during the analysis but not accommodated by the change. “Dynamic part change” and “dynamic part (re)binding” occur frequently owing to the way a composition-based application is structured - built with parts and bindings - to embrace dynamic evolution.

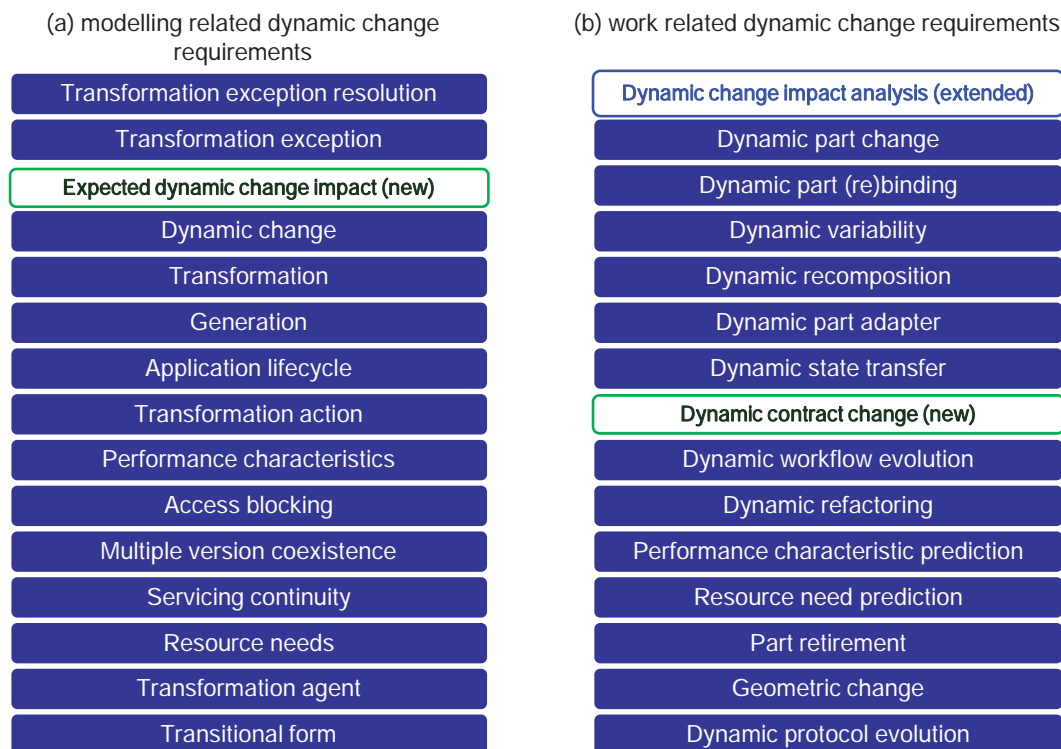
Table 5.10 Results of Wilcoxon signed-rank test for matched pairs on work related dynamic change requirements from Step 1 (cf. Table 5.4)

		Dynamic change impact analysis	Dynamic part change	Dynamic part (re)binding	Dynamic variability	Dynamic recomposition	Dynamic part adapter	Dynamic state transfer	Dynamic workflow evolution	Dynamic refactoring	Performance characteristic prediction	Resource need prediction	Part retirement	Geometric change	Dynamic protocol evolution
	Mean Rank	Significance Level (p)													
Dynamic change impact analysis	10.2		0.13	0.13	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dynamic part change	9.4	0.13		0.95	0.50	0.09	0.10	0.08	0.07	0.01	0.03	0.01	0.00	0.00	0.00
Dynamic part (re)binding	8.8	0.13	0.95		0.42	0.15	0.12	0.05	0.06	0.02	0.01	0.00	0.00	0.00	0.00
Dynamic variability	8.5	0.03	0.50	0.42		0.47	0.47	0.31	0.22	0.08	0.09	0.04	0.01	0.00	0.00
Dynamic recomposition	8.3	0.01	0.09	0.15	0.47		0.97	0.68	0.80	0.19	0.24	0.12	0.04	0.02	0.01
Dynamic part adapter	8.1	0.00	0.10	0.12	0.47	0.97		0.99	0.79	0.23	0.25	0.09	0.02	0.02	0.01
Dynamic state transfer	7.8	0.00	0.08	0.05	0.31	0.68	0.99		0.97	0.50	0.26	0.12	0.04	0.01	0.01
Dynamic workflow evolution	7.7	0.00	0.07	0.06	0.22	0.80	0.79	0.97		0.48	0.33	0.13	0.03	0.04	0.01
Dynamic refactoring	7.0	0.00	0.01	0.02	0.08	0.19	0.23	0.50	0.48		0.56	0.35	0.11	0.11	0.08
Performance characteristic prediction	6.5	0.00	0.03	0.01	0.09	0.24	0.25	0.26	0.33	0.56		0.43	0.36	0.31	0.15
Resource need prediction	6.0	0.00	0.01	0.00	0.04	0.12	0.09	0.12	0.13	0.35	0.43		0.61	0.54	0.21
Part retirement	5.8	0.00	0.00	0.00	0.01	0.04	0.02	0.04	0.03	0.11	0.36	0.61		0.84	0.60
Geometric change	5.6	0.00	0.00	0.00	0.00	0.02	0.02	0.01	0.04	0.11	0.31	0.54	0.84		0.54
Dynamic protocol evolution	5.3	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.08	0.15	0.21	0.60	0.54	

The three lowest ranked requirements, viz. “part retirement”, “geometric change” and “dynamic protocol evolution” (Table 5.10), occur relatively infrequently in a composition-based platform and this may explain why they were perceived as relatively less important. “Part retirement” and “dynamic protocol evolution” were also found to be not significant based on the Wilcoxon one-sample signed-rank test results ($p_{\text{wsr}}=0.11$ and 0.38, Table 5.8).

As reported in Section 5.2.3, there are extensions to the set of dynamic change

requirements rated by respondents in the survey. To assess their importance, the same expert review procedure as the one for additional quality factors (cf. Section 4.2.4) was used to incorporate these extensions into the importance rankings (cf. Table 5.9 and Table 5.10). Figure 5.5 shows the revised ranking orders for (a) modelling related, and (b) work related dynamic change requirements.



source: developed for this research

Figure 5.5 Importance rankings of dynamic change requirements after expert review

With respect to the modelling related requirements, the new entry “expected dynamic change impact” was ranked third. The experts commented that this requirement should be extremely important as in the work related requirement ranking order (“dynamic change impact analysis (extended)” in Figure 5.5(b)). Both “transformation exception” and “transformation exception resolution” still remained the most important and should be given the top priority for dynamic evolution.

With respect to the work related requirements, “dynamic change impact analysis” after refinement from the survey feedback still remains at the top spot. The new entry “dynamic contract change” was ranked just below “dynamic state transfer”. The experts explained that “dynamic contract change” is less common than “dynamic state transfer” but more common than “dynamic workflow evolution”.

5.3 STEP 3: DYNAMIC CHANGE REQUIREMENTS EXTENSION FROM METHODOLOGICAL PERSPECTIVE

In this Step, the same set of methodologies (listed in Section 2.4) that were reviewed for dynamic evolution quality factor identification in Task 1.1 (cf. Section 4.3) were also reviewed for dynamic change requirements. Then, the identified dynamic change requirements were used to extend (if necessary) the dynamic change requirement set after Step 2, by incorporating any additional requirements into the set.

Based on the same selection criteria for dynamic change requirements specified in Appendix A, requirements were identified from a number of methodologies (Table 5.11). Each identified requirement was then checked to see if it mapped to one or more requirements from the set of requirements after Step 2. As shown in Table 5.11, all requirements identified from the methodologies correspond to requirements in the set after Step 2 and hence no modification to the set is required.

Table 5.11 Analysis of potential dynamic change requirements elicited from reviewed methodologies

Methodology				Mapping to requirement as revised from Step 2
Name	Concern	Description	Relevant Literature	
AEM		Changes can include the <u>addition, removal, or replacement</u> of components and connectors.	Oreizy et al. (1999)	Work related: part level: dynamic part change
AEM		Changes can include ... <u>modifications to the configuration or parameters</u> of components and connectors.	Oreizy et al. (1999)	Work related: part level: dynamic part change
ASG	variation of service composition	Another aspect arises if service compositions are provided to several internal or external requestors. Often, they require slightly different functionality. To serve all of them optimally, different <u>variants of each service composition</u> are required.	Fahringer et al. (2007)	Work related: application level: dynamic variability
KobRA	variability	... rather than assemble every system in the family from scratch, it makes sense to build so-called 'frameworks', which 'hard-wire' the common aspects of the family, and allow the variable components to be ' <u>plugged in</u> ' as and when needed.	Atkinson et al. (2000)	Work related: application level: dynamic variability
SeCSE	rebinding	If the chosen candidate [i.e. one being used] has to be replaced (due to an error or a QoS decrease, for instance), the re-binding process will choose a new one from the alternatives list ... and will <u>modify the composition to bind</u> the new service.	SeCSE (2006)	Work related: part level: dynamic part (re)binding

<i>Methodology</i>				<i>Mapping to requirement as revised from Step 2</i>
<i>Name</i>	<i>Concern</i>	<i>Description</i>	<i>Relevant Literature</i>	
SeCSE	service composition replanning	When a service which participates in the composition fails, in the case there is no possibility to replace with other single service, there is the need to <u>modify the composition</u> in such a way the lost functionality can be recovered. The functionality may be split ... or may be <u>regrouped</u> with other part of the composition ...	SeCSE (2006)	Work related: application level: dynamic recomposition
SeCSE	variability point or variation point	With the variability points approach, we can define the places of the composition where there will be different alternatives to execute an abstract activity ... Depending on the context when the composition requested and the conditions defined for each variability point, one option will be chosen and executed. The <u>variances may be defined as different features available or as service realisation alternatives at runtime</u> .	SeCSE (2006)	Work related: application level: dynamic variability

Note: Underlined text highlights the relevance of each requirement with changes, maintenance and/or evolution as argued by respective methodologies.

5.4 STEP 4: EVALUATION OF SUPPORT FOR DYNAMIC CHANGE REQUIREMENTS IN METHODOLOGIES

After the incremental development of the set of dynamic change requirements in the previous steps, a feature analysis of existing methodologies (Section 2.3) was performed using the requirement set after Step 3, to determine what features from existing methodologies could be reused, what features could be enhanced for use and any additional support required. For consistency, the same scale points (Table Appendix B.1) used for evaluating the methodologies for their support for dynamic evolution quality factors (cf. Section 4.4) were also used in this step.

The analysis was conducted by the researcher and was limited to a qualitative review of the respective documentation of each methodology (Section 8.3 discusses this limitation). Table 5.12 reports the evaluation results. The actual features from the reviewed methodologies offering the support are detailed in Appendices B.2 and B.3. With respect to modelling related dynamic change requirements, the Unified Modelling Language (UML) (OMG 2010b) was also considered when evaluating Catalysis, Select Perspective, OPF, RUP and Kobra because these methodologies adopt UML as their underlying modelling language. Object Modelling Language (Firesmith et al. 1997) was also evaluated since it is adopted by OPF.

Table 5.12 Feature analysis results of selected methodologies

Dynamic Change Requirement		Methodology (C:supporting component-based, S:supporting SOA-based)												
		C	C	C	C	C	C	S	S	S	S	S	S	S
		AEM	Catalysis	EPIC	Kobra	OPF	Select Perspective	RUP	ASG	CBDI-SAE	ERL	P&H	SeCSE	SUPER
Modelling Related	Part level													
	Multiple version coexistence											L		L
	Resource needs		L	L							L	L		
	Performance characteristics		L	H ²		H		H	H		L			L
	Access blocking													
	Application level													
	Dynamic change	L			L					L			L	
	Transformation				L									
	- Transitional form													
	Generation				L			M						
	Application lifecycle				L	L		M				L	L	
	Servicing continuity		L											
	Others													
	Transformation agent	L							L				L	
	Transformation action		M ¹	M ¹	M ¹	M ¹	M ¹	M ¹						
	Transformation exception													
	+ Transformation exception resolution													
	Expected dynamic change impact (after Step 2)		L	M		L								
Work Related	Part level													
	Dynamic part change								L					
	Dynamic part adapter		L	L	M	L	L	H	L	L	L	L	M	M
	Part retirement					H			L					L
	Dynamic part (re)binding								L			L	M	L
	Resource need prediction													
	Performance characteristic prediction								H				L	
	Geometric change							L						
	Dynamic state transfer													
	Application level													
	- Dynamic protocol evolution													
	Dynamic workflow evolution													
	Dynamic recomposition		L						L				L	
	Dynamic refactoring		L		L	M	L	L				L	L	
	Dynamic variability				M		L	L				L	H	
	+ Dynamic change impact analysis (after Step 2)		L	L		L								
	Dynamic contract update (after Step 2)								L					

Notes:

1. UML offers elementary action objects “CreateLinkAction”, “CreateLinkObjectAction”, “DestroyLinkAction” and “DestroyObjectAction” to model change operations on an object model (OMG 2010b). The evaluation result of “transformation action” on a methodology adopting UML is equivalent to that for UML.
2. EPIC reiterates the list of performance characteristics from RUP in its documentation. This is not surprising since EPIC is an extension of RUP.
3. “After Step 2” dynamic evolution requirements refer to the requirements added/enhanced as a result of the survey results.
4. Dynamic change requirements prefixed with “+” and “-” are those ranked as the most and least important respectively. See Figure 5.5.

From a methodological viewpoint (i.e. columns in Table 5.12), no methodology addresses all the dynamic change requirements. Furthermore, none of the methodologies adequately support (“H”) more than one modelling related requirement

or more than one work related requirement, suggesting that they pay little attention to dynamic evolution as a first class problem in application development. OPF, RUP and ASG offer the best support, providing adequate support for two dynamic change requirements (two H's). ASG, for instance, prescribes a performance related technique (addressing the "performance characteristics prediction" work requirement) and comprehensive performance modelling attributes (addressing the "performance characteristics" modelling related dynamic change requirement). EPIC and SeCSE provide the next best level of support (one H).

The methodologies were then assessed to determine if a particular group of methodologies offers better support for dynamic evolution. The grouping was performed based on the type(s) of composition-based applications they support (component-based, SOA-based or both) as shown in Table 5.12. There is no apparent difference in the level of support between the best methodologies from each group (OPF from component-based, ASG from SOA-based, and RUP from both), each providing full support for two dynamic change requirements (two H's). However, methodologies addressing component-based development provide better coverage for the modelling related dynamic change requirements than those for SOA-based development (i.e. more M's and H's in Table 5.12). This may reflect the relative maturity of the component-based paradigm which started gaining popularity in the 1990s. The SOA-based paradigm is comparatively younger, focusing mainly on key issues, such as services, as fundamental elements for building applications. The initial request from the Object Management Group for a Specification for the UML Profile and Metamodel for Services only dates from 2006 (OMG 2008). The modelling concepts of dynamic evolution are yet to be incorporated into the SOA-based paradigm.

From the dynamic change requirement viewpoint (i.e. rows in Table 5.12), "performance characteristics" seem to be the best supported: EPIC, OPF, RUP and ASG model them as sets of measurable attributes (e.g. throughput). The notion of a "dynamic part adapter" is the requirement most widely supported by all the methodologies with RUP offering the best support (i.e. scoring "H"). The work related dynamic change requirement ranked as the most important (cf. Section 5.2.4) - "dynamic change impact analysis" - is inadequately supported (i.e. no "H"). Moreover, neither of the error related modelling requirements (i.e. "transformation exception" and "transformation exception resolution"), which were also top ranked (cf. Section 5.2.4), are supported by the methodologies assessed. The fact that the methodologies do not provide adequate support for dynamic change requirements regarded as the most important by respondents is a

concern, suggesting areas to which priority should be given when developing methodology extensions for dynamic evolution.

5.5 CONCLUSION

In this Chapter, the outcomes of Task 1.2 of this research - viz. Synthesise, assess and extend important dynamic change requirements - have been reported. This task performed four steps to the incremental development of an extended set of dynamic change requirements suitable for dynamic evolution in composition-based distributed applications.

The initial set of dynamic change requirements was elicited from the literature and relevant evaluation frameworks. The requirements were subsequently evaluated in a survey. Feedback from the survey was used to extend the initial set with new requirements. Then, the extended set was further extended via a review of selected development methodologies. To demonstrate the use of the extended set of dynamic change requirements, an evaluation of the selected development methodologies for their extent of support for these requirements was discussed. The set of dynamic change requirements identified in Task 1.2 are summarised in Table 5.13.

Table 5.13 Dynamic change requirements investigated in Task 1.2

<i>Scope</i>	<i>Modelling Related</i>	<i>Work Related</i>
Part	Multiple Version Coexistence Resource Needs Performance Characteristics Access Blocking	Dynamic Part Change Dynamic Part Adapter Part Retirement (see note 1) Dynamic Part (Re)Binding Resource Need Prediction Performance Characteristic Prediction Geometric Change Dynamic State Transfer
Application	Dynamic Change Transformation Generation Transitional Form (see note 1) Application Lifecycle Servicing Continuity	Dynamic Protocol Evolution (see note 1) Dynamic Workflow Evolution Dynamic Recomposition Dynamic Refactoring Dynamic Variability Dynamic Change Impact Analysis Dynamic Contract Update
Others	Transformation Agent Transformation Action Transformation Exception Transformation Exception Resolution Expected dynamic change impact	

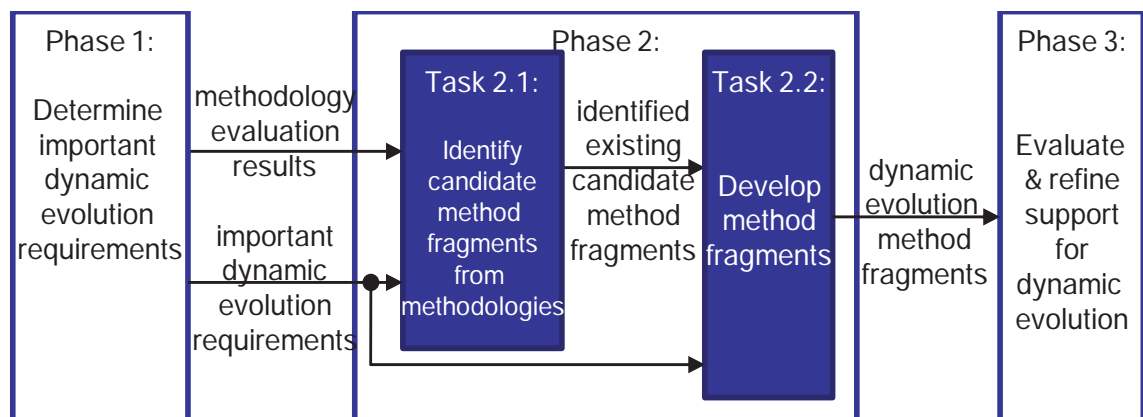
Notes:

1. These requirements were perceived by survey respondents as the least important (i.e. Step 2).

Chapter 6. DEVELOPMENT OF CONTINUUM

“Art and science have their meeting point in method.” - Edward Bulwer-Lytton, English dramatist, novelist and politician

In Chapter 4 and Chapter 5, the process of determining important dynamic evolution requirements for Continuum in Phase 1 and its outcomes were presented. This Chapter reports the outcomes from Phase 2 during which Continuum was subsequently developed to fulfil these requirements. Continuum is a methodology extension, comprising a suite of method fragments designed to be incorporated into an existing methodology via method engineering. Continuum developed in Phase 2 was subsequently evaluated and refined using the evaluation results in Phase 3. Figure 6.1 below illustrates the information flow among the tasks of Phase 2 and other phases:



source: developed for this research

Figure 6.1 Information flow in Phase 2 for developing Continuum

This Chapter is organised as follows. Section 6.1 summarises the set of important dynamic evolution requirements from Phase 1’s results, selected for the development of Continuum.

Section 6.2 presents the outcome of Task 2.1 of Phase 2, which sought candidate method fragments from existing methodologies suitable for reuse or requiring small enhancement to address these requirements.

Section 6.3 describes Continuum which was developed in Task 2.2 of Phase 2. The development involved a number of areas such as incorporating the method fragments identified from Task 2.1 as suitable for reuse into Continuum. A hypothetical application is used to guide the discussion of Continuum.

Section 6.4 summarises the correspondence between the dynamic evolution

requirements and Continuum's method fragments, as evidence of support for each requirement. For convenience, the version of Continuum presented in this Chapter also includes the refinements made to Continuum based on the evaluation results of Continuum obtained from Phase 3. The descriptions for the actual refinements together with the evaluation results are documented in the next Chapter. The full documentation for Continuum can be found in Appendix C. Section 6.5 concludes this Chapter.

6.1 REQUIREMENTS FOR CONTINUUM

In Phase 1 of this research, two types of dynamic evolution requirements were synthesised: *dynamic evolution quality factors* (cf. Chapter 4) and *dynamic change requirements* (cf. Chapter 5). Dynamic evolution quality factors are concerned with how well a distributed application and dynamic changes to it facilitate dynamic evolution (cf. Table 4.11). Dynamic change requirements characterise dynamic changes that a distributed application would accommodate and they are subdivided into *modelling* and *work* related dynamic change requirements (cf. Table 5.13). Since a vast number of requirements exist, it is logical to inspect their relative importance and focus on the more important ones so as to narrow the scope of the development of Continuum in Phase 2. The following requirements were omitted since they were considered the least important by the survey respondents:

- In the dynamic evolution quality factor category, *Efficiency* appears at the bottom of the importance ranking (Figure 4.5) and has the lowest mean rank (Table 4.7) based on the survey (Section 4.2.4);
- In the modelling related dynamic change requirement category, *transitional form* has the lowest importance ranking (Table 5.9). It is also the only modelling related requirement that was not perceived as important in the survey ($p_{wsr}=0.19$, Table 5.7); and
- In the work related dynamic change requirement category, neither *part retirement* nor *dynamic protocol evolution* was perceived to be important ($p_{wsr}=0.11$ and 0.38 , Table 5.8).

Accordingly, the set of dynamic evolution requirements that was selected for Continuum development are summarised in Table 6.1 (for dynamic evolution quality factors) and Table 6.2 (for dynamic change requirements).

Table 6.1 Summary of dynamic evolution quality factor requirements for Continuum (cf. Table 4.11)

Quality Factor	Quality Attribute
<i>Soundness of Change</i>	
Completeness	No missing functionality after a transformation
	No missing parts after a transformation
	No missing, illegal or broken bindings after a transformation
	(Also in consistency) assumptions and properties of a distributed application and its parts met by a change
Consistency	Compatible bindings
	Compatible communications protocol among parts
	All parts involved in a runtime change identified before a transformation
	No progression towards an error state after a transformation
	Synchronisation of application's and parts' states after a transformation
	A reachable state attained after a transformation
	No critical procedures executed before a transformation
	No pending messages, interactions or transactions before a transformation
	System invariants preserved from a transformation
	Adequate resources and support for new and replacement parts
	(Also in completeness) assumptions and properties of a distributed application and its parts met by a change
Correctness	Non-arbitrary and admissible changes
	No unintentional behaviour during and after a transformation
	Correct ordering of transformations
	Transformations at a right time
<i>Infusibility of Change</i>	
Locality	Application partitioning and change localisation to partitions
Maintainability	All parts clearly defined in interaction (or workflow) specifications
	No degradation in cost and ease of modifications
	No reduction in testability
	Clear and detailed interactions
Transparency	Transformations hidden from end users
	Transformation design and implementation hidden from application programmers
	Transformations hidden from parts unaffected by the transformations
	Transformation agents hidden from operating environment
<i>Changeability of Application</i>	
Autonomy	Self-control and self-governance of parts' own processing
Coordination	Transformations coordinated among multiple nodes/organisations
	Transformation agents tolerant of network unreliability during a transformation
Extensibility	Runtime extension/upgrade of an application with new functionality
	Runtime extension/upgrade of parts in an application with new functionality
	Runtime extension/upgrade of an application with new parts

<i>Quality Factor</i>	<i>Quality Attribute</i>
Loose Coupling	High level of independence between parts
	Parts having their own lifecycles and runtime environments
Separation of Concerns	Separating dynamic change concerns from functionality concerns
	Separating communication concerns from functionality concerns
	Separating security support from functionality concerns
	Separating realisations of parts from those of part clients
	Separating part specification from realisation concerns
<i>Robustness of Application</i>	
Fault Tolerance	High tolerance of faulty new and/or changed parts
	Barriers established to contain potentially faulty new and replacement parts
Recoverability	Restoration of an application and its parts to a state to continue to perform their functionality, after a failure caused by a transformation and/or its dynamic change(s)
Reliability	No compromise on intended functionality after a transformation
	Replacement parts fully satisfying their roles
Safety	Distributed application and its parts operating safely during and after a transformation
Security	Transformation agents secured from unauthorised access
	No security compromise by new and replacement parts after a transformation
	Access to for new and replacement parts restricted
	Dynamically updated security policy
	Separating security policy from security enforcement

Table 6.2 Summary of dynamic change requirements for Continuum (cf. Table 5.13)

<i>Requirement</i>	<i>Description</i>
<i>Modelling related, part level</i>	
Multiple version coexistence	Model the ability of a service to have <i>multiple versions</i> present in various regions of an application as the service evolves over time.
Resource needs	Model the changing <i>resource needs</i> of a service as it evolves.
Performance characteristics	Model the changing <i>performance characteristics</i> of a service as it evolves.
Access blocking	Model the ability to <i>block access</i> to a service while it is being updated or replaced.
<i>Modelling related, application level</i>	
Dynamic change	Model the notion of a <i>runtime change</i> which is the <i>intended result</i> of a modification to an application at runtime.
Transformation	Model the notion of a <i>transformation</i> which is an act of performing modifications to an application at runtime.
Generation	Model the notion of a <i>generation</i> which represents a stable version of an application at a particular time.
Application lifecycle	Model the notion of an <i>application lifecycle</i> which organises application evolution as a series of generations over time.
Servicing continuity	Model the ability of an application to <i>continuously offer some functionality</i> during a transformation.

<i>Requirement</i>	<i>Description</i>
<i>Modelling related, others</i>	
Transformation agent	Model the notion of a <i>transformation agent</i> responsible for performing transformations.
Transformation action	Model the <i>steps</i> undertaken in a transformation, e.g. adding a new service, followed by configuring the service.
Transformation exception	Model the notion of an <i>error condition</i> occurred during a transformation, e.g. Because of faulty changes.
Transformation exception resolution	Model the notion of a <i>resolution</i> - e.g. a rollback - to process and resolve errors caused by transformation exceptions.
Expected dynamic change impact	Model the impact of dynamic changes (scope and extent) expected on parts within the application and other applications affected by the changes.
<i>Work related, part level</i>	
Dynamic part change	Define support to <i>add, replace, remove</i> individual parts, and <i>modify their parameters</i> at runtime.
Dynamic part adapter	Define <i>adapters</i> for wrapping and plugging parts into an architecture at runtime, and resolving their mismatches.
Dynamic part (re)binding	Define support to <i>(re)bind one part to a new part at runtime</i> as the original part is replaced or upgraded by the new part.
Resource need prediction	Define support to <i>predict the resource needs</i> for a new part.
Performance characteristic prediction	Define support to <i>predict the performance characteristics</i> for a new part.
Geometric change	Define support to <i>relocate parts</i> to a different hosting environment as needed.
Dynamic state transfer	Define support to <i>transfer the state</i> from an instance of an old implementation of a part/workflow to an instance of the new implementation of the part/workflow as it evolves.
<i>Work related, application level</i>	
Dynamic workflow evolution	Define support to <i>evolve a workflow</i> definition while the workflow is operational.
Dynamic recomposition	Define support to <i>combine</i> several parts/workflows into a larger unit, <i>split</i> a larger part/workflow into smaller units or <i>reconfigure</i> the architectural structure at runtime.
Dynamic refactoring	Define support to <i>refactor</i> an application structure without functional changes at runtime, say, to reduce its complexity and improve its performance.
Dynamic variability	Define customisation points in an architecture to plug in or swap different parts at runtime to support limited <i>variations</i> in functionality (e.g. Using credit card vs. Account debit for payment).
Dynamic change impact analysis	Analyse the <i>impact of dynamic changes</i> to an application to determine which parts/workflows of the application, those parts/workflows outside the application, and other applications, will need to be affected and updated to accommodate the changes.
Dynamic contract update	Define support to <i>update contracts among parts</i> as the parts/contracts evolve.

6.2 TASK 2.1: METHOD FRAGMENT IDENTIFICATION FROM RELEVANT METHODOLOGIES

The objectives of this task were to identify and select features from relevant and

existing methodologies that can be reused without the need for enhancement, or can be enhanced with minimal effort to address the requirements. Selected features were documented as the International Standard ISO/IEC 24744 (ISO/IEC 2007) based method fragments to prepare them for incorporation into Continuum in Task 2.2. This task reused the assessment results of the selected methodologies in Phase 1 for their support of the dynamic evolution requirements. In particular, the results identify candidate features from the methodologies suitable for Continuum:

1. For reuse (i.e. rated as “H” for high in Phase 1), a feature explicitly and fully supports a particular requirement; and
2. For enhancement (i.e. rated as “M” for medium in Phase 1), a feature appears to support a particular requirement to some extent and needs a *small enhancement* to fulfil the requirement.

Although the assessment results are indicative of which of the features could *potentially* be incorporated for reuse/enhancement, they do not determine which of the features should be selected for Continuum. This is because a requirement may well be supported by similarly scored features from several methodologies. Accordingly, in this task the following steps were performed for each requirement to select the final set of features for reuse and enhancement in Continuum:

1. Determine if a feature exists in a methodology that fully supports the requirement (i.e. scored with “H”). If more than one feature is identified, select one as the *reusable* feature. In this case, the rationale for selecting a particular feature or integrating two or more into a better feature for the requirement is detailed in Table 6.3 and Table 6.4.
2. If no suitable feature is found in Step 1, identify the features with the medium score (i.e. “M”). If more than one feature is equally scored, select the feature requiring the minimum effort for enhancement, as the to-be-enhanced feature.
3. If no suitable feature is found, a *new* fragment is developed.

Features identified from Steps 1 and 2 above are presented in Table 6.3 and Table 6.4 for dynamic change and quality factor requirements respectively. The “Potential Source of Reuse/Enhancement” column summarises the features from the methodologies offering the support and their scores, as analysed in Phase 1 (cf. Appendix B). The “Decision for Continuum” column summarises the final features from the methodologies selected for Continuum and the rationale for choosing them. For brevity, only those requirements

that have corresponding features suitable for reuse/enhancement are shown in these tables. The following abbreviations are used in these tables for identifying respective methodologies:

- ASG: Adaptive Service Grid (Lehner et al. 2006)
- EPIC: Evolutionary Process for Integrating Commercial-off-the-Shelf Based Systems (Albert & Brownsword 2002)
- OPF: OPEN Process Framework (Firesmith & Henderson-Sellers 2002)
- RUP: Rational Unified Process (Kruchten 2003)
- SeCSE: Service Centric System Engineering (SeCSE 2007)
- SUPER: Semantics Utilised for Process Management within and between Enterprises (SUPER 2007)

Table 6.3 Features for reuse/enhancement for dynamic change requirements

Requirement	Potential Source(s) for Reuse/Enhancement	Decision for Continuum
Modelling Related		
<i>Part Level</i>		
Performance characteristics	H :OPF—capacity, latency, throughput, response time. H :EPIC/RUP—speed, efficiency, availability, accuracy, throughput, response time, recovery time. H :ASG—performance metrics: duration (response time, service time, network delay, residence time), capacity (throughput, workload, error rate).	Reuse ASG's definitions which are the most comprehensive and accompanied by performance engineering capabilities which also meet another requirement for Continuum (see the "performance characteristic prediction" requirement).
<i>Application Level</i>		
Generation	M :RUP—"software generation" from "evolution cycle".	Enhance RUP's notion of "generation" to cover dynamic evolution.
Application lifecycle	M :RUP—"evolution cycle".	Enhance RUP's notion of "evolution cycle" to cover dynamic lifecycle of application.
<i>Others</i>		
Transformation action	M :Unified Modelling Language (UML) (as used by many of the reviewed methodologies)—four low level actions relevant to object/link instantiation and destruction, but insufficient for a variety of scenarios in a transformation.	Enhance UML's action concept.
Expected dynamic change impact	M :EPIC—within the target resistance plan, EPIC characterises the impact of change with several attributes: "target that will be affected", "type of impact", and "level of disruption".	Enhance EPIC's notion of impact of change for dynamic change.

<i>Requirement</i>	<i>Potential Source(s) for Reuse/Enhancement</i>	<i>Decision for Continuum</i>
Work Related		
<i>Part Level</i>		
Dynamic part adapter	H :RUP–“service mediation” guideline to mediate interface, protocol and operational conflicts between parts. M :KobrA–“component adaptation” process to bridge between a component specification and a component to be reused. M :SeCSE–designing adapter in the “re-planning”, process to replace faulty services. M :SUPER–process and data mediators to bridge interactions between processes and to translate their data flows.	<i>Reuse</i> RUP’s “service mediation” guideline.
Dynamic part (re)binding	M :SeCSE–“binding and re-binding” process to find, select and execute appropriate service	<i>Enhance</i> SeCSE’s “binding and re-binding” process.
Performance characteristic prediction	H :ASG–“performance engineering” analysis and modelling tasks and techniques for SOA solutions (Kempter et al. 2007).	<i>Reuse</i> ASG’s “performance engineering” tasks.
<i>Application Level</i>		
Dynamic refactoring	M :OPF–“design refactoring” task.	<i>Enhance</i> OPF’s task to support dynamic refactoring.
Dynamic variability	M :KobrA–support for static variability in “framework engineering” process. H :SeCSE–“variation points management” process to design variation points, and “variation points realisation” process to design runtime variability in applications.	<i>Reuse</i> SeCSE’s “variation points management” and “variation points realisation” processes.

Table 6.4 Features for reuse/enhancement for dynamic evolution quality factors

<i>Quality Factor /Attribute</i>	<i>Potential Source of Reuse/Enhancement</i>	<i>Decision for Continuum</i>
Soundness of Change		
<i>Completeness</i>		
No missing functionality after a transformation	H :RUP–features the “review the design” task to check if a design model fulfil its requirements.	<i>Reuse</i> RUP’s “review the design” task.
<i>Consistency</i>		
System invariants preserved from a transformation	H :SeCSE–“regression testing” technique to derive invariants and test cases to verify them.	<i>Reuse</i> SeCSE’s “regression testing” technique.
Infusibility of Change		
<i>Locality</i>		
Application partitioning and applying changes to partitions	M :Catalysis–suggests several ways to partition an application into packages to confine propagation of changes.	<i>Enhance</i> Catalysis to support runtime changes.
<i>Maintainability</i>		
All parts clearly defined in interaction (or workflow) specifications	Well supported by many methodologies’ respective modelling languages (rated H for UML and BPMN for Catalysis, KobrA, OPF, P&H, RUP, SeCSE, Select Perspective).	<i>Reuse</i> respective modelling languages where appropriate to document interactions.

<i>Quality Factor /Attribute</i>	<i>Potential Source of Reuse/Enhancement</i>	<i>Decision for Continuum</i>
No degradation in cost and ease of modifications	<p>M:EPIC—defines characteristics of an evolvable architecture.</p> <p>M:OPF—suggests a number of mechanisms to implement maintainability.</p> <p>M:Select Perspective—defines characteristics of easily maintained components and applications.</p>	EPIC, OPF and Select Perspective's characterisations of an easily maintainable application overlap (e.g. proper and current documentation). However, OPF defines the most number of characteristics. Enhance OPF by augmenting its definitions with characteristics from EPIC and Select Perspective but missing in OPF.
Clear and detailed interactions	Well supported by many methodologies' respective modelling languages (rated H for UML and BPMN for Catalysis, Kobra, OPF, P&H, RUP, SeCSE and Select Perspective).	Reuse respective modelling languages where appropriate to document interactions.
Changeability of Application		
<i>Autonomy</i>		
Self-control and self-governance of parts' own processing	H :ERL—the "service autonomy" principle which distinguishes between "service-level autonomy" and "pure autonomy".	Reuse ERL's "service autonomy" principle.
<i>Loose Coupling</i>		
High level of independence between parts	<p>H:Catalysis—"decoupling process" pattern for objects and classes.</p> <p>H:P&H—the "service coupling" principle as a guideline for SOA.</p>	Reuse P&H's "service coupling" principle which is more suitable to composition-based models. Catalysis's pattern targets objects and classes which are less relevant/intuitive to composition-based models than P&H.
<i>Separation of Concerns</i>		
Separating communication concerns from functionality concerns	<p>H:RUP—communication concerns between two services are abstracted from services into a concept called "service channel", and addressed in the "service mediation" guideline.</p> <p>M:Catalysis— communication concerns are abstracted from components into the notion of an "action" and "collaborations" to model the interactions.</p>	Reuse RUP's "service channel" and "service mediation" to handle part communication.
Separating security support from functionality concerns	H :ERL—ERL's approach to security is based on the WS-Security standard (OASIS 2006). The standard defines mechanisms to separate security support from Web services functionality.	Reuse ERL's approach to support security, in the SOA context.
Separating realisations of parts from those of part clients	H :Select Perspective—uses two development processes to separate the development activities of component providers from those of component consumers, thus decoupling their realisation dependencies.	Reuse Select Perspective's notion of separating the development of parts from those of part clients.

<i>Quality Factor /Attribute</i>	<i>Potential Source of Reuse/Enhancement</i>	<i>Decision for Continuum</i>
Separation of part specification from implementation	Well supported by many methodologies (rated H for Catalysis, Select Perspective, RUP, EPIC, COMET, UML Components, P&H, SeCSE).	In summary, methodologies rated with H suggest that specifications and implementation of a part should be 1) regarded as separate concepts, 2) developed with separate activities; and 3) documented separately. Rather than reusing a feature from a particular methodology, Continuum adopts these criteria in its inspection checklist (see Dynamic Evolution Quality Inspection Report, Appendix C.2.2.3).
<i>Robustness of Application</i>		
<i>Reliability</i>		
Replacement parts fully satisfying their roles	H :SeCSE–“regression testing” technique to ensure a replacement part is in line with the expectations (both behaviour and QoS) for the original part.	Reuse SeCSE’s “regression testing” technique.
<i>Safety</i>		
Distributed application and its parts operating safely during and after a transformation	M :OPF–“work product safety engineering” process to ensure work products produced reach accepted levels of safety.	Enhance OPF’s process to cover composition-based distributed application with dynamic evolution capability.
<i>Security</i>		
Transformation agents secured from unauthorised access	M :OPF–“security engineering” process to ensure security needs of the work products are met.	Enhance OPF’s process to cover this requirement.
No security compromise after a transformation	Ditto	Enhance OPF’s process to cover this requirement.
Access to new and replacement parts restricted after a transformation	Ditto	Enhance OPF’s process to cover this requirement.

6.3 TASK 2.2: METHOD FRAGMENT DEVELOPMENT

This task developed Continuum to support the requirements identified in Phase 1. Continuum follows the paradigm of specifying a methodology’s elements as a set of method fragments, with a primary focus on dynamic evolution. The word Continuum originates from the Latin word “continuus” meaning “uninterrupted”, and signifies “a continuous sequence in which adjacent elements are not perceptibly different from each other, but the extremes are quite distinct” (Oxford English Dictionary). In this research, the word symbolises the ability of an application to continue running as it evolves. The development of Continuum comprises the following areas:

1. Incorporate the method fragments of reviewed methodologies that were selected for *reuse* in Task 2.1 into Continuum;
2. Enhance the method fragments of reviewed methodologies that were selected for *enhancement* in Task 2.1 and incorporate these fragments into Continuum;
3. For each requirement not addressed by any of the method fragments above, identify an approach from the literature that addresses the requirement; and
4. Develop *new* method fragments to meet the requirements that are otherwise not addressed by any of the method fragments above.

The description of Continuum is organised as follows. Section 6.3.1 describes a hypothetical application used to guide the discussion of Continuum in subsequent sections. Section 6.3.2 gives an introduction to Continuum. Section 6.3.3 presents key concepts and their relationships underpinning Continuum to model dynamic evolution. The next three sections present Continuum's method fragments for dynamic changes (Section 6.3.4), for dynamic evolution quality (Section 6.3.5) and for representing producers who execute dynamic evolution specific tasks during development (Section 6.3.6). Finally, Section 6.3.7 provides usage guidelines for Continuum such as how its process fragments could be used in a development lifecycle.

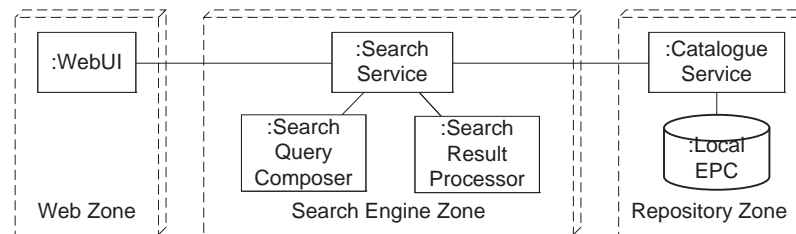
6.3.1 Electronic Product Catalogue Platform

A hypothetical application called Electronic Product Catalogue Platform (EPCP) is used throughout this Chapter to illustrate the features in Continuum. EPCP serves electronic product catalogues (EPC) on items of interest (e.g. merchandise) via the web, comprising:

- *Catalogue Service*, which provides access to EPC data;
- *Local EPC*, a database providing up-to-date catalogue information on items of interest, offered by an organisation in local currency;
- *Search Service*, which coordinates search invocations to the Catalogue Service using:
 - *Search Query Composer*, which builds search queries for search criteria entered by end users; and
 - *Search Result Processor*, which ranks and iterates through search results; and
- *Web User Interface (WebUI)*, for end users to enter search criteria and view search results.

Figure 6.2 illustrates EPCP. Solid boxes, connecting lines and the drum symbol are

notations from existing modelling languages of choice. Dashed boxes represent logical partitions in EPCP called zones using Continuum's notation (cf. Structural Configuration - Notational Extensions, Appendix C.2.2.8).



source: developed for this research

Figure 6.2. EPCP: current structure

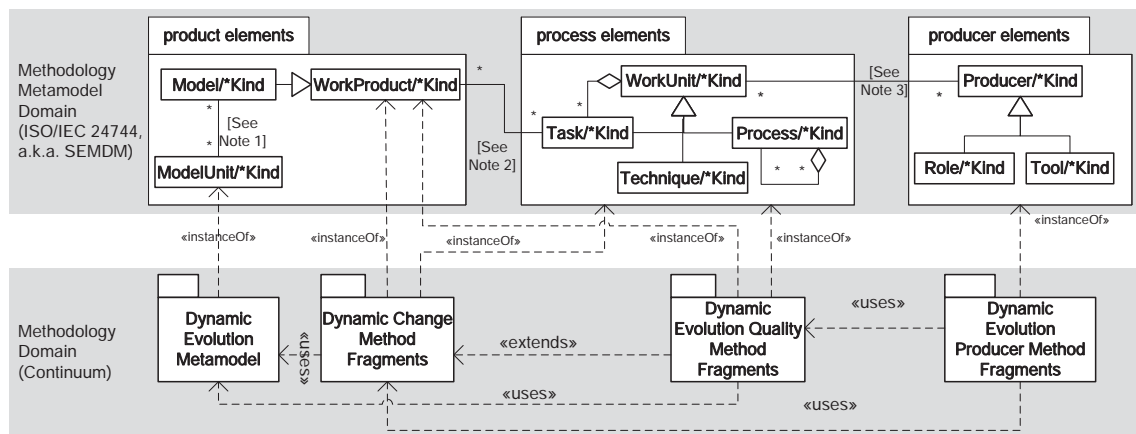
Suppose that after the initial successful launch of EPCP, its stakeholders decide that it should be *integrated with* two external partner EPC systems to provide additional catalogues for an increased variety of merchandise. Furthermore, EPCP will also support the *internationalisation* (a.k.a. “i18n”) capability (Arnold et al. 2005), meaning EPCP will handle catalogue items described in foreign languages and priced in different currencies. The EPCP scenario is inspired by the need for integrating different EPC systems together (Lincke & Schmid 1998). Since EPCP runs around the clock, it is desirable to progressively roll out these changes while reducing the interruption to EPCP.

6.3.2 Overview of Continuum

In specifying a full methodology or a methodological extension such as Continuum, a common approach is to instantiate elements from an existing methodology metamodel. More generally, a metamodel is a model of models (Atkinson & Kühne 2003). Likewise, a methodology metamodel is a model of methodologies (Gonzalez-Perez & Henderson-Sellers 2006b) offering (methodology) elements for construction and definitions of a methodology or a methodology extension. One such metamodel is the International Standard ISO/IEC 24744: Software Engineering - Metamodel for Development Methodologies (SEMDM) (ISO/IEC 2007) which supports both product and process aspects. The product aspect represents modelling needs of an endeavour, such as modelling concepts and the artefacts generated/used, whereas the process aspect represents what are performed in an endeavour. Because of its ability to support these two aspects, SEMDM is adopted for specifying Continuum.

Figure 6.3 depicts the major components of Continuum and their relationships with relevant methodology metamodel domain elements from SEMDM. In the “methodology

metamodel domain” (upper half of Figure 6.3) are the SEMDM elements called “powertype patterns” (Henderson-Sellers & Gonzalez-Perez 2005) all of which suffixed with “/*Kind” (Gonzalez-Perez & Henderson-Sellers 2006b). They are intended to be instantiated to create methodological elements (Henderson-Sellers & Gonzalez-Perez 2006), such as Continuum’s, in the “methodology domain” (lower half of Figure 6.3). (For simplicity the suffix “/*Kind” is suppressed from SEMDM element names in the remainder of this Chapter.)



source: developed for this research

Notes:

1. Model/*Kind is related to ModelUnit/*Kind via ModelUnitUsage/*Kind which is hidden for simplicity.
2. Task/*Kind is related to WorkProduct/*Kind via Action/*Kind which is hidden for simplicity.
3. Producer/*Kind is related to WorkUnit/*Kind via WorkPerformance/*Kind which is hidden for simplicity.

Figure 6.3. SEMDM and Continuum components

As shown in the upper half of Figure 6.3, three types of SEMDM elements are used for specifying Continuum via the instantiation mechanism:

- *Product elements* (upper left of Figure 6.3)

A *WorkProduct* is an artefact of interest in an endeavour, such as a *Model* to represent a particular view of an entity being modelled (e.g. class diagram). *ModelUnits* are atomic constructs used in a model to define the meaning of the model (e.g. class).

- *Process elements* (upper middle of Figure 6.3)

A *WorkUnit* is an abstraction of some job to be performed for a given purpose. More specifically, a *Process* is a large-grained work unit within a given area of expertise, whereas a *Task* and a *Technique* respectively focus on what must be done for a given purpose and how to achieve it for the given purpose. A logical

way to construct work units is to define tasks to be performed within each process, and supplement the tasks with techniques to achieve their given purpose (e.g. OPFRO 2009).

- *Producer elements* (upper right of Figure 6.3)

A *Producer* is a surrogate for an entity which has the responsibility of executing work units according to its areas of expertise. In particular, a *Role* defines a set of responsibilities that a producer can play to fulfil the objectives of certain work units, or a *Tool* which represents responsibilities that involve executing certain work units that can be automated.

The instantiation of SEMDM elements from the methodology metamodel domain into the methodology domain (i.e. from the upper to the lower half of Figure 6.3) establishes the following major components of Continuum:

- The dynamic evolution metamodel¹⁶ (Section 6.3.3), defining elementary concepts within the domain of dynamic evolution and the relationships among them. They are instantiated from SEMDM's ModelUnit powertype pattern.
- Dynamic change method fragments (Section 6.3.4), comprising work unit and work product fragments, to provide basic support for the analysis and design of changes to a running composition-based distributed application without requiring shutdown. They are instantiated from SEMDM's WorkProduct, Process, Task and Technique powertype patterns.
- Dynamic evolution quality method fragments (Section 6.3.5), comprising work unit and work product fragments, extend the capabilities of the dynamic change method fragments by addressing the quality aspects of the dynamic changes developed with the dynamic change method fragments. They are instantiated from SEMDM's WorkProduct, Process, Task and Technique powertype patterns.
- Dynamic evolution producer method fragments (Section 6.3.6), specifying who are responsible for executing work units in dynamic change and dynamic

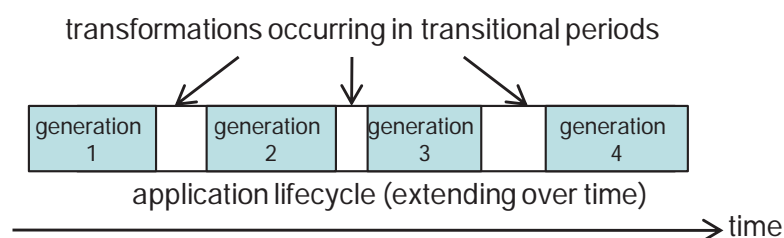
¹⁶ A metamodel has two meanings, "a model of models" and "a model of methodologies" (Gonzalez-Perez & Henderson-Sellers 2007). The former can also be seen as a domain ontology intended for a target domain (Rossi et al. 2004). It describes modelling concepts and the relationships among them (e.g. "classes" and "objects" in object-oriented modelling (ISO/IEC 2007)) They can be specified with instances of "model unit class" which are also methodology elements (ISO/IEC 2007) or method fragments (Henderson-Sellers & Ralyté 2010). In this research, the notion of *dynamic evolution metamodel* takes the meaning of a "model of models" which refers to a set of concepts relevant to dynamic evolution and their relationships.

evolution quality method fragments according to their areas of expertise. They are instantiated from SEMDM's Role and Tool powertype patterns.

Continuum components address a breadth of dynamic evolution problems sufficiently generic for a variety of composition-based distributed application types rather than for a particular composition-based application type. To enhance an existing methodology, the method engineering approach can be leveraged to incorporate Continuum components into the methodology's lifecycle to drive ongoing developments and dynamic changes to an application as they arise. Note that Continuum components focus on the analysis and design aspects of dynamic evolution. Conventional aspects of software development such as implementation, testing and configuration management, as well as those specialised for particular kinds of composition-based applications (e.g. component-based) are expected to be handled by features of existing methodologies.

6.3.3 Dynamic Evolution Metamodel

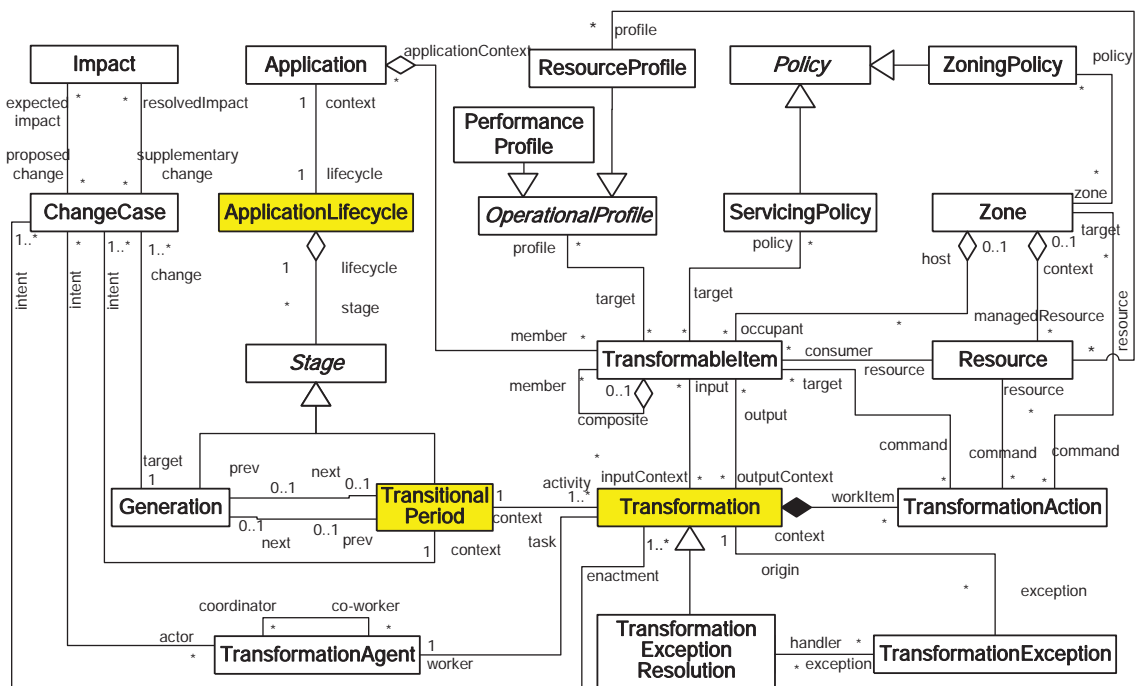
Continuum divides the notion of dynamic evolution into three concerns: application lifecycle, transitional period and transformation. Each concern symbolizes dynamic evolution on a different time scale so as to focus on particular issues at a time to handle its complexity. On a coarse-grained time scale, the *Application Lifecycle* concern characterises the progression of an application over its generations of releases during its operating lifespan. A *Generation* denotes the application is operating a particular version of its code, such as Figure 6.2 which shows the current generation of EPCP. The notion of a *Transitional Period* represents the period in the application lifecycle during which the application advances from one generation to the next via high-level runtime modifications called *Transformations*. On a fine-grained time scale, a *Transformation* concern prescribes low-level atomic runtime modification steps to the application's elements, such as initialising a new component before use. Figure 6.4 illustrates these concepts.



source: developed for this research

Figure 6.4. Application lifecycle, transitional periods and transformations

A metamodel generally defines concepts called model unit fragments (ISO/IEC 2007), representing pieces of coherent information about a topic and their relationship. It is modelled with object-oriented class diagrams in which each class corresponds to a model unit fragment. (Strictly speaking, a model unit fragment should be called a “clabject” (ISO/IEC 2007), having both the instance aspect from the SEMDM metamodel, and the class aspect from the methodology.) In Phase 1 of this research, individual concepts were identified from the literature, and subsequently validated and extended in a survey (cf. Chapter 5). The development of Continuum involved first a synthesis of a dynamic evolution metamodel which specifies these concepts as model unit fragments and defines the relationships among them. The resultant metamodel is shown in Figure 6.5 with the three key concerns highlighted.



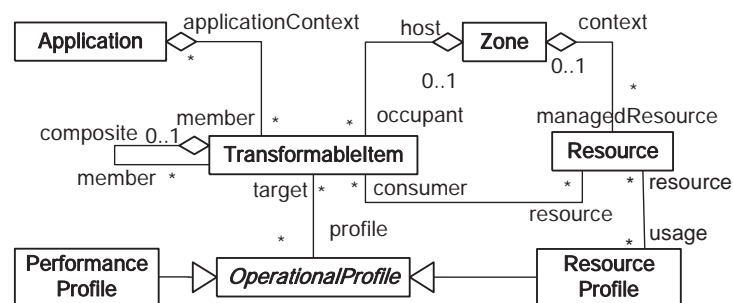
source: developed for this research

Figure 6.5 Dynamic evolution metamodel

Continuum’s dynamic evolution metamodel can be described along five related aspects: *structural foundation*, *application lifecycle*, *transitional period*, *transformation*, and *policy*.

Structural foundation related concepts (Figure 6.6: Application, OperationalProfile, PerformanceProfile, Resource, ResourceProfile, TransformableItem and Zone): These concepts represent abstractions for a composition-based distributed application which are the runtime structure of the application in terms of its elements and the relationships between them. They also prescribe constructs in its hosting environment

to support the running of the application. They are specified to a level of detail sufficient to articulate dynamic evolution in the application. Central to the structural foundation is *TransformableItem* denoting a runtime logical distributed entity that can be transformed into some other form. It epitomises a part, a binding or a composite thereof. It may consume specific *Resources* in order to function (Dearle 2007). An *OperationalProfile* represents distinctive characteristics of the *TransformableItem* such as resource needs and performance during its normal operation (i.e. *ResourceProfile* and *PerformanceProfile*). *OperationalProfiles* may impose conditions for runtime changes to be satisfied if a *TransformableItem* is to be modified or affected by those changes. At a high level of abstraction, an *Application* is a set of related *TransformableItems* together offering some functionality and computing capabilities. A *Zone* (Evans & Dickman 1999) defines a logical partition in a distributed environment (e.g. a logical node in a network) for hosting *TransformableItems*. In the EPCP example, its elements, which are *TransformableItems*, are scattered over three *Zones*: Web, Search Engine and Repository (Figure 6.2).

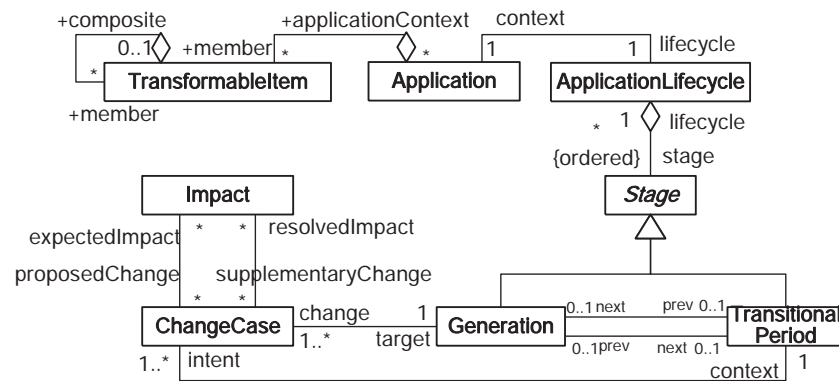


source: developed for this research

Figure 6.6 Structural foundation classes

Application lifecycle related concepts (Figure 6.7: *Application*, *ApplicationLifecycle*, *ChangeCase*, *Generation*, *Impact*, *Stage*, *TransformableItem* and *TransitionalPeriod*): An *ApplicationLifecycle* transcribes a series of stages over which an *Application* progresses, from the time the application becomes operational to the time it is retired. A *Stage* is a time period within the lifecycle of an application and can be a *Generation* or a *TransitionalPeriod*. During a *Generation*, an application is operating a particular version of its code. A *TransitionalPeriod*, on the other hand, designates the period between two successive *Generations* of an application during which it advances from one *Generation* to the next via a dynamic modification. A *TransitionalPeriod* is present since the modification almost never completes instantly. An *ApplicationLifecycle* can thus be conceived of as a sequence of alternate *Generations* and *TransitionalPeriods*.

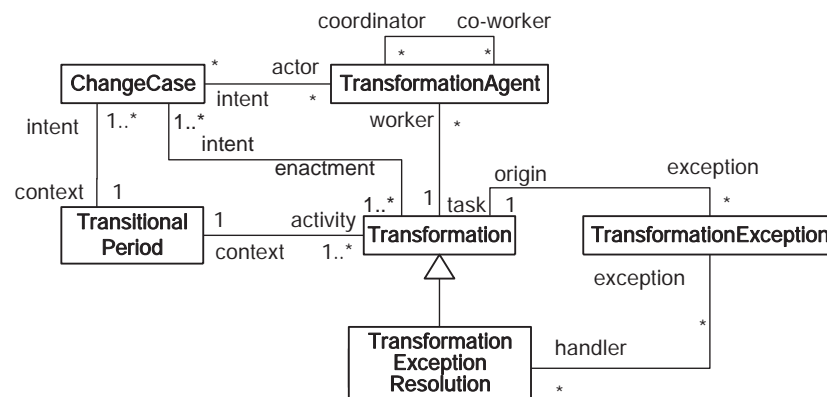
over time.



source: developed for this research

Figure 6.7 Application lifecycle related model unit fragments

Whilst requirements and change requests generally describe what an Application is intended to become, they may not always be sufficiently explicit for expressing dynamic changes to a running Application. Thus, Continuum extends the notion of a *ChangeCase* (Ecklund et al. 1996; Office of Government Commerce 2002). In Continuum, ChangeCases articulate dynamic changes to the properties and functionality of an Application's TransformableItems, to its structure, and to its Zones (e.g. "replace function p1 in component instance X with a new function p2") and to facilitate further understanding of changes required for the Application at runtime. ChangeCases may result in one or more *TransitionalPeriods* required to gradually roll out the intended changes based on their priorities, complexity etc. Once a change is proposed for an Application, the effect of the modification may ripple through several parts of an Application (Bohner 2002a). An *Impact* captures entities thought to be affected by a ChangeCase (TransformableItems, Zones etc.) and the extent of the impact.

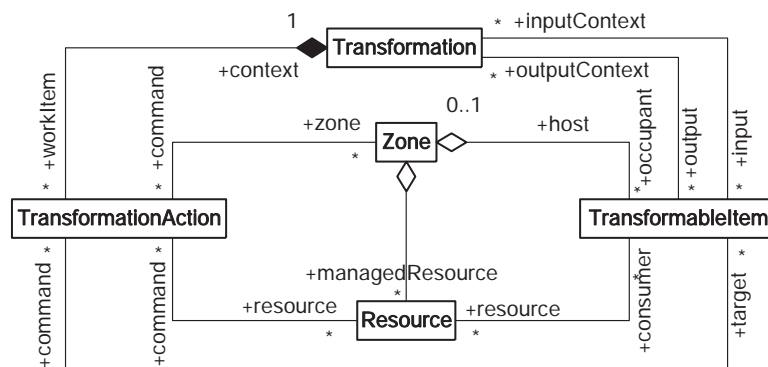


source: developed for this research

Figure 6.8 Transitional period related model unit fragments

Transitional Period related concepts (Figure 6.8: TransitionalPeriod, TransformationAgent, ChangeCase, Transformation, TransformationException and TransformationExceptionResolution): During a *TransitionalPeriod*, one or more *TransformationAgents* (e.g. upgrade manager (OMG 2003b)) perform a set of Transformations (i.e. the “how”) over a network in concert to realise a set of ChangeCases (i.e. the “what”). A *Transformation* represents some kind of high-level modification activity to an application. Since errors inevitably occur because of unforeseen circumstances while performing transformations, a *TransformationException* thus models such an error occurrence. Accordingly, a *TransformationExceptionResolution* designates a special kind of Transformation for resolving *TransformationExceptions*. An example resolution is to roll back changes made by an incorrect Transformation.

Transformation related concepts (Figure 6.9: Resource, TransformableItem, Transformation, TransformationAction and Zone): A *Transformation* captures the actual performance of runtime modification steps to an application. Each modification step, being atomic, stepwise, low-level and fine-grained (e.g. “bind a new component” to an application), is called a *TransformationAction*. Example TransformationActions include configuring the start-up state for a new TransformableItem before it operates, setting up Zones to host new TransformableItems, and allocating Resources to new TransformableItems.

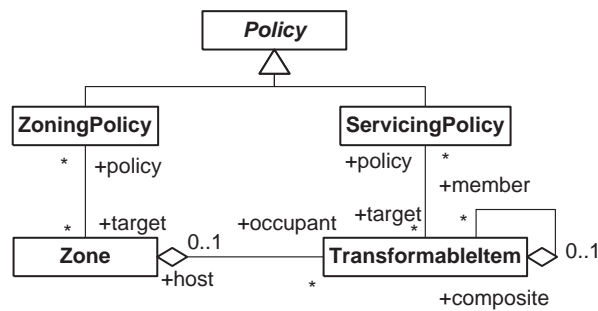


source: developed for this research

Figure 6.9 Transformation related model unit fragments

Policy related concepts (Figure 6.10: Policy, ServicingPolicy and ZoningPolicy): A *Policy* generally describes a set of rules for a particular purpose (ISO/IEC 2006). In dynamic evolution, a Policy is an abstract concept for articulating rules or behaviour desirable for dynamic evolution in a composition-based distributed application. An example network policy might confine the behaviour of a Transformation from breaking

communication links in an application. The Policy class is extended to model various rules in dynamic evolution. *ServicingPolicy*, for instance, denotes the extent to which a TransformableItem offers its services, especially useful during a TransitionalPeriod. *ZoningPolicy*, on the other hand, specifies rules for configuring a zone to host its TransformableItems as they evolve.



source: developed for this research

Figure 6.10 Policy related model unit fragments

6.3.4 Dynamic Change Method Fragments

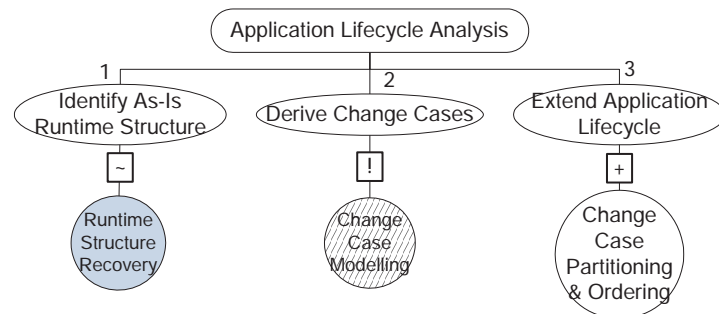
In light of the approach to divide dynamic evolution into three concerns, viz. application lifecycle, transitional period and transformation, Continuum prescribes four dynamic evolution specific process fragments (or processes), plus associated fragments, for the analysis and design of dynamic changes to composition-based distributed applications:

- *Application Lifecycle Analysis* to address the application lifecycle concern during analysis;
- *Transformation Identification* and *Transformation Agent Design* to tackle identification, sequencing and assignment of transformations during a transitional period; and
- *Transformation Design* to handle the detailed design of each transformation.

6.3.4.1 Fragments for Application Lifecycle Analysis

The Application Lifecycle Analysis process aims to define the roadmap for an application as successive stages in its lifecycle to accommodate changes elicited from requirements analysis. It does so by extending its lifecycle with new generations, with each transitional period between two successive generations labelled with associated change cases to accomplish during the period. Figure 6.11 shows the tasks iterated and techniques used in this process. The rounded rectangle, ellipse and circle notations correspond to process, task and technique instances as per SEMDM's notation (ISO/IEC 2010). The symbol on each link between a task and a technique

indicates the degree of suitability of the technique to the task: (“!”:mandatory, “+”:recommended, “~”:optional). For convenience, tasks are also annotated with numbers (not part of SEMDM’s notations) as suggestions for the order in which they can be iterated in this process. Additionally, a filled shape represents a technique reused from an existing methodology or the literature whereas a diagonally marked one represents a technique enhanced from an existing methodology or the literature.



source: developed for this research

Figure 6.11 Work units for Application Lifecycle Analysis

The tasks iterated in this process are:

1. *Identify As-Is Runtime Structure* determines the current or “as-is” model (Salinesi et al. 2004) of an application at runtime. This information forms the basis from which dynamic changes can be identified and articulated.
2. *Derive Change Cases* determines the change cases from requirements and change requests to the application, and records them in a Dynamic Application Change Document (cf. Appendix C.2.2.2).
3. *Extend Application Lifecycle* introduces new generations to an application lifecycle to progressively accommodate dynamic changes (specified as change cases) for the application. The updated application lifecycle is recorded in an Application Lifecycle Diagram (cf. Appendix C.2.2.1).

Table 6.5 summarises the technique(s) used and the circumstance(s) in which they should be used in various tasks of Application Lifecycle Analysis.

Table 6.5 Techniques used in Application Lifecycle Analysis

Task	Relevant Technique	Technique Purpose	Usage Criteria	Source of Reuse/ Enhancement
Identify As-Is Runtime Structure	Runtime Structure Recovery	Recover the information about the structure of a running application.	Optional - used when the detailed information about the application structure is unavailable or out-of-date.	Huang et al. (2006), Schmerl et al. (2006) (reused)

<i>Task</i>	<i>Relevant Technique</i>	<i>Technique Purpose</i>	<i>Usage Criteria</i>	<i>Source of Reuse/ Enhancement</i>
Derive Change Cases	Change Case Modelling	Express change cases for an application.	Mandatory	Salinesi et al. (2004) (enhanced)
Extend Application Lifecycle	Change Case Partitioning and Ordering	Divide a set of change cases into separate groups of logical and coherent change cases, and order the groups appropriately.	Recommended - used when change cases are complex and the number of change cases is high such that dividing them into smaller groups would assist in the identification of transitional periods.	

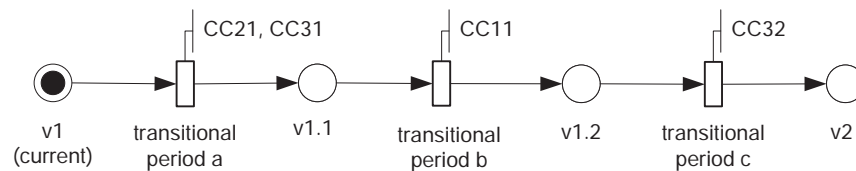
Example: To illustrate this process, consider the first objective which aims to obtain the “as-is” runtime structure of EPCP (i.e. using Task “Identify As-Is Runtime Structure”) as shown earlier in Figure 6.2. Next, the two enhancement requirements stated earlier in Section 6.3.1, being the i18n capability and integration with external EPC systems for EPCP, need to be translated into change cases (i.e. using Task “Derive Change Cases”). According to the architecture (Figure 6.2), these requirements mean the Catalogue Service is required to handle catalogues in different currencies and languages and to plug-in two external EPC systems. In the Web Zone, WebUI will be extended for end users to enter search criteria for and display catalogue items in different currencies and languages. The resulting change cases are specified in Table 6.6.

Table 6.6 EPCP: key change cases

<i>Change Case ID</i>	<i>Purpose</i>	<i>Description</i>	<i>Related Requirement</i>
CC11	Add i18n support to WebUI	WebUI extended to display a variety of products in different units of currency and languages.	i18n capability
CC21	Modify Search Service’s binding with old Catalogue Service to new Catalogue Service	Search Service modified to interact with the new Catalogue Service instead of the old one.	Ditto
CC31	Add i18n support to Catalogue Service	Catalogue Service extended to support catalogue data in a variety of currencies and languages.	Ditto
CC32	Add Catalogue Service bindings with two external EPC systems	Catalogue Service modified to integrate with two external EPC systems operated by business partners.	Integration with external EPC systems

Next, the application lifecycle of EPCP is extended (i.e. in Task “Extend Application Lifecycle”) to incorporate these change cases. To determine which change cases should be accomplished first, an analysis of the nature and purpose of the change cases is given. Since the two external EPC systems will hold catalogue items in a

foreign language, the EPCP application must support i18n before it is integrated with the EPCs (i.e. both CC31 and CC21 realised first). On the other hand, before the Catalogue Service serves catalogue items from external EPCs to WebUI, WebUI must be able to display them in foreign languages/currencies. This analysis suggests a sequential order in which the changes cases should be realised: from CC21/CC31 to CC11 and then followed by CC32. The resulting application lifecycle, taking into account this constraint, is shown in Figure 6.12 (notations discussed in Appendix C.2.2.1). The lifecycle designates three transitional periods - noted as “a”, “b” and “c” - to roll out the annotated change cases. Generation v2 on the right represents the final outcome of dynamic evolution, involving two intermediate generations: v1.1 (enhanced with the i18n capability) and v1.2 (enhanced with foreign currency and language support).

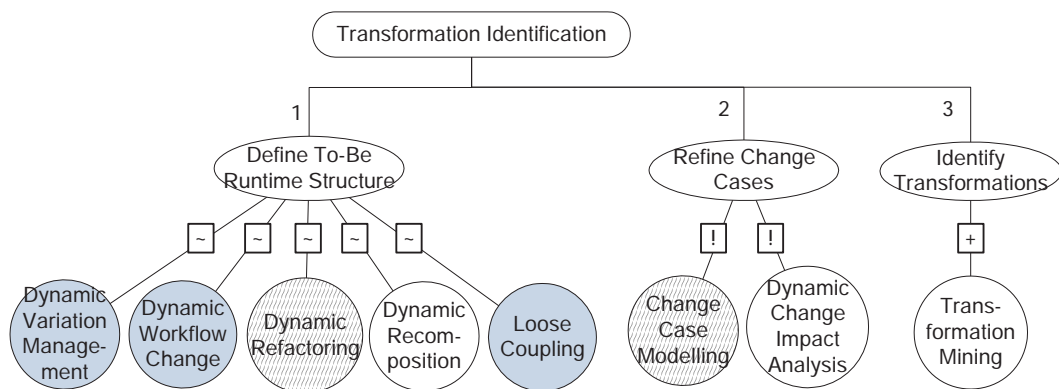


s
source: developed for this research

Figure 6.12. EPCP: application lifecycle diagram

6.3.4.2 Fragments for Transformation Identification

The Transformation Identification process aims at identifying a set of transformations to execute during a transitional period to advance an application from an “as-is” generation to a “to-be” generation that will accommodate the proposed changes for the “as-is” generation.



source: developed for this research

Figure 6.13. Work units for Transformation Identification

Figure 6.13 depicts the tasks iterated and techniques used in Transformation

Identification (using the same notations as in Figure 6.11). The tasks are:

1. *Define To-Be Runtime Structure* determines the expected or “to-be” model (Salinesi et al. 2004) of an application at runtime after proposed changes are accommodated into the application. The “to-be” runtime structure is derived from the static design of the “to-be” generation, the runtime structure of the “as-is” generation, and the change cases identified during analysis.
2. *Refine Change Cases* performs updates to the change cases defined during analysis. This Task also aims to uncover any potential change cases that could not be determined during analysis until the “to-be” runtime structure has been defined.
3. *Identify Transformations* determines the transformations to realise a set of change cases by progressing the application from the “as-is” into the “to-be” generation.

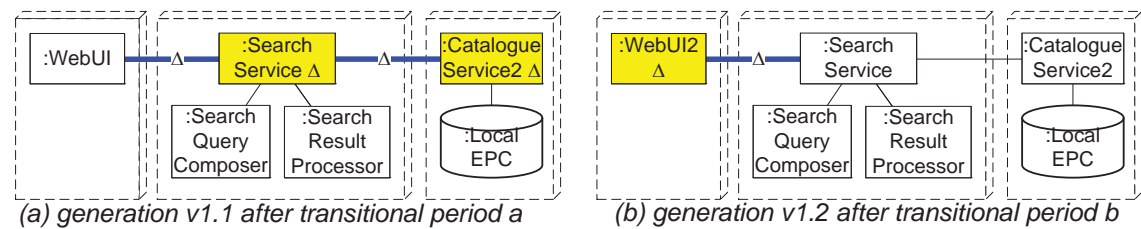
Table 6.7 summarises the techniques used and the circumstances in which they should be used in various tasks of Transformation Identification.

Table 6.7 Techniques used in Transformation Identification

Task	Relevant Technique	Technique Purpose	Usage Criteria	Source of Reuse / Enhancement
Define To-Be Runtime Structure	Dynamic Refactoring	Refactor a runtime structure, such as for the case of performance improvement, without functional changes.	Optional - used when transformations during a transitional period aim to improve an application.	OPF methodology (OPFRO 2009) (enhanced)
	Dynamic Recomposition	Compose several transformable items in a larger unit, decompose a larger one into smaller units, and reconfigure the structure at runtime.	Optional - used when transformations during a transitional period aim to recompose an application.	
	Dynamic Workflow Change	Dynamically change a running workflow.	Optional - used when a workflow is required to evolve.	Casati et al. (1998), Tosic et al. (2007), Ellis and Keddara (2000) (reused)
	Dynamic Variation Management	Define customisation points in a structure to plug in or swap different transformable items to support limited variations in functionality.	Optional - used when transformations during a transitional period aim to use variation points to alter an application.	SeCSE methodology (SeCSE 2007) (reused)

<i>Task</i>	<i>Relevant Technique</i>	<i>Technique Purpose</i>	<i>Usage Criteria</i>	<i>Source of Reuse / Enhancement</i>
	Loose Coupling	Reduce coupling among transformable items.	Optional - used as supplement to Dynamic Refactoring to further improve the flexibility of an application structure to evolve.	P&H methodology (Papazoglou & van den Heuvel 2006) (<i>reused</i>)
Refine Change Cases	Change Case Modelling	Capture newly uncovered dynamic changes.	Mandatory	Salinesi et al. (2004) (<i>enhanced</i>)
	Dynamic Change Impact Analysis	Determine the scope of dynamic changes.	Mandatory	
Identify Transformations	Transformation Mining	Determine transformations from change cases.	Recommended - used when a transitional period is complex, requiring the partition of work into several transformations, or when reuse of transformation patterns is sought.	

Example: In the EPCP example, it has been determined from the Application Lifecycle Analysis process that there are three transitional periods (Figure 6.12) and thus three “as-is”-and-to-be generation pairs: v1-v1.1, v1.1-v1.2 and v1.2-v2. The first task in this process (i.e. Task “Define To-Be Runtime Structure”) is to determine the runtime structures of these “to-be” generations. Generation v1 denotes the current design of EPCP and is shown earlier in Figure 6.2. The design for generation v2 is undertaken with conventional design activities. So it remains to derive generations v1.1 and v1.2. Given that change cases CC21, CC31 and CC11 aim to add the i18n capability to EPCP, affected transformable items (i.e. Catalogue Service and WebUI) require upgrade. Consequently, the generations to realise the upgrade are depicted in Figure 6.14, with changes from previous generations highlighted and labelled with the delta symbol “Δ”, as per Continuum’s Structural Configuration - Notational Extensions (cf. Appendix C.2.2.8).



source: developed for this research

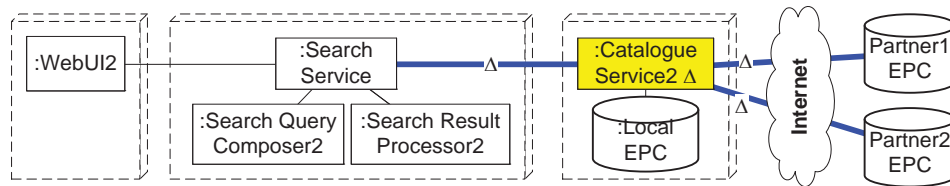
Figure 6.14 EPCP: to-be generations after transitional periods a and b

After the “to-be” generation has been defined, the gaps between the “as-is” and “to-be” generations can now be finalised (not possible during the Application Lifecycle Analysis process when the “to-be” generation was undefined), to uncover additional change cases to be incorporated into the set (i.e. using Task “Refine Change Cases”). Furthermore, the ripple effect of the proposed changes can also be scrutinised for additional change cases (cf. Technique “Dynamic Change Impact Analysis”, Appendix C.3.2.3) to account for parts also affected by the proposed changes. Take the case of “CC31: Add i18n support to Catalogue Service”. An analysis reveals that since the Catalogue Service will provide pricing information in different currencies and product descriptions in different languages, the Search Query Composer must be able to take the currency unit as a parameter in addition to a price range for a search. Similarly, the Search Result Processor will also be modified. This results in two new change cases as shown in Table 6.8:

Table 6.8 EPCP: impact set for change case CC1

Originating Change Case	Target	Impact Type	Level of Disruption	Supplementary Change Case(s)
CC31: Add i18n support to Catalogue Service	Search Query Composer	direct	high	CC22: Add i18n support to Search Query Composer
	Search Result Processor	direct	high	CC23: Add i18n support to Search Result Processor

Consequently, the runtime structure for generation v1.1 is updated to accommodate these two change cases. The same analysis is performed for the other change cases (CC11 and CC32) but yields no further change cases. Accordingly, the resultant generation v2 is now finalised as in Figure 6.15, with changes with generation v1.2 highlighted and labelled with “Δ”.



source: developed for this research

Figure 6.15 EPCP: generation v2 after transitional period c

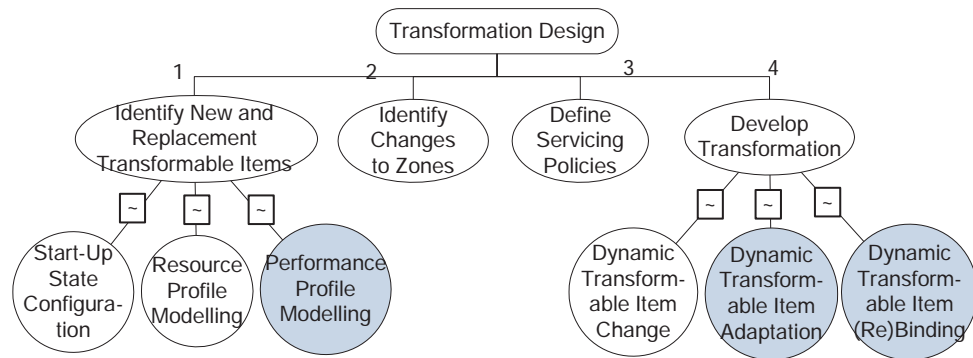
After Task “Refine Change Cases” has been completed, transformations are identified with Task “Identify Transformations”. For simplicity and illustration only, each change case will be realised by one or more transformations as listed in Table 6.9. Note that an extra transformation called “WebUI reconfiguration” is added for change case CC11 to handle existing web (browser) sessions with WebUI. It is further elaborated during the Transformation Design process.

Table 6.9 EPCP: responsible transformations for refined change cases

<i>Change Case</i>	<i>Responsible Transformation(s)</i>	<i>Designated Transitional Period</i>
CC31: Add i18n support to Catalogue Service	Catalogue Service2 deployment Catalogue Service removal	a
CC21: Modify Search Service’s binding with old Catalogue Service to new Catalogue Service	Search Service reconfiguration	a
CC22 Add i18n support to Search Query Composer	Search Query Composer2 deployment Search Query Composer removal	a
CC23: Add i18n support to Search Result Processor	Search Result Processor2 deployment Search Result Processor removal	a
CC11: Add i18n support to WebUI	WebUI2 deployment WebUI and WebUI2 reconfiguration WebUI removal	b
CC32: Add Catalogue Service bindings with two external EPC systems	Catalogue Service2 reconfiguration	c

6.3.4.3 Fragments for Transformation Design

The Transformation Design process aims to produce the detailed design for a transformation. It also identifies changes to zone configurations, say, to accommodate new and relocated transformable items, and specifies how transformable items affected by the transformation will offer their functions or services during and/or after the transformation.



source: developed for this research

Figure 6.16. Work units for Transformation Design

Figure 6.16 depicts the tasks iterated and techniques used in Transformation Design.

The tasks are:

1. *Identify New and Replacement Transformable Items* determines which transformable items are added to an application after a transformation.
2. *Identify Changes to Zones* determines changes to zone configurations and keep them current to support changes to transformable items (e.g. hosting new ones). These changes are recorded in a Zone Change Document (cf. Appendix C.2.2.11).
3. *Define Servicing Policies* determines the policies to regulate the services provided by transformable items (cf. Appendix C.2.1.11) that are impacted by transformations. For instance, when a transformable item is being upgraded, access to its services may be temporarily suspended.
4. *Develop Transformation* details the transformation actions to realise a transformation. The design of the transformation is captured in a Transformation Diagram (cf. Appendix C.2.2.9).

Table 6.10 summarises the techniques used and the circumstances in which they should be used in various tasks of Transformation Design.

Table 6.10 Techniques used in Transformation Design

<i>Task</i>	<i>Relevant Technique</i>	<i>Technique Purpose</i>	<i>Usage Criteria</i>	<i>Source of Reuse / Enhancement</i>
Identify New and Replacement Transformable Items	Performance Profile Modelling	Analyse and predict performance characteristics of new/replacement transformable items.	Optional - used when performance is a key requirement for an application. Performance profiles are captured in a New and Replacement Transformable Item Catalogue.	ASG methodology (Lehner et al. 2006) (reused)

<i>Task</i>	<i>Relevant Technique</i>	<i>Technique Purpose</i>	<i>Usage Criteria</i>	<i>Source of Reuse / Enhancement</i>
	Resource Profile Modelling	Analyse and predict the resource needs of new/replacement transformable items.	Optional - used when resources are limited. Resource profiles are recorded in a New and Replacement Transformable Item Catalogue (cf. Appendix C.2.2.6).	
	Start-up State Configuration	Determine states from which new/replacement transformable items start to operate after they have been placed into an application.	Optional - used when transformable items have the notion of states. If a replacement transformable item derives such a start-up state from its replacing transformable item, this will be specified with a State Map (cf. Appendix C.2.2.7).	
Develop Transformation	Dynamic Transformable Item Adaptation	Wrap and plug transformable items into an architecture at runtime.	Optional - used when transformable items have incompatible interfaces with its hosting application.	RUP methodology (Kruchten 2003) (<i>reused</i>)
	Dynamic Transformable Item (Re)binding	To (re)bind a transformable item to a composition of transformable items at runtime.	Optional - used when transformable items are required to be (re)bound to an application.	SeCSE methodology (SeCSE 2007) (<i>enhanced</i>)
	Dynamic Transformable Item Change	Provide design patterns to add/replace/remove transformable items at runtime.	Optional - used when a transformation adds/replaces/removes transformable items.	

Example: To demonstrate this process using EPCP, consider the transformations for EPCP listed in Table 6.9. (Normally, each transformation is separately developed by executing the Transformation Design process. For illustration only, however, they are discussed together in the following text.) The transformable items added to EPCP by their respective transformations are identified in Task “Identify New and Replacement Transformable Items”. They are summarised in Table 6.11. Next, an analysis is performed (i.e. Task “Identify Changes to Zones”) to determine changes to the zones to accommodate new and replacement transformable items. The right hand columns of Table 6.11 summarise the deployment of new and replacement transformable items to various zones and the removal of old ones from these zones by respective transformations. Since all transformable items will reside in existing zones, there is no need for creating new zones. On the other hand, the Repository Zone must be set up (e.g. opening network ports) to permit the new version of Catalogue Service (i.e. “Catalogue Service2”) to have secure and remote access to the two external EPC systems. This change requirement is also noted in Table 6.11.

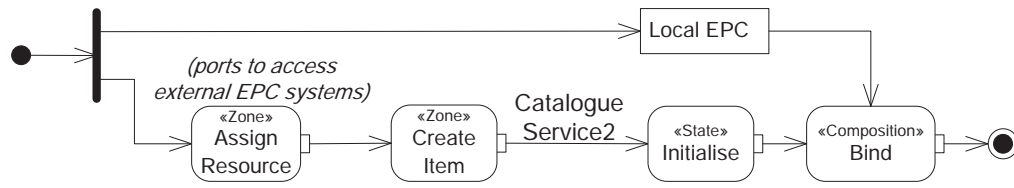
Table 6.11 EPCP: changes to zones

Transformation	New and Replacement Transformable Item(s)	Changes to Zone (+: addition, -: removal)		
		Web	Search Engine	Repository
Catalogue Service2 deployment	Catalogue Service2			+Catalogue Service2
Catalogue Service removal				-Catalogue Service
Search Service reconfiguration				
Search Query Composer2 deployment	Search Query Composer2		+Search Query Composer2	
Search Query Composer removal			-Search Query Composer	
Search Result Processor2 deployment	Search Result Processor2		+Search Result Processor2	
Search Result Processor removal			-Search Result Processor	
WebUI2 deployment	WebUI2	+WebUI2		
WebUI and WebUI2 reconfiguration				
WebUI removal		-WebUI		
Catalogue Service2 reconfiguration				secure and remote access to external EPC systems

Before defining appropriate servicing policies to individual transformable items during transformations are defined (i.e. using Task “Define Servicing Policies”). When doing so, it is useful to review the nature of the transformations. In essence, the transformations can be classified into three types: deployment, removal and reconfiguration. For the deployment transformation type, the design is straightforward since it does not interfere with the normal operations of EPCP and no specific servicing policies are needed. Then, Continuum’s “add” transformation pattern (cf. Appendix C.3.2.7) in which the steps are summarised below, can be used as the design for this type of this transformation (produced from Task “Develop Transformation”):

1. Assign adequate resources from the hosting zone to X (i.e. transformable item).
2. Create X in the hosting zone.
3. Initialise X to a start-up state.
4. Bind X to appropriate transformable items.

For instance, the transformation “Catalogue Service2 deployment” can be diagrammatically represented, as illustrated in Figure 6.17, using Continuum’s notation for transformation design (Appendix C.2.2.9).

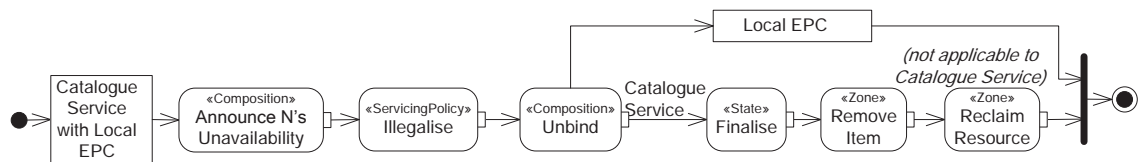


source: developed for this research

Figure 6.17 EPCP: applying deployment transformation pattern to Catalogue Service2

Likewise, the design for a removal transformation for all cases in EPCP is adapted from Continuum’s “removal” transformation pattern (Appendix C.3.2.7) (i.e. from Task “Develop Transformation”). Transformation actions 3 to 6 in the sequence below (see also Figure 6.18) have the reverse effect of the transformation actions in the deployment transformation:

1. Announce that X (i.e. transformable item) will no longer be available for use.
2. Disable access to X.
3. Unbind X from the rest of the application.
4. Set X to a shutdown state.
5. Destroy X in the hosting zone.
6. Release resources allocated to X to the hosting zone.



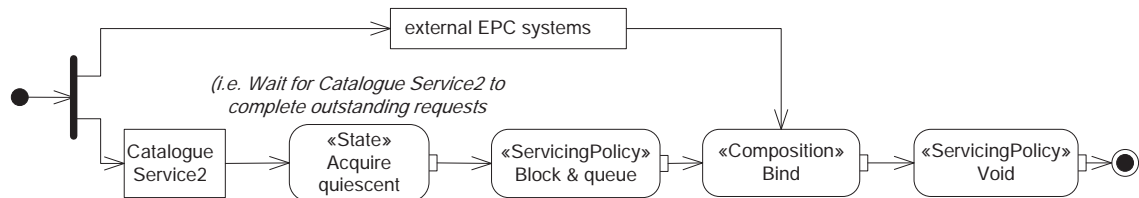
source: developed for this research

Figure 6.18 EPCP: applying removal transformation pattern to Catalogue Service

On the other hand, thought must be given to the rest of the transformations since they serve slightly different purposes. To begin with, the “Catalogue Service2 Reconfiguration” transformation aims to reconfigure Catalogue Service2, the new version of the Catalogue Service, to integrate it to the two external EPC systems. For data integrity, Catalogue Service2 needs to be temporarily out of service while it is being integrated with these two EPC systems. A simple strategy is to momentarily queue incoming search requests to Catalogue Service2 (cf. “blocked and queued” servicing policy, Appendix C.2.1.11), and hence the following sequence (also illustrated in Figure 6.19) (which is produced from Task “Develop Transformation”):

1. Wait for Catalogue Service2 to complete outstanding search requests.
2. Block and queue incoming requests to Catalogue Service2.

3. Bind Catalogue Service2 to two EPC systems.
4. Unblock incoming requests for Catalogue Service2.



source: developed for this research

Figure 6.19 EPCP: transformation design for “Catalogue Service2 Reconfiguration”

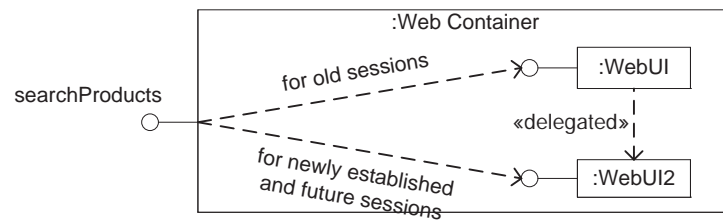
Similarly, the “Search Service reconfiguration” transformation can adopt the same strategy (also with the “blocked and queued” servicing policy) but with an extra step before Step 3 above (also produced from Task “Develop Transformation”):

1. Wait for the Search Service to complete outstanding search requests.
2. Block and queue incoming requests to the Search Service.
3. Unbind the Search Service from the old version of the Search Query Composer and the Search Result Processor.
4. Bind the Search Service to the new version of the Search Query Composer and the Search Result Processor.
5. Unblock incoming requests for the Search Service.

Lastly, the “WebUI and WebUI2 Reconfiguration” transformation is more complicated. At any time there may still be end users accessing EPCP and hence the presence of several web sessions. It is not ideal to switch EPCP over to the new user interface (i.e. WebUI2) instantly since end users may be confused with the spontaneity of the change in the user interface. A suitable strategy is to keep WebUI handling existing web sessions while all new and future web sessions are handled by WebUI2. Over time, all sessions served by WebUI will time out or be closed by end users. After that, WebUI is no longer required and can be removed from EPCP (to be handled by a removal transformation). Therefore, a delegation strategy is useful to this transformation (cf. “delegated” servicing policy, Appendix C.2.1.11) (which is an outcome of Task “Define Servicing Policies”). The corresponding sequence of transformation actions is thus (which is an outcome of Task “Develop Transformation”):

1. Redirect all incoming requests from new web sessions to WebUI2. The delegation mechanism can be implemented in the web container which hosts the old and new versions of the WebUI. This is depicted in Figure 6.20.
2. Wait for all web sessions served by WebUI to close or expire.

3. Disable access to WebUI.

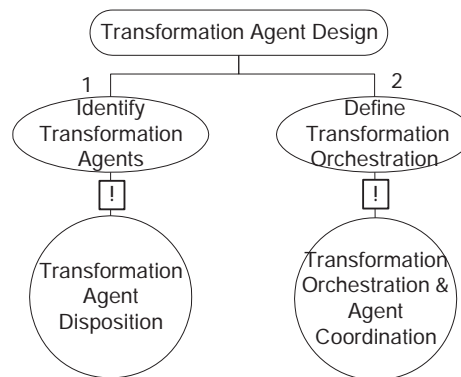


source: developed for this research

Figure 6.20 EPCP: delegating different search requests to WebUI and WebUI2

6.3.4.4 Fragments for Transformation Agent Design

The Transformation Agent Design process identifies the transformation agents to progress a particular generation (i.e. “as-is”) of an application to the next (i.e. “to-be”) during a transitional period, their responsibilities in terms of which transformations they perform, and the order in which they perform the assigned transformations.



source: developed for this research

Figure 6.21. Work units for Transformation Agent Design

Figure 6.21 depicts the tasks iterated and techniques used in Transformation Agent Design. The tasks are:

1. *Identify Transformation Agents* determines the transformation agents required to perform a set of transformations identified during a transitional period, to progress an application from a particular generation to the next.
2. *Define Transformation Orchestration* arranges transformations to be carried out (e.g. sequentially) during a transitional period and assigns them to the agents.

Table 6.12 summarises the techniques used and the circumstances in which they should be used in various tasks of Transformation Agent Design.

Table 6.12 Techniques used in Transformation Agent Design

<i>Task</i>	<i>Relevant Technique</i>	<i>Technique Purpose</i>	<i>Usage Criteria</i>	<i>Source of Reuse / Enhancement</i>
Identify Transformation Agents	Transformation Agent Disposition	Station transformation agents in different zones ready for transformations.	Mandatory	N/A
Define Transformation Orchestration	Transformation Orchestration and Agent Coordination	Design the orchestration of transformations and assign them to transformation agents which will coordinate with one another to carry out those transformations.	Mandatory	N/A

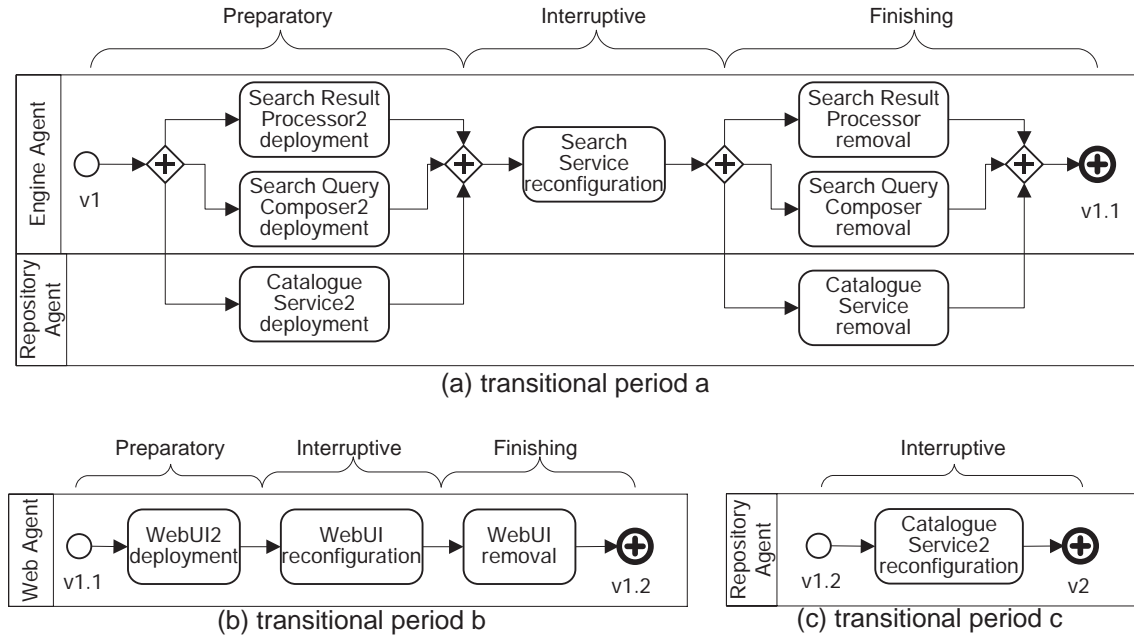
Example: Continuing on the EPCP example, transformation agents are first allocated in the architecture (i.e. using Task “Identify Transformation Agents”). A logical way is to put a transformation agent in each of the zones (Figure 6.2) to handle transformations within that zone: Web Agent, Engine Agent and Repository agent.

Next, transformations in each transitional period are assigned to transformation agents and arranged into an orchestration (i.e. in Task “Define Transformation Orchestration”). The orchestration specifies the order of precedence in which the transformations will be performed and can be represented with a “Transformation Orchestration Diagram” (Appendix C.2.2.10). In this regard, a transitional period is split into three phases - *preparatory, interruptive, finishing*¹⁷ - if applicable, to factor out transformations that will not interrupt an application from those that will, to confine the window of interruption (cf. Technique “Transformation Orchestration and Agent Coordination”, Appendix C.3.2.16).

Consider a transformation orchestration diagram for transitional period “a” for the EPCP example, as shown in Figure 6.22(a). During the preparatory phase, new versions of the Search Result Processor, the Search Query Composer and the Catalogue Service are set up ready for use in EPCP since their respective transformations will not affect the running of EPCP. During the interruptive phase, the Search Service will be temporarily out of service to allow it to bind to these new transformable items - leading to interruptions to end users. Finally, in the finishing phase, unused and old versions of these three transformable items can be removed from EPCP since their respective transformations do not affect EPCP. A similar design heuristics is applied to transitional

¹⁷ Li (2009) defines three stages for dynamic evolution based on types of modifications made: installation, transformation and removal. This classification is somewhat restrictive. For instance, state transfer (i.e. from an old transformable item to a replacement one) must only occur in the transformation stage and modifications in this stage are regarded as “instantaneous” i.e. very fast.

periods “b” and “c”, resulting in the Transformation Orchestration Diagrams (cf. Appendix C.2.2.10) shown in Figure 6.22(b) and Figure 6.22(c) respectively.



source: developed for this research

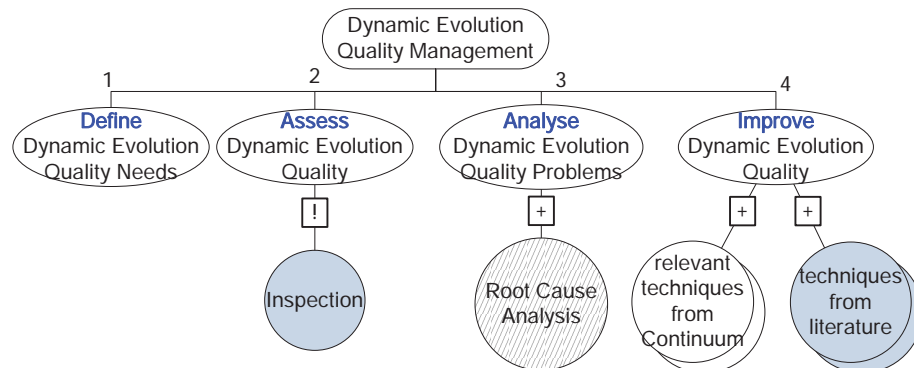
Figure 6.22 EPCP: orchestration designs for transformation agents

6.3.5 Dynamic Evolution Quality Method Fragments

Apart from the ability to accommodate changes to an application, quality is important in dynamic evolution. For example, low-quality changes may not only make the application harder to evolve but also inject live undetected errors into the application such that they may cause the application to fail instantly. For instance, an application may become compromised if a component responsible for protecting the application has been accidentally removed during a transformation. Hence, Continuum provides quality-oriented method fragments to address the quality aspects of dynamic evolution. Dynamic Evolution Quality Management, a process fragment (or process), aims to improve the quality of the work products produced with the four basic dynamic change process fragments: Application Lifecycle Analysis, Transformation Identification, Transformation Agent Design and Transformation Design. This process addresses a portfolio of dynamic evolution quality factors as summarised in Table 6.1.

A quality paradigm typically involves work to be performed in defining, measuring, analysing and improving quality (Fox & Frakes 1997). Likewise, Dynamic Evolution Quality Management is performed in an iterative manner using four tasks as shown in Figure 6.23, covering the define-assess-analyse-improve steps of a quality

improvement effort. Each stacked circle connected to the “Improve Dynamic Evolution Quality” task represents a collection of techniques:



source: developed for this research

Figure 6.23. Work units for Dynamic Evolution Quality Management

The tasks used in this process are:

1. *Define Dynamic Evolution Quality Needs* evaluates which dynamic evolution quality factors are relevant and important to an application at hand and documents the results in a Dynamic Evolution Quality Profile Report (cf. Appendix C.2.2.5).
2. *Assess Dynamic Evolution Quality* discovers quality problems of dynamic evolution work products, using a checklist of inspection questions for the quality factors. Note that the types of work products to assess depend on the quality factors to achieve. The results are recorded in a Dynamic Evolution Quality Inspection Report (cf. Appendix C.2.2.3). Table 6.13 summarises the technique(s) applied in this task.

Table 6.13 Technique(s) used in task Assess Dynamic Evolution Quality

Relevant Technique	Technique Purpose	Usage Criteria	Source of Reuse / Enhancement
Inspection	Evaluate work products in order to identify areas for resolution and improvement	Mandatory	Henderson-Sellers et al. (1998) (reused)

3. *Analyse Dynamic Evolution Quality Problems* determines the underlying causes for the identified quality problems such as defects and/or issues found in the work products. The results are recorded in a Dynamic Evolution Quality Problem Analysis Report (cf. Appendix C.2.2.4). Table 6.14 summarises the technique(s) applied in this task.

Table 6.14 Technique(s) used in task Analyse Dynamic Evolution Quality Problems

Relevant Technique	Technique Purpose	Usage Criteria	Source of Reuse / Enhancement
Root Cause Analysis	Determine the root causes which led to dynamic evolution quality problems.	Recommended	Leszak et al. (2002) (enhanced)

4. *Improve Dynamic Evolution Quality* revises dynamic evolution work products to further improve their quality. As shown in Table 6.15, this Task reuses a number of techniques from existing methodologies and the literature. In addition, Continuum offers several tasks and techniques to address other aspects of the quality factors.

Table 6.15 Recommended tasks/techniques used in Task Improve Dynamic Evolution Quality

Quality Factor	Recommended Task/Technique	Purpose (repeated from Table Appendix C.22)	Dynamic Evolution Quality Specific? [see note 2]	Source of Reuse
Soundness of Change				
Completeness	not applicable [see note 1]			
Consistency	Dynamic Change Impact Analysis	Identify transformable items affected by proposed changes.	no	
	Start-up State Configuration	Declare resuming states to avoid progression towards error states after a transformation.	no	
	Transformable Item Regression Testing	Identify and detect violations of invariants.	yes	SeCSE methodology (SeCSE 2006)
	Resource Profile Modelling	Capture adequate resources and support required for new and replacement transformable items.	no	
Correctness	Change Case Modelling	Capture proposed changes explicitly.	no	
	Define Servicing Policies	Restrict behaviour during transformations.	no	
Infusibility of Change				
Locality	Identify Changes to Zones	Provide guidance on accommodating transformable items in zones.	no	
	Dynamic Change Localisation	Confine changes to within zones.	yes	Evans and Dickman (1999)
Maintainability	Identify Changes to Zones	Keep the information about zones up-to-date.	no	
	Identify New and Replacement Transformable Items	Keep the information about transformable items in an application up-to-date.	no	
	Extend Application Lifecycle	Keep the information about an application lifecycle up-to-date.	no	

<i>Quality Factor</i>	<i>Recommended Task/Technique</i>	<i>Purpose (repeated from Table Appendix C.22)</i>	<i>Dynamic Evolution Quality Specific? [see note 2]</i>	<i>Source of Reuse</i>
	Identify As-Is Runtime Structure	Keep the information about runtime structure of an application up-to-date.	no	
	Define To-Be Runtime Structure	Ditto	no	
	Testability Analysis and Improvement	Improve testability of an application and its transformable items.	yes	Freedman (1991)
Transparency	Transformation Orchestration and Agent Coordination	Reduce the effects of transformations to end users.	no	
	Develop Transformation	Abstract transformations away from the business logic of an application to hide them from its programmers.	no	
	Identify Transformations	Ditto	no	
	Dynamic Change Localisation	Hide transformations from transformable items of an application unaffected by the transformations.	yes	Evans and Dickman (1999)
<i>Changeability of Application</i>				
Autonomy	Transformable Item Autonomy	Improve self-control and self-governance over transformable items.	yes	ERL methodology (Erl 2005)
Coordination	Define Transformation Orchestration	Organise transformation agents to facilitate the orchestration of transformations among multiple zones during a transitional period.	no	
	Secure and Reliable Transformation Agent Coordination	Address network unreliability when transformation agents coordinate with one another during a transitional period.	yes	
Extensibility	Dynamic Wrapper	Add new functionality to existing compositions.	yes	Truyen et al. (2001)
	Dynamic Change Localisation	Use zones to confine the scope of changes.	yes	Evans and Dickman (1999)
	Dynamic Variation Management	Add variation points to an architecture to plug alternative or additional transformable items to it.	no	SeCSE methodology (SeCSE 2006)
	Identify Changes to Zones	Support multiple versions of transformable items to co-exist.	no	
	Dynamic Wrapper	Add new functionality to existing transformable items.	yes	Truyen et al. (2001)

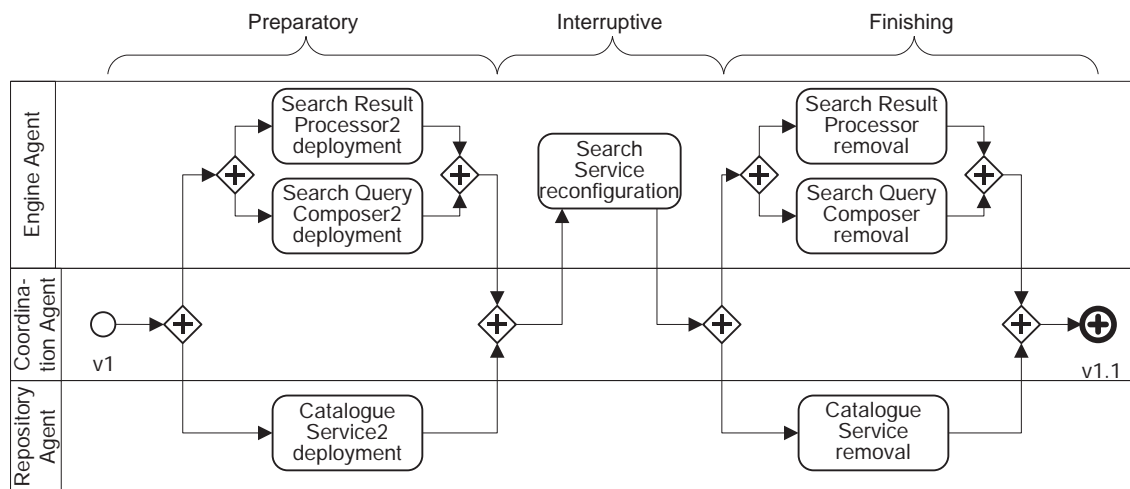
<i>Quality Factor</i>	<i>Recommended Task/Technique</i>	<i>Purpose (repeated from Table Appendix C.22)</i>	<i>Dynamic Evolution Quality Specific? [see note 2]</i>	<i>Source of Reuse</i>
Loose Coupling	Loose Coupling	Reduce coupling among transformable items.	no	P&H methodology (Papazoglou & van den Heuvel 2006)
Separation of Concerns	Transformable Item Mediation and Channelling	Abstract communication concerns from functionality.	yes	RUP (Kruchten 2003)
<i>Robustness of Application</i>				
Fault Tolerance	Dynamic Wrapper	Protect an application against potential faults from new and replacement transformable items.	yes	de Castro Guerra et al. (2003)
	Dynamic Wrapper	Establish barriers to contain the faults from new and replacement transformable items.	yes	Gama and Donsez (2010)
	Dynamic Change Localisation	Confine changes to within zones to isolate faults propagating from one zone to another.	yes	Evans and Dickman (1999)
Recoverability	Transformation Exception Management	Manage potential exceptions raised during the execution of a transformation.	yes	
	Recovery Blocks	Recover an application from errors caused by new and replacement transformable items after a transformation.	yes	Horning et al. (1974)
Reliability	Dynamic Wrapper	Limit the behaviour of new and replacement transformable items in case they behave unexpectedly, compromising an application.	yes	Voas (1998)
	Transformable Item Regression Testing	Check whether or not new and replacement transformable items behave as expected in an application.	yes	SeCSE methodology (SeCSE 2008)
Safety	Dynamic Evolution Safety Risk Management	Identify and mitigate safety risks associated with transformations and dynamic changes.	yes	
Security	the security aspect of Secure and Reliable Transformation Agent Coordination	Provide security for transformation agent control and coordination.	yes	Kon et al. (2000)
	Dynamic Wrapper	Protect the security of an application from being compromised by new and replacement transformable items and vice versa.	yes	Herrmann and Krumm (2001)
	Dynamic Wrapper	Provide access control, intrusion management and data input/output protection for transformable items.	yes	Badger and Feldman (1999)

Quality Factor	Recommended Task/Technique	Purpose (repeated from Table Appendix C.22)	Dynamic Evolution Quality Specific? [see note 2]	Source of Reuse
	Dynamic Security Policy and Enforcement Management	Manage security policy and enforcement changes in response to dynamic changes in an application.	yes	Grimm and Bershad (2001)

Notes:

1. The primary support for Completeness is by Inspection which is carried out in Task *Assess Dynamic Evolution Quality*. No further techniques are specifically defined to further support Completeness.
2. "Dynamic Evolution Quality Specific?": "no" - work units support both dynamic change and quality factor requirements, "yes" - work units support quality factor requirements only.

Example: Continuing on the EPCP example, several transformations are executed by two transformation agents in two zones during transitional period "a" (Figure 6.22(a)). With respect to Coordination, an inspection on the quality of the transformation orchestration design identifies that transformation agents can be further organised to facilitate the coordination of their transformations. This is attained by employing Technique "Secure and Reliable Transformation Agent Coordination" (Appendix C.3.2.11) as recommended in Table 6.15. This dedicates a new transformation agent to initiate and oversee transformations executed by the two agents. The improved design is shown in Figure 6.24 with the new agent labelled as "coordination agent". The improved orchestration starts with deploying the new version of three transformable items to their respective zones. Then, Search Service is reconfigured to use the new version of these transformable items. Finally, old versions of the transformable items are removed.



source: developed for this research

Figure 6.24 EPCP: use of coordination agent during transitional period a

6.3.6 Producer Method Fragments

Producers hold the responsibility of executing work units according to their areas of expertise (ISO/IEC 2007). In other words, the producer-work unit relationship defines who is doing what during software development to tackle dynamic evolution. Correspondingly, Continuum prescribes a set of basic producers as summarised below. The producer-work unit relationship is specified in the documentation of individual work units in Appendix C.3.1:

- A *Dynamic Evolution Analyst* (role-typed producer) is responsible for performing the analysis of dynamic evolution for a composition-based distributed application;
- A *Dynamic Evolution Designer* (role-typed producer) is responsible for carrying out the design of dynamic evolution for a composition-based distributed application; and
- A *Runtime Application Discovery Tool* (tool-typed producer) is a tool used for discovering information about a composition-based distributed application whilst it is running.

Although people and tool related issues are not a focus of this research (cf. Section 1.5), these role- and tool-typed producers are provided for completeness. The producer information is also useful for incorporating Continuum into a methodology tool. One such example is the Eclipse Process Framework (The Eclipse Foundation 2009) which mandates “roles” to be defined for tasks.

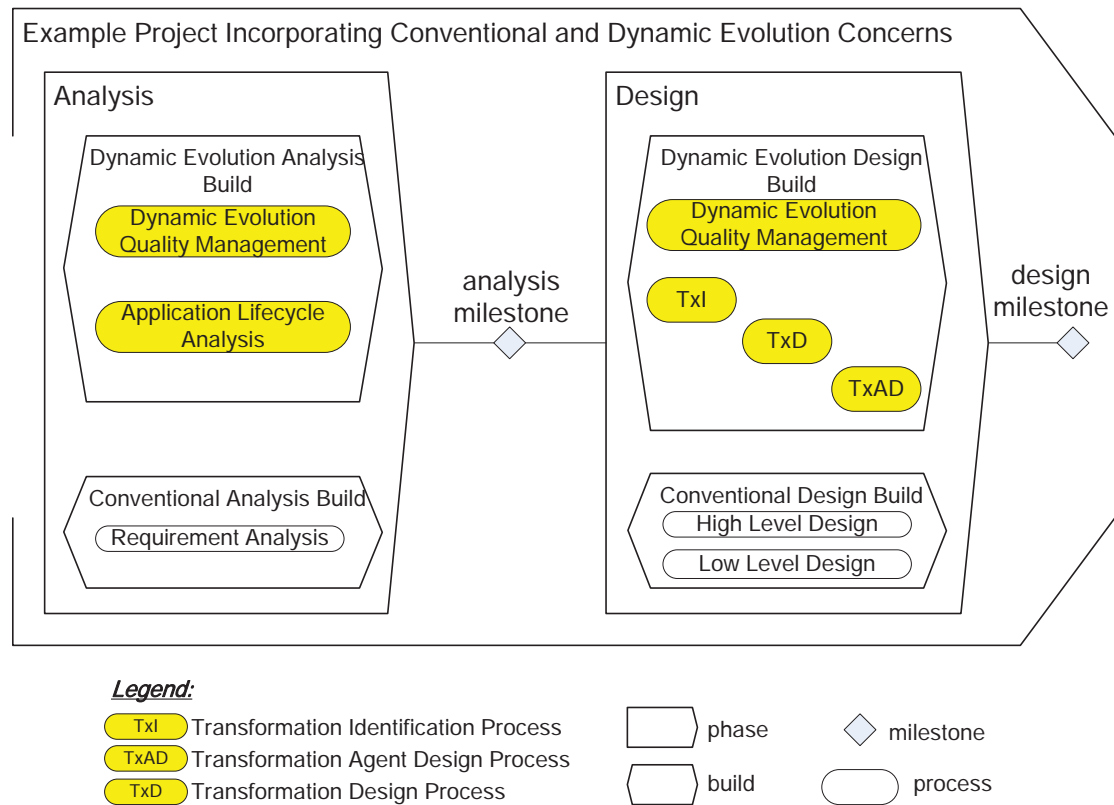
6.3.7 Usage Guidelines

As introduced in Section 6.3.3, the dynamic evolution metamodel (cf. Figure 6.5) provides a basis from which dynamic evolution can be described, expressed and modelled during development (e.g. an application lifecycle is represented by a sequence of generations and transitional periods). In particular, rules can be derived from the metamodel to govern model contents in work products. For example, in an Application Lifecycle Diagram (cf. Appendix C.2.2.1), one would expect to find generations and transitional periods for an application. SEMDM’s *ModelUnitUsage* class instances can be used as surrogates for specifying this type of information (see Appendix C.2.2).

To utilise Continuum in an endeavour, the following steps are suggested:

1. Construct a development lifecycle model which includes Continuum's processes, or incorporate them into an existing development lifecycle model. In the spirit of evolution, the lifecycle model should iteratively and progressively correct, refine and improve analysis and design work products.
2. Determine which Continuum techniques are appropriate for the particular application under consideration (Table 6.5, Table 6.7, Table 6.10 and Table 6.12).
3. Determine which quality factors are important to the business, the stakeholders, end users and the application, and tailor the Dynamic Evolution Quality Management process accordingly. For example, if security is not an issue for an in-house distributed application because it runs in an organisation's Intranet which is secure, security relevant inspection questions can be skipped when assessing the quality of dynamic evolution work products.

Figure 6.25 is one example development lifecycle model, informative as a guide for it is one of many ways to organise, link and sequence stages and processes in a project or programme setting to suit different situations. The lifecycle model has two concurrent streams of development activities, one for conventional application development (lower half) and the other devoted to dynamic evolution (upper half). It begins with the Analysis phase on the left, consisting of two builds, Conventional Analysis and Dynamic Evolution Analysis, each delivering a particular objective for the project. The Conventional Analysis build consists of a conventional Requirement Analysis process sourced from an existing methodology for eliciting, analysing, prioritising and approving requirements and change requests in an endeavour.



source: developed for this research

Figure 6.25. Example analysis and design project lifecycle with Continuum process fragments

The Dynamic Evolution Analysis build consists of Continuum's Application Lifecycle Analysis process which runs alongside Requirement Analysis to take approved requirements and change requests from the latter as inputs, and to plan for the application lifecycle and successive generations necessary to fulfil those requirements and change requests. This process also converts those requirements and change requests into change cases, and defines transitional periods to roll out the change cases. In a quality-aware endeavour, the Dynamic Evolution Analysis build complements this process with Continuum's Dynamic Evolution Quality Management process (top left of Figure 6.25) to continuously and progressively drive quality improvement of dynamic evolution work products produced with Continuum.

During the Design phase, requirements and change requests are inputted to a conventional Application Design process sourced from an existing methodology (as in the Conventional Design build in the lower right of Figure 6.25) which focuses on levelling the static design of the application to meet the new requirements and change requests, with little regard for dynamic evolution concerns. Likewise, change cases are inputted to the Dynamic Evolution Design build (upper right of Figure 6.25), during

which Continuum's Transformation Identification process is used to identify the transformations required to promote the running application to the new runtime structure. Next, one can continue with Continuum's Transformation Design process and then Continuum's Transformation Agent Design process (as sequenced inside the Dynamic Evolution Design build in Figure 6.25). In the former, the detailed design of all transformations is worked out before transformation agents are identified and assigned to these transformations whereas in the latter the execution order of these transformations is defined. Indeed, Transformation Identification, Transformation Design and Transformation Agent Design attend to different design aspects of transformations respectively: *what* is to be done, *how* it is to be done, and *who* is to do it. The Dynamic Evolution Design build is also augmented with Continuum's Dynamic Evolution Quality Management process (top right) to improve the quality of the design work products.

6.4 TRACEABILITY OF REQUIREMENTS AND METHOD FRAGMENTS

Table 6.16 and Table 6.17 show which Continuum's method fragments address each of the important dynamic evolution requirements considered for Continuum. In each row, method fragments in the "Method Fragment" column are those *directly* addressing the respective requirement in the "Requirement" column. Method fragments in the "Supplementary Method Fragment" column supplement those in the "Method Fragment" column to *further* and *indirectly* support their respective requirements. For instance, "ChangeCase" is a feature that represents the notion of a "dynamic change" requirement. But a means of producing and documenting ChangeCases should also be in place in application development. Thus, Continuum proposes Technique "Change Case Modelling" and Work Product "Dynamic Application Change Document" as supplementary fragments for modelling and recording ChangeCases.

Table 6.16 Traceability between important dynamic change requirements (Table 6.2) and Continuum's method fragments

Requirement	Method Fragment(s)	Supplementary Method Fragment(s)
Modelling Related		
<i>Part Level</i>		
Multiple version coexistence	ZoningPolicy used in Task Identify Changes to Zones	Task Identify Changes to Zones
Resource needs	ResourceProfile	New and Replacement Transformable Item Catalogue

<i>Requirement</i>	<i>Method Fragment(s)</i>	<i>Supplementary Method Fragment(s)</i>
Performance characteristics	PerformanceProfile (<i>reused</i> from ASG, cf. New and Replacement Transformable Item Catalogue)	New and Replacement Transformable Item Catalogue
Access blocking	ServicingPolicyType attribute in ServicingPolicy	Transformation Diagram, Structural Configuration - Notational Extensions Task Define Servicing Policies
<i>Application Level</i>		
Dynamic change	ChangeCase	Dynamic Application Change Document Tasks Derive Change Cases and Refine Change Cases Techniques Change Case Modelling and Change Case Partitioning and Ordering
Transformation	Transformation	Transformation Diagram Tasks Identify New and Replacement Transformable Items, Identify Changes to Zones, Define Servicing Policies and Identify Transformations Technique Transformation Mining
Generation	Generation (<i>enhanced</i> from RUP with additional semantics)	Application Lifecycle Diagram Tasks Identify As-Is Runtime Structure, Extend Application Lifecycle and Define To-Be Runtime Structure Technique Runtime Structure Recovery
Application lifecycle	ApplicationLifecycle (<i>enhanced</i> from RUP with the notion of a TransitionalPeriod and additional semantics)	Application Lifecycle Diagram Task Extend Application Lifecycle Technique Change Case Partitioning and Ordering
Servicing continuity	ServicingPolicyType attribute in ServicingPolicy	Transformation Diagram, Structural Configuration - Notational Extensions Task Define Servicing Policies
<i>Others</i>		
Transformation agent	TransformationAgent	Transformation Orchestration Diagram Tasks Identify Transformation Agents and Define Transformation Orchestration Techniques Transformation Agent Disposition and Transformation Orchestration and Agent Coordination
Transformation action	TransformationAction (<i>enhanced</i> from UML's action with various TransformationActionType values)	Transformation Diagram Task Develop Transformation
Transformation exception	TransformationException	Transformation Orchestration Diagram
Transformation exception resolution	TransformationExceptionResolution	Transformation Orchestration Diagram

<i>Requirement</i>	<i>Method Fragment(s)</i>	<i>Supplementary Method Fragment(s)</i>
Expected dynamic change impact	Impact(<i>enhanced</i> from EPIC with additional attributes and refined semantics for dynamic evolution)	Technique Dynamic Change Impact Analysis
Work Related		
Part Level		
Dynamic part change	Technique Dynamic Transformable Item Change	Task Develop Transformation
Dynamic part adapter	Technique Dynamic Transformable Item Adaptation (<i>reused</i> from RUP)	Task Develop Transformation
Dynamic part (re)binding	Technique Dynamic Transformable Item (Re)binding (<i>enhanced</i> from SeCSE with transformation patterns for performing the actual (re)binding at runtime)	Task Develop Transformation
Resource need prediction	Technique Resource Profile Modelling	New and Replacement Transformable Item Catalogue
Performance characteristic prediction	Technique Performance Profile Modelling (<i>reused</i> from ASG)	New and Replacement Transformable Item Catalogue
Geometric change	Task Identify Changes to Zones	Zone Change Document
Dynamic state transfer	Technique Start-up State Configuration	State Map
Application Level		
Dynamic workflow evolution	Technique: Dynamic Workflow Change (<i>reused</i> from several sources)	
Dynamic recomposition	Technique Dynamic Recomposition	
Dynamic refactoring	Technique Dynamic Refactoring (<i>enhanced</i> from OPF)	Technique Loose Coupling (<i>reused</i> from P&H)
Dynamic variability	Technique Dynamic Variation Management (<i>reused</i> from SeCSE)	
Dynamic change impact analysis	Technique Dynamic Change Impact Analysis	Dynamic Application Change Document
Dynamic contract update	Technique Dynamic Change Impact Analysis	

Notes:

1. Not listed in the Table 6.16 are the four process fragments - Application Lifecycle Analysis, Transformation Identification, Transformation Design and Transformation Agent Design - which link and sequence the tasks and techniques (cf. Figure 6.11, Figure 6.13, Figure 6.16 and Figure 6.21) to produce the desired outcomes for dynamic changes.

Table 6.17 Traceability between Continuum quality factor requirements (Table 6.1) and method fragments

Quality Factor and Attribute Requirement	Method Fragment(s)	Supplementary Method Fragment(s)
Soundness of Change		
Completeness		
No missing functionality after a transformation	I+A [see note 1] Inspection questions <i>reused</i> from criteria of RUP's "review the design" task	
No missing parts after a transformation	I+A	
No missing, illegal or broken bindings after a transformation	I+A	
(Also in consistency) assumptions and properties of a distributed application and its parts met by a change	I+A	
Consistency		
Compatible bindings	I+A	
Compatible communications protocol among parts	I+A	
All parts involved in a runtime change identified before a transformation	I+A	Technique Dynamic Change Impact Analysis
No progression towards an error state after a transformation	I+A	Technique Start-up State Configuration, and the notion of a <i>resuming state</i> in State Map
Synchronisation of application's and parts' states after a transformation	I+A	
A reachable state attained after a transformation	I+A	
No critical procedures executed before a transformation	I+A	The notion of a <i>quiescent state</i> in State Map
No pending messages, interactions or transactions before a transformation	I+A	The notion of a <i>quiescent state</i> in State Map
System invariants preserved from a transformation	I+A	Technique Transformable Item Regression Testing
Adequate resources and support for new and replacement parts	I+A	Technique Resource Profile Modelling
(Also in completeness) assumptions and properties of a distributed application and its parts met by a change	I+A	
Correctness		
Non-arbitrary and admissible changes	I+A	Technique Change Case Modelling
No unintentional behaviour during and after a transformation	I+A	Task Define Servicing Policies
Correct ordering of transformations	I+A	
Transformations at a right time	I+A	

<i>Quality Factor and Attribute Requirement</i>	<i>Method Fragment(s)</i>	<i>Supplementary Method Fragment(s)</i>
<i>Infusibility of Change</i>		
<i>Locality</i>		
Application partitioning and change localisation to partitions	I+A	Task Identify Changes to Zones -- Technique Dynamic Change Localisation (<i>reused</i> from Evans and Dickman (1999), in lieu of enhancing a Catalysis feature as originally planned in Table 6.4) [see note 2].
<i>Maintainability</i>		
All parts clearly defined in interaction (or workflow) specifications	I+A	
No degradation in cost and ease of modifications	I+A Inspection questions <i>enhanced</i> from EPIC, OPF and Select Perspective's merged attributes for easily maintainable applications	Tasks Identify Changes to Zones, Identify New and Replacement Transformable Items, Extend Application Lifecycle, Identify As-Is Runtime Structure, and Define To-Be Runtime Structure
No reduction in testability	I+A	Technique Testability Analysis and Improvement (<i>reused</i> from Freedman (1991))
Clear and detailed interactions	I+A	
<i>Transparency</i>		
Transformations hidden from end users	I+A	Technique Transformation Mining
Transformation design and implementation hidden from application programmers	I+A	Tasks Develop Transformation and Identify Transformations
Transformations hidden from parts unaffected by the transformations	I+A	Technique Dynamic Change Localisation (<i>reused</i> from Evans and Dickman (1999))
Transformation agents hidden from operating environment	I+A	
Separating part specification from realisation concerns	I+A	
<i>Changeability of Application</i>		
<i>Autonomy</i>		
Self-control and self-governance of parts' own processing	I+A	Technique Transformable Item Autonomy (<i>reused</i> from ERL)
<i>Coordination</i>		
Transformations coordinated among multiple nodes/organisations	I+A	Task Define Transformation Orchestration
Transformation agents tolerant of network unreliability during a transformation	I+A	Technique Secure and Reliable Transformation Agent Coordination

<i>Quality Factor and Attribute Requirement</i>	<i>Method Fragment(s)</i>	<i>Supplementary Method Fragment(s)</i>
<i>Extensibility</i>		
Runtime extension/upgrade of an application with new functionality	I+A	Techniques Dynamic Wrapper (<i>reused</i> from Truyen et al. (2001)) and Dynamic Change Localisation (<i>reused</i> from Evans and Dickman (1999)).
Runtime extension/upgrade of parts in an application with new functionality	I+A	Technique Dynamic Variation Management (<i>reused</i> from SeCSE) Task Identify Changes to Zones to support multi-version co-existence of transformable items.
Runtime extension/upgrade of an application with new parts	I+A	Techniques Dynamic Wrapper (<i>reused</i> from Truyen et al. (2001))
<i>Loose Coupling</i>		
High level of independence between parts	I+A	Technique Loose Coupling (<i>reused</i> from P&H)
Parts having their own lifecycles and runtime environments	I+A	
<i>Separation of Concerns</i>		
Separating dynamic change concerns from functionality concerns	I+A	
Separating communication concerns from functionality concerns	I+A	Technique Transformable Item Mediation and Channelling (<i>reused</i> from RUP)
Separating security support from functionality concerns	I+A Inspection questions derived (i.e. <i>reused</i>) from ERL's approach to security	
Separating realisations of parts from those of part clients	I+A Inspection questions derived (i.e. <i>reused</i>) from Select Perspective's notion of separate part and client development	
Separating part specification from realisation concerns	I+A Inspection questions derived from analysing several methodologies (cf. Table 6.4)	
<i>Robustness of Application</i>		
<i>Fault Tolerance</i>		
High tolerance of faulty new and/or changed parts	I+A	Technique Dynamic Wrapper (<i>reused</i> from de Castro Guerra et al. (2003))
Barriers established to contain potentially faulty new and replacement parts	I+A	Technique Dynamic Wrapper (<i>reused</i> from Gama and Donsez (2010))

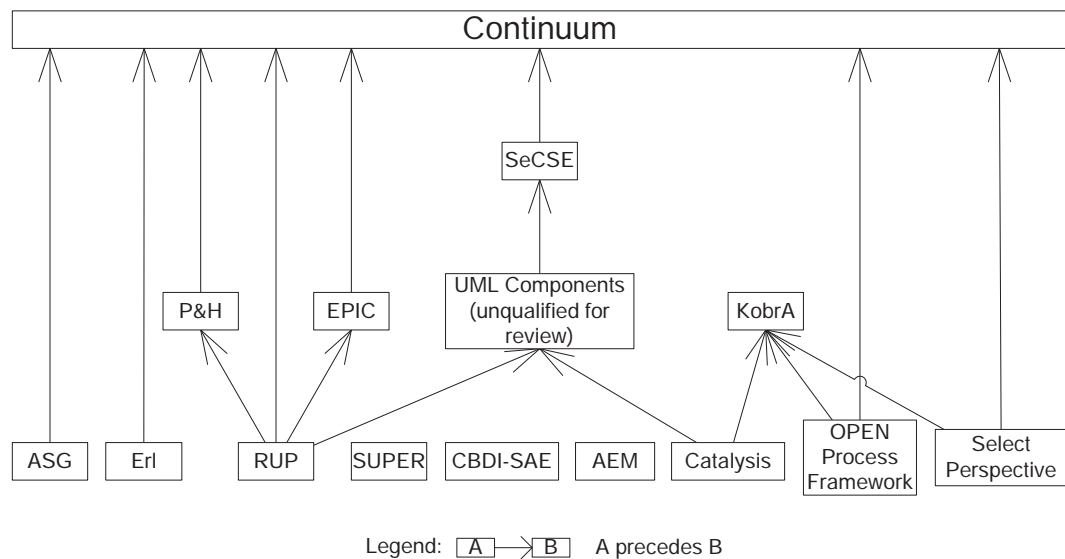
<i>Quality Factor and Attribute Requirement</i>	<i>Method Fragment(s)</i>	<i>Supplementary Method Fragment(s)</i>
<i>Recoverability</i>		
Restoration of an application and its parts to a state to continue to perform their functionality, after a failure caused by a transformation and/or its dynamic change(s)	I+A	Technique Transformation Exception Management and Recovery Blocks (<i>reused</i> from Horning et al. (1974))
<i>Reliability</i>		
No compromise on intended functionality after a transformation	I+A	Technique Dynamic Wrapper (<i>reused</i> from Voas (1998))
Replacement parts fully satisfying their roles	I+A	Technique Transformable Item Regression Testing (<i>reused</i> from SeCSE)
<i>Safety</i>		
Distributed application and its parts operating safely during and after a transformation	I+A	Technique Dynamic Evolution Safety Risk Management (<i>enhancement</i> for OPF's "safety engineering" process)
<i>Security</i>		
Transformation agents secured from unauthorised access	I+A	The security part of technique Secure and Reliable Transformation Agent Coordination (<i>reused</i> from Kon et al. (2000), in lieu of enhancing an OPF feature as originally planned in Table 6.4) [see note 2]
No security compromise by new and replacement parts after a transformation	I+A	Technique Dynamic Wrapper (<i>reused</i> from Herrmann and Krumm (2001), in lieu of enhancing an OPF feature as originally planned in Table 6.4) [see note 2]
Access to new and replacement parts restricted after a transformation	I+A	Technique Dynamic Wrapper (<i>reused</i> from Badger and Feldman (1999), in lieu of enhancing an OPF feature as originally planned in Table 6.4) [see note 2]
Dynamically updated security policy	I+A	Technique Dynamic Security Policy and Enforcement Management (<i>reused</i> from Grimm and Bershad (2001))
Separating security policy from security enforcement	I+A	Ditto

Notes:

1. "I+A" stands for inspection and analysis, referring to Techniques "Inspections" (reused from OPF) and "Root Cause Analysis" (enhanced from Leszak et al. (2002)). These techniques utilise work products Dynamic Evolution Quality Inspection Report (cf. Appendix C.2.2.3) and Dynamic Evolution Quality Problem Analysis Report (cf. Appendix C.2.2.4).
2. Enhancements were originally planned for a small set of fragments identified from existing methodologies to address certain dynamic evolution requirements (Table 6.3 and Table 6.4). Instead, since some techniques fully satisfying some of these requirements were also identified from the literature, the development of Continuum opted for adopting these techniques from the literature rather than performing fragment enhancement.
3. Not listed in Table 6.17 are the process fragment Dynamic Evolution Quality Management (cf. Section 6.3.5) and its tasks which together apply the techniques above in a lifecycle (cf. Figure 6.23) to drive quality improvement for dynamic evolution.
4. Task Improve Dynamic Evolution Quality (cf. Appendix C.3.1.2.4) applies to all quality factors but

omitted from the “Supplementary Method Fragment(s)” column for clarity reasons.

In section 2.4, the precedence relationships for a number of reviewed methodologies have been presented in Figure 2.5. Since Continuum reuses/enhances a number of these methodologies (cf Table 6.16 and Table 6.17), the relationships are extended, incorporating the relationships between Continuum and these methodologies, as depicted in Figure 6.26.



source: developed for this research

Figure 6.26 Precedence relationships of reviewed methodologies and Continuum (extended from Figure 2.5)

6.5 CONCLUSION

An extended introduction to Continuum, a methodological extension to support dynamic evolution, has been presented in this Chapter. It covers:

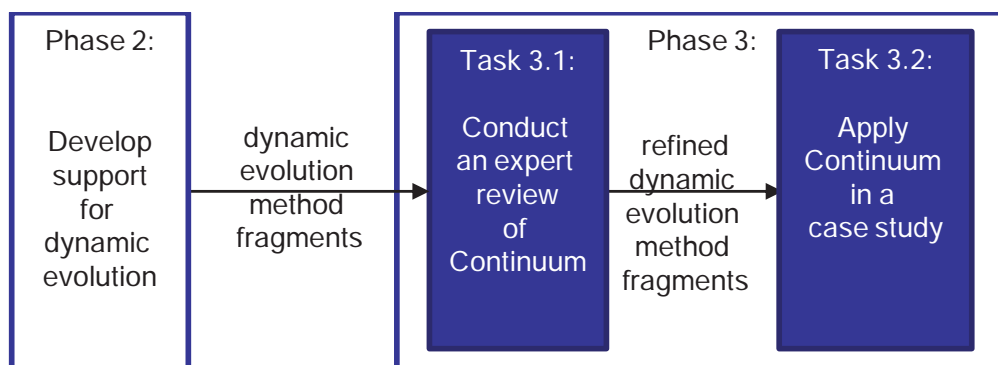
- the metamodel for describing dynamic evolution;
- work unit fragments (what to do) for addressing dynamic changes and quality aspects relevant to dynamic evolution;
- work product fragments (i.e. produced/used in these work unit fragments); and
- producer method fragments (who perform a particular work unit fragment).

The set of important dynamic evolution requirements that Continuum intends to address were briefly described. Some guidelines on using Continuum in an endeavour are also provided. The detailed specifications for Continuum can be found in Appendix C. The next Chapter describes the tasks of evaluations of Continuum and refinements made to it.

Chapter 7. EVALUATION AND REFINEMENT

“The first step towards amendment is the recognition of error.” - Seneca

This Chapter reports the evaluation and refinement of Continuum performed in Phase 3 of this research. In Task 3.1, an expert review of the version of Continuum produced from Phase 2 was used to refine Continuum. In Task 3.2, the version of Continuum updated after Task 3.1 was used in a case study, and subsequently refined based on the feedback from the case study results. Note that all refinements made to Continuum have been included in the version presented in Section 6.3 and Appendix C. Figure 7.1 below summarises the information flow for these tasks.



source: developed for this research

Figure 7.1 Information flow in Phase 3 for evaluating and refining Continuum

Task 3.1 is reported in Section 7.1. This is followed by Task 3.2 in Section 7.2. Section 7.3 concludes this Chapter.

7.1 TASK 3.1 EXPERT REVIEW OF CONTINUUM

The first task of Phase 3 aimed to conduct a preliminary assessment of the initial version of Continuum (i.e. developed from Phase 2, cf. Section 6.3) and refine it accordingly. In this regard, experts were asked to review the documentation of this version of Continuum consisting of two parts: the introduction section (similar to Section 6.3) and the full specifications section (similar to Appendix C). The former gave an in-depth overview of Continuum, its method fragments, how they could be fitted together and used in an endeavour. The latter provided full descriptions of individual fragments in Continuum which were organised according to the fragment types.

7.1.1 Selection of Experts

Two experts, one from the IT industry and the other from academia, took part in the

review. The former had experience in dealing with architectural aspects and implementation of live upgrades to distributed applications, while the latter did researches in dynamic adaptation in service-oriented systems. An advantage of having experts from two different sectors is that their knowledge and experience complemented each other and potentially improved the comprehensiveness of the review findings. Neither expert was aware of the identity of the other to avoid possible communication between them without the researcher's involvement.

7.1.2 Procedure

The review procedure consisted of the following sequential steps. Note that the experts never reviewed Continuum in parallel (i.e. at the same time):

1. Expert 1 reviewed *the initial version* of the documentation of Continuum, completed a review form (Appendix E.3), and if necessary annotated further comments on the documentation.
2. A meeting was held with Expert 1 to discuss the comments on the returned form and the documentation and to clarify and avoid possible misinterpretation of them.
3. The documentation was refined to address Expert 1's comments.
4. The refinements made to Continuum were discussed with Expert 1 via email communications to ensure that Expert 1 was satisfied that the refinements did address his/her comments. If not, go to Step 3.
5. Expert 2 reviewed the *refined version* of the documentation.
6. The clarification-refinement-discussion steps (i.e. 2 to 4) were repeated with Expert 2 using his/her comments.
7. Refinements according to Expert 2's comments were discussed with Expert 1 to ensure Expert 1 was also satisfied with the refinements. If not, further refinements and discussions were performed until both experts were satisfied.

Before commencing the review (i.e. Step 1 or 5), each expert was given the documentation of Continuum and the review form (Appendix E.3). The review form specifically requested each expert to comment on:

- strengths of Continuum;
- areas for improvement in Continuum (i.e. weaknesses); and
- suggestions, if any, for improvement to the areas above.

The review form also noted the size of the documentation (over one hundred and thirty

pages long) and how much time would be expected to complete the review (four to five days). This information served as a heads-up to experts to allocate sufficient time for the review. A deadline was then negotiated with each expert as to when the review form would be returned. Each expert was then sent a reminder close to the deadline.

7.1.3 Results

A summary of major events occurred in this task is shown in Table 7.1, with the longest events being the periods that the two experts spent on their first review of the documentation (two and three months respectively).

Table 7.1 Expert review timeline

<i>Period</i>	<i>Major Event(s)</i>
May to Jun 2009	Expert 1 reviewed initial version of Continuum.
Jul 2009	Expert 1 and Researcher discussed Expert 1's comments. Researcher refined Continuum based on Expert 1's comments. Expert 1 reviewed refinements.
Aug to Oct 2009	Expert 2 reviewed refined version of Continuum.
Nov 2009	Expert 2 and Researcher discussed Expert 2's comments. Researcher refined Continuum based on Expert 2's comments. Expert 2 reviewed refinements.
Nov 2009	Expert 1 reviewed refined version of Continuum and suggested further improvements.
Dec 2009	Researcher further refined Continuum based on Expert 1's further suggestions. Expert 1 reviewed further refinements.
Dec 2009 to Jan 2010	Expert 2 reviewed further refinements.

The experts' comments on the strengths of Continuum are given in Table 7.2. In summary, the strengths were observed from its content, structure, comprehensiveness and novelty in dealing with dynamic evolution.

Table 7.2 Expert comments on strengths of Continuum

<i>Expert</i>	<i>Strengths</i>
1	<ul style="list-style-type: none"> Continuum gives an impression of comprehensiveness and should be able to handle many of the dynamic evolution scenarios that it accounts for. Continuum is precise in its definitions, procedures and usage. It is clear in its definitions of what to do and when to do it. Continuum is prescriptive and offers a structural way for processes and work products to fit together, and a logical way to follow during analysis and design for dynamic evolution.
2	<ul style="list-style-type: none"> The documentation is well-written, detailed and easy to read, follow and understand. The methodology is well-presented and well-organised. The methodology extension of dynamic change identification and management is comprehensive, well-documented and provides links to relevant techniques. The extensions are novel and fill the gap in the existing methodology for the dynamic management of changes in software and services [in SOA environments]. Continuum and its metamodel open new doors and opportunities to works on dynamic changes of services [in SOA environments].

The areas pinpointed by the experts worthy of improvements (i.e. weaknesses of Continuum) are summarised below:

- clarity, completeness and semantics of the metamodel;
- clarity, completeness and relationship of the work units, and terminology used in them;
- notations and semantics of the work products;
- clarity, completeness, sequence, structure and style of the presentation;
- examples used to guide the discussion; and
- the people aspect (e.g. no producers missing in Continuum).

The areas that could be improved, the experts' suggestions on how these areas could be improved as well as the actual refinements made to Continuum to implement their suggestions are documented in Table Appendix F.1. The refinements include changes to the Continuum's method fragments, its level of details and examples in the documentation.

7.2 TASK 3.2 CASE STUDY APPLICATION OF CONTINUUM

After the expert review of the initial version of Continuum and the subsequent refinements made to it in Task 3.1 (cf. Section 7.1), this task was a case study in which the objective was to apply Continuum to the analysis and design of dynamic evolution for an application, and to use the results from it to further refine and improve Continuum. To achieve realism, the case study was managed as a project incorporating project schedules, deadlines and meetings.

An introduction to the case study encompassing the application used and the sponsor is presented in Section 7.2.1. The recruitment of case study participants is noted in Section 7.2.2. This is followed by a description of the case study procedure in Section 7.2.3 and the case study results in Section 7.2.4. A brief discussion on the lessons learned from the case study is given in Section 7.2.5.

7.2.1 Case Description

The Distributed Property Valuation system - also called "DPV" - used in the case study is a commercial and distributed application running around the clock. DPV coordinates activities among financial institutions and property valuers (i.e. individuals and staff of property valuation firms). Financial institutions use DPV to register valuations to be performed on properties of interest. DPV then notifies property valuers who carry out

the valuations on-site and return valuation reports to DPV. DPV checks and forwards completed valuation reports to the financial institutions. DPV adopts the standards proposed by the Lending Industry XML Initiative (LIXI)¹⁸ (LIXI 2010) specifying the formats and exchange of lending-related data which replace numerous incompatible and proprietary approaches.

The organisation that sponsored this case study (i.e. the sponsor) was outsourced to develop DPV for another company (i.e. the owner). The organisation was selected using convenience sampling with a personal contact at the organisation, for reasons of limited availability of industry sponsors and the preliminary nature of Continuum. The first version of DPV, labelled V1, was released in 2008. It consisted of:

- a web-based business process system for property lenders to initiate and track valuation workflows;
- a Mobile Job Application (MJA) running on personal digital assistant devices (PDA) for individual property valuers and staff from property valuation firms (collectively called “valuers”) to capture property valuation data on-site (i.e. the jobs); and
- a desktop sub-system for system administrators to manage property valuation data held in DPV.

The sponsor had been using an in-house agile software development methodology for five years. The methodology was intended for small development teams (e.g. four people for V1). Its major analysis and design activities were:

- *requirements consistency analysis*, reviewing consistency of requirements and the design¹⁹;
- *high-level design*, producing block diagrams for an architecture;
- *low-level design*, producing minimal structural diagrams;
- *technology and platform (off-the-shelf) evaluation and selection*, producing extensive documents;

¹⁸ LIXI is an independent non-profit organisation established for the Australian lending industry to facilitate lending information exchange among its member organisations more efficiently and at a lower cost. LIXI members include major banks, mortgage originators, mortgage brokers, mortgage insurers, property valuation firms, settlement agents, trustees and information technology providers.

¹⁹ Requirements gathering was handled by clients and thus not part of the in-house methodology.

- *effort estimation*, based on function point analysis and development tasks to perform;
- *migration planning*, involving ad hoc brainstorming (e.g. switching from old to new versions of a database); and
- *installation and operation planning*, involving ad hoc brainstorming on potential issues and plans.

Soon after the release of V1, the sponsor developed V2 which incorporated several enhancements to V1 such as a new database platform for storing property valuations and additional functionality. As several parts of DPV in V1 had to be ported to vastly different technology platforms in V2, the sponsor treated the upgrade from V1 to V2 as also a migration project and gave a lot of thoughts to change planning. During both development and installation/operation time, however, the development team had to think about and deal with changes and evolution themselves on an ad hoc basis. This was because the in-house methodology did not provide guidance, such as checklists, steps and patterns, for changes and evolution. During the upgrade from V1 to V2, the sponsor encountered several issues neither foreseen nor adequately addressed by its methodology causing unanticipated delays in completing the upgrade. Thus, the sponsor sought alternatives to deal with future upgrades.

At the time of the case study, a new version of DPV (i.e. V3) had been planned and was still subject to project approval and funding. The sponsor thus suggested applying Continuum to retrospectively re-design the upgrade from V1 to V2 in order to gain experience in dealing with dynamic evolution, for the benefit of future upgrades. Since V2 of DPV had already been designed and implemented, the sponsor decided not to repeat the conventional development activities in the case study project but to reuse what had already been developed from these activities instead.

7.2.2 Selection of Participants

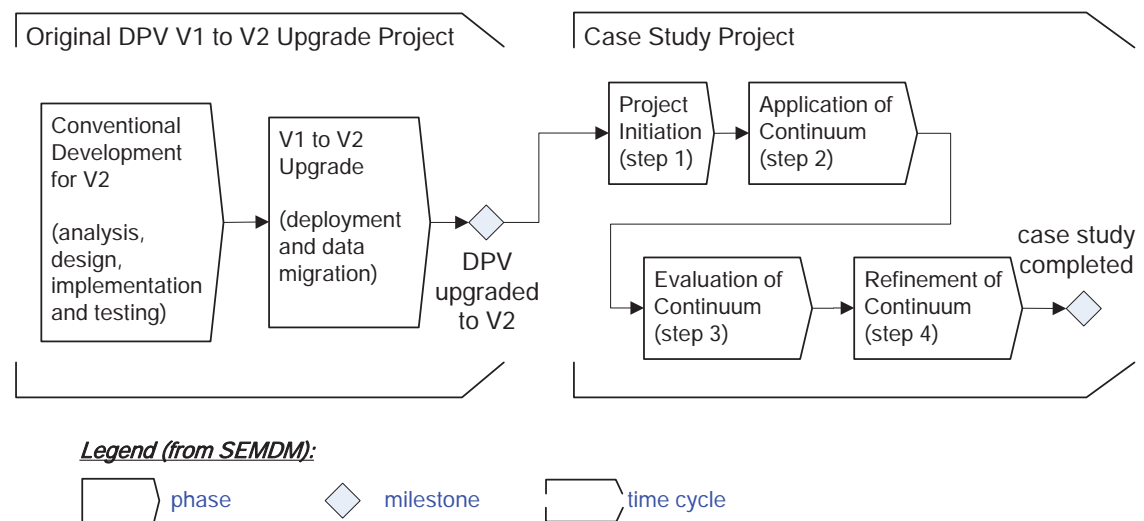
The sponsor organisation assigned two of its technical employees (i.e. the participants) to take part in the case study. One participant had been involved in the whole development lifecycle of DPV - covering analysis, design, implementation, testing and deployment - since its inception (i.e. from V1 onwards). The second participant joined the DPV development team after the V2 release to specifically perform requirements analysis of the latest version (i.e. V3). The latter participant was also familiar with the V1 and V2 features of DPV. Note that the participants were not the experts who reviewed earlier versions of Continuum in Task 3.1 (Section 7.1).

7.2.3 Procedure

The following steps were carried out in this task as described in the research design (cf. Section 3.2.3.2):

1. Initiate the project to kick off the case study.
2. Apply Continuum by analysing and designing dynamic evolution for an application and developing relevant dynamic evolution work products.
3. Evaluate Continuum by completing an evaluation form (Appendix E.4).
4. Wrap-up the project by refining Continuum based on the evaluation results and verifying the refinements made with the case study participants and the experts.

As highlighted in Section 7.2.1, the conventional design and analysis activities for DPV had been completed. Therefore, the project lifecycle for the original upgrade project was extended by incorporating the four steps above to concentrate the case study on the dynamic evolution aspects of DPV from V1 to V2. In this regard, a *lifecycle diagram* (ISO/IEC 2010), as depicted in Figure 7.2, can be used to represent the relationship between the original upgrade project and the case study project.



source: developed for this research

Figure 7.2 Case Study's lifecycle diagram

On the left of Figure 7.2 is the time cycle²⁰ for the original upgrade project which consisted of two phases: one covering the conventional development activities prior to the case study and the other covering the upgrade of DPV to V2.

²⁰ SEMDM defines a “time cycle” as a managed interval of time for the delivery of a product (ISO/IEC 2007).

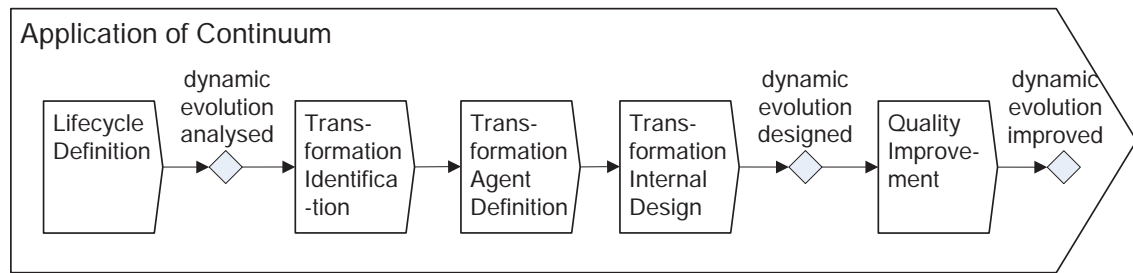
The right hand side of Figure 7.2 shows the time cycle for the case study project which was divided into four phases, corresponding to the execution of the four steps defined for this task. The activities of each phase of the case study project are now discussed.

7.2.3.1 Project Initiation

The case study project commenced with a kick-off meeting to communicate with the participants about the purpose, expectations and potential benefits of the case study. Furthermore, a project plan was structured to align the scope of work in the case study, the workload and involvement of the participants. An overall project schedule along with milestones was established. The schedule imposed deadlines to add realism to the case study. Early milestones let the participants and their sponsor see early results from the case study. In the meeting the participants gave an introductory talk on DPV and were handed the documentation for Continuum to read.

After the meeting, the participants received in-house training on Continuum in two hourly sessions (one week apart from each other). The first session covered topics such as key concepts and features of Continuum, how to navigate its documentation, and a brief introduction to method engineering using Continuum as an illustration. In the second session, examples in Section 6.3 were presented to the participants to showcase the use of Continuum. The presentation also drew on the characteristics of DPV, using the whiteboard where necessary, to demonstrate key concepts of Continuum (e.g. an example change case for DPV). The participants commented that examples were especially important since without a lifecycle model or being integrated into a full methodology, Continuum (alone) as a set of method fragments would be hard to understand.

Before applying Continuum, the participants refined the next phase “Application of Continuum” to guide them when to use particular features of Continuum. A logical approach was to define a sub-phase for each of the five Continuum processes (see Figure 7.3) to focus the sub-phase on particular dynamic evolution aspects supported by the associated process. Three milestones were also defined in “Application of Continuum”: “dynamic evolution analysed”, “dynamic evolution designed” and “dynamic evolution improved”. “Dynamic evolution analysed” signalled the end of the “Lifecycle Definition” sub-phase, during which the participants were expected to perform the analysis of DPV to evolve it from V1 to V2.



source: developed for this research

Figure 7.3 Configuration for case study's "Application of Continuum" phase

The milestone "dynamic evolution designed" flagged the completion of activities defined in the following sub-phases:

1. The "Transformation Identification" sub-phase, for identifying all transformations to advance DPV from V1 to V2;
2. The "Transformation Agent Definition" sub-phase, for identifying transformation agents and assigning them responsibilities (e.g. which agents would execute particular transformations defined in "Transformation Identification"); and
3. The "Transformation Design" sub-phase, for producing the internal design for each transformation identified in "Transformation Identification".

After "dynamic evolution designed", the quality of the dynamic evolution work products developed from prior sub-phases were to be assessed and subsequently improved in the "Quality Improvement" sub-phase. Finally, the milestone "dynamic evolution improved" indicated the completion of the "Application of Continuum" phase.

The sub-phases are further elaborated in Section 7.2.4.1.

7.2.3.2 Applying Continuum

After project initiation, the participants used Continuum to analyse and design dynamic evolution for DPV. Since the case study application of Continuum was expected to run for a few months, they were handed an evaluation form (Appendix E.4) to complete along the way, lest they forgot their experience of applying specific parts of Continuum earlier on. During each week, a meeting was held to keep everyone up-to-date about the case study application, to monitor its status and progress, and to establish work to complete in the following week. The last point involved:

- negotiating what objectives would be accomplished by the next meeting;
- determining which parts of the methodology would be used to accomplish the objectives; and

- getting commitment from participants to the objectives above.

In terms of *evidence of use*, the following materials were tracked in every meeting:

- topics discussed in relation to the dynamic evolution metamodel;
- work unit fragments discussed and used; and
- informal documentation (e.g. sketches on whiteboards) and formal documentation (i.e. work-in-progress and completed work products). (Note: Completed work products developed from using Continuum are documented in Appendix D.)

7.2.3.3 Evaluating Continuum

After applying Continuum to DPV (cf. Section 7.2.3.2), the participants completed and returned the evaluation form. For an early assessment of a development approach, Murphy et al. (1999) suggest focusing on *usefulness* (e.g. “Is it useful when addressing dynamic evolution) and *usability* (e.g. “Is it easy to understand and use?”). Thus, in the evaluation form the participants were asked to rate these aspects of Continuum. Furthermore, they were asked to evaluate its *strengths*, *weaknesses* and *completeness*²¹ by:

- rating the extent to which Continuum tackled each dynamic evolution issue (i.e. strengths and weaknesses);
- identifying areas in Continuum requiring improvement (i.e. weaknesses and completeness); and
- providing suggestions, if any, for improving these areas.

Follow-up face-to-face meetings were held to discuss and clarify the evaluation results in the returned forms. They also gave participants an opportunity to cross-check each other’s data.

7.2.3.4 Refining Continuum

The feedback obtained from the evaluation results was used to determine which parts of Continuum required updates based on the feedback. Refinements were iteratively

²¹ In work related to technology adoption for IS, Yang et al. (2005) propose a number of factors for measuring the service quality of a technology adoption model: “usefulness” of information provided, “usability” of information provided, “adequacy or completeness” of information provided, “accessibility” of IS (e.g. responsiveness), the means of “interaction” with IS, and “privacy and security” of information collected from users. The first three factors were also used in this evaluation. The last three were not applicable to Continuum and thus discarded.

made to Continuum and reviewed by both participants until they were satisfied. Next, this same refinement and review process was repeated with the two subject matter experts who took part in the expert review earlier in Task 3.1 (cf. Section 7.1).

7.2.4 Results

An overview of the major events occurred during the case study is shown in Table 7.3. The case study lasted ten months.

Table 7.3 Case study timeline

<i>Period</i>	<i>Phase</i>	<i>Major Event(s)</i>
Jan 2010	Project Initiation	Kick-off meeting.
Feb 2010		Case study participants attended Continuum training sessions.
Feb to Jun 2010	Application of Continuum	Case study participants performed analysis and design of dynamic evolution for DPV.
Jun 2010	Evaluation of Continuum	Case study participants completed evaluation of Continuum.
Jul to Aug 2010	Refinement of Continuum	Researcher refined Continuum based on participant evaluation results.
Aug 2010		Case study participants reviewed refinements made to Continuum.
Sep 2010		Expert 1 reviewed refinements made to Continuum since Expert 1's last review and suggested further improvements. Researcher further refined Continuum based on Expert 1's suggestions. Expert 1 reviewed further refinements.
Oct 2010		Case study participants reviewed further refinements based on Expert 1's suggestions.
Oct 2010		Expert 2 reviewed all refinements since Expert 2's last review.

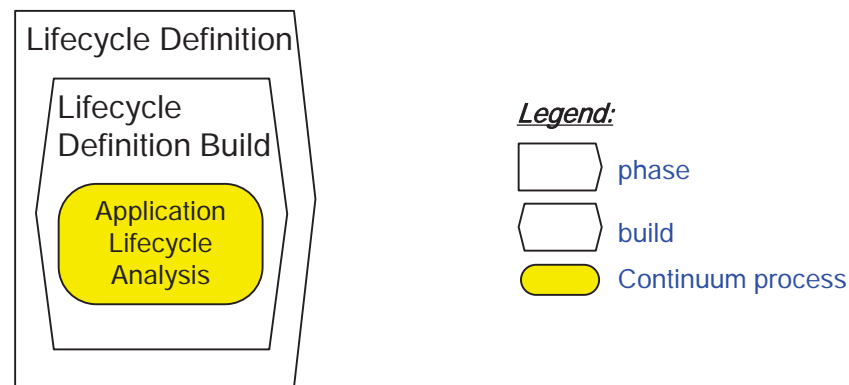
The case study results are documented in the following sections:

- Section 7.2.4.1, for the configuration of the sub-phases in “Application of Continuum” showing how Continuum processes were set up and used for DPV;
- Section 7.2.4.2, for the dynamic evolution issues encountered and the ratings for how well Continuum addressed these issues;
- Sections 7.2.4.3 and 7.2.4.4, for the evaluation results for usefulness and usability of Continuum respectively; and
- Table Appendix F.2, for the areas in Continuum requiring improvement, suggestions for how they could be improved, and the actual refinements made to Continuum to improve them.

7.2.4.1 Sub-Phase Configuration for Application of Continuum

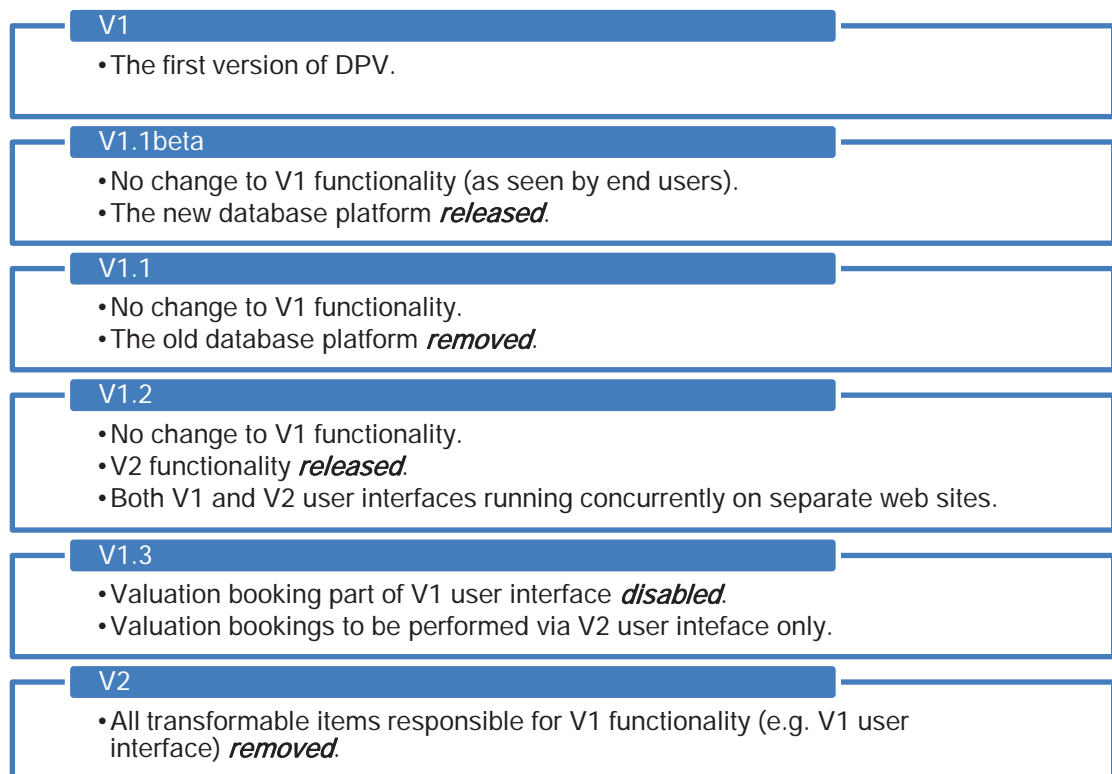
As defined during Project Initiation (cf. Section 7.1), the case study application of

Continuum began with the “Lifecycle Definition” sub-phase, during which the participants had to conduct an analysis of the dynamic evolution aspects of DPV. As discussed in Section 7.2.3.1 the conventional analysis for DPV had already been performed, the configuration for “Lifecycle Definition” was straightforward. That is, during “Lifecycle Definition”, the participants simply followed Continuum’s Application Lifecycle Analysis process without repeating the conventional analysis in a designated time interval called a “build” (ISO/IEC 2007). This is depicted in Figure 7.4.



source: developed for this research

Figure 7.4 Configuration for “Lifecycle Definition” sub-phase

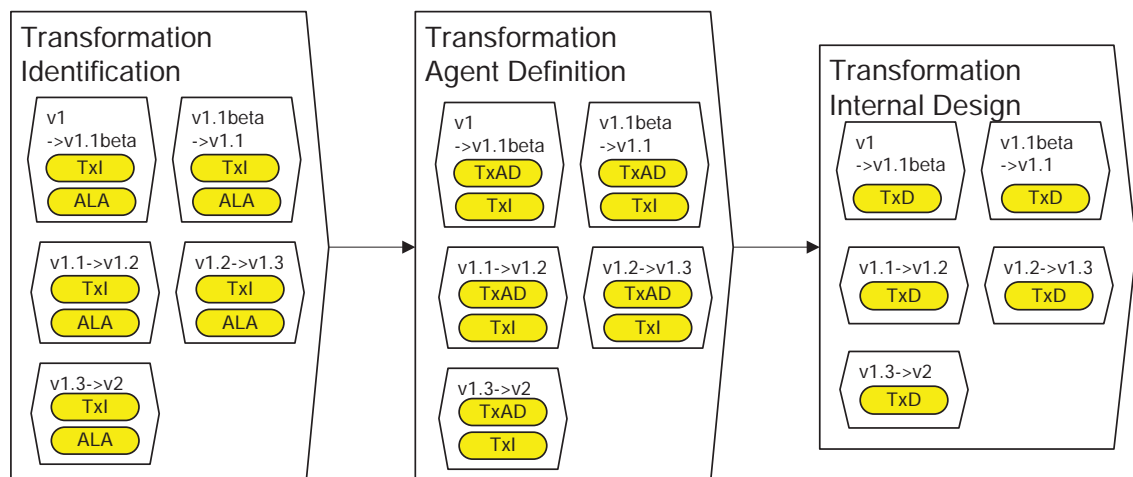


source: developed for this research

Figure 7.5 DPV: generation overview

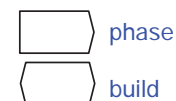
After completing “Lifecycle Definition”, the participants decided that in order to progress DPV from V1 to V2, DPV would be required to pass through four temporary generations (i.e. V1.1beta, V1.1, V1.2 and V1.3). A brief description of the generations from V1 to V2 is provided in Figure 7.5. See Appendix D for further details.

On account of the number of temporary generations between V1 and V2, the progression in effect would involve five transitional periods (i.e. from V1 to V1.1beta, from V1.1beta to V1.1, from V1.1 to V1.2, from V1.2 to V1.3, and from V1.3 to V2). This information was used to configure the next three sub-phases: “Transformation Identification”, “Transformation Agent Definition” and “Transformation Design”. More specifically, the participants assembled five homogeneous builds in each of these sub-phases, corresponding to the five transitional periods identified. The configurations for these sub-phases are shown in Figure 7.6 and briefly described afterwards.



Legend:

- ALA Continuum's Application Lifecycle Analysis Process
- TxI Continuum's Transformation Identification Process
- TxAD Continuum's Transformation Agent Design Process
- TxD Continuum's Transformation Design Process



source: developed for this research

Figure 7.6 Configuration for “Transformation Identification”, “Transformation Agent Definition” and “Transformation Design” sub-phases

- “Transformation Identification” sub-phase

Each build in this sub-phase was configured with Continuum's Transformation Identification and Application Lifecycle Analysis processes. The former was used to identify transformations to occur in the transitional period as targeted by the build. The latter was used alongside the former to update the application

lifecycle and change cases developed earlier from “Lifecycle Definition”. For example, when additional change cases were identified with the Transformation Identification process, they needed to be incorporated into the application lifecycle.

- “Transformation Agent Definition” sub-phase

Each build in this sub-phase was configured with Continuum’s Transformation Agent Design and Transformation Identification processes. The Transformation Agent Design process was used to determine the transformation agents to execute transformations during the transitional period targeted by the build. Transformations identified earlier during “Transformation Identification” were updated accordingly with the Transformation Identification process if required. For example, if two transformations assigned to the same transformation agent could be combined into a larger transformation, the transformations identified would thus require updates. The Application Lifecycle Analysis process was not added to the build since the outcomes of the Transformation Agent Design process were not expected to affect the design for the application lifecycle.

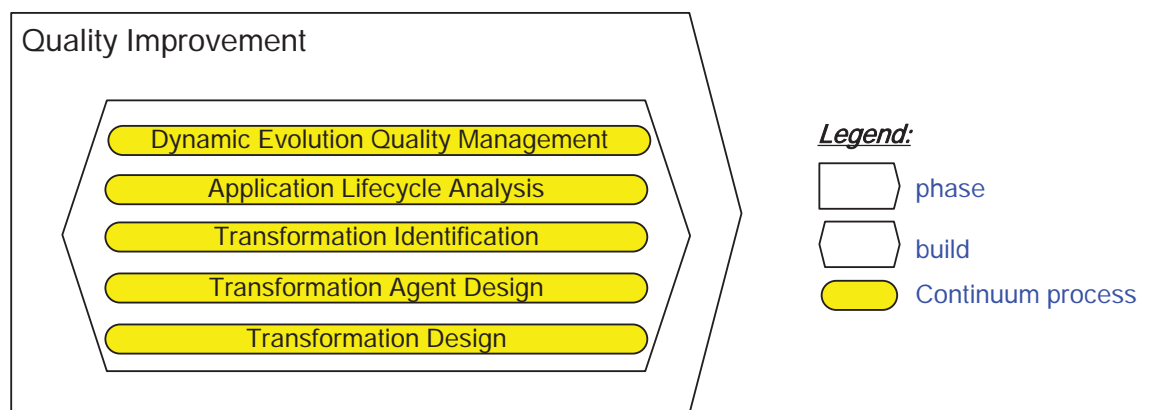
- “Transformation Design” sub-phase

Each build in this sub-phase (Figure 7.6) was configured with Continuum’s Transformation Design process, for designing how transformations work internally in terms of transformation actions. Since the internal behaviour of transformations would not affect other analysis and design aspects of dynamic evolution (e.g. change cases and transformation agents), the work products developed for the latter were not expected to require update. Hence, no other Continuum processes were used in the build.

Lastly, the “Application of Continuum” phase was finalised with the “Quality Improvement” sub-phase (cf. Figure 7.3) in which the primary objective was to improve the quality of the dynamic evolution work products developed earlier on. As illustrated in Figure 7.7, “Quality Improvement” had only one build which was assembled with all of Continuum’s processes. The primary process in the build was Dynamic Evolution Quality Management used to assess the quality of the dynamic evolution work products developed so far and to improve them according to the quality assessment. This process was supplemented with Continuum’s other four processes (Application Lifecycle Analysis, Transformation Identification, Transformation Agent Design and

Transformation Design) to further update and refine the work products developed. For instance, the participants identified that there was no mechanism to protect DPV from a potential failure triggered by a critical transformation (cf. Table Appendix D.14). This led to a number of changes to the work products, e.g.:

- the incorporation of a recovery transformation into a Transformation Orchestration Diagram via the Transformation Agent Design process; and
- the internal design for the recovery transformation above via the Transformation Design process.



source: developed for this research

Figure 7.7 Configuration for "Quality Improvement" sub-phase

7.2.4.2 Dealing with Dynamic Evolution Issues

In this section of the evaluation form, the participants were asked to jointly identify significant analysis/design issues in dynamic evolution in situ, i.e. encountered in the case study, and describe how Continuum was used to tackle the issues. Then, each participant rated the extent to which Continuum tackled each issue on a 1-7 Likert scale (1 for "extremely badly", 7 for "extremely well"). The information about the issues and the rating scores is shown in Table 7.4.

Table 7.4 Dynamic evolution issues encountered in case study and addressed with Continuum

Dynamic Evolution Issue Encountered	How Continuum Was Used to Address The Issue	Rating Score for Continuum	
		Range	Median
The overall interruption time to the system due to upgrade (i.e. dynamic evolution) is too long.	Allocating transformations to interruptive and non-interruptive (i.e. preparatory and finishing) phases, to reduce overall interruption time.	6-6	6.0
All users require to be trained for using the new system (V2) almost at the same time as when V2 is released, which is not practical and costly.	Use of a temporary generation in transit to make both V1 and V2 versions of the system functions available to end users, to give end users enough time to complete training.	6-6	6.0

<i>Dynamic Evolution Issue Encountered</i>	<i>How Continuum Was Used to Address The Issue</i>	<i>Rating Score for Continuum</i>	
		<i>Range</i>	<i>Median</i>
The new version of the system (V2) is not backwards compatible with other systems. This means jobs created with V1 of the system and still in progress can no longer be completed by the new version after the system upgrade to V2.	Use of a temporary generation in transit to make both V1 and V2 versions of the system functions available to end users, to enable them to complete jobs created with V1 of the system while using V2 to create and complete new jobs.	5-6	5.5
The scope of change for each system upgrade is too big and too complex.	Breakdown of changes into change cases, transformations and transformation actions.	6-6	6.0
Coordination among different roles (i.e. transformation agents) during application deployment is unplanned and chaotic.	Use of Transformation Orchestration Diagram and Notation to model coordination among different people.	6-7	6.5
System upgrade is not transparent to end users.	Overall system upgrade refined into small units of transformations, and only a subset of which will interrupt the system; use of «Blocked and Queued» ServicingPolicy to cache incoming requests from end users to further improve transparency.	6-7	6.5
Not all required changes and upgrades are identified from requirements.	Explicit use of change cases and associated tasks/techniques to progressively identify and analyse changes and upgrades.	4-6	5.0
<i>Quality aspects of dynamic evolution are not covered.</i>	<i>Explicit and dedicated Process “Dynamic Evolution Quality Management” to cover quality aspects.</i>	6-6	6.0
overall score		4-7	5.9

Continuum achieved median scores ranging from 5.0 (“well”) to 6.5 (between “very well” and “extremely well”). For instance, Continuum scored best in its ability to specify coordination among roles to perform transformations, and its ability to improve the transparency of transformations to end users of DPV. Note also that all issues were encountered in both the case study and in the previous actual upgrade project for DPV (i.e. from V1 to V2) with one exception; the issue concerning quality aspects of dynamic evolution (*italicised* in Table 7.4) was not addressed in the original upgrade project but was in the case study. One participant explained that practitioners of the upgrade project were unfamiliar with and unaware of quality aspects of dynamic evolution and quality was hence not contemplated in the upgrade project. After consulting Continuum, however, the participants felt that quality played an important role in dynamic evolution and subsequently it was applied in the case study.

7.2.4.3 Usefulness

For evaluation of usefulness, the case study aimed at comparing Continuum with the sponsor’s in-house methodology to see which one was more useful. A common technique for this kind of comparison is *member checking* which calls for each participant to perform a qualitative comparison from his/her past experience (Seaman

1999) Hence, the evaluation form asked participants to list the dynamic evolution issues encountered in both the case study (i.e. using Continuum) and in the previous project, which actually upgraded DPV (i.e. using the in-house methodology), and to determine which methodology worked better for each issue.

Only one of two participants in the case study was also involved in the previous upgrade project and thus able to evaluate Continuum's usefulness compared with the previous approach. The comparison results reported by this participant are listed in Table 7.5.

Table 7.5 Comparison of Continuum and in-house methodology in tackling recurrent dynamic evolution issues

<i>Dynamic Evolution Issue Encountered</i>	<i>How the Issue was Addressed</i>		<i>Extent to Which the Issue was Addressed</i>		
	<i>with Continuum</i>	<i>with In-House Methodology</i>	<i>Better with Continuum</i>	<i>About the Same</i>	<i>Better with In-House Methodology</i>
The overall interruption time to the system due to upgrade (i.e. dynamic evolution) is too long.	Allocating transformations to interruptive and non-interruptive (i.e. preparatory and finishing) phases, to reduce overall interruption time.	All changes planned to be applied to the system in one long time period (also the interruption time).	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
All users require to be trained for using the new system (V2) almost at the same time as when V2 is released, which is not practical and costly.	Use of a temporary generation in transit to make both V1 and V2 versions of the system functions available to end users, to give end users enough time to complete training.	A temporary generation as in the Continuum case considered but not probable since the upgrade had to be completed in one time period.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The new version of the system (V2) is not backwards compatible with other systems. This means valuations created with V1 of the system and still in progress can no longer be completed by the new version after the system upgrade to V2.	Use of a temporary generation in transit to make both V1 and V2 versions of the system functions available to end users, to enable them to complete jobs created with V1 of the system while using V2 to create and complete new jobs.	Not addressed (incomplete valuations in V1 of the system, if any, expected to be manually re-created in V2 of the system as part of the upgrade).	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The scope of change for each system upgrade is too big and too complex.	Breakdown of changes into change cases, transformations and transformation actions.	Changes identified at a high-level only.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Coordination among different roles (i.e. transformation agents) during application deployment is unplanned and chaotic.	Use of Transformation Orchestration Diagram and Notation to model coordination among different people.	Not addressed.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

<i>Dynamic Evolution Issue Encountered</i>	<i>How the Issue was Addressed</i>		<i>Extent to Which the Issue was Addressed</i>		
	<i>with Continuum</i>	<i>with In-House Methodology</i>	<i>Better with Continuum</i>	<i>About the Same</i>	<i>Better with In-House Methodology</i>
System upgrade is not transparent to end users.	Overall system upgrade refined into small units of transformations, and only a subset of which will interrupt the system; use of «Blocked and Queued» ServicingPolicy to cache incoming requests from end users to further improve transparency.	Not addressed.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Not all required changes and upgrades are identified from requirements.	Explicit use of change cases and associated tasks/techniques to progressively identify and analyse changes and upgrades.	Changes identified (but limited to high-level ones).	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Continuum appeared to handle most issues better than the in-house methodology except one, shown at the bottom of Table 7.5. More specifically, all changes and upgrades were successfully identified from requirements in both the case study (i.e. with Continuum) and the previous project (i.e. with the in-house methodology). During a follow-up meeting, this participant clarified that the identification in the previous project was confined to “high-level” changes and upgrades only. However, this participant acknowledged that Continuum offers useful constructs to map identified high-level changes to low-level changes and to detailed designs (i.e. of transformation actions). Such mapping was not supported by the in-house methodology.

7.2.4.4 Usability

For usability, the case study asked the participants to evaluate method fragments in Continuum with respect to *understandability* (e.g. “Is ‘X’ easy to understand?”) and *ease of use* (e.g. “Is ‘X’ easy to use?”)²², depending on the type of method fragment under assessment. In the evaluation form, they were asked to assign ratings to method fragments on a 1-7 Likert scale (1 for “strongly disagree”, 7 for “strongly agree”). The evaluation results gave some confidence in the quality of Continuum.

For model unit fragments (i.e. concepts) in Continuum’s metamodel, the participants

²² In work related to product quality, the ISO/IEC 9126-1 standard (ISO/IEC 2001) divides usability into three sub-characteristics: understandability, learnability and operability. Learnability is not in scope for this research as Continuum could be presented to the participants via better media, such as a development tool, which could influence learnability. Operability concerns the capability of a product to be used and liked by a user. This definition is equivalent to *ease of use* in this research.

were asked to rate the fragments' understandability (i.e. "Is concept 'X' easy to understand?") but not ease of use, since model unit fragments are not directly used; they manifest in work unit and work product fragments which are actually *used*. Table 7.6 presents the scores for the metamodel. Continuum's model unit fragments achieved median scores ranging from 5.5 (for the "Transitional Period" aspect) to 6.5 (for the "Structural Foundation" aspect).

Table 7.6 Understandability ratings for Continuum's metamodel

Metamodel Aspect	Related Model Units	Understandability	
		Range	Median
Structural Foundation	Application, Resource OperationalProfile, TransformableItem and Zone	6-7	6.5
Application Lifecycle	Application, ApplicationLifecycle, ChangeCase, Generation, Impact, Stage, TransformableItem and TransitionalPeriod	6-6	6.0
Transitional Period	TransitionalPeriod, TransformationAgent, ChangeCase, Transformation, TransformationException and TransformationExceptionResolution	5-6	5.5
Transformation	Resource, TransformableItem, Transformation, TransformationAction and Zone	5-7	6.0
Policy	Policy, ServicingPolicy and ZoningPolicy	6-6	6.0
overall score		5-7	6.0 (average)

For work unit and work product fragments of Continuum, the participants were asked to evaluate both *understandability* (i.e. "Is 'X' easy to understand?") and *ease of use* (i.e. "Is 'X' easy to use to, such as to produce relevant models/diagrams/documents?"). Table 7.7 presents the scores for Continuum's work unit and work product fragments. Overall, the task and technique fragments scored an average of 6.0 for both understandability and ease of use. The work product fragments scored averages of 5.9 and 6.0 for understandability (of the fragments) and ease of use respectively. Process fragments were not individually rated. Instead, the scores for each process fragment were obtained by taking the average of the scores of the process fragment itself, its associated task, technique and work product fragments. The process fragments attained an average score of 6.0 for both understandability and ease of use.

Table 7.7 Usability ratings for Continuum's work unit and work product fragments

Method Fragment Kind	Work Unit/Product Fragment	Understandability		Ease of Use	
		Range	Median	Range	Median
Process Application Lifecycle Analysis					
Task	Identify As-Is Runtime Structure	6-7	6.5	6-7	6.5
	Derive Change Cases	5-6	5.5	5-6	5.5

Method Fragment Kind	Work Unit/Product Fragment	Understandability		Ease of Use	
		Range	Median	Range	Median
	Extend Application Lifecycle	6-7	6.5	6-7	6.5
Technique	Change Case Modelling	5-6	5.5	5-6	5.5
	Change Case Partitioning and Ordering	5-6	5.5	5-6	5.5
	Runtime Structure Recovery (reused)	6-6	6.0	6-6	6.0
Work Product	Application Lifecycle Diagram	6-7	6.5	6-7	6.5
	Dynamic Application Change Document	6-6	6.0	6-6	6.0
	Structural Configuration - Notational Extensions	5-6	5.5	6-6	6.0
process score (including its task, technique and work product method fragments)		5-7	5.9 (average)	5-7	6.0 (average)
Process Transformation Identification					
Task	Define To-Be Runtime Structure	7-7	7.0	7-7	7.0
	Refine Change Cases	5-7	6.0	5-7	6.0
	Identify Transformations	7-7	7.0	7-7	7.0
Technique	Dynamic Variation Management (reused)	6-6	6.0	6-6	6.0
	Dynamic Workflow Change (reused)	6-6	6.0	6-6	6.0
	Dynamic Refactoring	6-6	6.0	6-6	6.0
	Dynamic Recomposition	6-6	6.0	6-6	6.0
	Change Case Modelling	5-7	6.0	5-7	6.0
	Dynamic Change Impact Analysis	6-7	6.5	6-7	6.5
	Transformation Sizing	5-6	5.5	5-6	5.5
Work Product	Dynamic Application Change Document	6-6	6.0	6-6	6.0
	Structural Configuration - Notational Extensions	5-6	5.5	6-6	6.0
process score (including its task, technique and work product method fragments)		5-7	6.1 (average)	5-7	6.2 (average)
Process Transformation Agent Design					
Task	Identify Transformation Agents	6-6	6.0	6-6	6.0
	Define Transformation Orchestration	6-6	6.0	6-6	6.0
Work Product	Transformation Orchestration Diagram	6-6	6.0	6-6	6.0
process score (including its task, technique and work product method fragments)		6-6	6.0 (average)	6-6	6.0 (average)
Process Transformation Design					
Task	Identify New and Replacement Transformable Items	6-7	6.5	6-7	6.5
	Identify Changes to Zones	6-7	6.5	6-7	6.5
	Define Servicing Policies	5-6	5.5	5-6	5.5
	Develop Transformation	6-7	6.5	5-7	6.0

<i>Method Fragment Kind</i>	<i>Work Unit/Product Fragment</i>	<i>Understandability</i>		<i>Ease of Use</i>	
		<i>Range</i>	<i>Median</i>	<i>Range</i>	<i>Median</i>
<i>Technique</i>	Resource Profile Modelling	6-6	6.0	6-6	6.0
	Start-up State Configuration	5-6	5.5	5-6	5.5
	Performance Profile Modelling	4-6	5.0	4-6	5.0
	Dynamic Transformable Item Adaptation	4-6	5.0	4-6	5.0
	Dynamic Transformable Item (Re)binding	5-6	5.5	5-6	5.5
	Dynamic Transformable Item Change	6-6	6.0	6-6	6.0
<i>Work Product</i>	Transformation Diagram	6-6	6.0	6-6	6.0
	State Map	5-6	5.5	5-6	5.5
	New and Replacement Transformable Item Catalogue	6-7	6.5	6-7	6.5
	Zone Change Document	5-6	5.5	5-6	5.5
process score (including its task, technique and work product method fragments)		4-7	5.8 (average)	4-7	5.8 (average)
<i>Process Dynamic Evolution Quality Management</i>					
<i>Task</i>	Define Dynamic Evolution Quality Needs	6-7	6.5	6-7	6.5
	Assess Dynamic Evolution Quality	6-7	6.5	6-7	6.5
	Analyse Dynamic Evolution Quality Problems	6-6	6.0	6-6	6.0
	Improve Dynamic Evolution Quality	6-6	6.0	6-6	6.0
<i>Technique</i>	Dynamic Change Localisation (reused)	5-7	6.0	5-7	6.0
	Dynamic Evolution Safety Risk Management	5-6	5.5	5-6	5.5
	Dynamic Security Policy and Enforcement Management (reused)	5-7	6.0	5-7	6.0
	Dynamic Wrapper (reused)	6-7	6.5	6-7	6.5
	Inspection (reused)	6-7	6.5	6-7	6.5
	Loose Coupling (reused)	6-7	6.5	6-7	6.5
	Recovery Blocks (reused)	6-6	6.0	6-6	6.0
	Secure and Reliable Transformation Agent Coordination	5-7	6.0	5-7	6.0
	Start-up State Configuration	6-7	6.5	6-7	6.5
	Testability Analysis and Improvement (reused)	5-6	5.5	5-6	5.5
	Transformable Item Autonomy (reused)	4-6	5.0	4-6	5.0
	Transformable Item Mediation and Channelling (reused)	5-6	5.5	5-6	5.5
	Transformable Item Regression Testing (reused)	5-6	5.5	5-6	5.5
	Transformation Exception Management	6-7	6.5	5-7	6.0
	Transformation Sizing	5-5	5.0	5-5	5.0
<i>Work Product</i>	Dynamic Evolution Quality Profile Report	6-6	6.0	6-6	6.0

Method Fragment Kind	Work Unit/Product Fragment	Understandability		Ease of Use	
		Range	Median	Range	Median
	Dynamic Evolution Quality Inspection Report	6-6	6.0	6-6	6.0
	process score (including its task, technique and work product method fragments)	4-7	6.0 (average)	4-7	6.0 (average)
	overall process score	4-7	6.0 (average)	4-7	6.0 (average)

For each work product fragment of Continuum, the participants were also asked to assess the understandability of the actual work product(s) (model(s), diagram(s), document(s) etc.) *developed* with the work product fragment for the case study. The rating question corresponds to “Are the models/diagrams/documents produced from work product fragment ‘X’ easy to understand?”. Table 7.8 presents the scores for the actual work products developed which are documented in Appendix D. They scored an average of 6.0 for understandability.

Table 7.8 Understandability ratings for work products developed with Continuum

Work Product Kind	Actual Work Product Developed for Case Study	Understandability	
		Range	Median
Application Lifecycle Diagram	Figure Appendix D.2	6-7	6.5
Dynamic Application Change Document	Table Appendix D.4 (for change cases for generation V1) Table Appendix D.5 (for change cases for generation V1.1beta), Table Appendix D.6 (for change cases for generation V1.1), Table Appendix D.7 (for change cases for generation V1.2), Table Appendix D.8 (for change cases for generation V1.3), Table Appendix D.2 (for analysed change impacts)	6-6	6.0
Structural Configuration - Notational Extensions	Figure Appendix D.1 (for generation V1), Figure Appendix D.4 (for generation V1.1beta), Figure Appendix D.5 (for generation V1.1), Figure Appendix D.6 (for generation V1.2), Figure Appendix D.7 (for generation V1.3), Figure Appendix D.8 (for generation V2), Figure Appendix D.9 (for disposition of transformation agents)	6-6	6.0
Transformation Orchestration Diagram	Figure Appendix D.10 (for progressing generation V1 to V1.1beta), Figure Appendix D.11 (for progressing generation V1.1beta to V1.1), Figure Appendix D.12 (for progressing generation V1.1 to V1.2), Figure Appendix D.13 (for progressing generation V1.2 to V1.3), Figure Appendix D.14 (for progressing generation V1.3 to V2)	6-6	6.0
Transformation Diagram	Figure Appendix D.15, Figure Appendix D.16, Figure Appendix D.17, Figure Appendix D.18	6-6	6.0
State Map	Table Appendix D.10	5-6	5.5

<i>Work Product Kind</i>	<i>Actual Work Product Developed for Case Study</i>	<i>Understandability</i>	
		<i>Range</i>	<i>Median</i>
New and Replacement Transformable Item Catalogue	Table Appendix D.9	6-7	6.5
Zone Change Document	Table Appendix D.11	5-6	5.5
Dynamic Evolution Quality Profile Report	Table Appendix D.13 (only overall scores of dynamic evolution quality factors shown in Table Appendix D.13)	6-6	6.0
Dynamic Evolution Quality Inspection Report	Table Appendix D.14 (only method fragments with defects/issues shown in Table Appendix D.14)	6-6	6.0
overall score		5-7	6.0 (average)

7.2.5 Lessons Learned

While the case participants were enthusiastic about a new approach for dealing with dynamic evolution relevant to their project, lessons learned from the case study are discussed with a view to facilitate Continuum's adoption in an industrial setting. To begin with, Continuum expects people who contemplate in learning and using it to have certain knowledge and/or experience in relevant areas. From a method user's viewpoint, the case study participants' familiarity with specific kinds of composition-based distributed applications and change management indeed helped them comprehend Continuum and dynamic evolution. From a method engineer's viewpoint, their experience with a methodology helped them quickly grasp the notion of method engineering and attain the skills for configuring Continuum to suit the case study.

With respect to learning, the case study conducted two training sessions for the participants (Section 7.2.3.1). While they gained basic knowledge about Continuum, it would have been more effective if the training sessions had been scheduled farther apart than in the case study (e.g. at least weeks). This is because the participants would have reasonable time to read through Continuum documentation between the two sessions and to be able to ask in-depth questions about Continuum during the second session. It was also observed that the participants had to take care of their normal duties in the period between the two sessions such that they had limited time to read through Continuum. On the other hand, it was effective for the participants to learn to apply Continuum in a real project setting (cf. Section 7.2.3.1) and by solving a real dynamic evolution problem relevant to their responsibilities. Other possibilities to help lower the learning curve include offering exercises for learners, providing sample

Continuum configurations for a variety of contexts and additional sample dynamic evolution work products for specific kinds of composition-based distributed applications.

Based on the case study, other factors that may contribute to the successful adoption of Continuum in the industry include:

- *Tool support*: From the experience of developing a methodology tool, one participant noted that it would be easier to use Continuum in the form of a methodology tool (further elaborated in Section 8.4.3) than as a document.
- *Cost control*: Adopting a new technology/methodology invariably entails cost overheads (Murphy et al. 1999). Although the sponsor generously offered two participants to take part in the case study, cost overheads - studying, training, development time, etc. - should be analysed, discussed and controlled with an adopting organisation.
- *Experienced participation*: It was not an intent for any researcher to participate in a case study. However, someone familiar with Continuum would be beneficial to an endeavour in the form of mentoring others and participating in development requiring dynamic evolution.

7.3 CONCLUSION

This Chapter presented how Continuum was progressively evaluated and refined in Phase 3 of this research programme to produce the final version of Continuum (documented in Section 6.3 and Appendix C). The first task of Phase 3 conducted a review of Continuum's documentation by two subject matter experts both experienced in the field of dynamic evolution. The results of the review were used to refine Continuum. After that, the second and final task of Phase 3 involved two participants in a case study to apply the updated version of Continuum to retrospectively analyse and design dynamic evolution for a real-world application. Feedback from the results was collected from the participants and Continuum was refined accordingly. The evaluation aspect of these tasks covered usefulness, usability, strengths, weaknesses and completeness. The refinement aspect of these tasks resulted in improvements to various areas of Continuum, including its metamodel, work unit fragments, work product fragments, structure, organisation, presentation and examples used. The next Chapter concludes this research.

Chapter 8. CONCLUSIONS

“From the end spring new beginnings.” - Pliny the Elder

This Chapter concludes this research with a summary of the investigations carried out in this research (Section 8.1), a discussion of the contributions of this research (Section 8.2), a statement of validity and reliability threats to the findings of this research (Section 8.3) and recommendations for future work in both research and practice (Section 8.4).

8.1 SUMMARY OF INVESTIGATIONS

The aim of this research was to develop extensions to existing software development methodologies to support the analysis and design aspects of dynamic evolution for composition-based distributed applications (Section 1.2). Correspondingly, this research organised its investigations into two research questions (Section 1.2). On this basis, this research implemented a design science research programme consisting of three phases of activities to investigate the two research questions. In Phase 1 (Chapter 4 and Chapter 5), the investigation focused on answering the first research question (i.e. RQ1):

“What are the important requirements for consideration in composition-based distributed application development to support dynamic evolution?”

To manage the complexity of dynamic evolution, the investigation distinguished between two types of requirements:

- *Dynamic change requirements*

These requirements characterise the changes that a distributed application could accommodate at runtime.

- *Dynamic evolution quality factors*

These requirements concern how well a distributed application and the dynamic changes to it are designed to facilitate dynamic evolution.

Both types of requirements are sufficiently generic to a variety of composition-based distributed applications, including component-based and service-oriented applications, both a main focus of this research. Each type of requirement was determined using the same process. An initial set of requirements was first synthesised from evaluation

frameworks and a systematic literature review. They were then extended in two steps: a survey of practitioners and researchers; and identification of dynamic evolution requirements already considered in a number of selected methodologies.

The importance rankings of the initial set of dynamic evolution requirements were determined from the analysis of survey data. They were then revised using expert opinions to account for additional requirements suggested by survey respondents and those identified from the literature published after the survey. Important dynamic evolution requirements are summarised in Table 8.1. A small number of requirements that were deemed to be the least important - viz. Efficiency, Transitional Form, Part Retirement and Dynamic Protocol Evolution (cf. Section 6.1) - are discarded from Table 8.1.

Table 8.1 Important dynamic evolution requirements

<i>Type</i>	<i>Category</i>	<i>Requirements</i>	
Dynamic Change Requirement	Modelling Related	Part Level	Multiple Version Coexistence Resource Needs Performance Characteristics Access Blocking
		Application Level	Dynamic Change Transformation Generation Application Lifecycle Servicing Continuity
		Others	Transformation Agent Transformation Action Transformation Exception Transformation Exception Resolution Expected dynamic change impact
	Work Related	Part Level	Dynamic Part Change Dynamic Part Adapter Dynamic Part (Re)Binding Resource Need Prediction Performance Characteristic Prediction Geometric Change Dynamic State Transfer
		Application Level	Dynamic Workflow Evolution Dynamic Recomposition Dynamic Refactoring Dynamic Variability Dynamic Change Impact Analysis Dynamic Contract Update

<i>Type</i>	<i>Category</i>	<i>Requirements</i>
Dynamic Evolution Quality Factor	Soundness of Change	Completeness Consistency Correctness
	Infusibility of Change	Locality Maintainability Transparency
	Changeability of Application	Autonomy Coordination Extensibility Loose Coupling Separation of Concerns
	Robustness of Application	Fault Tolerance Recoverability Reliability Safety Security

The second research question (i.e. RQ2), as repeated below, was addressed in Phases 2 and 3 by developing Continuum which is a methodological extension comprising a set of the International Standard ISO/IEC 24744 (ISO/IEC 2007) based method fragments:

“How can these important requirements be addressed with method fragments used in composition-based distributed application development?”

Continuum supports the design and analysis of dynamic evolution during composition-based distributed application development. An initial version of Continuum was constructed in Phase 2 to specifically address the important dynamic evolution requirements determined as a result of the first research question (RQ1). Afterwards, Continuum was progressively evaluated and refined in Phase 3 via the expert view and case study application of Continuum to reach its final version. To summarise, Continuum comprises:

- *Dynamic evolution metamodel* (Section 6.3.3)

The metamodel offers constructs to express, communicate and reason about dynamic evolution during software development.

- *Dynamic change method fragments* (Section 6.3.4)

These fragments provide basic support for the analysis and design of changes to a running composition-based distributed application without requiring shutdown.

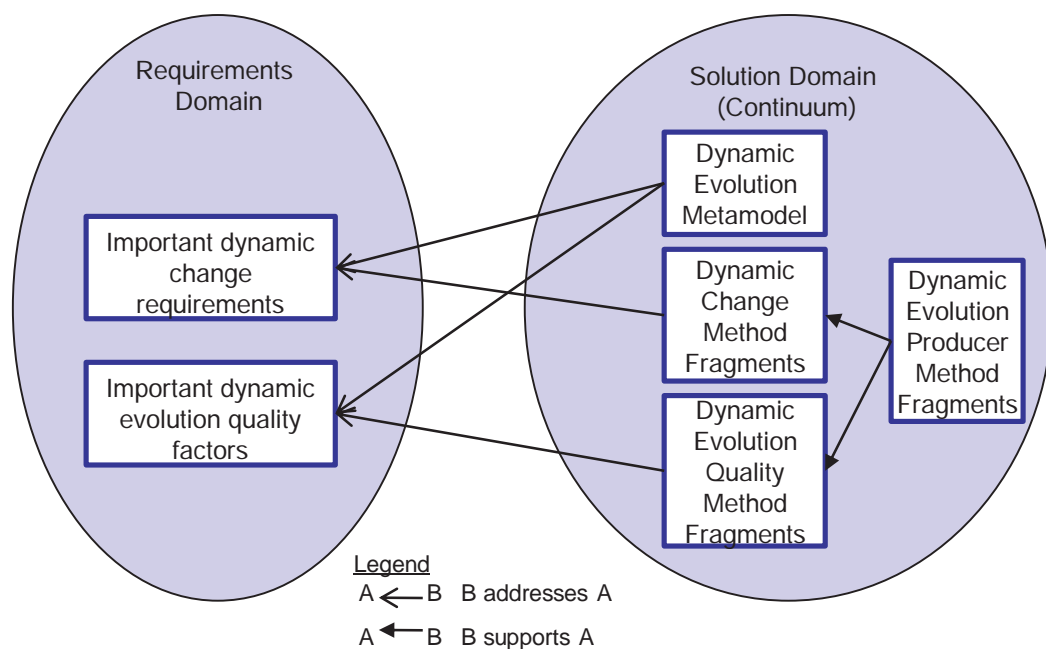
- *Dynamic evolution quality method fragments* (Section 6.3.5)

These fragments further enhance the quality of the work products delivered with Continuum's dynamic change method fragments.

- *Dynamic evolution producer method fragments* (Section 6.3.6)

These fragments define roles responsible for executing particular work units specified in Continuum's dynamic change and dynamic evolution quality method fragments according to their areas of expertise.

The relationship between Continuum's method fragments and the important dynamic evolution requirements identified in this research is summarised in Figure 8.1. The information on which individual method fragments fulfil a particular dynamic evolution requirement was provided in Section 6.4.



source: developed for this research

Figure 8.1 Major relationships among research artefacts developed in this research

8.2 CONTRIBUTIONS OF RESEARCH

As highlighted in Section 1.1.3, poor understanding and support for dynamic evolution is apparent in the literature concerning methodology development and research. This is reflected by two gaps: 1) the lack of a comprehensive set of dynamic evolution requirements from the literature, evaluation frameworks and methodologies; and 2) inadequate support for dynamic evolution in methodologies. The first gap was observed from the publications (i.e. the literature, evaluation frameworks and methodologies)

reviewed for dynamic evolution requirement synthesis in this research. None offer a broad range of dynamic evolution requirements covering many aspects of dynamic evolution considered in this research (e.g. Sections 4.1.5, 4.3, 5.1.3 and 5.3). The second gap was evidenced from a feature analysis of a number of relevant methodologies performed in this research using the dynamic evolution requirements proposed in this research as the analysis criteria (cf. Sections 4.4 and 5.4).

Regarding the first gap, the investigation outcomes contribute to both research and practice by proposing a comprehensive set of important dynamic evolution requirements for explicit consideration in methodologies supporting composition-based distributed application development (cf. Section 8.1). This contribution represents an improvement to existing and relevant evaluation frameworks and development methodologies in which dynamic evolution has not been explicitly considered and/or addressed (Sections 2.3 and 2.4).

With respect to software development concerning dynamic evolution, it is hoped that the proposed requirements will assist methodology or method users (Nuseibeh et al. 1996) in becoming more familiar with various aspects of dynamic evolution for a composition-based distributed application and the level of support provided by their own methodology for dynamic evolution. For instance, if security and safety are key non-negotiable concerns in their business, method(ology) users can utilise relevant requirements from the proposed set to examine if a methodology satisfactorily addresses these concerns. Moreover, the proposed requirements offer method(ology) users a means to communicate their specific needs for dynamic evolution to methodology or method engineers (Nuseibeh et al. 1996) who will then configure a methodology attuned to those needs, or choose the methodology of best fit. Method(ology) engineers can also apply the proposed requirements as criteria to identify gaps in a methodology in terms of weaknesses for supporting dynamic evolution (e.g. lack of fault tolerance) which also represent areas for change to enhance the methodology's support for dynamic evolution.

To fill the second gap (i.e. lack of methodological support for dynamic evolution), this research contributes to research and practice by proposing Continuum to specifically address the analysis and design aspects of dynamic evolution as a separate concern. Leveraging the situational method engineering approach, method(ology) engineers can select fragments from Continuum, orchestrate and sequence them to run alongside conventional software development activities in a methodology to suit a particular endeavour. The flexibility of extending a methodology to support dynamic evolution this

way also separates the dynamic evolution concern from conventional analysis and design concerns in a methodology. This separation allows method(ology) users responsible for application development to focus on conventional activities, independently from method(ology) users who specifically deal with dynamic evolution as a distinct concern during application development.

8.3 VALIDITY AND RELIABILITY THREATS

Limitations of this research and the way in which the research techniques were applied pose threats to the findings of this research. The threats are examined along five dimensions which are commonly used for empirical and social research paradigms (Wohlin et al. 2000; Yin 2003): conclusion validity, construct validity, internal validity, external validity and reliability.

8.3.1 Conclusion Validity

Conclusion validity concerns whether conclusions can be correctly drawn about the relations between a treatment and an outcome (Wohlin et al. 2000). In the surveys conducted in this research, respondents could misinterpret questionnaire forms because of potentially poor wording and bad rating scales used. This would affect the importance rankings of the dynamic evolution requirements and limit conclusion validity (Sections 4.2 and 5.2). This was addressed by pilot testing and refining questionnaire forms to improve their quality, clarity and completeness.

Misinterpretation of the feedback collected from survey respondents, experts and case study participants could also lead to the wrong conclusions drawn from the feedback and a threat to conclusion validity. To counter this, follow-up meetings were held with subjects to discuss and clarify their feedback.

8.3.2 Construct Validity

Construct validity concerns whether appropriate means of measurement for the concepts being studied have been established (Wohlin et al. 2000; Yin 2003). One threat to construct validity observed in the case study is experimenter expectancy (Wohlin et al. 2000). Since one of the two participants had involved in the previous upgrade project prior to the case study, this participant might have certain positive or negative expectations of the case study and bias the results. To lessen this effect, the case study had both participants cross-check each other's data (e.g. Section 7.2.3.3). In order to address construct validity of the case study data further, multiple sources of

evidence were collected: evaluation forms, project meetings, follow-up meetings and documentation (whiteboard drawings and hard copies).

8.3.3 Internal Validity

Internal validity concerns the extent to which the causal relationship of the theory being evaluated is established from the measured data (Wohlin et al. 2000; Yin 2003). There are several threats to internal validity of this research's outcome - purposeful sampling used in the surveys, for instance (Sections 4.2 and 5.2). In particular, since the survey recruitment targeted an adequately sized group of people with appropriate experience and/or knowledge in composition-based technologies to achieve credibility and authenticity of the responses collected, this led to selection bias which weakens internal validity. A larger sample size randomly sourced from wider geographic regions would improve the confidence of the survey outcomes.

The expert review of Continuum relied on the opinion from a small number of experts (Section 7.1) which might be biased and limited by their own experience and knowledge in dynamic evolution. Internal validity is therefore dampened. A more thorough review effort would involve a mix of people with different experience, knowledge and skill set, including experts, practitioners experienced in dealing with dynamic evolution and experienced users of relevant methodologies, in conducting assessment and review activities. As the number of people participating in such a review grows, they should be coordinated and managed to ensure the review results are systematically collated and analysed.

In the case study, the sponsor provided only two participants to evaluate Continuum and only one of them was able to compare Continuum with the in-house methodology based on this participant's own experience (Sections 7.2.4.3 and 7.2.4.4). Both limitations compromise internal validity of the evaluation and comparison results. The former can be alleviated by sourcing a sponsor that can provide an increased number of participants. The latter can be addressed by undertaking a controlled and replicated experiment in lieu of a case study approach to evaluate Continuum. In a simple experiment, participants apply a methodology to develop dynamic evolution, repeat the development using the same methodology enhanced with Continuum, and then compare the outcomes from the two development activities. A threat to this experiment is the learning effect (Basili et al. 1999), in which case the participants gain experience in dealing with dynamic evolution in the first instance and may subsequently improve the solution when dealing with dynamic evolution again (i.e. when using the

methodology enhanced with Continuum). This effect can be alleviated by applying a between-subjects design (Basili et al. 1999; Pfleeger 1995), which varies the participants, the dynamic evolution problems to solve and the number of experiments.

Other threats to internal validity include the fact that the methodology evaluations (to demonstrate a use of the dynamic evolution requirements and to identify method fragments for reuse) were performed by only one person (Sections 4.4 and 5.4). In addition, the evaluation results were subject to the quality of the information provided in the documentation of the reviewed methodologies. Some confidence could be gained by involving experienced users of the methodologies to take part in the evaluation. Significant differences in the evaluation results would then be resolved by post-evaluation discussions (Iivari & Kerola 1983).

8.3.4 External Validity

External validity refers to the degree to which a technique is generalisable to other situations (projects, application types etc.) (Wohlin et al. 2000; Yin 2003). Since the practical use of dynamic evolution is still in its infancy, the number of respondents interested in both dynamic evolution and the surveys was small (thirty-six for dynamic change requirements and forty for dynamic evolution quality factors). It cannot be concluded whether the findings are representative of the whole IS community from the industry and academia, in which case the small sample sizes compromise external validity. Similarly, the small number of experts participating in various stages of this research is also a threat to external validity since their opinions cannot be generalisable to experts from the industry and academia.

While the single case study offered Continuum to be evaluated in a particular context (i.e. property valuations) in depth, this limits external validity because the case study results cannot be generalisable to other applications and contexts. This threat can be alleviated by applying Continuum in different application contexts (see Section 8.4.2).

Another threat to external validity is this research investigated mainly two types of composition-based distributed applications - component-based and SOA-based - for the development of the dynamic evolution requirements and Continuum which cannot be regarded as generalisable to all kinds of composition-based distributed applications. This is because there are potentially other (and perhaps less common) types of composition-based distributed applications which have not been accounted for in this research. The rationale for restricting the investigation to these two types of applications is because they are commonly used in practice and a vast amount of

relevant literature exists. The outcome of this research, however, aims to be sufficiently generic to these two types and is hopefully also suitable for other types of composition-based distributed applications.

8.3.5 Reliability

Reliability relates to whether the results of a technique can be replicated by other researchers (Yin 2003). For that matter, this research documented the procedures for the techniques used, including the systematic literature review (Appendix A), surveys (Sections 4.2 and 5.2), expert reviews (Section 7.1.2), and the case study (Section 7.2.3), as well as the instruments (e.g. Table Appendix B.1) and forms used (Appendix E). Lee (1989), however, cautioned that in a replicated case study, not all conditions in a previous development endeavour are likely to recur and this situation represents a potential threat to reliability.

8.4 RECOMMENDATIONS FOR FUTURE WORK

Apart from the suggestions for addressing the limitations of this research (cf. Section 8.3), there are a few unexplored research areas suggesting potential topics for future work that can be pursued from the industry and academia. They are described next.

8.4.1 Extension

There are several directions for extending the work in this research. For example, since Continuum and the proposed dynamic evolution requirements are intended to be sufficiently generic to different types of composition-based distributed applications, one could utilise these results as a baseline to develop further enhancements tailored to applications of a specific type such as those based on SOA.

As noted in Section 1.4, this research focuses on two types of composition-based distributed applications (i.e. component-based and SOA-based) in order to manage its scope. Continuum can be extended for other types of composition-based distributed applications. An example is agent-oriented systems consisting of agents operating in a distributed heterogeneous environment. Agents resemble active components or objects interacting with one another to achieve a common objective (Fortino et al. 2004; Zambonelli et al. 2003).

In consideration of the scope of Continuum which is currently limited to analysis and design, a possibility is to extend Continuum to cover other aspects of the development lifecycle. This includes *implementation*, *testing*, and *deployment* of changes to a

running application. One expert undertaking the review of Continuum also suggested that during deployment, *monitoring* the execution of transformations and checking their *conformance* to the design of dynamic evolution are interesting areas of concern since they are required to support both the detection of faults and the application of remedies to faults. They provide an opportunity for extending Continuum.

On a different note, research in dynamic adaptation has been gathering pace. Dynamic adaptation requires dynamic evolution and needs to be able to deal with changes in the environment in which an application operates (see Section 2.1.2). Logically, an investigation into this environmental or contextual factor and extending the work of this research to handle dynamic adaptation represents another opportunity for future work.

8.4.2 Applying Continuum to a Variety of Applications and Domains

In this research, Continuum was applied to the analysis and design of dynamic evolution for a property valuation system and subsequently evaluated by two participants (Section 7.2). To further strengthen its validity and quality, future work should apply Continuum to a wide range of applications from a variety of domains and improve it accordingly. This diversity also stems from an observation that dynamic evolution is increasingly recognised as a capability in many application domains, research projects and commercial systems (Oreizy et al. 2008). Applications would preferably be sourced from the real world to give Continuum an opportunity to be used to solve real dynamic evolution problems in a practical setting. In addition, as noted earlier in Section 8.3.3, more participants should be involved in future applications and evaluations of Continuum.

8.4.3 Tool Implementation

The current version of Continuum is documented on paper (introduction in Sections 6.3.2 to 6.3.7, and full details in Appendix C). A future endeavour would be to incorporate Continuum into an existing methodbase or a repository supporting method fragments, an exemplar of which being the Open Process Framework (OPF) Repository (OPFRO 2009). This would allow Continuum to be integrated with other methodologies implemented in the same methodbase. For instance, a methodology *plug-in* is firstly constructed for Continuum in accordance with the Eclipse Process Framework (EPF), an open-source tool for methodology construction and tailoring (The Eclipse Foundation 2009). Afterwards, the plug-in is then incorporated into OpenUP, a small methodology also built with EPF (The Eclipse Foundation 2009). Nonetheless,

EPF and OpenUP are not based on the International Standard ISO/IEC 24744 (ISO/IEC 2007).

8.5 CONCLUDING REMARKS

Dynamic evolution will in no doubt bring agility to the business in accommodating changes in their increasingly critical modern distributed applications. However, dynamic evolution has not been well supported by existing methodologies. Through a design science research programme, this research proposed a comprehensive set of dynamic evolution requirements and associated methodological support to fill this gap. These improvements are a step forward to embracing dynamic evolution in this kind of application. Hopefully, the outcomes of this research will help to stimulate further research in dynamic evolution, and to harness methodologies to increase its uptake in practice.

BIBLIOGRAPHY

- Aassine, S & El Jai, MC 2002, 'Vegetation dynamics modelling: a method for coupling local and space dynamics', *Ecological Modelling*, vol. 154, no. 3, pp. 237-249.
- Abran, A & Moore, JW 2004, *Guide to the Software Engineering Body of Knowledge: 2004 Version*, Report Number 0-7695-2330-7, IEEE Computer Society, viewed 13 Sep 2010 <http://www2.computer.org/portal/web/store?product_id=RN0000001&category_id=ReadyNotes>.
- Adamek, J & Plasil, F 2005, 'Component composition errors and update atomicity: static analysis', *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, no. 5, pp. 363-377.
- Adman, P 1997, 'Book Review: Understanding and Evaluating Methodologies - NIMSAD: A Systemic Framework. Nimal Jayaratna.', *Systems Research and Behavioral Science*, vol. 14, no. 5, pp. 352-354.
- Agnew, B, Hofmeister, C & Purtilo, J 1994a, 'Planning for change: a reconfiguration language for distributed systems', *Distributed Systems Engineering: Special Issue on Configurable Distributed Systems*, vol. 1, no. 5, pp. 313-322.
- Agnew, B, Hofmeister, C & Purtilo, J 1994b, 'Planning for change: a reconfiguration language for distributed systems', *Proceedings of the 2nd International Workshop on Configurable Distributed Systems (IWCDs'94)* pp. 15-22.
- Akram, MS, Medjahed, B & Bouguettaya, A 2003, 'Supporting dynamic changes in web service environments', in *Service-Oriented Computing - ICSOC 2003*, vol. 2910/2003, Springer, Berlin / Heidelberg, pp. 319-334.
- Aksit, M & Choukair, Z 2003, 'Dynamic, adaptive and reconfigurable systems overview and prospective vision', *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops*, IEEE Computer Society, pp. 84-89.
- Albert, C & Brownsword, L 2002, *Evolutionary Process for Integrating COTS-Based Systems (EPIC)*, Technical Report CMU/SEI-2002-TR-005, Software Engineering Institute, Pittsburgh PA, viewed 3 Aug 2007 <<http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr009.pdf>>.
- Allen, P 2007, *The Service Oriented Process*, Report Number 1745-1884, Everware-CBD Inc., viewed 3 Jun 2008 <http://www.cbdforum.com/secure/interact/2007-02/service_oriented_process.php>.
- Allen, P & Brown, P 2007, *Architected Solution Delivery: Enhancing the Service Oriented Process*, Report Number 1745-1884, Everware-CBD Inc., viewed 3 Jun 2008 <http://www.cbdforum.com/secure/interact/2007-11/architected_solution_delivery_enhancing_service_oriented_process.php>.
- Allen, R & Garlan, D 1997, 'A formal basis for architectural connection', *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 3, pp. 213-249.
- Almeida, JPA, Sinderen, MV & Nieuwenhuis, L 2001, 'Transparent dynamic reconfiguration for CORBA', *Proceedings of the 3rd International Symposium on Distributed Objects and Applications*, IEEE Computer Society, pp. 197-207.
- Altunel, Y & Tolun, M, R. 2007, 'Component-based software development with component variants', *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*, ACTA Press, Innsbruck, Austria, pp. 235-241.
- Alur, D, Mals, D & Crupi, J 2003, *Core J2EE patterns: Best Practices and Design Strategies*, 2nd edn, Prentice Hall PTR, Upper Saddle River, NJ, USA.
- Ammon, Rv, Ertlmaier, T, Etzion, O, Kofman, A & Paulus, T 2010, 'Integrating complex events for collaborating and dynamically changing business processes', in A Dan, F Gittler & F Toumani (eds), *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, vol. 6275, Springer Berlin / Heidelberg, pp. 370-384.
- Andersson, J & Bosch, J 2005, 'Development and use of dynamic product-line architectures', *IEE Proceedings - Software*, vol. 152, no. 1, pp. 13-26.
- Andersson, T & von Hellens, LA 1997, 'Information systems work quality', *Information and Software Technology*, vol. 39, no. 12, pp. 837-844.

- Andrade, L, Fiadeiro, JL, Gouveia, J & Koutsoukos, G 2002, 'Separating computation, coordination and configuration', *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 14, no. 5, pp. 353-369.
- Andrews, JD & Moss, TR 2002, *Reliability and Risk Assessment*, 2nd edn, Professional Engineering Publishing, London.
- Andrikopoulos, V, Benbernou, S & Papazoglou, M 2008, 'Managing the evolution of service specifications', in, *Advanced Information Systems Engineering*, Springer, Berlin / Heidelberg, pp. 359-374.
- Apperly, H, Hofman, R, Latchem, S, Maybank, B, McGibbon, B, Piper, D, Simons, C & Hoffman, R 2003, *Service- and Component-based Development: Using the Select Perspective*, Addison-Wesley Professional.
- Arapis, C 1995, 'A temporal perspective of composite objects', in O Nierstrasz & D Tsichritzis (eds), *Object-Oriented Software Composition*, Prentice Hall Inc., pp. 123-152.
- Arnold, HJ 1965, 'Small sample power of the one sample Wilcoxon test for non-normal shift alternatives', *The Annals of Mathematical Statistics*, vol. 36, no. 6, pp. 1767-1778.
- Arnold, K, Gosling, J & Holmes, D 2005, *The Java(TM) Programming Language (4th Edition)*, Addison-Wesley Professional.
- Arnold, P, Bodoff, S, Coleman, D, Gilchrist, H & Hayes, F 1991, *Criteria for Comparing Object-Oriented Development Methods*, HP Labs, viewed 6 Apr 2008 <<http://www.hpl.hp.com/techreports/91/HPL-91-51.html>>.
- Arnold, RS & Böhner, SA 1996, 'An introduction to software change impact analysis', in RS Arnold & SA Böhner (eds), *Software Change Impact Analysis*, IEEE Computer Society, pp. 1-26.
- Arsanjani, A 2004, *Service-Oriented Modeling and Architecture*, IBM Corp., viewed 6 Jun 2008 <<http://www.ibm.com/developerworks/library/ws-soa-design1/>>.
- Arshad, N, Heimbigner, D & Wolf, A 2007, 'Deployment and dynamic reconfiguration planning for distributed software systems', *Software Quality Journal*, vol. 15, no. 3, pp. 265-281.
- Asadi, M & Ramsin, R 2008, 'MDA-Based Methodologies: An Analytical Survey', in, *Model Driven Architecture - Foundations and Applications*, Springer, pp. 419-431.
- Atkinson, C, Bayer, J, Bunse, C, Kamsties, E, Laitenberger, O, Laqua, R, Muthig, D, Paech, B, Wüst, J & Zettel, J 2002, *Component-based Product Line Engineering with UML*, Addison-Wesley.
- Atkinson, C, Bayer, J & Muthig, D 2000, 'Component-based product line development: the KobrA approach', *Proceedings of the first conference on Software product lines : experience and research directions: experience and research directions*, Kluwer Academic Publishers, Denver, Colorado, United States, pp. 289-309.
- Atkinson, C & Kühne, T 2003, 'Model-driven development: a metamodeling foundation', *IEEE Software*, vol. 20, no. 5, pp. 36-41.
- Autili, M, Berardinelli, L, Cortellessa, V, Di Marco, A, Di Ruscio, D, Inverardi, P & Tivoli, M 2007, 'A development process for self-adapting service oriented applications', in B Krämer, K-J Lin & P Narasimhan (eds), *Service-Oriented Computing - ICSOC 2007*, vol. 4749, Springer Berlin / Heidelberg, pp. 442-448.
- Baglietto, P, Maresca, M, Parodi, A & Zingirian, N 2005, 'Stepwise deployment methodology of a service oriented architecture for business communities', *Information and Software Technology*, vol. 47, no. 6, pp. 427-436.
- Bajec, M, Vavpotic, D & Krisper, M 2007, 'Practice-driven approach for creating project-specific software development methods', *Information and Software Technology*, vol. 49, no. 4, pp. 345-365.
- Balsamo, S, Marco, AD, Inverardi, P & Simeoni, M 2004, 'Model-based performance prediction in software development: a survey', *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 295-310.
- Barbacci, M, Klein, MH, Longstaff, TA & Weinstock, CB 1995, *Quality Attributes*, Software Engineering Institute, Pittsburgh PA, viewed 27 Jul 2010 <<http://www.sei.cmu.edu/reports/95tr021.pdf>>.
- Barbier, F & Henderson-Sellers, B 2000, 'Object modelling languages: An evaluation and some key expectations for the future', *Annals of Software Engineering*, vol. 10, no. 1-4, pp. 67-101.
- Basili, VR, Shull, F & Lanubile, F 1999, 'Building knowledge through families of experiments', *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 456-473.

- Bass, L, Clements, P & Kazman, R 2003, *Software Architecture in Practice*, 2nd edn, Addison Wesley.
- Bastida, L 2008, 'A methodology for dynamic service composition', *Proceedings of the 7th International Conference on Composition-Based Software Systems, 2008 (ICCBSS 2008)*, IEEE Computer Society, pp. 33-42.
- Beck, K & Andres, C 2005, *Extreme Programming Explained: Embrace Change*, 2nd edn, Addison-Wesley Longman, Boston, MA.
- Ben-Shaul, I, Holder, O & Lavva, B 2001, 'Dynamic adaptation and deployment of distributed components in Hadas', *IEEE Transactions on Software Engineering*, vol. 27, no. 9, pp. 769-787.
- Bengtsson, P, Lassing, N, Bosch, J & van Vliet, H 2004, 'Architecture-level modifiability analysis (ALMA)', *Journal of Systems and Software*, vol. 69, no. 1-2, pp. 129-147.
- Bennett, KH & Rajlich, VT 2000, 'Software maintenance and evolution: a roadmap', *Proceedings of the 22nd International Conference on Software Engineering: The Future of Software Engineering*, ACM, Limerick, Ireland, pp. 73-87.
- Bianculli, D, Jurca, R, Binder, W, Ghezzi, C & Faltings, B 2007, 'Automated dynamic maintenance of composite services based on service reputation', in B Krämer, K-J Lin & P Narasimhan (eds), *Service-Oriented Computing - ICSOC 2007*, vol. 4749, Springer Berlin / Heidelberg, pp. 449-455.
- Bidan, C, Issarny, V, Saridakis, T & Zarras, A 1998, 'A dynamic reconfiguration service for CORBA', *Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDs'98)*, IEEE Press, Annapolis, Maryland, pp. 35-42.
- Blaha, M & Premerlani, W 1996, 'A catalog of object model transformations', *Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE'96)*, IEEE Computer Society, Monterey, California, pp. 87-97.
- Blake, MB 2007, 'Decomposing composition: service-oriented software engineers', *IEEE Software*, vol. 24, no. 6, pp. 68-77.
- Bloom, T & Day, M 1993, 'Reconfiguration and module replacement in Argus: theory and practice', *Software Engineering Journal*, vol. 8, no. 2, pp. 102-108.
- Bobkowska, AE 2005, 'A framework for methodologies of visual modeling language evaluation', *Proceedings of the 2005 Symposia on Metainformatics*, ACM, Esbjerg, Denmark.
- Bock, C 2003, 'UML 2 Activity and Action Models Part 2: Actions', *Journal of Object Technology*, vol. 2, no. 5, pp. 41-56.
- Boertien, N, Steen, MWA & Jonkers, H 2005, 'Evaluation of component-based development methods', in J Krogstie, TA Halpin & K Siau (eds), *Information Modeling Methods and Methodologies*, Idea Group, pp. 323-343.
- Bohmann, T, Junginger, M & Krcmar, H 2003, 'Modular service architectures: a concept and method for engineering IT services', *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 3 - Volume 3*, ed. M Junginger, IEEE p. 74.71.
- Bohner, SA 1996, 'Impact analysis in the software change process: a year 2000 perspective', *Proceedings of the 1996 International Conference on Software Maintenance*, IEEE Computer Society, Monterey, CA, USA, pp. 42-51.
- Bohner, SA 2002a, 'Extending software change impact analysis into COTS components', *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, IEEE Computer Society, pp. 175-182.
- Bohner, SA 2002b, 'Software change impacts-an evolving perspective', *Proceedings of the 18th International Conference on Software Maintenance (ICSM'02)*, IEEE Computer Society, pp. 263-272.
- Bradbury, JS, Cordy, JR, Dingel, J & Wermelinger, M 2004, 'A survey of self-management in dynamic software architecture specifications', *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, ACM, Newport Beach, California, pp. 28-33.
- Breivold, HP & Crnkovic, I 2010, 'A Systematic Review on Architecting for Software Evolvability', *Proceedings of the 21st Australian Software Engineering Conference (ASWEC'10)*, IEEE Computer Society, Auckland, New Zealand pp. 13-22.
- Brereton, P & Budgen, D 2000, 'Component-based systems: a classification of issues', *Computer*, vol. 33, no. 11, pp. 54-62.
- Brereton, P, Kitchenham, BA, Budgen, D, Turner, M & Khalil, M 2007, 'Lessons from applying the systematic literature review process within the software engineering domain',

- Journal of Systems and Software*, vol. 80, no. 4, pp. 571-583.
- Brinkkemper, S 1996, 'Method engineering: engineering of information systems development methods and tools', *Information and Software Technology*, vol. 38, no. 4, pp. 275-280.
- Brinkkemper, S, Saeki, M & Harmsen, F 1998, 'Assembly techniques for method engineering', in B Pernici & C Thanos (eds), *Proceedings of the 10th International Conference on Advanced Information Systems Engineering*, Springer-Verlag London, UK, pp. 381-400.
- Brown, A, Johnston, S & Kelly, K 2002, *Using service-oriented architecture and component-based development to build web service applications*, Rational Software Corp. (now part of IBM Corp.), viewed 2 Jul 2007 <<http://www.ibm.com/developerworks/rational/library/510.html>>.
- Brown, AW & Wallnau, KC 1996, 'Engineering of component-based systems', in AW Brown (ed.), *Component-Based Software Engineering*, IEEE Computer Society, pp. 7-15.
- Brown, AW & Wallnau, KC 1998, 'The current state of CBSE', *IEEE Software*, vol. 14, no. 5, pp. 37-46.
- Bruneton, E, Coupaye, T & Stefani, JB 2002, 'Recursive and dynamic software composition with sharing', *Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02)*, Malaga, Spain.
- Brykczynski, B 1999, 'A survey of software inspection checklists', *SIGSOFT Softw. Eng. Notes*, vol. 24, no. 1, pp. 82-89.
- Bucchiarone, A, Cappiello, C, Di Nitto, E, Kazhamiakin, R, Mazza, V & Pistore, M 2010, 'Design for adaptation of service-based applications: main issues and requirements', in A Dan, F Gittler & F Toumani (eds), *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, vol. 6275, Springer Berlin / Heidelberg, pp. 467-476.
- Bucher, T, Klesse, M, Kurpjuweit, S & Winter, R 2007, 'Situational method engineering: on the differentiation of "context" and "project type"', in J Ralyté, S Brinkkemper & B Henderson-Sellers (eds), *Situational Method Engineering: Fundamentals and Experiences*, vol. 244, Springer Boston, pp. 33-48.
- Buckley, J, Mens, T, Zenger, M, Rashid, A & Kniesel, G 2005, 'Towards a taxonomy of software change', *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, no. 5, pp. 309-332.
- Burstein, F & Gregor, S 1999, 'The systems development or engineering approach to research in information systems: an action research perspective', *Proceedings of the 10th Australasian Conference on Information Systems*, eds B Hope & P Yoong, Wellington, New Zealand, pp. 122-134.
- Canal, C, Pimentel, E & Troya, JM 1999, 'Specification and refinement of dynamic software architectures', *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, Kluwer, B.V., pp. 107-126.
- Canal, C, Poizat, P & Salaun, G 2008, 'Model-based adaptation of behavioral mismatching components', *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 546-563.
- Cao, F, Bryant, BR, Liu, S & Zhao, W 2005, 'A non-invasive approach to dynamic web services provisioning', *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005)*, IEEE Computer Society, Orlando, pp. 229-236.
- Carzaniga, A, Fuggetta, A, Hall, RS, van der Hoek, A, Heimbigner, D & Wolf, AL 1998, *A Characterization Framework for Software Deployment Technologies*, Dept. of Computer Science, University of Colorado, viewed 19 Jan 2008 <<http://www.ics.uci.edu/~andre/papers/T3.pdf>>.
- Casati, F, Ceri, S, Pernici, B & Pozzi, G 1998, 'Workflow evolution', *Data & Knowledge Engineering*, vol. 24, no. 3, pp. 211-238.
- Cavano, JP & McCall, JA 1978, 'A framework for the measurement of software quality', *Proceedings of the software quality assurance workshop on Functional and performance issues*, ACM, pp. 133-139.
- Cervantes, H & Hall, RS 2005, 'Technical concepts of service orientation', in Z Stojanovic & A Dahanayake (eds), *Service-oriented Software System Engineering: Challenges and Practices*, IGI Global, Hershey, PA, pp. 1-26.
- Challenger, J, Dantzig, P, Iyengar, A & Witting, K 2005, 'A fragment-based approach for efficiently creating dynamic web content', *ACM Transactions on Internet Technology*, vol. 5, no. 2, pp. 359-389.
- Chang, SH & Kim, SD 2007, 'A Service-Oriented Analysis and Design Approach to Developing

- Adaptable Services', *Proceedings of the IEEE International Conference on Services Computing (SCC'07)*, IEEE Computer Society, pp. 204 - 211.
- Chapin, N, Hale, JE, Kham, KM, Ramil, JF & Tan, W-G 2001, 'Types of software evolution and software maintenance', *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 13, no. 1, pp. 3-30.
- Chaudet, C, Greenwood, RM, Oquendo, F & Warboys, BC 2000, 'Architecture-driven software engineering: specifying generating, and evolving component-based software systems', *IEEE Proceedings - Software*, vol. 147, no. 6, pp. 203-214.
- Cheesman, J & Daniels, J 2001, *UML Components: A Simple Process for Specifying Component-Based Software*, Addison-Wesley, Boston, MA, USA.
- Chen, H, Yu, J, Chen, R, Zang, B & Yew, P-C 2007, 'POLUS: A POverful Live Updating System', paper presented to the *Proceedings of the 29th international conference on Software Engineering*.
- Chen, W-K, Hiltunen, MA & Schlichting, RD 2001, 'Constructing adaptive software in distributed systems', *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001)*, pp. 635-643.
- Chen, W & Hirschheim, R 2004, 'A paradigmatic and methodological examination of information systems research from 1991 to 2001', *Information Systems Journal*, vol. 14, no. 3, pp. 197-235.
- Chen, X 2002, 'Extending RMI to support dynamic reconfiguration of distributed systems', *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, pp. 401-408.
- Chikofsky, EJ & Cross, JH, II 1990, 'Reverse engineering and design recovery: a taxonomy', *IEEE Software*, vol. 7, no. 1, pp. 13-17.
- Chimi, CJ & Russell, DL 2009, 'The Likert scale: a proposal for improvement using quasi-continuous variables', *Proceedings of the Information Systems Education Conference 2009*, v 26, Washington DC.
- Chin, W, Marcolin, B & Newsted, P 1996, 'A Partial Least Squares latent variable modeling approach for measuring interaction effects: results from a Monte Carlo simulation study and voice mail emotion/adoption study', *Proceedings of the International Conference on Information Systems (ICIS 1996)*, Association for Information Systems, viewed 6 Jul 2011 <<http://aisel.aisnet.org/icis1996/6>>.
- Cho, ES, Kim, SD & Rhew, SY 2004, 'A domain analysis and modeling methodology for component development', *International Journal of Software Engineering and Knowledge Engineering*, vol. 14, no. 2, pp. 221-254.
- Christensen, E, Curbera, F, Meredith, G & Weerawarana, S 2001, *Web Services Description Language (WSDL) 1.1*, W3C, viewed 2 Jul 2007 <<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>>.
- Chung, L, Nixon, BA, Yu, E & Mylopoulos, J 1999, *Non-Functional Requirements in Software Engineering* Springer.
- Clitherow, D, Brookbanks, M, Clayton, N & Spear, G 2008, 'Combining high availability and disaster recovery solutions for critical IT environments', *IBM Systems Journal*, vol. 47, no. 4, pp. 563-575.
- Conradi, R & Westfechtel, B 1998, 'Versions models for software configuration management', *ACM Computing Surveys*, vol. 30, no. 2, pp. 232-282.
- Cook, JE & Dage, JA 1999, 'Highly reliable upgrading of components', *Proceedings of the 21st IEEE International Conference on Software Engineering (ICSE'99)*, IEEE Computer Society, Los Angeles CA, USA, pp. 203-212.
- Cook, S, He, J & Harrison, R 2001, 'Dynamic and static views of software evolution', *Proceedings of the IEEE International Conference on Software Maintenance, 2001.*, IEEE Computer Society, pp. 592-601.
- Coyle, L, Hinchey, M, Nuseibeh, B & Fiadeiro, JL 2010, 'Guest editors' introduction: evolving critical systems', *Computer*, vol. 43, no. 5, pp. 28-33.
- Cronbach, LJ 1951, 'Coefficient alpha and the internal structure of tests', *Psychometrika*, vol. 16, no. 3, pp. 297-334.
- Cugola, G, Frey, D, Murphy, AL & Picco, GP 2004, 'Minimizing the reconfiguration overhead in content-based publish-subscribe', *Proceedings of the 2004 ACM symposium on Applied computing*, ACM, Nicosia, Cyprus, pp. 1134-1140.
- Curbera, F, Khalaf, R, Mukhi, N, Tai, S & Weerawarana, S 2003, 'The next step in Web

- services', *Communications of the ACM*, vol. 46, no. 10, pp. 29-34.
- D'Souza, DF & Wills, AC 1998, *Objects, Components and Frameworks with UML: The Catalysis Approach*, Addison-Wesley.
- Dahanayake, ANW, Sol, HG & Stojanovic, Z 2003, 'Methodology Evaluation Framework for Component-Based System Development', *Journal of Database Management*, vol. 14, no. 1, pp. 1-26.
- Darwin, C 1859, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, 6th edn, John Murray, Albemarle Street, London.
- Date, CJ 2003, *An Introduction to Database Systems*, 8th edn, Addison Wesley.
- de Castro Guerra, PA, Rubira, CMF, Romanovsky, A & de Lemos, R 2003, 'Integrating COTS software components into dependable software architectures', *Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 139-142.
- de Champeaux, D & Faure, P 1992, 'A comparative study of object-oriented analysis methods', *Journal of Object-Oriented Programming*, vol. 5, no. 1, pp. 21-33.
- de Jonge, M, Muskens, J & Chaudron, M 2003, 'Scenario-based prediction of run-time resource consumption in component-based software systems', *Proceedings of the 6th ICSE Workshop on Component-based Software Engineering (CBSE6)*, pp. 19-24.
- de Lara, E, Chopra, Y, Kumar, R, Vaghela, N, Wallach, DS & Zwaenepoel, W 2005, 'Iterative adaptation for mobile clients using existing APIs', *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 10, pp. 966-981.
- de Paula, VC, Justo, GRR & Cunha, PRF 2000, 'Specifying and verifying reconfigurable software architectures', *Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems*, IEEE Computer Society, p. 21.
- Dearle, A 2007, 'Software deployment, past, present and future', *Future of Software Engineering (FOSE '07)*, IEEE Computer Society, pp. 269-284.
- Dellarocas, C 1997, 'A coordination perspective on software system design', *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering (SEKE'97)*, Knowledge Systems Institute, Madrid, Spain, pp. 569-578.
- Dellarocas, C, Klein, M & Shrobe, H 1998, 'An architecture for constructing self-evolving software systems', *Proceedings of the 3rd International Workshop on Software Architecture*, ACM, Orlando, Florida, United States, pp. 29-32.
- Deprez, JC, Monfilsc, FF, Ciolkowskf, M & Soto, MASM 2007, 'Defining software evolvability from a Free/Open-Source Software', *Proceedings of the 3rd International IEEE Workshop on Software Evolvability* ed. FF Monfilsc, IEEE Computer Society, pp. 29-35.
- Desnos, N, Huchard, M, Urtado, C, Vauttier, S & Tremblay, G 2007, 'Automated and unanticipated flexible component substitution', in H Schmidt, I Crnkovic, G Heineman & J Stafford (eds), *Component-Based Software Engineering*, vol. 4608, Springer Berlin / Heidelberg, pp. 33-48.
- Dijkstra, E 1976, *A discipline of programming*, Prentice Hall Inc.
- Ding, L & Medvidovic, N 2001, 'Focus: a light-weight, incremental approach to software architecture recovery and evolution', *Proceedings of the 2001 Working IEEE/IFIP Conference on Software Architecture (WISCA'01)*, IEEE Computer Society, Amsterdam, The Netherlands, p. 191.
- Dodd, J, Allen, P, Butler, J, Olding, S, Veryard, R & Wilkes, L 2007, *CBDI-SAE Meta Model for SOA version 2.0*, Everware-CBDI Inc., viewed 12 Aug 2010 <http://www.cbdiforum.com/public/CBDI_SAE_META_MODEL_FOR_SOA_V2.0.php3>
- Dromey, RG 1995, 'A model for software product quality', *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 146-162.
- Dustdar, S & Schreiner, W 2005, 'A survey on web services composition', *International Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1-30.
- Ebraert, P, D'Hondt, T & Mens, T 2004, 'Enabling dynamic software evolution through automatic refactorings', *Proceedings of the Workshop on Software Evolution Transformations (SET2004)*, pp. 3-7.
- Ecklund, EF, Delcambre, LML & Freiling, MJ 1996, 'Change cases: use cases that identify future requirements', *SIGPLAN Notices*, vol. 31, no. 10, pp. 342-358.
- Elfatraty, A 2007, 'Dealing with change: components versus services', *Communications of the*

- ACM, vol. 50, no. 8, pp. 35-39.
- Ellis, C & Keddara, K 2000, 'ML-DEWS: modeling language to support dynamic evolution within workflow systems', *Computer Supported Cooperative Work (CSCW)*, vol. 9, no. 3, pp. 293-333.
- Endler, M 1993, 'A model for distributed management of dynamic changes', *Proceedings of the 4th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'93)*, Long Branch, USA.
- Eriksson, I & Törn, A 1991, 'A model for IS quality [information systems]', *Software Engineering Journal*, vol. 6, no. 4, pp. 152-158.
- Erl, T 2005, *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall Inc.
- Erlikh, L 2000, 'Leveraging Legacy System Dollars for E-business', *IT Pro*, pp. 17-23.
- Ernst, MD, Perkins, JH, Guo, PJ, McCamant, S, Pacheco, C, Tschantz, MS & Xiao, C 2007, 'The Daikon system for dynamic detection of likely invariants', *Science of Computer Programming*, vol. 69, no. 1-3, pp. 35-45.
- Erradi, A, Anand, S & Kulkarni, N 2006a, 'SOAF: an architectural framework for service definition and realization', *Proceedings of the IEEE International Conference on Services Computing, 2006. (SCC '06)* ed. A Sriram, IEEE Press, pp. 151-158.
- Erradi, A, Maheshwari, P & Tosic, V 2006b, 'Policy-driven middleware for self-adaptation of web services compositions', in M van Steen & M Henning (eds), *Middleware 2006*, vol. 4290/2006, Springer, Berlin / Heidelberg, pp. 62-80.
- Etzkorn, G 1992, 'Change programming in distributed systems', *International Workshop on Configurable Distributed Systems (IWCDs'92)*, IEEE Computer Society, pp. 140-151.
- Evans, H & Dickman, P 1999, 'Zones, contracts and absorbing change: an approach to software evolution', *Proceedings of the 14th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99)*, vol. 34(10), ACM, pp. 415-434.
- Fahringer, T, Krause, H, Kuropka, D, Mayer, H, Ocampo, A, Schreder, B, Staab, S, Tröger, P, Wahler, A & Zaremba, M 2007, *Adaptive Services Grid - White Paper: ASG technology advantages and disadvantages, exploitation possibilities and its business impact*, Adaptive Services Grid (ASG), viewed 22 Jan 2010 <http://tb0.asg-platform.org/download/downloadrequest.php?asgdocument=white_paper_ASG.pdf>.
- Fayad, M & Cline, MP 1996, 'Aspects of software adaptability', *Communications of the ACM*, vol. 39, no. 10, pp. 58-59.
- Feiler, P & Li, J 1998, 'Managing inconsistency in reconfigurable systems', *IEEE Proceedings - Software*, vol. 145, no. 5, pp. 172-179.
- Feng, T & Maletic, JI 2006, 'Applying dynamic change impact analysis in component-based architecture design', *Proceedings of the 7th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2006)*, ed. JI Maletic, IEEE Computer Society, pp. 43-48.
- Firesmith, DG 2003, *Common Concepts Underlying Safety, Security, and Survivability Engineering*, Software Engineering Institute, Pittsburgh PA, viewed 27 Jul 2010 <<http://www.sei.cmu.edu/reports/03tn033.pdf>>.
- Firesmith, DG & Henderson-Sellers, B 2002, *The OPEN Process Framework: An Introduction*, Addison-Wesley, Harlow, Herts, U.K.
- Firesmith, DG, Henderson-Sellers, B & Graham, IM 1997, *OPEN Modelling Language (OML) Reference Manual*, Cambridge University Press.
- Fitzgerald, B, Russo, NL & O'Kane, T 2003, 'Software development method tailoring at Motorola', *Communications of the ACM*, vol. 46, no. 4, pp. 64-70.
- Fortino, G, Russo, W & Zimeo, E 2004, 'A Statecharts-based software development process for mobile agents', *Information and Software Technology*, vol. 46, no. 13, pp. 907-921.
- Fowler, M, Beck, K, Brant, J, Opdyke, W & Roberts, D 1999, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional.
- Fox, C & Frakes, W 1997, 'The quality approach: is it delivering?', *Communications of the ACM*, vol. 40, no. 6, pp. 24-29.
- Fragopoulou, P, Mastroianni, C, Montero, R, Andrzejak, A & Kondo, D 2010, 'Self-* and Adaptive Mechanisms for Large Scale Distributed Systems', in F Desprez, V Getov, T Priol & R Yahyapour (eds), *Grids, P2P and Services Computing*, Springer US, pp. 147-156.

- Fraser, T, Badger, L & Feldman, M 1999, 'Hardening COTS software with generic software wrappers', *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, IEEE Computer Society, Oakland, CA, pp. 2-16.
- Freedman, RS 1991, 'Testability of software components', *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 553-564.
- Friedman, M 1937, 'The use of ranks to avoid the assumption of normality implicit in the analysis of variance', *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675-701.
- Fung, KH, Low, GC & Ray, PK 2004, 'Embracing dynamic evolution in distributed systems', *IEEE Software*, vol. 21, no. 2, pp. 49-55.
- Gama, K & Donsez, D 2010, 'A self-healing component sandbox for untrustworthy third party code execution', in L Grunske, R Reussner & F Plasil (eds), *Component-Based Software Engineering*, vol. 6092, Springer Berlin / Heidelberg, pp. 130-149.
- Gamma, E, Helm, R, Johnson, R & Vlissides, J 1995, *Design Patterns: Elements of Reusable Software*, Addison-Wesley.
- Ganesan, R & Sengupta, S 2001, 'O2BC: A technique for the design of component-based applications', *Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Language and Systems*, pp. 46-55.
- Garlan, D, Allen, R & Ockerbloom, J 1994, 'Exploiting style in architectural design environments', *SIGSOFT Software Engineering Notes*, vol. 19, no. 5, pp. 175-188.
- Gärtner, FC 1999, 'Fundamentals of fault-tolerant distributed computing in asynchronous environments', *ACM Computing Surveys*, vol. 31, no. 1, pp. 1-26.
- Georgiadis, I, Magee, J & Kramer, J 2002, 'Self-organising software architectures for distributed systems', *Proceedings of the 1st workshop on Self-healing systems*, ACM, Charleston, South Carolina, pp. 33-38.
- GigaTS 2001, *GigaTS Handbook: Rapid Service Development Methodology*, Telematica Instituut, viewed 26 Aug 2007 <<http://rsd.demo.telin.nl/handbook/>>.
- Gil, Y, Deelman, E, Ellisman, M, Fahringer, T, Fox, G, Gannon, D, Goble, C, Livny, M, Moreau, L & Myers, J 2007, 'Examining the Challenges of Scientific Workflows', *Computer*, vol. 40, no. 12, pp. 24-32.
- Gilb, T 1988, *Principles of Software Eng. Management*, Addison Wesley.
- Goldman, KJ, Swaminathan, B, McCartney, P, Anderson, MD & Sethuraman, R 1995, 'The programmers' playground: I/O abstraction for user-configurable distributed applications', *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 735-746.
- Gonzalez-Perez, C & Henderson-Sellers, B 2005, 'Templates and resources in software development methodologies', *Journal of Object Technology*, vol. 4, no. 4, pp. 173-190.
- Gonzalez-Perez, C & Henderson-Sellers, B 2006a, 'An Ontology for Software Development Methodologies and Endeavours', in C Calero, F Ruiz & M Piattini (eds), *Ontologies for Software Engineering and Technology*, Springer-Verlag, Berlin, pp. 123-151.
- Gonzalez-Perez, C & Henderson-Sellers, B 2006b, 'A powertype-based metamodeling framework', *Software and Systems Modeling*, vol. 5, no. 1, pp. 72-90.
- Gonzalez-Perez, C & Henderson-Sellers, B 2007, 'Modelling software development methodologies: A conceptual foundation', *Journal of Systems and Software*, vol. 80, no. 11, pp. 1778-1796.
- Goudarzi, KM & Kramer, J 1996, 'Maintaining node consistency in the face of dynamic change', *Proceedings of the 3rd International Conference on Configurable Distributed Systems 1996 (ICCDs'96)*, IEEE Computer Society, Annapolis, Maryland, pp. 62-69.
- Grady, RB 1992, *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Inc.
- Graham, I 1991, *Object-oriented methods*, Addison-Wesley.
- Graham, IM, Henderson-Sellers, B & Younessi, H 1997, *The OPEN Process Specification*, Addison-Wesley.
- Gregersen, AR & Jørgensen, BN 2009, 'Dynamic update of Java applications—balancing change flexibility vs programming transparency', *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 2, pp. 81-112.
- Gregor, S 2002, 'Design theory in information systems', *Australasian Journal of Information Systems*, vol. 10, no. 1.
- Gregor, S 2006, 'The nature of theory in information systems', *MIS Quarterly*, vol. 30, no. 3, pp. 611-642.

- Grimán, A, Pérez, M, Mendoza, L & Losavio, F 2006, 'Feature analysis for architectural evaluation methods', *Journal of Systems and Software*, vol. 79, no. 6, pp. 871-888.
- Grimm, R & Bershad, BN 2001, 'Separating access control policy, enforcement, and functionality in extensible systems', *ACM Transactions on Computer Systems*, vol. 19, no. 1, pp. 36-70.
- Grundy, J, Ding, G & Hosking, J 2005, 'Deployed software component testing using dynamic validation agents', *Journal of Systems and Software*, vol. 74, no. 1, pp. 5-14.
- Gu, Q & Lago, P 2007, 'A stakeholder-driven service life cycle model for SOA', *Proceedings of the 2nd International Workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE Joint Meeting*, ACM, Dubrovnik, Croatia, pp. 1-7.
- Gupta, D, Jalote, P & Barua, G 1996, 'A formal framework for on-line software version change', *IEEE Transactions on Software Engineering*, vol. 22, no. 2, pp. 120-131.
- Haerder, T & Reuter, A 1983, 'Principles of transaction-oriented database recovery', *ACM Computing Surveys*, vol. 15, no. 4, pp. 287-317.
- Haire, B, Henderson-Sellers, B & Lowe, D 2001, 'Supporting web development in the OPEN process: additional tasks', *Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development*, IEEE Computer Society, pp. 383-389.
- Hall, RS, Heimbigner, D & Wolf, AL 1999, 'A cooperative approach to support software deployment using the software dock', *Proceedings of the 21st International Conference on Software Engineering*, IEEE Computer Society, Los Angeles, California, United States, pp. 174-183.
- Ham, DH, Kim, JS, Cho, JH & Ha, SJ 2004, 'MaRMI-III: A methodology for component-based development', *ETRI Journal*, vol. 26, no. 2, pp. 167-180.
- Harmsen, F, Brinkkemper, S & Oei, JLH 1994, 'Situational method engineering for informational system project approaches', *Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle*, Elsevier Science Inc., pp. 169-194.
- Hauptmann, S & Wasel, J 1996, 'On-line maintenance with on-the-fly software replacement', *Proceedings of the 3rd International Conference on Configurable Distributed Systems 1996 (ICCDs'96)*, IEEE Computer Society, Annapolis, Maryland, pp. 70-80.
- Heider, W, Froschauer, R, Gruenbacher, P, Rabiser, R & Dhungana, D 2010, 'Simulating evolution in model-based product line engineering', *Information and Software Technology*, vol. 52, no. 7, pp. 758-769.
- Henderson-Sellers, B 2003, 'Method engineering for OO systems development', *Communications of the ACM*, vol. 46, no. 10, pp. 73-78.
- Henderson-Sellers, B 2006, 'Method engineering: theory and practice', *Information Systems Technology and its Applications. 5th International Conference ISTA'2006*, vol. P-84, eds D Karagiannis & HC Mayr, Klagenfurt, Austria, pp. 13-23.
- Henderson-Sellers, B, Collins, G, Due, R & Graham, I 2001, 'A qualitative comparison of two processes for object-oriented software development', *Information and Software Technology*, vol. 43, no. 12, pp. 705-724.
- Henderson-Sellers, B & Gonzalez-Perez, C 2005, 'Connecting powertypes and stereotypes', *Journal of Object Technology*, vol. 4, no. 7, pp. 83-96.
- Henderson-Sellers, B & Gonzalez-Perez, C 2006, 'On the ease of extending a powertype-based methodology metamodel, keynote paper', in S Brockmans, J Jung & Y Sure (eds), *Meta-Modelling and Ontologies, Proceedings of the 2nd Workshop on Meta-Modelling (WoMM 2006)*, vol. P-96, Gesellschaft für Informatik, Bonn, pp. 11-25.
- Henderson-Sellers, B, Gonzalez-Perez, C, Serour, MK & Firesmith, DG 2005, 'Method engineering and COTS evaluation', *Proceedings of the second international workshop on Models and processes for the evaluation of off-the-shelf components*, ACM, St. Louis, Missouri, pp. 1-4.
- Henderson-Sellers, B & Ralyté, J 2010, 'Situational method engineering: state-of-the-art review', *Journal of Universal Computer Science*, vol. 16, no. 3, pp. 424-478.
- Henderson-Sellers, B, Simons, A & Younessi, H 1998, *The OPEN toolbox of techniques*, ACM/Addison-Wesley Publishing Co.
- Herrmann, P & Krumm, H 2001, 'Trust-adapted enforcement of security policies in distributed component-structured applications', *Proceedings of the 6th IEEE Symposium on Computers and Communications*, IEEE Computer Society, pp. 2-8.

- Herzum, P & Sims, O 2000, *Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*, John Wiley & Sons Inc.
- Hevner, AR, March, ST, Park, J & Ram, S 2004, 'Design science in information systems research', *MIS Quarterly*, vol. 28, no. 1, pp. 75-106.
- Hicks, M & Nettles, S 2005, 'Dynamic software updating', *ACM Transactions on Programming Languages and Systems*, vol. 27, no. 6, pp. 1049-1096.
- Hillman, J & Warren, I 2004, 'An open framework for dynamic reconfiguration', *Proceedings of the 26th IEEE International Conference on Software Engineering (ICSE'04)*, IEEE Computer Society, pp. 594-603.
- Hnětynka, P & Plášil, F 2006, 'Dynamic reconfiguration and access to services in hierarchical component models', in I Gorton, GT Heineman, I Crnkovic, HW Schmidt, JA Stafford, CA Szyperski & K Wallnau (eds), *Component-Based Software Engineering*, vol. 4063/2006, Springer, Berlin / Heidelberg, pp. 352-359.
- Hofmeister, C 1993, 'Dynamic Reconfiguration of Distributed Applications', Ph. D. thesis, University of Maryland.
- Holder, O, Ben-Shaul, I & Gazit, H 1999, 'System support for dynamic layout of distributed applications', *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*, pp. 403-411.
- Hong, S, van den Goor, G & Brinkkemper, S 1993, 'A formal approach to the comparison of object-oriented analysis and design methodologies', *Proceeding of the 26th Hawaii International Conference on System Sciences* vol. iv, ed. G van den Goor, pp. 689-698.
- Horie, M, Pang, JC, Manning, EG & Shoja, GC 1998, 'Using meta-interfaces to support secure dynamic system reconfiguration', *Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDs'98)*, pp. 164-171.
- Horning, JJ, Lauer, HC, Melliar-Smith, PM & Randell, B 1974, 'A program structure for error detection and recovery', *Proceedings of an International Symposium on Operating Systems*, Springer-Verlag, pp. 171-187.
- Howard, P 2006, *Compuware Uniface 9: Product Evaluation*, Bloor Research, viewed 25 Aug 2007 <http://www.compuware.com/products/uniface/resources/6197_ENG_HTML.asp>.
- Hu, JM & Grefen, P 2003, 'Conceptual framework and architecture for service mediating workflow management', *Information and Software Technology*, vol. 45, no. 13, pp. 929-939.
- Huang, G, Mei, H & Yang, F-Q 2006, 'Runtime recovery and manipulation of software architecture of component-based systems', *Automated Software Engineering*, vol. 13, no. 2, pp. 257-281.
- Hubbers, J-W & Verhoef, D 2000, *Tutorial: Component Based Analysis and Design*, viewed 25 Aug 2007 <http://www.dsv.su.se/conferences/caise00/tutorials2.html#component_based_analysis_design>.
- Huhns, MN & Singh, MP 2005, 'Service-oriented computing: key concepts and principles', *IEEE Internet Computing*, vol. 9, no. 1, pp. 75-81.
- IEEE 2009, *SEVOCAB: Software and Systems Engineering Vocabulary*, IEEE, viewed 10 Jan 2009 <http://pascal.computer.org/sev_display/>.
- IEEE Computer Society 1998, *IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications*, IEEE Computer Society.
- livari, J 1995, 'Object-orientation as structural, functional and behavioural modelling: a comparison of six methods for object-oriented analysis', *Information and Software Technology*, vol. 37, no. 3, pp. 155-163.
- livari, J, Hirschheim, R & Klein, HK 1998, 'A Paradigmatic Analysis Contrasting Information Systems Development Approaches and Methodologies', *Information Systems Research*, vol. 9, no. 2, pp. 164-193.
- livari, J & Kerola, P 1983, 'A sociocybernetic framework for the feature analysis of information systems design methodologies', in TW Olle, H Sol & C Tully (eds), *Information System Design Methodologies: A Feature Analysis*, Elsevier B.V., Amsterdam, pp. 87-140.
- ISO/IEC 1995, *Open Distributed Processing - Reference Model: Part 1 to Part 4 (ITU-T Rec. X.901 to X.904 / ISO/IEC 10746-1 to 10746-4)*, ISO/IEC Press.
- ISO/IEC 1998, *ISO/IEC 15026:1998 Information technology -- System and software integrity levels*, ISO/IEC Press.
- ISO/IEC 2001, *Software engineering -- Product quality -- Part 1: Quality model ISO/IEC 9126-*

- 1:2001, ISO/IEC Press.
- ISO/IEC 2004, *ISO/IEC 15909-1:2004: Systems and software engineering -- High-level Petri nets -- Part 1: Concepts, definitions and graphical notation*, ISO/IEC Press.
- ISO/IEC 2005, *ISO/IEC 25000:2005: Software engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE*, ISO/IEC Press.
- ISO/IEC 2006, *ISO/IEC 15414:2006 Information technology -- Open distributed processing -- Reference model -- Enterprise language*, ISO/IEC Press.
- ISO/IEC 2007, *Software Engineering -- Metamodel for Development Methodologies ISO/IEC 24744:2007*, ISO/IEC Press.
- ISO/IEC 2008, *Software and Systems Engineering Vocabulary ISO/IEC FCD 24765*, ISO/IEC Press.
- ISO/IEC 2010, *ISO/IEC 24744:2007/Amd 1:2010 : Software Engineering - Metamodel for Development Methodologies - Amendment 1: Notation*, ISO/IEC Press.
- ISO/IEC 2011, *ISO/IEC FDIS 25010: Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models*, ISO/IEC Press.
- Janssens, N, Joosen, W & Verbaeten, P 2005, 'NeCoMan: middleware for safe distributed-service adaptation in programmable networks', *Distributed Systems Online, IEEE*, vol. 6, no. 7.
- Jayaratna, N 1994, *Understanding and Evaluating Methodologies: NIMSAD, a Systematic Framework*, McGraw-Hill Inc.
- Jayaswal, K 2005, 'No time for down time', in K Jayaswal (ed.), *Administering Data Centers: Servers, Storage, and Voice over IP*, John Wiley & Sons Inc.
- Jiao, W & Mei, H 2005, 'Dynamic architectural connectors in cooperative software systems', *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2005)*, ed. M Hong, IEEE Computer Society, pp. 477-486.
- Jones, CB, Romanovsky, AB & Welch, I 2002, 'A structured approach to handling on-line interface upgrades', *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment (COMPSAC'02)*, IEEE Computer Society, pp. 1000-1005.
- Jones, S & Morris, M 2005, *A methodology for Service Architectures (OASIS SOA Adoption Blueprints draft)*, Capgemini UK plc, <<http://www.oasis-open.org/committees/download.php/15071/A%20methodology%20for%20Service%20Architectures%201%202%204%20-%20OASIS%20Contribution.pdf>>.
- Kaminski, P, Müller, H & Litoiu, M 2006, 'A design for adaptive web service evolution', *Proceedings of the 2006 International Workshop on Self-adaptation and Self-managing Systems*, ACM, Shanghai, China, pp. 86-92.
- Karamanolis, CT & Magee, JN 1996, 'A replication protocol to support dynamically configurable groups of servers', *Proceedings of the 3rd International Conference on Configurable Distributed Systems (ICCDs'96)*, pp. 161-168.
- Karastoyanova, D, Houspanossian, A, Cilia, M, Leymann, F & Buchmann, A 2005, 'Extending BPEL for run time adaptability', *Proceedings of the 9th IEEE International EDOC Enterprise Computing Conference (EDOC'05)* pp. 15-26.
- Karlsson, F & Ågerfalk, PJ 2004, 'Method configuration: adapting to situational characteristics while creating reusable assets', *Information and Software Technology*, vol. 46, no. 9, pp. 619-633.
- Karsai, G, Massacci, F, Osterweil, L & Schieferdecker, I 2010, 'Evolving embedded systems', *Computer*, vol. 43, no. 5, pp. 34-40.
- Kataoka, Y, Notkin, D, Ernst, MD & Griswold, WG 2001, 'Automated support for program refactoring using invariants', *Proceedings of the 17th IEEE International Conference on Software Maintenance (ICSM'01)*, IEEE Computer Society, Florence, Italy pp. 736-743.
- Kemerer, CF & Slaughter, S 1999, 'An empirical approach to studying software evolution', *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 493-509.
- Kempter, B, Olkovich, L & Rachinskiy, E 2007, *Adaptive Services Grid Deliverable D6-III-3: Performance Engineering Methodology* Adaptive Services Grid (ASG), viewed 20 Jun 2008 <http://tb0.asg-platform.org/download/downloadrequest.php?asgdocument=D6.III-3_Performance_Engineering.pdf>.
- Kerlinger, FN 1986, *Foundations of behavioral research*, 3rd edn, Holt, Rinehart & Winston, New York.

- Khalaf, R, Mukhi, N & Weerawarana, S 2003, 'Service-oriented composition in BPEL4WS', *Proceedings of the 2003 World Wide Web Conference, Web Services Track*, ACM, Budapest, Hungary, viewed 14 Jul 2007 <http://www.www2003.org/cdrom/papers/alternate/P768/choreo_html/p768-khalaf.htm>.
- Kim, I-G, Bae, D-H & Hong, J-E 2007, 'A component composition model providing dynamic, flexible, and hierarchical composition of components for supporting software evolution', *Journal of Systems and Software*, vol. 80, no. 11, pp. 1797-1816.
- Kim, Y & Yun, H 2006, 'An approach to modeling service-oriented development process', *Proceedings of the IEEE International Conference on Services Computing (SCC'06)*, IEEE Computer Society, pp. 273-276.
- Kitchenham, B, Brereton, OP, Budgen, D, Turner, M, Bailey, J & Linkman, S 2009, 'Systematic literature reviews in software engineering - A systematic literature review', *Information and Software Technology*, vol. 51, no. 1, pp. 7-15.
- Kitchenham, B, Linkman, S & Law, D 1997, 'DESMET: a methodology for evaluating software engineering methods and tools', *Computing & Control Engineering Journal*, vol. 8, no. 3, pp. 120-126.
- Kitchenham, BA 1996, 'Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods', *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 1, pp. 11-14.
- Kitchenham, BA 2004, *Procedures for Undertaking Systematic Reviews*, Joint Technical Report, Computer Science Department, Keele University (TR/SE-0401) and National ICT Australia Ltd (0400011T.1).
- Kitchenham, BA & Jones, L 1997, 'Evaluating software engineering methods and tools part 6: identifying and scoring features', *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 2, pp. 16-18.
- Klein, M & Dellarocas, C 2000, 'A knowledge-based approach to handling exceptions in workflow systems', *Computer Supported Cooperative Work (CSCW)*, vol. 9, no. 3, pp. 399-412.
- Kluth, A 2004, 'Survey: Information Technology: Make it simple', *The Economist*, vol. 17, pp. 1-14.
- Kniesel, G 1999, 'Type-safe delegation for run-time component adaptation', in R Guerraoui (ed.), *ECOOP' 99 – Object-Oriented Programming*, vol. 1628, Springer Berlin / Heidelberg, pp. 668-668.
- Knight, JC 2002, 'Safety critical systems: challenges and directions', *Proceedings of the 24th International Conference on Software Engineering*, ACM, Orlando, Florida, pp. 547-550.
- Kon, F & Campbell, RH 2000, 'Dependence management in component-based distributed systems', *IEEE Concurrency*, vol. 8, no. 1, pp. 26-36.
- Kon, F, Campbell, RH & Nahrstedt, K 2001, 'Using dynamic configuration to manage a scalable multimedia distribution system', *Computer Communications*, vol. 24, pp. 105-123.
- Kon, F, Gill, B, Anand, M, Campbell, R & Mickunas, M 2000, 'Secure dynamic reconfiguration of scalable CORBA systems with mobile agents', in *Agent Systems, Mobile Agents, and Applications*, Springer, Berlin / Heidelberg, pp. 86-98.
- Koning, M, Sun, C-a, Sinnema, M & Avgeriou, P 2009, 'VxBPEL: supporting variability for Web services in BPEL', *Information and Software Technology*, vol. 51, no. 2, pp. 258-269.
- Koskinen, J 2004, *Software Maintenance Costs*, University of Jyväskylä, Finland, viewed 22 Mar 2007 <<http://www.cs.jyu.fi/~koskinen/smcosts.htm>>.
- Kramer, J & Magee, J 1990, 'The evolving philosophers problem: dynamic change management', *IEEE Transactions on Software Engineering*, vol. 16, no. 11, pp. 1293-1306.
- Kramer, J & Magee, J 1998, 'Analysing dynamic change in distributed software architectures', *IEEE Proceedings - Software*, vol. 145, no. 5, pp. 146-154.
- Kramer, J & Magee, J 2007, 'Self-managed systems: an architectural challenge', *Future of Software Engineering, 2007 (FOSE '07)*, IEEE Computer Society, pp. 259-268.
- Kruchten, P 2003, *The Rational Unified Process: An Introduction*, 3rd edn, Addison-Wesley.
- Kuhn, TS 1996, *The Structure of Scientific Revolutions*, 3rd edn, University of Chicago Press.
- Kulkarni, SS & Biyani, KN 2004, 'Correctness of component-based adaptation', in I Crnkovic, JA Stafford, HW Schmidt & K Wallnau (eds), *Component-Based Software Engineering*, vol. 3054, Springer Berlin / Heidelberg, pp. 48-58.
- Kung, CH 1983, 'An analysis of three conceptual models with time perspective', in TW Olle, H

- Sol & C Tully (eds), *Information System Design Methodologies: A Feature Analysis*, Elsevier B.V., Amsterdam, pp. 141-168.
- Ladd, DA 2009, 'Everybody likes Likert: using a variable-interval slider to collect interval-level individual options', *Proceedings of the International Conference on Information Systems (ICIS 2009)*, Association for Information Systems, viewed 6 Jul 2011 <<http://aisel.aisnet.org/icis2009/100>>.
- Lamport, L 1977, 'Proving the correctness of multiprocess programs', *IEEE Transactions on Software Engineering*, vol. 3, no. 2, pp. 125-143.
- Lamport, L, Shostak, R & Pease, M 1982, 'The Byzantine Generals Problem', *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382-401.
- Laprie, JC 1995, 'Dependable computing and fault tolerance: concepts and terminology', *Proceedings of the 25th International Symposium on Fault-Tolerant Computing: Highlights from Twenty-Five Years*, IEEE, pp. 2-11.
- Lassing, N, Rijsenbrij, D & van Vliet, H 2003, 'How well can we predict changes at architecture design time?', *Journal of Systems and Software*, vol. 65, no. 2, pp. 141-153.
- LDJ Trust 2003, "Freedom" *Service-Oriented Methodology*, viewed 5 Jun 2008 <<http://www.jreality.com/freedom/index.html>>.
- Lee, AS 1989, 'A scientific methodology for MIS case studies', *MIS Quarterly*, vol. 13, no. 1, pp. 33-50.
- Lee, SP, Chan, LP & Lee, EW 2006, 'Web Services Implementation Methodology for SOA application', *IEEE International Conference on Industrial Informatics*, IEEE, pp. 335-340.
- Lee, YF & Chang, RC 2005, 'Java-based component framework for dynamic reconfiguration', *IEE Proceedings - Software*, vol. 152, no. 3, pp. 110-118.
- Léger, M, Ledoux, T & Coupaye, T 2010, 'Reliable dynamic reconfigurations in a reflective component model', in L Grunske, R Reussner & F Plasil (eds), *Component-Based Software Engineering*, vol. 6092, Springer Berlin / Heidelberg, pp. 74-92.
- Lehman, MM & Ramil, JF 2001, 'Rules and tools for software evolution planning and management', *Annals of Software Engineering*, vol. 11, pp. 15-44.
- Lehner, T, Bayer, J, Bella, F & Ocampo, A 2006, *Adaptive Services Grid Deliverable D6.III-2: ASG Development Process - Application and Service Engineering* Adaptive Services Grid (ASG), viewed 13 Jun 2008 <<http://tb0.asg-platform.org/download/downloadrequest.php?asgdocument=D6.III-2.pdf>>.
- Leitner, P, Michlmayr, A, Rosenberg, F & Dustdar, S 2008, 'End-to-end versioning support for Web Services', *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 1*, IEEE Computer Society, pp. 59-66.
- Leszak, M, Perry, DE & Stoll, D 2002, 'Classification and evaluation of defects in a project retrospective', *Journal of Systems and Software*, vol. 61, no. 3, pp. 173-187.
- Leveson, NG 1986, 'Software safety: why, what, and how', *ACM Computing Surveys*, vol. 18, no. 2, pp. 125-163.
- Li, G, Han, Y, Zhao, Z, Wang, J & Wagner, RM 2006, 'Facilitating dynamic service compositions by adaptable service connectors', *International Journal of Web Services Research*, vol. 3, no. 1, pp. 68-85.
- Li, W 2009, 'DynaQoS[®]-RDF: a best effort for QoS-assurance of dynamic reconfiguration of dataflow systems', *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 1, pp. 19-48.
- Liang, Y 2000, 'An approach to assessing and comparing object-oriented analysis methods', *Journal of Object-Oriented Programming*, vol. 13, no. 3, pp. 27-33.
- Lientz, BP & Swanson, EB 1980, *Software Maintenance Management*, Addison-Wesley.
- Lim, AS 1996, 'Abstraction and composition techniques for reconfiguration of large-scale complex applications', *Proceedings of the 3rd International Conference on Configurable Distributed Systems (ICCDs'96)* pp. 186-193.
- Lincke, D-M & Schmid, B 1998, 'Mediating electronic product catalogs', *Communications of the ACM*, vol. 41, no. 7, pp. 86-88.
- Lindqvist, U & Jonsson, E 1998, 'A map of security risks associated with using COTS', *Computer*, vol. 31, no. 6, pp. 60-66.
- Liu, M, Ma, D & Zhao, Y 2009, 'An approach to identifying conversation dependency in service oriented system during dynamic evolution', *Proceedings of the 2009 ACM symposium on Applied Computing*, ACM, Honolulu, Hawaii, pp. 1072-1073.
- LIXI 2010, *Welcome to the LIXI home page*, viewed 8 May 2010 <<http://www.lixi.org.au/>>.

- Lopes, A, Wermelinger, M & Fiadeiro, JL 2003, 'Higher-Order Architectural Connectors', *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 1, pp. 64-104.
- Loulou, I, Jmaiel, M, Drira, K & Kacem, AH 2010, 'P/S-CoM: Building correct by design publish/subscribe architectural styles with safe reconfiguration', *Journal of Systems and Software*, vol. 83, no. 3, pp. 412-428.
- Lovrek, I, Jezic, G, Kusek, M, Ljubi, I, Caric, A, Huljenic, D, Desic, S & Labor, O 2003, 'Improving software maintenance by using agent-based remote maintenance shell', *Proceedings of the 19th International Conference on Software Maintenance (ICSM 2003)*, IEEE Computer Society, pp. 440-449.
- Maes, P 1987, 'Concepts and experiments in computational reflection', *Conference proceedings on Object-oriented programming systems, languages and applications*, ACM, Orlando, Florida, United States, pp. 147-155.
- Magee, J & Kramer, J 1996, 'Dynamic structure in software architectures', *Proceedings of the 4th ACM SIGSOFT symposium on Foundations of Software Engineering*, ACM, San Francisco, California, United States, pp. 3-14.
- Mak, R, Walton, J, Keely, L, Heher, D & Chan, L 2005, 'A reliable service-oriented architecture for NASA's Mars Exploration Rover mission', *Proceedings of 2005 IEEE Aerospace Conference*, IEEE, Big Sky, MT, pp. 1-14.
- Mao, C, Zhang, J & Lu, Y 2007, 'Using dependence matrix to support change impact analysis for CBS', *International Conference on Computational Science and its Applications, 2007 (ICCSA 2007)*, ed. J Zhang, IEEE Computer Society, pp. 192-200.
- March, ST & Smith, GF 1995, 'Design and natural science research on information technology', *Decision Support Systems*, vol. 15, no. 4, pp. 251-266.
- Mari, M & Eila, N 2003, 'The impact of maintainability on component-based software systems', *Proceedings of the 29th Euromicro Conference*, pp. 25-32.
- Martínez-Beneito, MA, López-Quilez, A & Botella-Rocamora, P 2008, 'An autoregressive approach to spatio-temporal disease mapping', *Statistics in Medicine*, vol. 27, no. 15, pp. 2874-2889.
- Matinlassi, M 2004, 'Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA', *Proceedings of the 26th International Conference on Software Engineering*, IEEE Computer Society, pp. 127-136.
- McKinley, PK, Sadjadi, SM, Kasten, EP & Cheng, BHC 2004, 'Composing adaptive software', *Computer*, vol. 37, no. 7, pp. 56-64.
- Meadows, C & McLean, J 1998, 'Security and dependability: then and now', *Proceedings of Computer Security, Dependability and Assurance: From Needs to Solutions (CSDA'98)*, IEEE Computer Society, pp. 166-170.
- Medvidovic, N, Rosenblum, DS & Taylor, RN 1999, 'A language and environment for architecture-based software development and evolution', *Proceedings of the 21st IEEE International Conference on Software Engineering (ICSE'99)*, IEEE Press, pp. 44-53.
- Medvidovic, N & Taylor, R 1997, 'Exploiting architectural style to develop a family of applications', *IEEE Proceedings Software Engineering*, vol. 144, no. 5, pp. 237-248.
- Mens, T 2008, 'Introduction and roadmap: history and challenges of software evolution', in T Mens & S Demeyer (eds), *Software Evolution*, Springer, pp. 1-11.
- Mens, T & D'Hondt, T 2000, 'Automating support for software evolution in UML', *Automated Software Engineering*, vol. 7, no. 1, pp. 39-59.
- Mens, T & Demeyer, S 2008, *Software Evolution*, Springer.
- Mens, T, Magee, J & Rumpe, B 2010, 'Evolving software architecture descriptions of critical systems', *Computer*, vol. 43, no. 5, pp. 42-48.
- Meredith, LG & Bjorg, S 2003, 'Contracts and types', *Communications of the ACM*, vol. 46, no. 10, pp. 41-47.
- Merideth, MG, Iyengar, A, Mikalsen, T, Tai, S, Rouvellou, I & Narasimhan, P 2005, 'Thema: Byzantine-fault-tolerant middleware for web-service applications', *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, IEEE Computer Society, pp. 131-142.
- Milanovic, N & Malek, M 2004, 'Current solutions for Web Service composition', *IEEE Internet Computing*, vol. 8, no. 6, pp. 51-59.
- Milazzo, M, Pappalardo, G, Tramontana, E & Ursino, G 2005, 'Handling run-time updates in distributed applications', *Proceedings of the 2005 ACM symposium on Applied computing (SAC'05)*, ACM, Santa Fe, New Mexico, pp. 1375-1380.

- Mittal, K 2006, *Service Oriented Unified Process*, viewed 3 Jun 2008 <<http://www.kunalmittal.com/html/soup.shtml>>.
- Monarchi, DE & Puhr, GI 1992, 'A research typology for object-oriented analysis and design', *Communications of the ACM*, vol. 35, no. 9, pp. 35-47.
- Morandini, M, Penserini, L & Perini, A 2008, 'Towards goal-oriented development of self-adaptive systems', *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ACM, Leipzig, Germany, pp. 9-16.
- Morrison, J & George, J, F. 1995, 'Exploring the software engineering component in MIS research', *Communications of the ACM*, vol. 38, no. 7, pp. 80-91.
- Morrison, R, Balasubramaniam, D, Kirby, G, Mickan, K, Warboys, B, Greenwood, R, Robertson, I & Snowdon, B 2007, 'A framework for supporting dynamic systems co-evolution', *Automated Software Engineering*, vol. 14, no. 3, pp. 261-292.
- Motahari Nezhad, HR, Benatallah, B, Martens, A, Curbera, F & Casati, F 2007, 'Semi-automated adaptation of service interactions', *Proceedings of the 16th international conference on World Wide Web*, ACM, Banff, Alberta, Canada, pp. 993-1002.
- Mukhija, A & Glinz, M 2005, 'Runtime adaptation of applications through dynamic recomposition of components', in, *Systems Aspects in Organic and Pervasive Computing - ARCS 2005*, vol. 3432/2005, Springer, Berlin / Heidelberg, pp. 124-138.
- Murphy, GC, Walker, RJ & Baniassad, ELA 1999, 'Evaluating emerging software development technologies: lessons learned from assessing Aspect-Oriented Programming', *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 438-455.
- Muskens, J & Chaudron, M 2004, 'Prediction of run-time resource consumption in multi-task component-based software systems', in, *Component-Based Software Engineering*, Springer, Berlin / Heidelberg, pp. 162-177.
- Myers, MD 2004, *Qualitative research in information systems*, MISQ Discovery, viewed 24 Sep 2010 <http://www.misq.org/discovery/MISQD_isworld/>.
- Ngwenyama, OK 1991, 'The critical social theory approach to information systems: problems and challenges', in HE Nissen, HK Klein & H R. (eds), *Information Systems Research: Contemporary Approaches and Emergent Traditions*, INFORMS, New York, NY, pp. 267-280.
- Niehaves, B & Stahl, BC 2006, 'Criticality, epistemology and behaviour vs. design - information systems research across different sets of paradigms', *Proceedings of the 14th European Conference on Information Systems*, Göteborg, Sweden.
- Nuseibeh, B & Easterbrook, S 2000, 'Requirements engineering: a roadmap', *Proceedings of the Conference on The Future of Software Engineering*, ACM, Limerick, Ireland, pp. 35-46.
- Nuseibeh, B, Finkelstein, A & Kramer, J 1996, 'Method engineering for multi-perspective software development', *Information and Software Technology*, vol. 38, no. 4, pp. 267-274.
- OASIS 2006, *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, OASIS, viewed 18 Jun 2008 <<http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>>.
- OASIS 2007, *Web Services Coordination (WS-Coordination) Version 1.1*, OASIS, viewed 4 Jul 2007 <<http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-os.pdf>>.
- Office of Government Commerce 2002, *Application Management*, The Stationery Office, London.
- Office of Government Commerce 2005a, *ITIL Service Support*, The Stationery Office, London.
- Office of Government Commerce 2005b, *PRINCE2: Managing Successful Projects*, 4th edn, TSO (The Stationery Office), London.
- Okoli, C & Pawlowski, SD 2004, 'The Delphi method as a research tool: an example, design considerations and applications', *Information & Management*, vol. 42, no. 1, pp. 15-29.
- Olle, TW, Sol, H & Tully, C 1983, *Information System Design Methodologies: A Feature Analysis*, Elsevier B.V., Amsterdam.
- Olle, TW, Sol, H & Verrijn-Stuart, AA 1986, *Information Systems Design Methodologies : Improving the Practice* Elsevier B.V., Amsterdam.
- OMG 2003a, *MDA Guide Version 1.0.1*, Report Number omg/2003-06-01, viewed 25 Jul 2007 <<http://www.omg.org/docs/omg/03-06-01.pdf>>.
- OMG 2003b, *Online Upgrades: Draft Adopted Specification*, Report Number ptc/03-08-07,

- viewed 21 Jul 2010 <<http://www.omg.org/cgi-bin/doc?ptc/2003-08-07>>.
- OMG 2005, *UML Profile for Schedulability, Performance, and Time Specification v1.1*, Report Number formal/2005-01-02, viewed 21 Jul 2010 <<http://www.omg.org/cgi-bin/doc?formal/2005-01-02>>.
- OMG 2008, *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)*, Report Number ad/2008-08-04, viewed 29 May 2009 <<http://www.omg.org/cgi-bin/doc?ad/2008-08-04>>.
- OMG 2009, *Business Process Modeling Notation Specification v1.2*, Report Number formal/2009-01-03, viewed 21 Jul 2010 <<http://www.omg.org/spec/BPMN/1.2>>.
- OMG 2010a, *OMG Unified Modeling Language™(OMG UML), Infrastructure, Version 2.3*, Report Number formal/2010-05-03, viewed 21 Jul 2010 <<http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/>>.
- OMG 2010b, *OMG Unified Modeling Language™(OMG UML), Superstructure, Version 2.3*, Report Number formal/2010-05-05, viewed 21 Jul 2010 <<http://www.omg.org/spec/UML/2.3/Superstructure/PDF/>>.
- OPFRO 2009, *OPEN Process Framework Repository Organization*, viewed 21 Jul 2010 <<http://www.opfro.org/>>.
- Oreizy, P, Gorlick, MM, Taylor, RN, Heimbigner, D, Johnson, G, Medvidovic, N, Quilici, A, Rosenblum, D & Wolf, A 1999, 'An architecture-based approach to self-adaptive software', *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 54-62.
- Oreizy, P, Medvidovic, N & Taylor, RN 1998, 'Architecture-based runtime software evolution', *Proceedings of the 20th IEEE International Conference on Software Engineering (ICSE'98)*, IEEE Computer Society, Kyoto, Japan, pp. 177-186.
- Oreizy, P, Medvidovic, N & Taylor, RN 2008, 'Runtime software adaptation: framework, approaches, and styles', *Companion of the 30th International Conference on Software Engineering*, ACM, Leipzig, Germany, pp. 899-910.
- Oreizy, P & Taylor, RN 1998, 'On the role of software architectures in runtime system reconfiguration', *Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDs'98)*, pp. 61-70.
- Oriol, M & Serugendo, GDM 2004, 'Disconnected service architecture for unanticipated run-time evolution of code', *IEE Proceedings - Software*, vol. 151, no. 2, pp. 95-107.
- Orlikowski, WJ & Baroudi, JJ 1991, 'Studying Information Technology in Organizations: Research Approaches and Assumptions', *Information Systems Research*, vol. 2, no. 1, pp. 1-28.
- Oueichek, I & Rousset de Pina, X 1996, 'Dynamic configuration management in the Guide object-oriented distributed system', *Proceedings of the 3rd International Conference on Configurable Distributed Systems (ICCDs'96)*, pp. 28-35.
- Pahl, C 2004, 'Adaptive development and maintenance of user-centric software systems', *Information and Software Technology*, vol. 46, no. 14, pp. 973-986.
- Paige, RF, Ostroff, JS & Brooke, PJ 2000, 'Principles for modelling language design', *Information and Software Technology*, vol. 42, no. 10, pp. 665-675.
- Papazoglou, M 2008, 'The challenges of service evolution', in *Advanced Information Systems Engineering*, vol. 5074/2008, Springer, Berlin / Heidelberg, pp. 1-15.
- Papazoglou, M & Kratz, B 2007, 'Web services technology in support of business transactions', *Service Oriented Computing and Applications*, vol. 1, no. 1, pp. 51-63.
- Papazoglou, MP & Georgakopoulos, D 2003, 'Service-oriented computing: introduction', *Communications of the ACM*, vol. 46, no. 10, pp. 24-28.
- Papazoglou, MP, Traverso, P, Dustdar, S & Leymann, F 2008, 'Service-oriented computing: A research roadmap', *International Journal of Cooperative Information Systems*, vol. 17, no. 2, pp. 223-255.
- Papazoglou, MP & van den Heuvel, W-J 2006, 'Service-oriented design and development methodology', *International Journal of Web Engineering and Technology*, vol. 2, no. 4, pp. 412-442.
- Parzyjegl, H, Muhl, GG & Jaeger, MA 2006, 'Reconfiguring publish/subscribe overlay topologies', *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW 2006)*, pp. 29-29.
- Pasley, J 2005, 'How BPEL and SOA are changing Web services development', *Internet Computing, IEEE*, vol. 9, no. 3, pp. 60-67.
- Patton, MQ 2002, *Qualitative Research and Evaluation Methods*, Sage.

- Paulk, M, Curtis, B, Chrissis, M & Weber, C 1993, *Capability Maturity Model for Software (Version 1.1)*, Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Pittsburgh PA.
- Peffers, K, Tuunanen, T, Rothenberger, M & Chatterjee, S 2008, 'A design science research methodology for information systems research', *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45-77.
- Pellegrinelli, S 1997, 'Programme management: organising project-based change', *International Journal of Project Management*, vol. 15, no. 3, pp. 141-149.
- Pelliccione, P, Tivoli, M, Bucchiarone, A & Polini, A 2008, 'An architectural approach to the correct and automatic assembly of evolving component-based systems', *Journal of Systems and Software*, vol. 81, no. 12, pp. 2237-2251.
- Peltz, C 1999, 'A hierarchical technique for composing COM based components', *Proceedings of the 2nd International Workshop on Component-Based Software Engineering (CBSE'99)*, IEEE Computer Society, pp. 17-18, viewed 17 Dec 2007 <<http://www.sei.cmu.edu/pacc/icse99/papers/26/26.pdf>>.
- Petrenko, M, Poshyvattyk, D, Rajlich, V & Buchta, J 2007, 'Teaching Software Evolution in Open Source', *Computer*, vol. 40, no. 11, pp. 25-31.
- Pfleeger, SL 1995, 'Experimental design and analysis in software engineering', *Annals of Software Engineering*, vol. 1, pp. 219-253.
- Pfleeger, SL 1999, 'Albert Einstein and empirical software engineering', *Computer*, vol. 32, no. 10, pp. 32-38.
- Pfleeger, SL & Bohner, SA 1990, 'A framework for software maintenance metrics', *Proceedings of the International Conference on Software Maintenance*, IEEE, San Diego, C.A., pp. 320-327.
- Plášil, F, Bálek, D & Janeček, R 1998, 'SOFA/DCUP: architecture for component trading and dynamic updating', *Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDs'98)*, pp. 43-51.
- Ponnekanti, SR & Fox, A 2004, 'Interoperability among independently evolving web services', *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, Springer-Verlag New York, Inc., Toronto, Canada, pp. 331-351.
- Pressman, RS 2005, *Software Engineering: A Practitioners' Approach*, 6th edn, McGraw-Hill Int.
- Rajlich, V & Gosavi, P 2004, 'Incremental change in object-oriented programming', *IEEE Software*, vol. 21, no. 4, pp. 62-69.
- Ralyté, J, Deneckère, R & Rolland, C 2003, 'Towards a generic model for situational method engineering', in *Proceedings of the 15th Conference on Advanced Information Systems Engineering, CAISE'03*, vol. 2681/2010, Springer Berlin / Heidelberg, Klagenfurt/Velden, Austria, pp. 95-110.
- Ralyté, J & Rolland, C 2001, 'An assembly process model for method engineering', in *Proceedings of the 13th International Conference on Advanced Information Systems Engineering, CAiSE 2001*, vol. 2068/2001, Springer Berlin / Heidelberg, Interlaken, Switzerland, pp. 267-283.
- Ralyté, J, Rolland, C & Deneckère, R 2004, 'Towards a meta-tool for change-centric method engineering: a typology of generic operators', in A Persson & J Stirna (eds), *Advanced Information Systems Engineering*, vol. 3084, Springer Berlin / Heidelberg, pp. 695-717.
- Ramsin, R & Paige, R, F 2008, 'Process-centered review of object oriented software development methodologies', *ACM Computing Surveys*, vol. 40, no. 1, pp. 1-89.
- Rasche, A & Polze, A 2003, 'Configuration and Dynamic reconfiguration of component-based applications with Microsoft .NET', *Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)*, IEEE Computer Society, pp. 164-171.
- Redmond, B & Cahill, V 2006, 'Supporting unanticipated dynamic adaptation of application behaviour', in B Magnusson (ed.), *ECOOP 2002 – Object-Oriented Programming*, vol. 2374, Springer Berlin / Heidelberg, pp. 29-53.
- Robertson, P & Williams, B 2006, 'Automatic recovery from software failure', *Communications of the ACM*, vol. 49, no. 3, pp. 41-47.
- Robey, D 1996, 'Diversity in Information Systems Research: Threat, Promise, and Responsibility', *Information Systems Research*, vol. 7, no. 4, pp. 400-408.
- Roddick, JF 1992, 'Schema evolution in database systems: an annotated bibliography', *ACM SIGMOD Record*, vol. 21, no. 4, pp. 35-40.

- Rossi, M, Ramesh, B, Lyytinen, K & Tolvanen, J-P 2004, 'Managing evolutionary method engineering by method rationale', *Journal of the Association for Information Systems*, vol. 5, no. 9, pp. 356-391.
- Rowe, D, Leaney, J & Lowe, D 1998, 'Defining systems evolvability', *Proceedings of the 11th IEEE Conference on Computer Based Systems (ECBS'98)*, IEEE Press, Jerusalem, Israel, pp. 45-52.
- Ryan, ND & Wolf, AL 2004, 'Using event-based translation to support dynamic protocol evolution', *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, IEEE Computer Society, pp. 408-417.
- Salinesi, C, Etien, A & Zoukar, I 2004, 'A systematic approach to express IS evolution requirements using gap modelling and similarity modelling techniques', in *Advanced Information Systems Engineering*, Springer, Berlin / Heidelberg, pp. 338-352.
- Schmerl, B, Aldrich, J, Garlan, D, Kazman, R & Yan, H 2006, 'Discovering architectures from running systems', *IEEE Transactions on Software Engineering*, vol. 32, no. 7, pp. 454-466.
- Schuster, S 2008, 'Forward', *Proceedings of the 7th International Conference on Composition-Based Software Systems (ICCBSS 2008)*, IEEE Computer Society, Madrid, Spain
- Seaman, CB 1999, 'Qualitative methods in empirical studies of software engineering', *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557-572.
- SeCSE 2006, *Report on Methodological Approach to Design Service Compositions (v 2.0)* The SeCSE consortium, viewed 9 Aug 2009 <<http://www.secse-project.eu/wp-content/uploads/2007/08/a3d32-report-on-methodological-approach-to-design-service-composition-v2.zip>>.
- SeCSE 2007, *SeCSE Methodology, Version 3*, The SeCSE consortium, viewed 9 Aug 2009 <<http://www.secse-project.eu/wp-content/uploads/2007/08/a5-d4-2-secse-methodology-version-3.pdf>>.
- SeCSE 2008, *Testing Method Definition*, The SeCSE consortium, viewed 9 Aug 2009 <<http://www.secse-project.eu/wp-content/uploads/a1d34-testing-method-definition-v4-final.pdf>>.
- Segal, ME 2002, 'Online software upgrading: new research directions and practical considerations', *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment (COMPSAC'02)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 977-981.
- Segal, ME & Frieder, O 1993, 'On-the-fly program modification: systems for dynamic updating', *IEEE Software*, vol. 10, no. 2, pp. 53-65.
- SEI 2006, *CMMI for Development, Version 1.2, Technical Report CMU/SEI-2006-TR-008*, Software Engineering Institute, Pittsburgh PA, viewed 17 Apr 2007 <<http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr008.pdf>>.
- Shatz, M 1993, *Development of Distributed Software: Concepts and Tools*, Macmillan Publ. Co.
- Shaw, M, DeLine, R, Klein, DV, Ross, TL, Young, DM & Zelesnik, GZ 1995, 'Abstractions for software architecture and tools to support them', *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 314-335.
- Shaw, M, DeLine, R & Zelesnik, G 1996, 'Abstractions and Implementations for architectural connections', *Proceedings of the 3rd International Conference on Configurable Distributed Systems (ICCDs'96)*, pp. 2-10.
- Shaw, M & Garlan, D 1996, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall Inc.
- Shrivastava, SK & Wheeler, SM 1998, 'Architectural support for dynamic reconfiguration of distributed workflow applications', *IEEE Proceedings - Software*, vol. 145, no. 5, pp. 155-162.
- Siau, K & Rossi, M 1998, 'Evaluation of information modeling methods -- a review', *Proceedings of the 31st Annual Hawaii International Conference on System Sciences - Volume 5*, IEEE Computer Society, pp. 314-422.
- Siau, K & Rossi, M 2007, 'Evaluation techniques for systems analysis and design modelling methods - a review and comparative analysis', *Information Systems Journal*.
- Siljee, J, Bosloper, I, Nijhuis, J & Hammer, D 2005, 'DySOA: making service systems self-adaptive', in B Benatallah, F Casati & P Traverso (eds), *Service-Oriented Computing - ICSSOC 2005*, vol. 3826, Springer Berlin / Heidelberg, pp. 255-268.

- SINTEF 2007, *COMET - component and model-based development methodology*, SINTEF ICT, viewed 20 Jun 2007 <<http://www.modelbased.net/comet/>>.
- Sol, HG 1992, 'Information systems development: a problem solving approach', in WW Cotterman & JA Senn (eds), *Challenges and Strategies for Research in Systems Development*, John Wiley & Sons Inc., pp. 151-161.
- Sparling, M 2000, 'Lessons learned through six years of component-based development', *Communications of the ACM*, vol. 43, no. 10, pp. 47-53.
- Stal, M 2006, 'Using architectural patterns and blueprints for service-oriented architecture', *IEEE Software*, vol. 23, no. 2, pp. 54-61.
- Standards Australia 2004, *AS/NZS 4360:2004 Risk Management*, 3rd edn.
- Stojanovic, Z, Dahanayake, A & Sol, H 2004a, 'An evaluation framework for component-based and service-oriented system development methodologies', in K Siau (ed.), *Advanced Topics in Database Research*, vol. 3, IGI Global, pp. 45-68.
- Stojanovic, Z, Dahanayake, A & Sol, H 2004b, 'Modeling and design of service-oriented architecture', *IEEE International Conference on Systems, Man and Cybernetics*, IEEE Computer Society, pp. 4147-4152.
- Sun, P & Jiang, CJ 2009, 'Analysis of workflow dynamic changes based on Petri net', *Information and Software Technology*, vol. 51, no. 2, pp. 284-292.
- SUPER 2007, *Semantic Business Process Life Cycle version 1.0*.
- Swaminathan, B & Goldman, KJ 1996, 'Data handles and virtual connections: high-level support for anonymous reconfiguration', *Proceedings of the 3rd International Conference on Configurable Distributed Systems (ICCDs'96)*, pp. 19-26.
- Szyperski, C 2003, *Component Software: Beyond Object Oriented Programming*, 2nd edn, Addison-Wesley.
- Taentzer, G, Goedicke, M & Meyer, T 2000, 'Dynamic change management by distributed graph transformation: towards configurable distributed systems', in H Ehrig, G Engels, H-J Kreowski & G Rozenberg (eds), *Theory and Application of Graph Transformations*, vol. 1764, Springer Berlin / Heidelberg, pp. 179-193.
- ter Hofstede, AHM & Verhoef, TF 1997, 'On the feasibility of situational method engineering', *Information Systems*, vol. 22, no. 6-7, pp. 401-422.
- The Eclipse Foundation 2009, *Eclipse Process Framework Project (EPF)*, viewed 7 Nov 2009 <<http://www.eclipse.org/epf/>>.
- The Object Agency Inc. 1995, *A Comparison of Object-Oriented Development Methodologies*, viewed 5 Jun 2007 <<http://www.toa.com/smn?mcr.html>>.
- Torres-Pomales, W 2000, *Software Fault Tolerance: A Tutorial*, NASA Langley Research Center, Hampton, Virginia, viewed 18 Aug 2009 <http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20000120144_2000175863.pdf>.
- Tosic, V, Erradi, A & Maheshwari, P 2007, 'WS-Policy4MASC - a WS-Policy extension used in the MASC middleware', *Proceedings of the IEEE International Conference on Services Computing (SCC 2007)*, IEEE Computer Society, pp. 458-465.
- Tran, Q-NN & Low, G 2008, 'MOBMAS: A methodology for ontology-based multi-agent systems development', *Information and Software Technology*, vol. 50, no. 7-8, pp. 697-722.
- Truyen, E, Vanhaute, B, Jørgensen, BN, Joosen, W & Verbaeten, P 2001, 'Dynamic and selective combination of extensions in component-based applications', *Proceedings of the 23rd International Conference on Software Engineering*, IEEE Computer Society, Toronto, Ontario, Canada, pp. 233-242.
- Tsai, WT, Fan, C, Chen, Y, Paul, R & Chung, J-Y 2006, 'Architecture classification for SOA-based applications', *Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06) - Volume 00*, IEEE Computer Society, pp. 295-302.
- Tsai, WT, Liu, X & Chen, Y 2005, 'Distributed policy specification and enforcement in service-oriented business systems', *Proceedings of the IEEE International Conference on e-Business Engineering*, IEEE Computer Society, pp. 10-17.
- Tsai, WT, Song, W, Paul, R, Cao, Z & Huang, H 2004, 'Services-oriented dynamic reconfiguration framework for dependable distributed computing', *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04) - Volume 01*, IEEE Computer Society, Washington, DC, pp. 554-559.
- Unhelkar, B 1997, 'Effect of granularity of object-oriented design on modelling an enterprise, and its application to financial risk management', Ph. D. thesis, The University of

- Technology, Sydney, Sydney.
- Vaishnavi, V & Kuechler, W 2004, *Design Research in Information Systems*, Association for Information Systems, viewed 07 Feb 2010 <<http://desrist.org/design-research-in-information-systems>>.
- van Gurp, J, Bosch, J & Svahnberg, M 2001, 'On the notion of variability in software product lines', *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01) - Volume 00*, IEEE Computer Society, pp. 45-54.
- Vandewoude, Y & Berbers, Y 2002, 'Run-time evolution for embedded component-oriented systems', *Proceedings of the 18th International Conference on Software Maintenance*, ed. Y Berbers, IEEE Computer Society, pp. 242-245.
- Vandewoude, Y, Ebraert, P, Berbers, Y & D'Hondt, T 2007, 'Tranquility: a low disruptive alternative to quiescence for ensuring safe dynamic updates', *IEEE Transactions on Software Engineering*, vol. 33, no. 12, pp. 856-868.
- Velasco Elizondo, P & Lau, K-K 2010, 'A catalogue of component connectors to support development with reuse', *Journal of Systems and Software*, vol. 83, no. 7, pp. 1165-1178.
- Venable, JR & Travis, J 1999, 'Using a group support system for the distributed application of soft systems methodology', *Proceedings of the 10th Australasian Conference on Information Systems*, eds B Hope & P Yoong, Wellington, New Zealand, pp. 1105-1117.
- Veryard, R 1998, *SCIPIO: Aims, Principles and Structure v0.9*, viewed 30 Aug 2007 <<http://www.users.globalnet.co.uk/~rxv/scipio/SCIPIOap.PDF>>.
- Voas, J 1998, 'Certifying off-the-shelf software components', *Computer*, vol. 31, no. 6, pp. 53-59.
- W3C 2003, *SOAP Version 1.2*, W3C, viewed 26 Dec 2006 <<http://www.w3.org/TR/soap12/>>.
- Walls, JG, Widmeyer, GR & El Sawy, OA 1992, 'Building an information system design theory for vigilant EIS', *Information Systems Research*, vol. 3, no. 1, pp. 36-59.
- Wang, G, Ungar, L & Klawitter, D 1999, 'Component assembly for OO distributed systems', *Computer*, vol. 32, no. 7, pp. 71-78.
- Wang, Q, Chen, F, Mei, H & Yang, F 2002, 'An application server to support online evolution', *Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002)*, pp. 131-140.
- Wang, Q, Shen, J, Wang, X & Mei, H 2006, 'A component-based approach to online software evolution', *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 3, pp. 181-205.
- Warren, I & Sommerville, I 1996, 'A model for dynamic configuration which preserves application integrity', *Proceedings of the 3rd International Conference on Configurable Distributed Systems 1996 (ICCDs'96)*, IEEE Computer Society, Annapolis, Maryland, pp. 81-88.
- Wasserman, AI, Freeman, P & Porcella, M 1983, 'Characteristics of software development methodologies', in TW Olle, H Sol & C Tully (eds), *Information System Design Methodologies: A Feature Analysis*, Elsevier B.V., Amsterdam, pp. 37-62.
- Wermelinger, M 1998, 'Towards a chemical model for software architecture reconfiguration', *IEE Proceedings - Software*, vol. 145, no. 5, pp. 130-136.
- White, SA 2004, *Workflow Patterns with BPMN and UML*, IBM Corp., viewed 7 Jul 2009 <<http://www.bpmn.org/Documents/Notations%20and%20Workflow%20Patterns.pdf>>.
- Wienberg, A, Matthes, F & Boger, M 1999, 'Modeling dynamic software components in UML', in, *Proceedings of the 2nd International Conference on Unified Modelling Language (UML'99)*, Springer-Verlag, pp. 204-219.
- Wilcoxon, F 1945, 'Individual comparisons by ranking methods', *Biometrics Bulletin*, vol. 1, no. 6, pp. 80-83.
- Wohlin, C, Runeson, P, Höst, M, Ohlsson, MC, Regnell, B & Wesslén, A 2000, *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers.
- Wold, S, Sjöström, M & Eriksson, L 2001, 'PLS-regression: a basic tool of chemometrics', *Chemometrics and Intelligent Laboratory Systems*, vol. 58, no. 2, pp. 109-130.
- Wood, B, Pethia, R, Gold, LR & Firth, R 1988, *A Guide to the Assessment of Software Development Methods*, Technical Report CMUSEI-88-TR-8, Software Engineering Institute.
- WS-I 2006, *About WS-I*, Web-Service Interoperability Organisation, viewed 11 Jul 2007 <<http://www.ws-i.org/about/>>.
- Wu, M, Etta, P & Zheng, Y 2005, 'Information Systems and Health Care IV: Real-time ROC analysis to evaluate radiologists' performance of interpreting mammography',

- Communications of AIS*, vol. 2005, no. 16, pp. 340-355.
- Xiao, H, Guo, J & Zou, Y 2007, 'Supporting change impact analysis for service oriented business applications', *International Workshop on Systems Development in SOA Environments, 2007 (SDSOA '07)*, ed. J Guo, IEEE Computer Society, p. 6.
- Yan, H, Garlan, D, Schmerl, B, Aldrich, J & Kazman, R 2004, 'DiscoTect: a system for discovering architectures from running systems', *Proceedings of the 26th International Conference on Software Engineering*, IEEE Computer Society, pp. 470-479.
- Yang, J 2003, 'Web service componentization', *Communications of the ACM*, vol. 46, no. 10, pp. 35-40.
- Yang, Z, Cai, S, Zhou, Z & Zhou, N 2005, 'Development and validation of an instrument to measure user perceived service quality of information presenting Web portals', *Information & Management*, vol. 42, no. 4, pp. 575-589.
- Yau, SS, Collofello, JS & MacGregor, TM 1993, 'Ripple effect analysis of software maintenance', in M Shepperd (ed.), *Software Engineering Metrics I: Measures and Validations*, McGraw-Hill Inc., pp. 71-82.
- Yellin, DM & Strom, RE 1997, 'Protocol specifications and component adaptors', *ACM Transactions on Programming Languages and Systems*, vol. 19, no. 2, pp. 292-333.
- Yen, I-L, Ma, H, B. Bastani, F & Mei, H 2008, 'QoS-reconfigurable Web services and compositions for high-assurance systems', *Computer*, vol. 41, no. 8, pp. 48-55.
- Yin, RK 2003, *Case Study Research: Design and Methods*, 3rd edn, SAGE Publications.
- Yu, L, Mishra, A & Ramaswamy, S 2010, 'Component co-evolution and component dependency: speculations and verifications', *IET Software*, vol. 4, no. 4, pp. 252-267.
- Yu, P, Ma, X & Lu, J 2005, 'Dynamic software architecture oriented service composition and evolution', *Proceedings of the 5th International Conference on Computer and Information Technology (CIT'05)*, IEEE Computer Society, pp. 1123-1129.
- Yun, GW & Trumbo, CW 2000, 'Comparative response to a survey executed by post, e-mail, & web form', *Journal of Computer-Mediated Communication*, vol. 6, no. 1, pp. 0-0.
- Zambonelli, F, Jennings, NR & Wooldridge, M 2003, 'Developing multiagent systems: The Gaia methodology', *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 3, pp. 317-370.
- Zelkowitz, MV & Wallace, DR 1998, 'Experimental models for validating technology', *Computer*, vol. 31, no. 5, pp. 23-31.
- Zenger, M 2004, 'Programming Language Abstractions for Extensible Software Components', Ph. D. thesis, EPFL, Lausanne.
- Zhang, H, Urtado, C & Vauttier, S 2009, 'Connector-driven process for the gradual evolution of component-based software', *Proceedings of the 2009 Australian Software Engineering Conference*, IEEE Computer Society, Gold Coast, Australia, pp. 246-255.
- Zhang, J & Cheng, BHC 2006, 'Model-based development of dynamically adaptive software', *Proceedings of the 28th International Conference on Software Engineering*, ACM, Shanghai, China, pp. 371-380.
- Zhang, J, Cheng, BHC, Yang, Z & McKinley, PK 2005, 'Enabling safe dynamic component-based software adaptation', in A Romanovsky, R de Lemos & C Gacek (eds), *Architecting Dependable Systems III*, Springer-Verlag, pp. 194-211.
- Zhang, P, Zhou, Y & Li, B 2007, 'A service-oriented methodology supporting automatic synthesis and verification of component behavior model', *8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, vol. 1, IEEE Computer Society, pp. 511-516.
- Zhao, J, Yang, H, Xiang, L & Xu, B 2002, 'Change impact analysis to support architectural evolution', *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 14, no. 5, pp. 317-333.
- Zhao, W 2009, 'Design and implementation of a Byzantine fault tolerance framework for Web services', *Journal of Systems and Software*, vol. 82, no. 6, pp. 1004-1015.
- Zieba, B & van Sinderen, M 2006, 'Preservation of correctness during system reconfiguration in data distribution service for real-time systems (DDS)', *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW 2006)*, pp. 30-30.
- Zimmermann, M & Drobniak, O 1994, 'Specification and implementation of reconfigurable distributed applications', *Proceedings of the 2nd International Workshop on Configurable Distributed Systems (IWDCS'94)*, pp. 23-34.

Zimmermann, O, Krogdahl, P & Gee, C 2004, *Elements of Service-Oriented Analysis and Design*, IBM Corp., viewed 6 Jun 2008
<<http://www.ibm.com/developerworks/webservices/library/ws-soad1/>>.

Appendix A. SYSTEMATIC LITERATURE REVIEW

This Appendix describes the steps undertaken to perform the systematic literature review to identify feature requirements for dynamic evolution (or “dynamic evolution requirements” for short) from the literature. This research distinguishes between two types of dynamic evolution requirements to handle the complexity of dynamic evolution:

- *Dynamic evolution quality factor requirements* (or “dynamic evolution quality factors” for short), concerned with how well a distributed application and dynamic changes to it are designed to facilitate dynamic evolution; and
- *Dynamic change requirements*, accounting for the characteristics of dynamic changes to a distributed application including various kinds of changes that a distributed application would accommodate; who enacts such changes; and the notion of errors arisen from such changes.

The systematic literature review steps undertaken are described below. They follow the guidelines from Kitchenham et al. (2009) which are based on the original review guidelines proposed by Kitchenham (2004). Any differences in the treatment for the two types of dynamic evolution requirements are noted accordingly:

1. Research question

The objective of the review is to answer the following research question:

What dynamic evolution requirements are considered in the literature and should be addressed during software development for composition-based distributed applications?

2. Search process

The search process was a search of relevant articles from conference proceedings and journals, covering the period 1994 to 2010. The search used their respective journal and proceeding databases rather than search engines since the latter are not designed to support systematic literature review (Brereton et al. 2007). A rationale for selecting particular conference proceedings and journals for the search process was because most of them are known to publish research on maintenance, evolution and distributed systems. Table Appendix A.1 lists the selected journals and conference proceedings.

Table Appendix A.1 Selected Journal and conference proceedings for feature requirement synthesis

<i>Source</i>	<i>Title</i>	<i>Abbreviation</i>
Journals	Communications of the ACM	CACM
	ACM Transactions on Computer Systems	TOCS
	ACM Transactions on Software Engineering and Methodology	TOSEM
	European Journal of Information Systems	EJIS
	IEEE Computer	
	IEEE Software	
	IEEE Transactions on Software Engineering	TSE
	IET Software (formerly "IEE Proceedings Software" and "Software Engineering Journal")	IETS
	Information and Software Technology	IST
	Information Systems Journal	ISJ
	Information Systems Research	ISR
	Journal of Information Technology	JIT
	Journal of Software Maintenance and Evolution: Research and Practice (formerly "Journal of Software Maintenance")	JSME
	Journal of Systems and Software	JSS
	MIS Quarterly	MISQ
	Requirements Engineering	
Conference Proceedings	International Symposium on Component-Based Software Engineering (formerly "ICSE Workshop on Component-Based Software Engineering")	CBSE
	International Conference on Configurable Distributed Systems	IWCDS, ICCDS
	International Conference on Distributed Computing Systems	ICDCS
	International Conference on Service-Oriented Computing	ICSOC
	International Conference on Software Engineering	ICSE
	International Conference on Software Maintenance	ICSM

3. Inclusion and exclusion criteria

The following criteria were used to filter articles from the journals and conference proceedings listed in Step 2:

- An included article must be in English.
- An included article must address software development.
- An included article's title or abstract must have one of the following search terms or variations of it (e.g. "adapt" in verb form vs. "adaptability" in noun form): dynamic, runtime, online, adaptation, change, evolution, maintenance, (re)configuration, update, and upgrade.

- The topic described by an included article must be relevant to dynamic evolution since it is studied under different terms: live, running, runtime, dynamic, online etc.
- The work described by an included article must be sufficiently generic for composition-based distributed applications, including component- or SOA-based types.
- The article must be a “full paper”. Example short articles that are excluded are “poster papers” in conference proceedings.
- Only the most recent version of the same study was included in the review. Duplicate and earlier reports of the same study, even when they appear in different journals/conference proceedings, are excluded;
- Articles on the following topics were excluded:
 - hardware evolution;
 - data and schema evolution (e.g. Roddick 1992); and
 - studies targeting object- and aspect-oriented techniques, since they may be too low level and unsuitable for composition-based distributed applications (Andrade et al. 2002).

4. Quality assessment

To improve the credibility of the dynamic evolution requirements synthesised, the following quality criteria were used to filter the articles selected in Step 3:

- The work described by an included article must be complete rather than in-progress.
- An included article solves at least one problem or aspect relevant to dynamic evolution.
- An included article has been cited.

5. Data collection

The articles remained after Step 4 were examined for “raw” dynamic evolution requirements (raw requirements were synthesised into actual dynamic evolution requirements in the next step). The following selection criteria were used to guide the identification of raw requirements to focus on areas relevant to this research:

- An *included* raw requirement is relevant to dynamic evolution of composition-based distributed applications;

- An *included* raw requirement can be addressed and/or reasoned about during analysis and/or design of such applications;
- An *included* raw requirement should be sufficiently generic to a variety of composition-based distributed applications, including component-based and service-oriented applications which are the main focus of this research;
- An *included* raw requirement is concerned with to a technical aspect (e.g. structural) of dynamic evolution; and
- A raw requirement that falls into the areas generally excluded from this research (Section 1.5) is *excluded*.

For each raw requirement identified, the following data were recorded:

- the original textual description for the requirement;
- bibliographical information (author name(s), article title, journal/conference name etc.); and
- the kind of composition-based distributed application(s) for which the associated article is intended.

6. Data analysis:

After identification in Step 5, the set of “raw” requirements was revised to reconcile the meaning of overlapping requirements, to remove duplicate requirements, to change the descriptions to use standardised terms (e.g. transformation) and to narrow the descriptions to be specific to dynamic evolution. Groupings were formed using appropriate schemes as stated in the research design (cf. Section 3.2.1), in accordance with the types of requirements:

- *For dynamic evolution quality factors*

The revised requirements are called quality attributes. Quality attributes characterising the same quality aspect (e.g. security) were grouped to form the definitions for the factor²³. Quality factors were then categorised into four groups (soundness of change, infusibility of change, changeability of application, and robustness of application) according to their similarities and contexts. The category-factor-attribute hierarchy corresponds to the quality

²³ A quality attribute describes an assessable characteristic possessed by a quality factor. An example assessable characteristic is “no missing component”.

levels (i.e. characteristic, sub-characteristic and attribute) in the quality model of the International Standard ISO/IEC 9126-1 (ISO/IEC 2001).

- *For dynamic change requirements*

The revised requirements are called dynamic change requirements. The dynamic change requirements were categorised along two dimensions: methodology and application. The methodology dimension determines whether a dynamic change requirement is a modelling related (concept, notation and model) or work related (what must be done and how to do it) concern to be addressed by a methodology to support dynamic evolution. The application dimension classifies dynamic change requirements in line with their areas of concern: for individual parts, for the application as a whole, and for all other situations.

7. Derivation (refinement)

A complementary search was performed on the bibliographies of the articles that offer dynamic evolution requirements (after Step 5) for additional articles. This was to account for important and cited articles not covered in the journals and conference proceedings (cf. Table Appendix A.1) and the period searched in Step 2. Afterwards, Steps 3-6 were repeated for the additional articles.

Appendix B. FEATURE ANALYSIS RESULTS OF DEVELOPMENT METHODOLOGIES

This Appendix documents the detailed results of evaluating selected methodologies (cf. Section 2.4) for their extent of support for the dynamic evolution requirements proposed in this research. The evaluation was limited to a qualitative review of the respective documentation of the methodologies. The results are organised into three areas of concern as per the requirement types:

- dynamic evolution quality factors and attributes (Appendix B.1); and
- dynamic change requirements, which are further divided into
 - modelling related dynamic change requirements (Appendix B.2); and
 - work related dynamic change requirements (Appendix B.3)

The results are summarised elsewhere in Table 5.12 for dynamic change requirements and in Table 4.9 for dynamic evolution quality factors. To rate a methodological feature for its extent of support for a requirement, the scaling points in Table Appendix B.1 were developed for and used in the evaluation:

Table Appendix B.1 Scale points for scoring a methodology's feature

<i>Scale Point</i>	<i>Definition</i>
H (high)	A feature appears explicitly in a methodology and fully supports the feature requirement, providing opportunities for <i>reuse</i> .
M (medium)	A feature appears explicitly in a methodology and supports the feature requirement to a limited extent and needs <i>small enhancement</i> .
L (low)	A feature appears explicitly in a methodology, does not adequately address many aspects of the feature requirement and needs <i>significant enhancement</i> . This may take as much effort as new development.
[blank]	A methodology fails to recognise any support for the requirement. This identifies an area in the methodology for <i>new development</i> .

B.1 EVALUATION RESULTS OF SUPPORT FOR DYNAMIC EVOLUTION QUALITY FACTORS

This section documents the detailed results of evaluating selected methodologies (see Section 2.4) for their extent of support for the dynamic evolution quality factors proposed in this research (see Section 4.4 for summary). Each subsection reports the evaluation results for a particular dynamic evolution quality factor. Each subsection is divided by lines of boxed text, each of which representing a quality attribute of the respective quality factor. The text below each quality attribute (i.e. boxed text) documents the evaluation results for the attribute using the following format. Each methodology that offers support to a quality attribute is given an evaluation score and a brief description of the support. The evaluation score indicates the extent of support as

per Table Appendix B.1 (i.e. “**H**” for high, “**M**” for medium, and “**L**” for low). An evaluation score appearing in bold font indicates that the corresponding feature of a methodology is a potential source for reuse or enhancement (i.e. rated as “**H**” or “**M**”). The brief description part records the name of a methodology and briefly describes its feature that offers the support.

B.1.1 Soundness of Change

B.1.1.1 Completeness

no missing functionality after a transformation

H: RUP’s “review the design” task evaluates a design model as a whole to check if it fulfils its requirements. In each iteration of a development lifecycle, it verifies that there is no missing behaviour by checking to see that all scenarios (i.e. use cases) specified have been completely realised by respective designs.

L: OPF offers “inspections” which is a generic technique to evaluate work products to identify defects and issues.

no missing parts after a transformation

no missing, illegal or broken bindings after a transformation

L: RUP highlights the importance of configuration management in ensuring the overall completeness of an application. RUP also has several tasks (e.g. “perform configuration audit”) under the “configuration and change management” process to manage changes.

L: EPIC’s checklist for a “component dossier document” includes items for identifying what a component offers, what capabilities are expected from the component to build the system, and how the missing capabilities can be fulfilled.

(Also in <i>Consistency</i>) assumptions and properties of a distributed application and its parts met by a change

See Consistency in Appendix B.1.1.2 for the quality attribute above.

B.1.1.2 Consistency

all parts involved in a runtime change identified before a transformation

L: RUP highlights the importance of configuration management in supporting the consistency of an application.

adequate resources and support for new and changed parts
--

No direct support is identified for the quality attribute listed above.

system invariants preserved from a transformation

H: In SeCSE's "regression testing" technique, possible invariants are automatically generated using tools. Test cases are then developed to verify if the generated invariants are violated. This technique also uses known invariants, which are captured as part of specifications of services, to generate additional test cases and check the design against the invariants.

(Also in *Completeness*) assumptions and properties of a distributed application and its parts met by a change

L: Kobra defines six consistency rules (Atkinson et al. 2002, pp94 & 334) to ensure that artefacts in a product line are mutually consistent with one another as they are subject to constant demand for changes:

- *intra-diagram* rule: well-formed individual diagrams;
- *inter-diagram* rule: diagrams at the same abstraction level consistent with one another;
- *realisation* rule: component's realisation correctly representing its specification;
- *clientship* rule: clients and servers fulfilling their contract;
- *containment* rule: component relationships at runtime consistent with their relationships at development time; and
- *specialisation* rule: specialised components (e.g. application's) conforming to the component from which it is specialised (e.g. framework's).

L: SeCSE defines several online testing approaches to detect SLA (service-level-agreement) and QoS (quality-of-service) violations.

L: RUP has several tasks (e.g. "perform configuration audit", "develop deployment plan") under the "configuration and change management" process to verify changes.

compatible connections, meaning for component-based system: types and directions of connected ports matching, and for service-oriented system: compatibility between service consumers and providers

compatible communications protocols among parts

no progression towards an error state after a transformation

synchronisation of application's and parts' states after a transformation, and specific to SOA: synchronisation of state information of messages and new services after a

transformation

a reachable state attained after a transformation

no critical procedures executed before a transformation

no pending messages, interactions or transactions before a transformation

No direct support is identified for the quality attributes listed above.

B.1.1.3 Correctness

non-arbitrary and admissible changes

L: AEM checks modifications in the “maintain consistency and system integrity” task.

L: RUP highlights the importance of configuration management in supporting correctness of an application. In its “configuration and change management” process, several tasks (e.g. “confirm duplicate or rejected change requests”) are specified to manage changes. However, these tasks are oriented towards change requests from the business and are not explicitly related to dynamic aspects of changes.

L: OPF prescribes the task “configuration control” to manage changes on work products under configuration management. One of its responsibilities is to evaluate each change request, and then either approves, rejects, or postpones it.

correct ordering of transformations

no unintentional behaviour during and after a transformation

transformations at a right time

No direct support is identified for the quality attributes listed above.

B.1.2 Infusibility of Change

B.1.2.1 Efficiency

easily executed transformations

quickly executed transformations

resource efficient transformations

Minimal disruptions to application functions and their users during a transformation

Minimal degradation to application performance during and after a transformation

No direct support is identified for the quality attributes listed above.

B.1.2.2 Locality

application partitioning and change localisation to partitions

M: To confine the propagation of changes, Catalysis suggests using packages to partition model elements into separate areas, with explicit dependencies between them (D'Souza & Wills 1998, pp298-9). Catalysis also lists a number of ways to partition a system into packages, such as vertical slicing (from user perspectives), horizontal slicing (i.e. business vs. technical), and domain driven partitioning (e.g. user interface vs. persistence).

L: OPF uses both layering and partitioning to organise the grouping of functionality provided by various parts of an architecture.

L: In Select Perspective, business components (i.e. the business layer) are separated from the database schema for data management (i.e. the resource layer), to avoid major ripples between them. That is, changes that occur in the business components are expected to have little or no effect on the database structure.

B.1.2.3 Maintainability

no degradation in cost and ease of modifications

M: In Select Perspective, easily maintained components and applications mean:

- interfaces are properly designed, documented and tested;
- tools and templates are available to streamline implementation; and
- component specification and implementation information is published and properly catalogued to facilitate search and retrieval for reuse.

M: In EPIC, an “evolvable” (i.e. easy to accommodate changes) architecture exhibits these characteristics:

- layered to group components;
- highly modular with optimally scoped and sized components;
- well-defined and standard based component interfaces; and
- common mechanisms (e.g. means of component communication).

M: OPF suggests a number of mechanisms to implement maintainability:

- layered architectures;
- modular software;
- information hiding of implementation;

- well-defined interfaces;
- object-orientation and component-based development;
- complete and current documentation; and
- adherence to project conventions.

all parts clearly defined in interaction (or workflow) specifications

No direct support is identified for the quality attribute listed above.

no reduction in testability

No direct support is identified for the quality attribute listed above. RUP, however, offers the “verify changes in build” task in the “configuration and change management” to deal with testing and verification of changes. But it does not address testability of an application. Testability is also noted as a systemic quality but neither its definition nor how it can be improved is provided.

clear and detailed interactions

No direct support is identified for the quality attribute listed above.

B.1.2.4 Transparency

transformation design and implementation hidden from application programmers

transformation agents hidden from operating environment

No direct support is identified for the quality attributes listed above.

transformations hidden from end users

L: AEM constrains the design of an architecture to follow the C2-style topology (Medvidovic et al. 1999) to hide runtime changes in one layer from components in the layer below it.

RUP acknowledges component “run-time replaceability” as a way to support upgrade with “no loss of availability” but does not describe how it can be supported.

transformations hidden from parts unaffected by the transformations

No direct support is identified for the quality attribute listed above.

B.1.3 Flexibility of Application

B.1.3.1 Autonomy

self-control and self-governance of parts' own processing

H: ERL provides the “service autonomy” principle, and distinguishes between “service-level autonomy” and “pure autonomy”.

L: P&H discusses “distributed governance” for business units.

B.1.3.2 Configurability

distributed application configurable to different policies for dynamic changes and transformations

No direct support is identified for the quality attribute listed above.

B.1.3.3 Coordination

transformations coordinated among multiple nodes/organisations

transformation agents tolerant of network unreliability during a transformation

No direct support is identified for the quality attributes listed above.

B.1.3.4 Extensibility

runtime extension/upgrade of an application with new functionality

L: ERL contends that operations and messages should be designed in a way that are business activity-agnostic such that functionality extensions would minimise and avoid changes to existing interfaces (Erl 2005, p. 558).

runtime extension/upgrade of parts in an application with new functionality

No direct support is identified for the quality attribute listed above. Nevertheless, ERL acknowledges “extensibility” but does not properly deal with it.

runtime extension/upgrade of an application with new parts

L: ERL notes that when handling new requirements, composing existing services without modifying their interfaces should be considered first.

B.1.3.5 Flexibility

any part of a distributed application to be changeable at runtime

L: AEM advocates the notion of a “C2-style” architecture in which components are distributed in layers. Components in one layer only use and communicate with those in the layer above it. Thus, components in one layer are oblivious to runtime changes to those in the layer below it.

distributed application accommodating a variety of runtime changes
--

No direct support is identified for the quality attribute listed above.

B.1.3.6 Loose Coupling

high level of independence between parts
--

H: Catalysis offers the “decoupling process” pattern for objects and classes.

M: ERL offers the “service loose coupling” principle to design a loosely coupled architecture. Its “service contract” principle reduces the processing logic from using different services as this would lead to tight dependencies (i.e. tight coupling).

L: Kobra notes the use of layers to improve low coupling between a product-line framework and applications using the framework.

L: OPF notes the use of tiers and coupling for application design. It also offers the “collaboration analysis” technique to reduce system coupling.

H: P&H offers the “service coupling” principle as a guideline for SOA-based applications.

M: RUP offers the “solution partitioning” and “package coupling” principles and the “design subsystem” guidelines to address coupling and cohesion.

parts having their own lifecycles and runtime environments
--

No direct support is identified for the quality attributes listed above.

B.1.3.7 Separation of Concerns

separating dynamic change concerns from functionality concerns
--

separating communication concerns from functionality concerns

separating security support from functionality
--

No direct support is identified for the quality attributes listed above.

separating realisations of parts (e.g. service providers) from those of their clients (e.g. service consumers)
--

L: CBDI-SAE notes it is important for component consumers to be separated from component providers.

H: Select Perspective uses two development processes to keep development activities of component providers independent from those of component consumers, thereby decoupling their realisation dependencies.

separating part specification from realisation concerns

H: ASG separates service specification in a service discovery database from the service providers offering the service.

H: Catalysis maintains component specification from implementation in its techniques.

L: CBDI-SAE notes it is important for service realisation to be separated from implementation as a key objective.

H: ERL offers the “service abstraction” principle. Its purpose is to hide the underlying details of the service so that only the service contract is available and of concern to service requestors.

H: Kobra clearly distinguishes among component specification, realisation and implementation throughout its lifecycle of activities.

H: P&H offers the “process realisation analysis” activity which utilises the separation of specification from implementation to implement Web services in different ways.

H: RUP uses different activities (“identify design elements”, “identify services” etc.) to separately address identification, design and realisation of services.

H: SeCSE offers tasks to define abstract service composition, and others to design alternative concrete services.

H: Select Perspective defines four levels of component abstractions - specification, implementation, executable and deployment - and offers various tasks to support their development separately.

B.1.4 Robustness of Application

B.1.4.1 Fault tolerance

high tolerance of faulty new and/or changed parts

L: OPF catalogues “exception handling” as a basic technique to trap and handle errors at the code level during the execution of an application.

L: In RUP, designing with “redundancy” is a mechanism to handle fault tolerance, say, to alleviate reliance on faulty servers. To handle a faulty server (but not the parts running on it), parts on the server are designed to be “location transparent”; they can be relocated to a server on another node to continue to operate. In addition, RUP suggests diagnostics on a running application be available to monitor its faults/errors.

M: SeCSE employs several processes together addressing faulty services (SeCSE 2007, pp. 28-31, 38-39). The “Service monitoring” process, which is complemented with tool support, observes and checks at runtime that each service conforms to its expected functional and non-functional behaviour. If a service is found to deviate from its expected behaviour, the “recovery management” process is notified to recover the associated application from the problematic behaviour. It firstly uses the “binding and re-binding” process which attempts to look for alternative services offering comparative functionality, and select the most appropriate ones. If no alternative is found, recovery management instructs the “runtime service composition management and re-planning” process to generate an alternative service, composed of other existing services, to substitute the problematic service.

barriers established to contain potentially faulty new and replacement parts

L: OPF offers “exception handling” as a basic technique to trap and handle errors at the code level during the execution of an application.

B.1.4.2 Recoverability

Restoration of an application and its parts to a state to continue to perform their functionality, after a failure caused by a transformation and/or its change(s)

L: OPF defines a set of disaster recovery activities to deal with major disasters, whether natural or man-made.

L: SeCSE defines the “recovery management” process to recover a system from problematic behaviour but it is not specific to dynamic evolution (SeCSE 2007, pp. 29-30).

B.1.4.3 Reliability

no compromise on intended functionality after a transformation

L: In EPIC, reliability of components must be benchmarked and evaluated against the overall reliability requirements for the overall system.

L: RUP acknowledges the need for evaluating reliability during the “review the architecture” activity, but lacks support for improving reliability.

replacement parts fully satisfying their roles

L: In EPIC, reliability of components must be benchmarked and evaluated against the overall reliability requirements for the overall system.

L: RUP acknowledges the need for evaluating reliability during the “review the architecture” activity, but lacks support for evaluating and improving reliability.

H: SeCSE’s “regression testing” technique (SeCSE 2008) aims at ensuring that a replacement service is accessible, performs its operations, and behaves as expected after it replaces the obsolete original service. This technique checks both the functionality and QoS aspects of the replacement service. “QoS testing” and “agreement testing” are related techniques for measuring the boundaries of the service-level-agreements provided by services, but they do not adequately address the dynamic evolution requirement above.

B.1.4.4 Safety

distributed application and its parts operating safely during and after a transformation

L: AEM suggests architectural changes are analysed during its “enact changes” task to check if they render an application to become unsafe.

M: In OPF, there is a full discipline dedicated to safety: “Work Product Safety Engineering”. This discipline is to ensure work products - systems, applications and components - do not exceed accepted levels of safety risks. Safety control begins with the “safety program planning” activity to define an overall safety programme plan for a particular application. Then, during the “safety risk analysis” activity acceptable safety risks are identified and safeguards are determined. Next, the “safety monitoring” activity is executed to monitor and report the status of the safety programme. If safety incidents are detected, the “incident investigation” activity is conducted to determine the causes of safety incidents and appropriate steps required to prevent their recurrence. The “safety compliance assessment” activity is used to analyse if an application conforms to a safety programme, standards and/or other safety requirements. Optionally, the application can be certified for compliance in the “safety certification” activity.

B.1.4.5 Security

transformation agents secured from unauthorised access
--

no security compromise after a transformation

access to new and replacement parts restricted after a transformation

L: EPIC's checklist for a "component dossier document" includes items for checking if components meet the security requirements for the overall system.

M: OPF defines a process called "Security Engineering" to address security in an endeavour. It is structured as a generic framework to tackle security but it is not specific to a particular problem domain. Its tasks are performed to ensure security needs of the work products are met. In "security risk assessment", security risks on work products are assessed. The assessment results are used in the "security policy production" activity to create approved security policies and privacy statements. When the policies are ready, the "security enforcement" activity puts in place the mechanisms to enforce them. Then, activities for "preventing" and "detering" security threats from occurring can be activated. Where appropriate, the "security auditing" and "security threat detection and analysis" activities are conducted on business units, data centres etc. for security assessment and impact analysis from threats.

L: In RUP, overall security of an architecture is evaluated in the "review the architecture" activity but it lacks guidelines for doing such an evaluation.

dynamically updated security policy

separating security policy from security enforcement
--

No direct support is identified for the quality attributes listed above.

B.2 EVALUATION RESULTS OF SUPPORT FOR MODELLING RELATED DYNAMIC CHANGE REQUIREMENTS

Table Appendix B.2 presents the detailed results of evaluating selected methodologies (see Section 2.4) for their extent of support for the dynamic change requirements concerning the modelling aspect of a methodology (see Section 5.4 for summary). The evaluation took into account the following variations:

1. Catalysis, Select Perspective, Kobra and RUP adopt UML as their modelling approach with or without extensions. Thus, their evaluations thus included the evaluation of UML.

2. OPF is designed to integrate with UML and Open Modelling Language (OML, Firesmith et al. 1997) as its modelling approach. In the evaluation, OPF's modelling capabilities were assessed in conjunction with UML and OML.
3. P&H and SUPER adopt BPMN (OMG 2009) as their modelling approach with or without extensions. Thus, they were evaluated together with BPMN.

The first letter of each evaluation score represents a scale point ("H" for high, "M" for medium, "L" for low). See Table Appendix B.1 for its definitions.

Table Appendix B.2 Evaluation of support for modelling related change requirements (see Sections 5.1.4 and 5.2.3 for definitions)

	<i>AEM</i>	<i>ASG</i>	<i>Catalysis</i>	<i>CBDI-SAE</i>	<i>EPIC</i>	<i>Erl</i>	<i>Kobra</i>	<i>OPF</i>	<i>P&H</i>	<i>RUP</i>	<i>SeCSE</i>	<i>Select Perspective</i>	<i>SUPER</i>
Service Level													
multiple version coexistence							—	—	L: multiple service versions			—	L: multiple versions of business process model
resource needs			L: evident only in an example		L: part of performance L: resource management noted for maintenance		—	—	L: resource reallocation for SLA compliance			—	
performance characteristics		H: performance metrics model	L: captured as part of use cases		H: speed, efficiency, availability, accuracy, throughput, response time, recovery time, resource usage [also reiterated in RUP] L: some performance attributes as part of technical constraints			H: capacity, latency, throughput, response time		H: speed, efficiency, availability, accuracy, throughput, response time, recovery time, resource usage [also reiterated in EPIC]			L: performance data noted
access blocking													
Application Level													
dynamic change	L: "change descriptions" noted		L: one change per package release				L: support for change at design time only				—	L: runtime change limited to replacement of failed services via "re-planning"	

© 2011 UNSW page 264

	<i>AEM</i>	<i>ASG</i>	<i>Catalysis</i>	<i>CBDI-SAE</i>	<i>EPIC</i>	<i>Erl</i>	<i>KobraA</i>	<i>OPF</i>	<i>P&H</i>	<i>RUP</i>	<i>SeCSE</i>	<i>Select Perspective</i>	<i>SUPER</i>
<i>transformation compensation</i>		L: automated re-planning for failed composition	—		—		—	—	—	—		—	
<i>expected dynamic change impact</i>			L: impact of change treated as change from one level of abstraction to the highest level		M: target resistance plan			L: noted in change request form					

Notes:

1. UML offers elementary action objects “CreateLinkAction”, “CreateLinkObjectAction”, “DestroyLinkAction” and “DestroyObjectAction” to model change operations on an object model. The rating for UML with respect to “transformation action” is given medium (i.e. “M”).

B.3 EVALUATION RESULTS OF SUPPORT FOR WORK RELATED DYNAMIC CHANGE REQUIREMENTS

Table Appendix B.3 presents the detailed results of evaluating selected methodologies (see Section 2.4) for their extent of support for the dynamic change requirements concerning the kinds of work to be performed during development (see Section 5.4 for summary). The first letter of each evaluation score represents a scale point (cf. Table Appendix B.1).

Table Appendix B.3 Evaluation of support for work related change requirements (see Sections 5.1.4 and 5.2.3 for definitions)

	<i>AEM</i>	<i>ASG</i>	<i>Catalysis</i>	<i>CBDI-SAE</i>	<i>EPIC</i>	<i>Erl</i>	<i>KobraA</i>	<i>OPF</i>	<i>P&H</i>	<i>RUP</i>	<i>SeCSE</i>	<i>Select Perspective</i>	<i>SUPER</i>
Service Level													
<i>dynamic part change</i>		L: automated “re-planning” to replace faulty composition									L: “runtime service composition management and re-planning” process to replace a faulty service		

© 2011 UNSW page 266

<i>dynamic contract update</i>	<i>dynamic change impact analysis</i>	<i>dynamic variability</i>	<i>dynamic refactoring</i>	<i>dynamic recomposition</i>	
					<i>AEM</i>
L: "re-negotiate contracts" for exception handling			L: automated "re-planning" for faulty composition	L: automated "re-planning" for faulty composition	<i>ASG</i>
	L: change impact noted		L: design time refactoring	L: runtime composition briefed in upgradability	<i>Catalysis</i>
					<i>CBDI-SAE</i>
	L: scope of change impact assessment noted throughout the methodology	[see RUP]			<i>EPIC</i>
					<i>Erl</i>
		M: support for static variability only	L: "tree refactoring" (design time)		<i>Kobra</i>
	L: change impact noted in change request form		M: "design refactoring"		<i>OPF</i>
		L: "design for service reusability"	L: design time business process refinement noted		<i>P&H</i>
		L: "asset based development"	L: code level refactoring technique with eXtreme Programming™		<i>RUP</i>
		H: "variation point management" and "variation points realisation" processes	L: "runtime service composition management and re-planning" process for faulty services	L: "runtime service composition management and re-planning" process for faulty services	<i>SeCSE</i>
		L: "commonality" noted	L: parameter, message, code refactoring		<i>Select Perspective</i>
					<i>SUPER</i>

Appendix C. DETAILED SPECIFICATIONS FOR CONTINUUM

This Appendix documents Continuum method fragments which are specified using the International Standard ISO/IEC 24744 for Software Engineering Metamodel for Development Methodologies (SEMDM) (ISO/IEC 2007). (Strictly speaking, “method fragments” should be termed “method fragment kinds”. For convenience and ease of reading, the “kind” suffix has been dropped.) This standard adoption aims to help vendors to implement and import these fragments in an existing method repository, and practitioners to consistently learn and apply these fragments. An introduction to Continuum can be found in Section 6.3.

Continuum’s method fragments are described under three headings:

- *Producer Method Fragments* (Appendix C.1) describes agents (people, roles, tools etc.) responsible for executing work units.
- *Work Product Fragments* (Appendix C.2) describes artefacts used and/or produced in development, plus the modelling concepts and notations applied in the artefacts.
- *Work Unit Fragments* (Appendix C.3) describes what must be done for given purposes and how they are achieved in development.

C.1 PRODUCER METHOD FRAGMENTS

Producer method fragments concern producers who are agents having the responsibility of executing work units according to their areas of expertise (ISO/IEC 2007). Continuum prescribes two specific types of producer method fragments: role and tool (ISO/IEC 2007). A role is a collection of responsibilities that a producer can play to fulfil objectives of certain work units. A tool assists a producer in executing the producer’s responsibilities in an automated way.

C.1.1 Dynamic Evolution Analyst

This role is mainly involved in executing the work units relating to the analysis aspects of dynamic evolution. Its responsibilities include:

- identification and analysis of dynamic evolution requirements in terms of dynamic changes to an application;
- planning the roll out of dynamic changes through an application lifecycle;

- analysis of dynamic evolution quality aspects important to an application; and
- assessment and improvement of dynamic changes and the application lifecycle with respect to dynamic evolution quality.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
name	String	The name of a DynamicEvolutionAnalyst.

Associations: See the description for the "agents" field of a task fragment specification in the Appendix C.3.1.

C.1.2 Dynamic Evolution Designer

This role is primarily involved in executing the work units relating to the design aspects of dynamic evolution. Its responsibilities include:

- development of design-related dynamic evolution work products;
- selection and use of relevant design patterns for the development of the above; and
- assessment and improvement of quality aspects of design-related dynamic evolution work products.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
name	String	The name of a DynamicEvolutionDesigner.

Associations: See the description for the "agents" field of a task fragment specification in the Appendix C.3.1.

C.1.3 Runtime Application Discovery Tool

This tool is used for providing a secure means of discovering information about an application whilst it is running. Its responsibilities include:

- inspecting the runtime structure of the application for discovering its transformable items and their binding relationships;
- identification of the zones in which the transformable items are hosted;
- identification of the version information of each transformable item; and
- determination of the resources used by the transformable items.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
name	String	The name of a RuntimeApplicationDiscoveryTool.

Associations: See the description for the "agents" field of a task fragment specification in the Appendix C.3.1.

C.2 WORK PRODUCT FRAGMENTS

Work product fragments concern model artefacts - software and hardware, models, documents etc. - used, created and/or modified in development, plus the modelling concepts and notations applied in the artefacts (ISO/IEC 2007).

Continuum prescribes two types of product fragments:

- *Model Unit fragments*, which depict atomic elements, concepts and constructs as low level semantic building blocks for creating models of interest relevant to dynamic evolution; and
- *Diagram, Document and Notation fragments*, which depict these models visually and/or textually.

Work product fragments of each type are described next in alphabetical order.

C.2.1 Model Unit Fragments

Model Unit fragments are instances of SEMDM ModelUnit/*Kind, and specified with attributes and associations, with the same semantics as those for object-oriented classes. Attributes hold data values about an instance of a model unit fragment. The data values can be of standard data types (e.g. integer and string) or custom enumerated data types. An association defines a relationship of one model unit fragment with another. A visual representation of the model unit fragments and their associations are depicted in Figure 6.5 as part of the introduction to Continuum.

C.2.1.1 Application

An Application is a set of inter-related TransformableItems to offer some functionality and computing capabilities to the needs of a business (Dearle 2007).

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
name	String	The name of an Application.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
member	TransformableItem	TransformableItems that together provide the functionality of an Application as a whole.
lifecycle	ApplicationLifecycle	The various Stages through which an Application passes.

C.2.1.2 ApplicationLifecycle

An ApplicationLifecycle, a.k.a. a software lifecycle, is defined as “the period of time that begins when a software product is conceived and ends when the software is no longer available for use” (IEEE 2009). In this context, Within an ApplicationLifecycle, an Application progresses through successive Stages as it evolves dynamically.

Attributes: none

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
context	Application	The Application exhibiting the lifecycle.
stage	Stage	Successive Stages in a lifecycle that an Application progresses.

C.2.1.3 ChangeCase

A ChangeCase articulates a proposed dynamic change to properties and functionality of an Application’s TransformableItems, contracts between TransformableItems, to its structure, and to its Zones so as to facilitate further understanding of changes required for the Application at runtime²⁴. The change specified by a ChangeCase is confined to a particular element of an application as the recipient of the change. A ChangeCase is slightly different from a user or business requirement or a change request, which is specified in terms understood by end users, stakeholders and the business, and independent of the structure and configuration of an application. For instance, a change request may propose a change to an application as a whole, “Add a new function to application Z”, which can be translated to a change case, “Add a new function to component X in application Z”. This change case might then be mapped to a transformation called “component X upgrade” which replaces the existing component X with a new version that has the new function.

A ChangeCase is specified relative to the as-is (i.e. current) Generation of an application. ChangeCases effectively articulate goals for TransformationAgents (i.e. the actors for the change cases) to achieve during a TransitionalPeriod.

²⁴ The original definition of ChangeCase articulates changes specific to an Application’s functionality (Ecklund et al. 1996; Office of Government Commerce 2002).

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
id	String	A unique identifier of the ChangeCase.
purpose	String	The purpose of a ChangeCase in a sentence, including a gap operator, and a recipient of the change if required.
description	String	A long description of a ChangeCase.
relatedRequirement	String	Requirements from which a ChangeCase is derived.
expectedImpact	Impact	The expected impact of a ChangeCase has on the recipient of the change.
resolvedImpact	Impact	The impact that a ChangeCase intends to resolve because of the impact caused by another ChangeCase.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
actor	TransformationAgent	The entity responsible for realising the ChangeCase.
context	TransitionalPeriod	The period during and after which a ChangeCase will be realised in an Application.
target	Generation	A particular “as-is” Generation of an Application for which a dynamic change is specified using a ChangeCase.
enactment	Transformation	The transformation which realises the change case.

C.2.1.4 Generation

A Generation of an Application represents a stable state of the Application. That means the Application structure is operating a particular version of its code and not involving in any runtime modification.

Generation is a subclass of Stage.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
name	String	The name of a Generation.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
change	ChangeCase	A change requirement for this Generation.
next	TransitionalPeriod	The TransitionalPeriod following this Generation.
prev	TransitionalPeriod	The TransitionalPeriod preceding this Generation.

C.2.1.5 Impact

An impact identifies a target (e.g. a TransformableItem, a Zone) thought to be affected both directly and/or indirectly by a proposed ChangeCase, and the extent to which the impact has on the target.

The notions of “Target”, “Impact Type” and “Level of Disruption” for an Impact are reused

from “target resistance plans” in the EPIC methodology (Albert & Brownsword 2002). The enumerated values for the latter two attributes are customised for dynamic evolution (cf. Table Appendix C.1 and Table Appendix C.2).

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
impactType	ImpactType	The type of the impact on the “target”.
disruptionLevel	ImpactDisruptionLevel	The level of disruption the impact has on the “target”.
target	String	An entity (TransformableItem, Zone etc.) thought to be affected by the ChangeCase.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
proposedChange	ChangeCase	The change case that leads to the Impact.
supplimentaryChange	ChangeCase	A supplementary ChangeCase (or more) that must also be realised in order to resolve the impact.

C.2.1.5.1 Custom attribute type ImpactType

ImpactType enumerates the possible kinds of impacts that a ChangeCase has on a target thought to be affected by the ChangeCase. The default value for ImpactType is “direct”.

Table Appendix C.1 Enumerated values of ImpactType

<i>Name</i>	<i>Semantics</i>
direct (default)	A change case describes a change that directly modifies or applies to a target identified by the Impact.
indirect	A change case describes a change that modifies or applies to a target. Via a ripple effect, the change causes a further and indirect change to another target identified by the Impact.

C.2.1.5.2 Custom attribute type ImpactDisruptionLevel

ImpactDisruptionLevel enumerates the possible severity levels of disruptions that an impact has on a target thought to be affected by a ChangeCase.

Table Appendix C.2 Enumerated values of ImpactDisruptionLevel

<i>Name</i>	<i>Semantics</i>
high	The target will no longer function.
medium	The target will continue to offer a subset of its functions or services not affected by the change.
low	The target will continue to offer all its functions or services.

C.2.1.6 OperationalProfile

An OperationalProfile is an abstract description about distinctive operational characteristics of the TransformableItem, such as resource needs and performance expectations, during its normal operation.

Attributes: none

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
target	TransformableItem	The TransformableItem to which this OperationalProfile applies.

C.2.1.7 Policy

A Policy is an abstract concept for articulating rules or behaviour desirable for dynamic evolution in a composition-based distributed application. The Policy class is intended to be extended (using the class inheritance mechanism) to model various facets of dynamic evolution. Through class extension, Continuum can identify and classify various types of policies suitable for dynamic evolution. On the other hand, Continuum does not offer any formalism or language for expressing policies to allow methodology users to choose and adopt various ways of expressing policies as appropriate.

Attributes: none

Associations: none

C.2.1.8 PerformanceProfile

A PerformanceProfile specifies observable performance expectations for a TransformableItem when it carries out a particular function. For example, the minimum number of user requests a web server should handle at a time is a performance characteristic of the web server. Continuum reuses Adaptive Service Grid's performance metrics model (Kempton et al. 2007). Metrics are incorporated into a PerformanceProfile as attributes. This is beyond the scope of Continuum to fully describe the metrics model here.

PerformanceProfile is a subclass of OperationalProfile.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
responseTime	String	The duration from the start of a request is initiated (e.g. click of a search button) to an observable response of an application or a TransformableItem is received.
serviceTime	String	The portion of the responseTime that a request is actually serviced by an application or a TransformableItem.
networkDelay	String	The portion of the responseTime that is attributed to the latency caused by the network.
residenceTime	String	The portion of the responseTime that a request is spent in a queue waiting to be processed.
throughput	String	The number of requests processed per time unit (e.g., bits/sec, hits/sec, applied use cases/sec).

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
workload	String	The number of requests (e.g. click of a search button) per time unit an application or a TransformableItem has to handle.
errorRate	String	The number of times an application or a TransformableItem responds with errors over the number of requests handled by it.

Associations: none

C.2.1.9 Resource

A Resource is a finite computing artefact that a TransformableItem requires in order to function (Dearle 2007). Resources are offered by and managed within Zones.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
name	String	The name of a Resource.
description	String	A description of a Resource.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
command	TransformationAction	The command causing adjustments to the allocation of the Resource.
consumer	TransformableItem	The TransformableItem that will use the Resource.
context	Zone	The environment that manages the Resource.
profile	ResourceProfile	The usage profile for the Resource.

C.2.1.10 ResourceProfile

A ResourceProfile specifies the resources used or consumed by a TransformableItem in order to function.

ResourceProfile is a subclass of OperationalProfile.

Attributes: none

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
resource	Resource	The Resource for which the ResourceProfile is targeted.

C.2.1.11 ServicingPolicy

A ServicingPolicy specifies the condition under which a construct, be it a TransformableItem or its method(s), is offering its services or functions, such as during a TransitionalPeriod.

Exclusion is a subclass of Policy.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
type	ServicingPolicyType	The type of a ServicingPolicy characterising the access to a TransformableItem.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
target	TransformableItem	The TransformableItem(s) to which the ServicingPolicy applies.

C.2.1.11.1 Custom attribute type ServicingPolicyType

ServicingPolicyType enumerates the possible kinds of policy for a TransformableItem in offering its services, and especially useful during a TransitionalPeriod. The default value for ServicingPolicyType is “void”.

Table Appendix C.3 Enumerated values of ServicingPolicyType

<i>Name</i>	<i>Semantics</i>
Illegal	Use of a service is banned and illegal.
Delegated	Use of a service is not available but another TransformableItem (i.e. the delegate) is offering an equivalent service.
Blocked and Queued	Requests to use a service are blocked and queued on a first-come first-served order. Note that the delay caused by a service being “blocked and queued” may be sufficiently long such that the service might be regarded as unavailable. If this apparent “downtime” is unacceptable, a design should investigate the “delegated” ServicingPolicyType option which means requests to the service will be forwarded to and handled by an alternative TransformableItem.
Void (default)	Use of a service is permitted as normal.

C.2.1.12 Stage

A Stage is an abstract designated period during the lifespan of an Application.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
startTime	Timestamp	The time instant at which a Stage begins.
endTime	Timestamp	The time instant at which a Stage ends.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
lifecycle	ApplicationLifecycle	The lifecycle in which a particular Stage exhibits.

C.2.1.13 TransformableItem

A TransformableItem is an abstract concept for a runtime logical entity that can be transformed into some other form (e.g. a newer version) by some means. It epitomises a building block in a composition-based distributed application: a part of some functionality, a binding for connecting parts and mediating their interactions, a composite of smaller parts, or a workflow of parts thereof. TransformableItems are

distributed in a network which facilitates connectivity among them. Example TransformableItems include components and connectors in component-based applications (Evans & Dickman 1999) and services and bindings in service-oriented applications (Papazoglou & Georgakopoulos 2003).

Each TransformableItem is assigned a unique instance identifier, to distinguish itself from others running the same code at runtime.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
instanceId	String	A unique identifier to identify a particular TransformableItem in an Application.
state	TransformableItem StateType	A condition of a TransformableItem indicating when it can join and leave a Transformation.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
applicationContext	Application	The Application to which a TransformableItem belongs.
command	TransformationAction	A unit of modification to be applied to a TransformableItem.
host	Zone	An environment in which a TransformableItem is deployed and run.
inputContext	Transformation	The Transformation in which a TransformableItem will be changed.
outputContext	Transformation	The Transformation in which a TransformableItem was changed.
profile	OperationalProfile	A profile describing an operational characteristic of a TransformableItem.
policy	ServicingPolicy	The policy defining rules for a TransformableItem to offer its services during a Transformation.
resource	Resource	A resource used by a TransformableItem.

C.2.1.13.1 Custom attribute type TransformableItemStateType

This enumeration type defines the possible and externally observable states of a TransformableItem from a configuration or transformation perspective. These states can be used to specify State Maps (Appendices C.3.2.12 and C.2.2.7). The enumerated values are intended to annotate additional semantics to existing states of a TransformableItem under normal operations.

Table Appendix C.4 Enumerated values of TransformableItemStateType

<i>Name</i>	<i>Semantics</i>
Quiescent	A state of an existing TransformableItem to indicate it is ready to participate in a Transformation (Kramer & Magee 1990). For example, a TransformableItem is in a quiescent state when it is not performing any transaction (e.g. updating a database record). A quiescent state is, however, a necessary but not sufficient condition for a transformation to start.

<i>Name</i>	<i>Semantics</i>
Resuming	A state of a new TransformableItem from which the TransformableItem operates after a Transformation completes. A resuming state may very well be the initial state (i.e. the state when a TransformableItem is firstly instantiated) but this is not always the case. When a database façade is added to an application, it is desirable to configure the façade to the 'database available' state, ready for access, rather than the 'database shutdown' state, which is the initial state.
Resolved	A state of an existing TransformableItem from which the TransformableItem continues to operate after a TransformationException has been detected and resolved.

C.2.1.14 Transformation

A Transformation is a mapping from some existing entities, such as TransformableItems, as the input to entities in another form as the output. A Transformation can be executed without requiring an Application to be shutdown. A Transformation is modelled as a set of atomic steps of runtime modifications called TransformationActions (cf. Appendix C.2.1.15).

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
name	String	The name of a Transformation.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
context	TransitionalPeriod	The context in which a Transformation occurs.
exception	TransformationException	An exception that occurs in a Transformation.
input	TransformableItem	A TransformableItem to be changed in a Transformation.
intent	ChangeCase	A change intended to be realised.
output	TransformableItem	A TransformableItem changed by a Transformation.
worker	TransformationAgent	The body responsible for performing a Transformation.
workItem	TransformationAction	A unit of work in a Transformation.

C.2.1.15 TransformationAction

A TransformationAction is a unit of work performed by a TransformationAgent on TransformableItems, Resources or Zones.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
behaviour	String	A verb or short description about what a TransformationAction is intended to do (e.g. "create", "replace" and "remove").
type	TransformationActionType	The type of a TransformationAction characterising its behaviour.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
context	Transformation	The context in which a TransformationAction is executed.
resource	Resource	A resource (re)configured by a TransformationAction.
target	TransformableItem	A TransformableItem affected by a TransformationAction.
zone	Zone	A Zone configured by a TransformationAction.

C.2.1.15.1 Custom attribute type TransformationActionType

TransformationActionType enumerates the possible kinds of behaviour of a TransformationAction at a high level of abstraction. The kinds can be extended by defining new behaviours to add new meanings to a TransformationAction. Alternatively, new kinds can also be derived from existing kinds to enhance the semantics of the original ones. For instance, the “composition” kind can be further refined into “create”, “delete”, “bind” and “unbind” to designate the kinds of modifications for an application’s structure.

Table Appendix C.5 Enumerated values of TransformationActionType

<i>Name</i>	<i>Semantics</i>
Composition	An action type relating to a change in the composition of a TransformableItem and/or its elements: <ul style="list-style-type: none"> • establishing TransformableItem binding • managing resources • notifying clients of a TransformableItem about modifications
Location	An action type designating that a TransformableItem is being relocated.
State	An action type designating that the state of a TransformableItem is being configured (e.g. according to a State Map, Appendix C.2.2.7).
Zone	An action type designating that the environment in which TransformableItems operate (e.g. the Zone) is being configured, such as allocating resources to a TransformableItem.
Property	An action type relating to a modification to a property (i.e. operating parameter) of a TransformableItem.
ServicingPolicy	An action type designating that a servicing policy is being assigned to a TransformableItem or one of its services or functions.

C.2.1.16 TransformationAgent

TransformationAgents are mobile entities over the network (e.g. Lovrek et al. 2003), responsible for executing Transformations and therefore realising ChangeCases during a TransitionalPeriod. In complex scenarios, TransformationAgents may collaborate with one another to perform their assigned Transformations. It should be emphasised that TransformationAgent is a role. In a self-modifiable Application, for instance, its TransformableItems may also play the role of a TransformationAgent to execute Transformations on themselves. In another case, a TransformationAgent can also be a surrogate of a person who manually performs modifications to an application.

Attributes: none

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
task	Transformation	A unit of work performed by a TransformationAgent.
intent	ChangeCase	A change intended for a TransformationAgent to realise.
coordinator	TransformationAgent	A coordinator for a group of TransformationAgents.
co-worker	TransformationAgent	A peer TransformationAgent with a common interest to be achieved by the coordinator.

C.2.1.17 TransformationException

A TransformationException is an abstract concept representing an error condition occurred during a Transformation, such as the situation when a transformation does not complete within a time limit.

Attributes:

<i>Name</i>	<i>Type</i>	<i>Semantics</i>
reason	String	A description of the error condition.

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
origin	Transformation	The entity from which the error condition originated.
handler	TransformationExceptionResolution	The transformation to process or resolve a TransformationException.

C.2.1.18 TransformationExceptionResolution

A TransformationExceptionResolution designates a special kind of Transformation for resolving one or more TransformationExceptions should they occur. The “Resolution” part of the name is adopted from Dellarocas et al. (1998) concerning exception management.

TransformationExceptionResolution is a subclass of Transformation.

Attributes: none

Associations:

<i>Role</i>	<i>To Class</i>	<i>Semantics</i>
exception	TransformationException	The exception resolved by a TransformationExceptionResolution.

C.2.1.19 TransitionalPeriod

A TransitionalPeriod is a special Stage during which an Application advances from one Generation to another *in transitu* as modifications are underway. It is temporary and short lived compared with the lifecycle of an Application.

Transition is a subclass of Stage.

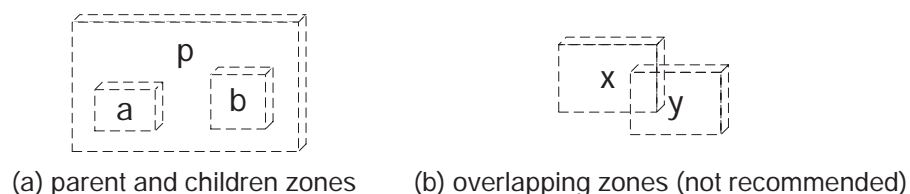
Attributes: none

Associations:

Role	To Class	Semantics
intent	ChangeCase	A set of ChangeCases to be realised during a Transitional Period.
next	Generation	The Generation after a TransitionalPeriod has occurred.
prev	Generation	The Generation before a TransitionalPeriod starts.

C.2.1.20 Zone

A Zone is a *disjoint* partition or region (Evans & Dickman 1999) which establishes a boundary for its TransformableItems to help to manage their lifecycles and to enforce policies on them (e.g. security). A Zone is also a *controlled environment* (cf. "container", Dearle 2007) under the management and/or guardian of some kind of authority (e.g. Transformation Agents). Zones are related to each other in a hierarchy (e.g. Figure Appendix C.1(b)). An enclosing Zone (i.e. parent Zone) can have other Zones (i.e. children Zones) fully enclosed within its boundary, as illustrated in Figure Appendix C.1(a) consisting of a parent "p" and two children "a" and "b". "Disjoint" means boundaries of children zones should not overlap or else this would complicate the domains of responsibilities by the respective authorities, such as when identifying Transformation Agents and their transformations.



source: developed for this research

Figure Appendix C.1 Example hierarchical zones and illegal zones

Attributes: none

Associations:

Role	To Class	Semantics
managedResource	Resource	A Resource offered and managed by a Zone.
command	TransformationAction	A unit of transformation work to be applied to a Zone.
occupant	TransformableItem	The TransformableItems occupying a Zone.
policy	ZoningPolicy	The policy defining rules for configuring a Zone (or more).

C.2.1.21 ZoningPolicy

A ZoningPolicy is a rule for configuring one zone or more in which an application

operates as it evolves.

ZoningPolicy is a subclass of Policy.

Attributes: none

Associations:

Role	To Class	Semantics
target	Zone	The Zone to which the ZoningPolicy applies.

C.2.2 Diagram and Document Fragments

Continuum's diagram and document fragments depict dynamic evolution models, which are abstract representations of some subjects of interest relevant to dynamic evolution. These fragments offer their own notations, which may be extended or reused from existing notations for modelling similar concepts. They are instances of SEMDM WorkProduct/*Kind.

Table Appendix C.6 lists various dynamic evolution model unit fragments (i.e. instances of ModelUnit/*Kind) used in diagram and document fragments (instances of Model/*Kind). The entry in each cell (e.g. "1..n") is a *multiplicity* using the UML notation (OMG 2010b), which specifies a range for the possible number of occurrences of a model unit fragment (i.e. row) in a diagram or document fragment (i.e. column). The presence of such an entry also implies an instance of SEMDM's ModelUnitUsage/*Kind in Continuum.

Table Appendix C.6 Model Unit fragments and their usage in model artefacts

Model Unit Fragment	Application Lifecycle Diagram	Dynamic Application Change Document	Dynamic Evolution Quality Inspection Report	Dynamic Evolution Quality Problem Analysis Report	Dynamic Evolution Quality Profile Report	New and Replacement Transformable Item Catalogue	State Map	Structural Configuration - Notational Extensions	Transformation Diagram	Transformation Orchestration Diagram	Zone Change Document
Application	1..1										
ApplicationLifecycle	1..1										
ChangeCase	0..n	1..n									
Generation	1..n	1..n								2..2	
Impact		0..n									
OperationalProfile						1..n					
Policy (abstract)											
Resource		0..n							0..n		
ServicingPolicy								0..n			
Stage (abstract)											
TransformableItem		0..n				1..n	2..2	0..n	0..n		0..n

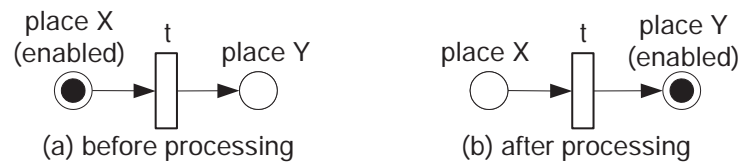
<i>Model Unit Fragment</i>	<i>Application Lifecycle Diagram</i>	<i>Dynamic Application Change Document</i>	<i>Dynamic Evolution Quality Inspection Report</i>	<i>Dynamic Evolution Quality Problem Analysis Report</i>	<i>Dynamic Evolution Quality Profile Report</i>	<i>New and Replacement Transformable Item Catalogue</i>	<i>State Map</i>	<i>Structural Configuration - Notational Extensions</i>	<i>Transformation Diagram</i>	<i>Transformation Orchestration Diagram</i>	<i>Zone Change Document</i>
Transformation							1..1	0..n	1..1	1..n	
TransformationAction									1..n		
TransformationAgent									1..n	1..n	
TransformationException										0..n	
TransformationException Resolution										0..n	
TransitionalPeriod	0..n							0..n		1..1	
Zone		0..n						0..n	0..n		1..n
ZoningPolicy											1..n

Note: Dynamic Evolution Quality Inspection Report, Dynamic Evolution Quality Problem Analysis Report and Dynamic Evolution Quality Profile Report do not document artefacts using any of the model units. They are however displayed in the table for completeness.

C.2.2.1 Application Lifecycle Diagram and Notation

Description: An Application Lifecycle Diagram depicts the evolution landscape of an application, in terms of generations and transitional periods, using the notion of Petri nets with minor extensions. In a nutshell, a Petri net²⁵ is a directed graph with mathematical and graphical facets to model processes and systems that exhibit concurrent, asynchronous, distributed, parallel and/or non-deterministic behaviour (ISO/IEC 2004). The basic premises of a Petri net are “places” and “transitions”, being the two node types drawn respectively as circles and boxes, plus “arcs” linking the nodes. A Petri net is labelled with markings, called “tokens” drawn as filled circles, to denote the state of a net. Tokens move along arcs and nodes in a Petri net to describe the underlying behaviour in action. To model data processing in action, for instance, Petri net places represent inputs to and outputs of a process, and a Petri net transition as the process. The presence of a token in an input or output place is interpreted as data presenting to or produced from the process respectively. Figure Appendix C.2 illustrates the modelling of a simple process, via the traversal of a token from input place X to output place Y via transition t.

²⁵ It is beyond the scope of this research to fully describe and review Petri net. Research in Petri net has been extensive and well published (e.g. the International Standard ISO/IEC 15909-1:2004, ISO/IEC 2004).



source: developed for this research

Figure Appendix C.2 Example Petri nets in action

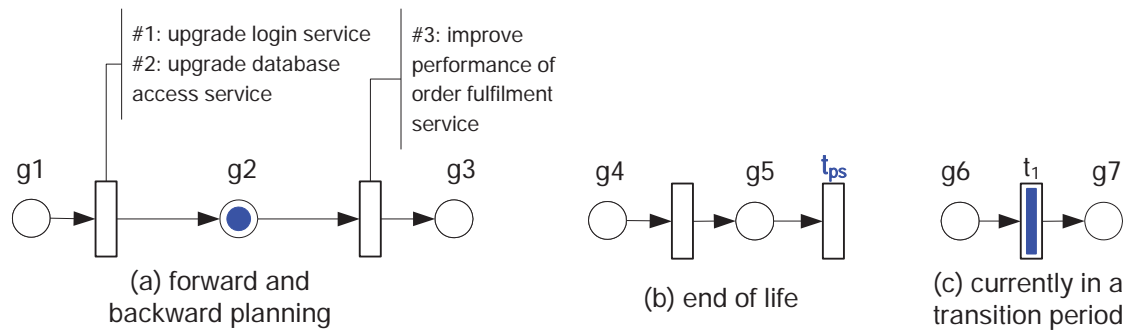
To empower Petri net in modelling an application lifecycle, a mapping is defined in Table Appendix C.7. Those marked with asterisks are extensions to the basic Petri net concepts. They will be clarified in the examples after Table Appendix C.7.

Table Appendix C.7 Notations for Application Lifecycle Diagram

<i>Continuum Notation</i>	<i>Continuum Semantics</i>	<i>Equivalent Petri net Semantics</i>
	Generation (past or future)	Place
	Generation (current)	Place marked with single Token
	Transitional Period (past or future)	Transition
	Transitional Period (current)	* Transition marked with a small filled box
	entering or exiting a Transitional Period	directed Arc
	Change Cases associated with a Transitional period	* annotation to Transition

Figure Appendix C.3 depicts three uses of these notations for one application:

- An application is defined to pass through two transitional periods: from release “g1” to the current release “g2” (marked with a token), and from “g2” to the next release “g3”. “g1” is a *source place* with no inbound arc (ISO/IEC 2004), representing a first or initial release of the application.
- The evolution history of a decommissioned application is shown. Notice that none of the nodes are marked with a token. The final transitional period labelled “t_{ps}” is an indefinite period meaning that the application has reached the end of its application lifecycle and is no longer in use. It is equivalent to a “sinking transition” (ISO/IEC 2004) in Petri net.
- Transitional period “t₁” is explicitly marked with a token to indicate that an application is currently in a transitional period. Marking a transitional period this way enables the diagram to emphasise the latency aspects of a transitional period (or more), which is unsupported by ordinary Petri net transitions.



source: developed for this research

Figure Appendix C.3 Example application lifecycle diagrams

Each transitional period can be further annotated with change case tags, to designate the change cases that are to be realised in the transitional period, as a way of evolution planning. In this regard, Figure Appendix C.3(a) illustrates in which of the transitional periods the three change cases (“#1”, “#2” and “#3”) are to be realised.

There are restrictions for applying Petri net notations in an Application Lifecycle Diagram:

- There is one and only one token in the diagram;
- Each Petri net place can hold at most one Petri net token; and
- Arc traversal is directed by relevant software development initiatives.

C.2.2.2 Dynamic Application Change Document

Description: A Dynamic Application Change Document catalogues dynamic changes proposed for an application. In particular, dynamic changes are specified as change cases and accompanied impact sets. Change cases are derived from requirements and change requests with respect to the current runtime structure and configuration of an application. Each impact set identifies the entities in an application or those outside an application thought to be impacted by the associated change case should the change case be realised. A Dynamic Application Change Document uses the table structures of Table Appendix C.8 and Table Appendix C.9 as templates to record change cases. The semantics of each column is described in Appendices C.2.1.3 and C.2.1.5.

Table Appendix C.8 Example documentation of change case and impact

Change Case				Impact			
<i>Id</i>	<i>Purpose</i>	<i>Description</i>	<i>Related Requirement</i>	<i>Target</i>	<i>Impact Type</i>	<i>Level of Disruption</i>	<i>Supplementary Change Case</i>
generation v1.4							
CC8	Add visual features to UI (user interface)	The original UI displays the seating capacity and the location of a selected lecture room. The new UI will display the audio/visual equipments available, and a map showing the room location.	Use Case 5: Display details of selected room	Search Engine (it must now provide new search data fields to the UI)	indirect	medium	CC50: Replace search query composer

Table Appendix C.9 Example documentation of change case and transformation

<i>Change Case ID</i>	<i>Change Case Purpose</i>	<i>Enactment (i.e. Name of responsible transformation(s))</i>
CC8	Add visual features to UI	transformation "UI upgrade"

C.2.2.3 Dynamic Evolution Quality Inspection Report

Description: An inspection checklist consists of items, documented as inspection questions, to inspectors or reviewers with hints and recommendations for finding defects and issues (Brykczynski 1999). In particular, the Dynamic Evolution Quality Inspection Report offers a checklist of inspection questions for evaluating the state of dynamic evolution quality. During an inspection, inspection results are recorded against the checklist in such a report, to be used as inputs for improving the quality for dynamic evolution afterwards. As shown in Table Appendix C.10, inspection questions are grouped under their respective quality categories and quality factors. The questions were derived from the attributes of the quality factors (cf. Table 6.1) along with enhancements from the relevant literature and methodologies (as cited in Table Appendix C.10). Note that not all quality factors are suitable or important for any endeavour at hand. Accordingly, an inspection should select the inspection questions relevant to the dynamic evolution quality factors of interest.

Table Appendix C.10 Template for dynamic evolution quality inspection report

Inspection Question(s)	Type of artefact to inspect: "A" for analysis "D" for design	"Y", "N" or "N/A" (yes, no or not applicable)	If "No", describe all defects/issues
Soundness of Change [i.e. quality category]			
Completeness [i.e. quality factor]			
Functionality (reused and converted into questions from RUP) After a transformation is applied to an application: <ul style="list-style-type: none"> Is the functionality offered by the application (i.e. external functionality) covered by its realisations (i.e. of respective transformable items)? Is the internal functionality relevant to and/or supporting external functionality) covered by its realisations (i.e. of respective transformable items)? 	D		
Transformable items and workflows After a transformation is applied to an application, <ul style="list-style-type: none"> Are all transformable items (including workflows) specified in the application present? Are all transformable items participating in workflows present? 	D		
Bindings After a transformation is applied to an application, <ul style="list-style-type: none"> Are all bindings specified in the application present? Are all broken bindings (i.e. without required transformable items) identified and removed from the application? Are all illegal bindings identified and removed from the application? 	D		
Assumptions and properties <ul style="list-style-type: none"> Are assumptions and properties of an application and its parts met by change cases and transformations? 	A, D		
Consistency			
Bindings <ul style="list-style-type: none"> Are bindings compatible (.e.g. interface) with their associated transformable items after a transformation? 	D		
Protocols <ul style="list-style-type: none"> Do transformable items use compatible protocols when communicating with one another after a transformation? 	D		
Dynamic change impact <ul style="list-style-type: none"> Have all transformable items to be involved in a dynamic change been identified? 	D		
Resuming state (Appendix C.2.1.13.1) <ul style="list-style-type: none"> Have resuming states been identified for all new and replacement transformable items to be placed into an application by a transformation? Are all resuming states reachable (i.e. which can be progressively transitioned from the start-up states of their respective transformable items)? 	D		
State maps <ul style="list-style-type: none"> Have state maps been specified for all transformable items affected by a transformation, to ensure they do not progress towards any error state but continue from their resuming states after a transformation? 	D		

<i>Inspection Question(s)</i>	<i>Type of artefact to inspect: "A" for analysis "D" for design</i>	<i>"Y", "N" or "N/A" (yes, no or not applicable)</i>	<i>If "No", describe all defects/ issues</i>
<i>State synchronisation</i> <ul style="list-style-type: none"> Are the resuming states of all new and replacement transformable items synchronised with the state of the application and vice versa after a transformation? (For instance, if a new component is required to keep a database available for use by an application, it will do so after a transformation.) 	D		
<i>Quiescent state</i> (Appendix C.2.1.13.1) <ul style="list-style-type: none"> Have quiescent states been identified for all transformable items affected by a transformation? During each quiescent state (from which a transformation will commence): <ul style="list-style-type: none"> Are critical procedures prevented from execution? Are message exchanges, interactions and transactions prevented from being started? 	D		
<i>System invariants</i> <ul style="list-style-type: none"> Are all system invariants preserved from transformations? 	D		
<i>Resources and support for transformable items</i> <ul style="list-style-type: none"> Have adequate resources and support been identified and provided for new and replacement transformable items? 	D		
<i>Assumptions and properties</i> <ul style="list-style-type: none"> [See Completeness] 	A, D		
<i>Correctness</i>			
<i>Dynamic changes</i> <ul style="list-style-type: none"> Are proposed dynamic changes non-arbitrary, meaning <ul style="list-style-type: none"> all proposed changes are captured (e.g. using ChangeCase, Appendix C.2.1.3); and recipients of all proposed changes are identified in an application? Are proposed dynamic changes feasible (i.e. corresponding transitional periods and transformations identifiable to realise the changes)? 	A, D		
<i>Behaviour</i> <ul style="list-style-type: none"> Are appropriate servicing policies (Appendix C.2.1.11.1) declared for all transformable items affected by a transformation when it is executed? 	D		
<i>Transformation ordering</i> <ul style="list-style-type: none"> Are transformations ordered as appropriate in all transformation orchestration diagrams? Are transformation actions ordered correctly in all transformation diagrams? 	D		
<i>Transformations</i> <ul style="list-style-type: none"> During each transitional period, are transformations broken into sufficiently short and simple units to facilitate execution? Is each transformation executed at a right time (e.g. avoiding interference with normal business activities (Carzaniga et al. 1998))? 	D		

<i>Inspection Question(s)</i>	<i>Type of artefact to inspect: "A" for analysis "D" for design</i>	<i>"Y", "N" or "NA" (yes, no or not applicable)</i>	<i>If "No", describe all defects/ issues</i>
<i>Infusibility of Change</i>			
<i>Locality</i>			
<i>Zoning and change localisation</i> <ul style="list-style-type: none"> Are zoning policies (Appendix C.3.1.4.2) followed when defining zones? Are changes local to a set of transformable items (e.g. in a zone) rather than global to an application? 	D		
<i>Maintainability</i>			
<i>Transformable items</i> <ul style="list-style-type: none"> Are transformable items that will participate in interactions (or workflows), after a transformation, explicitly defined in these interactions (using relevant modelling languages UML, BPMN etc.)? 	D		
<i>Cost and ease of modifications</i> (merged and enhanced from Select Perspective, EPIC and OPF): <ul style="list-style-type: none"> Are specifications for new and replacement transformable items well-defined, standard based, tested, catalogued and current after changes? Are realisations for new and replacement transformable items appropriately documented and catalogued after changes? Are changes to zones of an application identified (e.g. in Zone Change Document, Appendix C.2.2.11)? Is an application appropriately layered after changes? Are new and replacement transformable items identified (e.g. in New and Replacement Transformable Item Catalogue, Appendix C.2.2.6)? After changes are made, will the documentation be current for: <ul style="list-style-type: none"> the application lifecycle (i.e. up-to-date Application Lifecycle Diagram, Appendix C.2.2.1); and the application structure (e.g. using Structural Configuration - Notational Extensions, Appendix C.2.2.8)? Are project conventions followed for <ul style="list-style-type: none"> the application lifecycle (e.g. Appendix C.2.2.1); change specifications (e.g. Appendix C.3.2.1); state maps (e.g. Appendix C.2.2.7); the application structure (e.g. UML, BPMN and Appendix C.2.2.8); and transformations (e.g. Appendix C.2.2.9)? 	A, D		
<i>Testability</i> <ul style="list-style-type: none"> Is the documentation for input and output characteristics, and states of transformable items current after changes? Do the changes degrade application observability (Appendix C.3.2.17.12)? Do the changes degrade application controllability (Appendix C.3.2.17.12)? 	D		
<i>Interactions (or workflows)</i> <ul style="list-style-type: none"> Are interactions (or workflows), after a transformation, clearly documented, detailed (e.g. with UML, BPMN) and current after changes? 	D		

<i>Inspection Question(s)</i>	<i>Type of artefact to inspect: "A" for analysis "D" for design</i>	<i>"Y", "N" or "NA" (yes, no or not applicable)</i>	<i>If "No", describe all defects/ issues</i>
Transparency			
<i>End users</i> <ul style="list-style-type: none"> Are transformations, when executed, hidden from end users? 	D		
<i>Application programmers</i> <ul style="list-style-type: none"> Are the design and implementation for transformations hidden from application programmers such that they do not need to take care of transformations when developing the business logic of an application? 	D		
<i>Transformable items</i> <ul style="list-style-type: none"> Are transformations hidden from transformable items of an application unaffected by the transformations? 	D		
<i>Zones</i> <ul style="list-style-type: none"> Are transformation agents abstracted from zones in which an application operates? 	D		
Changeability of Application			
Autonomy			
<i>Transformable items</i> <ul style="list-style-type: none"> Are transformable items designed to have control and governance of their own processing? 	D		
Coordination			
<i>Transformation agents</i> <ul style="list-style-type: none"> Are transformation agents organised to facilitate the coordination of transformations among multiple zones during a transitional period? 	D		
<i>Network</i> <ul style="list-style-type: none"> Do transformation agents have a means of tolerating network unreliability when they coordinate with one another during a transitional period? 	D		
Extensibility			
<i>Application with new functionality</i> <ul style="list-style-type: none"> Does the application support extensions of its compositions with new functionality? (customised from Zenger (2004, pp. 9-10)) Are compositions defined with the specifications rather than the realisations of transformable items? Are zones used to confine the scope of changes? 	D		
<i>Application with new transformable items</i> (customised from Zenger (2004, pp. 3,10)) <ul style="list-style-type: none"> Are variation points defined in an architecture to plug in alternative or additional transformable items where appropriate? Are bindings between transformable items dynamic instead of static? Does the application support multiple versions of transformable items to co-exist? 	D		
<i>Parts with new functionality</i> <ul style="list-style-type: none"> Does the architecture support extensions of its transformable items with new functionality? 	D		

<i>Inspection Question(s)</i>	<i>Type of artefact to inspect: "A" for analysis "D" for design</i>	<i>"Y", "N" or "N/A" (yes, no or not applicable)</i>	<i>If "No", describe all defects/ issues</i>
<i>Loose Coupling</i>			
<i>Transformable items</i> <ul style="list-style-type: none"> Are transformable items loosely coupled from one another? Do transformable items have their own lifecycles where appropriate? Are transformable items allocated their own runtime environments where appropriate? 	D		
<i>Separation of Concerns</i>			
<i>Functionality vs. dynamic changes</i> <ul style="list-style-type: none"> Are dynamic change concerns (e.g. change cases, transformations, transitional periods, application lifecycle) explicitly defined? Are dynamic change concerns separated from functionality concerns (e.g. specifications and realisations for transformable items)? 	D		
<i>Functionality vs. communication</i> <ul style="list-style-type: none"> Are channels defined in bindings between communicating transformable items? Are communication concerns between transformable items mediated in channels or their bindings? 	D		
<i>Functionality vs. security</i> (reused and converted into questions from ERL) <ul style="list-style-type: none"> Is the support for security separated from functionality of transformable items? Are relevant standards followed for security concerns where appropriate (e.g. use of WS-Security (OASIS 2006) as in ERL (2005))? 	D		
<i>Transformable item realisation vs. clients</i> (reused and converted into questions from Select Perspective) <ul style="list-style-type: none"> Are realisations for transformable items hidden from their clients? Does the development for clients of transformable items use the specifications for the transformable items only but not their realisations? Is the development for the realisations for transformable items separated from the development for their clients (e.g. separate activities used in Select Perspective (Apperly et al. 2003))? 	D		
<i>Transformable item specification vs. realisation</i> <ul style="list-style-type: none"> Is the specification for a transformable item distinguished from its realisation as separate concepts? Are the specification and realisation for a transformable item developed with separate activities? Are the specification and realisation for a transformable item documented separately? 	D		

<i>Inspection Question(s)</i>	<i>Type of artefact to inspect: "A" for analysis "D" for design</i>	<i>"Y", "N" or "NA" (yes, no or not applicable)</i>	<i>If "No", describe all defects/ issues</i>
<i>Robustness of Application</i>			
<i>Fault Tolerance</i>			
<i>New and replacement transformable items</i> <ul style="list-style-type: none"> Is an application protected from potential faults of new and replacement transformable items? Are barriers established to contain potentially faulty new and replacement transformable items to minimise their impact on the application? 	D		
<i>Recoverability</i>			
<i>Failure caused by transformation</i> <ul style="list-style-type: none"> Are transformations declared with exceptions and exception handlers where appropriate to restore an application to continue to offer its functionality, for failures caused by transformations? 	D		
<i>Failure caused by dynamic change</i> <ul style="list-style-type: none"> Is an application restored to a state to continue to operate for failures caused by dynamic changes (i.e. after a transformation)? 	D		
<i>Reliability</i>			
<i>Control</i> <ul style="list-style-type: none"> Does an application limit the behaviour of its new and replacement transformable items to avoid itself from being compromised? 	D		
<i>New and replacement transformable items</i> <ul style="list-style-type: none"> Do new and replacement transformable items behave as expected, both functionally and non-functionally, in an application? 	D		
<i>Safety</i>			
<i>In-transformation</i> <ul style="list-style-type: none"> Do transformations not lead an application and its transformable items to operate unsafely? 	D		
<i>Post-transformation</i> <ul style="list-style-type: none"> Do dynamic changes not lead an application and its transformable items to operate unsafely? 	D		
<i>Security</i>			
<i>Transformation agents</i> <ul style="list-style-type: none"> Are transformation agents secured from unauthorised access? 	D		
<i>Application</i> <ul style="list-style-type: none"> Is the security of an application protected from being compromised by new and replacement transformable items after a transformation? 	D		
<i>New and replacement transformable items</i> <ul style="list-style-type: none"> Is access to new and replacement items restricted where appropriate to ensure they are used as intended after a transformation? 	D		

<i>Inspection Question(s)</i>	<i>Type of artefact to inspect: "A" for analysis "D" for design</i>	<i>"Y", "N" or "NA" (yes, no or not applicable)</i>	<i>If "No", describe all defects/ issues</i>
<i>Security policies</i> <ul style="list-style-type: none"> Are security policies dynamically updatable and reloadable in an application? 	D		
<i>Security policies and enforcement</i> <ul style="list-style-type: none"> Are the declarations for security policies separated from their enforcement? 	D		

C.2.2.4 Dynamic Evolution Quality Problem Analysis Report

Description: A Dynamic Evolution Quality Problem Analysis Report documents the information about the defects/issues found and analysed for dynamic evolution quality. Each defect/issue consists of the following attributes:

- *Phase detected:* the development phase in which a defect or an issue was detected
- *Defect/Issue type:* the type of the defect or issue found
- *Defect/Issue location:* the work product in which the defect or issue has actually been found
- *Root cause(s):* the trigger that led to the defect or issue

Table Appendix C.11 provides an example as well as a template for documenting a Dynamic Evolution Quality Problem Analysis Report. See Technique Root Cause Analysis for ranges of values suitable for these attributes (cf. Appendix C.3.2.10).

Table Appendix C.11 Example dynamic evolution quality problem analysis report

<i>Defect/Issue Description</i>	<i>Phase Detected</i>	<i>Defect/Issue Type</i>	<i>Defect/Issue Location</i>	<i>Root Cause(s)</i>
Cache component not initialised after deployment	Dynamic evolution design	TransformableItem	State Map for cache component	Transformation Design incomplete
				Individual mistake

C.2.2.5 Dynamic Evolution Quality Profile Report

Description: A Dynamic Evolution Quality Profile Report records assessment results on dynamic evolution quality factors for an application. An assessment aims at grading these factors for their importance, with the results providing a basis from which quality factors can be ranked and prioritised to be dealt with in an endeavour. The importance for each quality factor is likely to be different for each particular situation (i.e. the application). For instance, Security could be seen as more important to an application with components distributed in a wide area network than an application running in a

secure local area network. At other times, quality factors are re-assessed over time to adapt to the changing nature of a business and the needs of end users. For example, initial releases of a simple application might ignore “extensibility” but as it grows in size and complexity, application extension becomes increasingly important.

The Dynamic Evolution Quality Profile Report is produced by filling out the template as shown in Table Appendix C.12. A quality factor is represented by one or more quality attributes, each further exemplified by a consequence in the table should the attribute fail to be met. A quality attribute is assessed for an application by assigning an importance score to the attribute (between 1 for not important at all and 7 for extremely important). Each quality factor is then scored by taking the average of the scores of its quality attributes. Its score is recorded in the rightmost column of the template.

Table Appendix C.12 Template for dynamic evolution quality profile report

Quality Factor	Quality Attribute	Possible Adverse Consequences	Importance Score (1-not important at all 7-extremely important)	
			Quality Attribute	Quality Factor
Soundness of Change				
Completeness	No missing functionality after a transformation	An application missing functionality to operate		
	No missing transformable items after a transformation	An application missing transformable items to operate		
	No missing, illegal or broken bindings after a transformation	Transformable items unable to interact through missing, illegal or broken bindings		
	(Also in Consistency) assumptions and properties of a distributed application and its transformable items met by a change	Assumptions and properties of a application and its transformable items violated		
Consistency	Compatible bindings	Transformable items unable to bind to one another		
	Compatible communications protocol among transformable items	Transformable items unable to communicate		
	All transformable items involved in a runtime change identified before a transformation	Transformable items requiring modifications left out of a transformation		
	No progression towards an error state after a transformation	Application failure after a transformation		
	Synchronisation of application's and transformable items' states after a transformation	An application viewing a new transformable item as not ready for use yet the item has been initiated and operating		

Quality Factor	Quality Attribute	Possible Adverse Consequences	Importance Score (1-not important at all 7-extremely important)	
			Quality Attribute	Quality Factor
	A reachable state attained after a transformation	Unknown application behaviour (due to being in an unknown state)		
	No critical procedures executed before a transformation	Critical procedures in execution aborted during a transformation		
	No pending messages, interactions or transactions before a transformation	Pending messages, interactions or transactions aborted during a transformation		
	System invariants preserved from a transformation	System invariants violated		
	Adequate resources and support for new and replacement transformable items	New and replacement transformable items unable to operate		
	(Also in Completeness) assumptions and properties of a distributed application and its transformable items met by a change	Assumptions and properties of an application and its transformable items violated		
Correctness	Non-arbitrary and admissible changes	Changes too complex and error prone to implement in an application		
	No unintentional behaviour during and after a transformation	An application behaving erratically during and after a transformation		
	Correct ordering of transformations	Transformable items inserted into an application in a wrong order		
	Transformations at a right time	Transformations interrupting an application during busy business operating hours		
Infusibility of Change				
Locality	Application partitioning and change localisation to partitions	A small change in one part of an application triggering changes to many other parts that have not been accounted for		
Maintainability	All transformable items clearly defined in interaction (or workflow) specifications	Transformable items participating in interactions not accounted for change as their interactions change		
	No degradation in cost and ease of modifications	An application more costly and difficult to evolve		
	No reduction in testability	An application more difficult to test		
	Clear and detailed interactions	Unable to evolve interactions because of lack of documentation on interactions		

			Importance Score (1-not important at all 7-extremely important)	
Quality Factor	Quality Attribute	Possible Adverse Consequences	Quality Attribute	Quality Factor
Transparency	Transformations hidden from end users	End users noticing and dissatisfied with interruptions caused by transformations		
	Transformation design and implementation hidden from application programmers	Harder to focus on normal application development as transformation design and implementation are mixed with application code		
	Transformations hidden from transformable items unaffected by the transformations	Potential interruptions to transformable items unaffected by the transformation		
	Transformation agents hidden from operating environment	Transformation agents exposed to transformable items that run in the operating environment		
Changeability of Application				
Autonomy	Self-control and self-governance of transformable items' own processing	Transformable items unable to evolve independently		
Coordination	Transformations coordinated among multiple nodes	Changes in different nodes out of sync with one another		
	Transformation agents tolerant of network unreliability during a transformation	Transformation agents unable to complete transformations executed remotely		
Extensibility	Runtime extension/upgrade of an application with new functionality	An application unable to accommodate new functionality with ease		
	Runtime extension/upgrade of transformable items in an application with new functionality	Transformable items unable to accommodate new functionality with ease		
	Runtime extension/upgrade of an application with new transformable items	An application unable to accommodate new transformable items with ease		
Loose Coupling	High level of independence between transformable items	Increase in likelihood of changes to one transformable item causing changes to others		
	Transformable items having their own lifecycles and runtime environments	Ditto		
Separation of Concerns	Separating dynamic change concerns (i.e. via transformations) from functionality concerns	Transformation design tangled up with functional design, making the latter harder to evolve		
	Separating communication concerns from functionality concerns	Harder to evolve communication design and functionality independently		

Quality Factor	Quality Attribute	Possible Adverse Consequences	Importance Score (1-not important at all 7-extremely important)	
			Quality Attribute	Quality Factor
	Separating security support from functionality concerns	Harder to evolve security support and functionality independently		
	Separating realisations of transformable items from those of transformable item clients	Harder to evolve transformable items from their clients independently		
	Separating transformable item specification from realisation concerns	Harder to evolve transformable items without also requiring updates to their clients		
Robustness of Application				
Fault Tolerance	High tolerance of faulty new and/or changed transformable items	An application producing faulty results in the presence of faults from new and replacement transformable items		
	Barriers established to contain potentially faulty new and replacement transformable items	Faults from new and replacement transformable items rippling through and impacting other parts of an application		
Recoverability	Restoration of an application and its parts to a state to continue to perform their functionality, after a failure caused by a transformation and/or its dynamic change(s)	An application unable to continue operation after transformation failures		
Reliability	No compromise on intended functionality after a transformation	Functionality compromised by new and replacement transformable items		
	Replacement transformable items fully satisfying their roles	New and replacement transformable items behaving unexpectedly		
Safety	Distributed application and its transformable items operating safely during and after a transformation	Potential safety hazards triggered during and/or after a transformation		
Security	Transformation agents secured from unauthorised access	Unauthorised changes to application		
	No security compromise by new and replacement transformable items after a transformation	New and replacement transformable items introducing security compromise in application		
	Access to for new and replacement transformable items restricted	New and replacement transformable items unprotected from misuse by distrusted ones		
	Dynamically updated security policy	Security policies not easily updated to adapt to changing security requirements		

<i>Quality Factor</i>	<i>Quality Attribute</i>	<i>Possible Adverse Consequences</i>	<i>Importance Score (1-not important at all 7-extremely important)</i>	
			<i>Quality Attribute</i>	<i>Quality Factor</i>
	Separating security policy from security enforcement	Unable to update security enforcement without changes to policies		

C.2.2.6 New and Replacement Transformable Item Catalogue

Description: A New and Replacement Transformable Item Catalogue documents Transformable Items (new and replacement) to be added to an application, their state configurations from which they start up to operate, and their Operational Profiles (Appendix C.2.1.6). Continuum supports two kinds of operational profile, viz. resource and performance profiles (cf. Appendices C.2.1.10 and C.2.1.8). Operational profiles can be used to verify if these profiles will still be met after changes are made to an application at runtime. For simple resource and performance profiles, the template shown in Table Appendix C.13 can be used for such a catalogue:

Table Appendix C.13 Example documentation of simple new and replacement transformable item catalogue

<i>New and Replacement Transformable Items</i>				
<i>Responsible Transformation(s)</i>	<i>Name</i>	<i>Change in Resource Profile (current vs. new)</i>	<i>Change in Performance Profile (current vs. new)</i>	<i>Start-up State Configuration</i>
<i>progressing from generations V3 to V4</i>				
data store upgrade	new data store	exclusive memory: 4TB/16TB (current/new)		state map XYZ (cf. example in Table Appendix C.15)
Web UI upgrade	new Web UI	exclusive memory: 512MB/4GB (current/new)	minimum number of users: 2000/5000 (current/new)	N/A

For a more sophisticated resource profile, an estimate for a resource consumed by a transformable item can be treated as consisting of the static (i.e. steady usage) and dynamic parts (i.e. on-demand usage) (see resource profile modelling in Appendix C.3.2.9). Thus, the example template in Table Appendix C.14 can be used to supplement Table Appendix C.13. The “exclusive use” column indicates if a resource allocation will be dedicated exclusively to a consumer, or can be shared with other consumers. The “current” and “new” columns register the current and new usages for a resource before and after a transformation. The “static” and “dynamic” columns register

the steady-state and on-demand usage estimates for a resource.

Table Appendix C.14 Example documentation of resource profile

<i>Resource</i>	<i>Resource Consumer</i>	<i>Exclusive Use</i>	<i>Static (GB)</i>		<i>Dynamic (GB)</i>	
			<i>Current</i>	<i>New</i>	<i>Current</i>	<i>New</i>
network drive space	new data store	Yes	2000	4000	2000	12000
	new Web UI	Yes	¼	2	¼	2

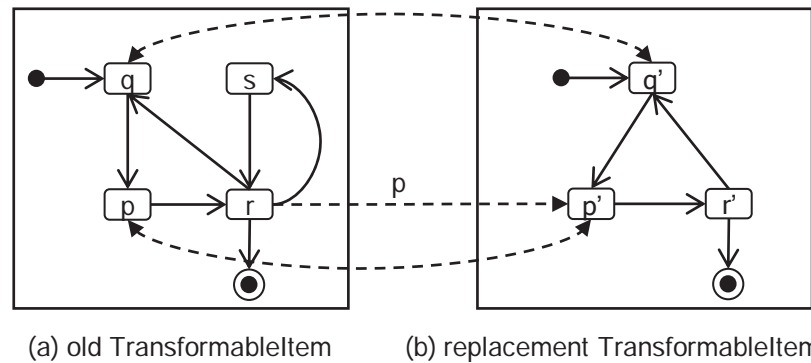
For complex performance profiles, Continuum reuses Adaptive Service Grid's (ASG's) performance metrics model (Kempton et al. 2007) which comprehensively models performance characteristics for services, applications and their operating environment. This is summarised in the descriptions for the *Performance Profile* model unit fragment (Appendix C.2.1.8). For reference only, there are alternative approaches for performance analysis and prediction, such as those surveyed by Balsamo et al. (2004).

C.2.2.7 State Map

Description: A State Map describes the mapping between the states of an existing TransformableItem and the states of the TransformableItem replacing it (i.e. the replacement) after a transformation is complete. A state map is defined as a tuple (Q, M, V) where Q and V are quiescent and resolved states of an existing or old TransformableItem, and M is a resuming state of a replacement TransformableItem (see Appendix C.2.1.13.1 for state definitions). This tuple is depicted in graphical and tabular ways. In the former, two UML state machines (OMG 2010b) for the old and replacement TransformableItems are interconnected with the following arc types:

- A double-ended arrow dashed arc represents a two-way mapping between a state of the old TransformableItem, being both a quiescent and a resolved state, and a resuming state of the replacement TransformableItem. This means when a transformation occurs and the old TransformableItem happens to be at a quiescent state connected by the arc, the replacement TransformableItem will commence from the resuming state connected by the arc after the transformation. If the transformation fails, the old TransformableItem is set to continue to operate from the same quiescent state pointed to by the arc. The arc connecting state p of the old to state p' of the replacement TransformableItem in Figure Appendix C.4 illustrates this scenario.
- A single-arrow dashed arc from a state of the old TransformableItem to a state of its replacement defines a forward mapping from the former to the latter, with the resolved state being annotated on the arc. Its semantics is similar to a

double-arrow dashed arc, except that the resolved state of the old TransformableItem is the state annotated on the arc rather than the state connected by the arc. So, if the transformation fails, the old TransformableItem is set to continue to operate from the designated resolved state. Figure Appendix C.4 illustrates this scenario, where the arc linking state *r* of an old TransformableItem to state *p'* of its replacement TransformableItem designates state *p* as the resolved state if a transformation fails.



source: developed for this research

Figure Appendix C.4 Example state map for two UML state machines

In the tabular form, rows and columns of a state map correspond to the observable states of the old TransformableItem and the states of the replacement TransformableItem. State identifiers in the row and column headings are further annotated with “Q” for Quiescent, “M” for resuMing and “V” for resoluEd states. Table Appendix C.15 and Figure Appendix C.4 describe the same information.

Table Appendix C.15 Tabular representation of state map in Figure Appendix C.4

resolved state		replacement state		
		<i>p'</i> : M	<i>q'</i> : M	<i>r'</i>
old state	<i>p</i> : QV	p		
	<i>q</i> : QV		q	
	<i>r</i> : Q	p		
	<i>s</i>			

Note that certain TransformableItems have no notion of state, such as stateless services in a SOA environment (Milanovic & Malek 2004). As such, State Map is not applicable.

C.2.2.8 Structural Configuration - Notational Extensions

Description: A structural configuration generally depicts the runtime structure of an

application at some stage in its lifecycle. As various mature modelling languages and notations already exist to model the structural configurations of different types of composition-based distributed applications, Continuum does not prescribe its own set of extensive notations for modelling the same kinds of concepts. Instead, Continuum offers several notations, listed in Table Appendix C.16, as generic extensions to those languages to enhance their richness and support for dynamic evolution. These notations are sufficiently generic for different types of composition-based distributed applications, and sufficiently detailed to assist in modelling an application undergoing dynamic changes with Continuum.

Table Appendix C.16 Dynamic evolution related notations for structural configurations







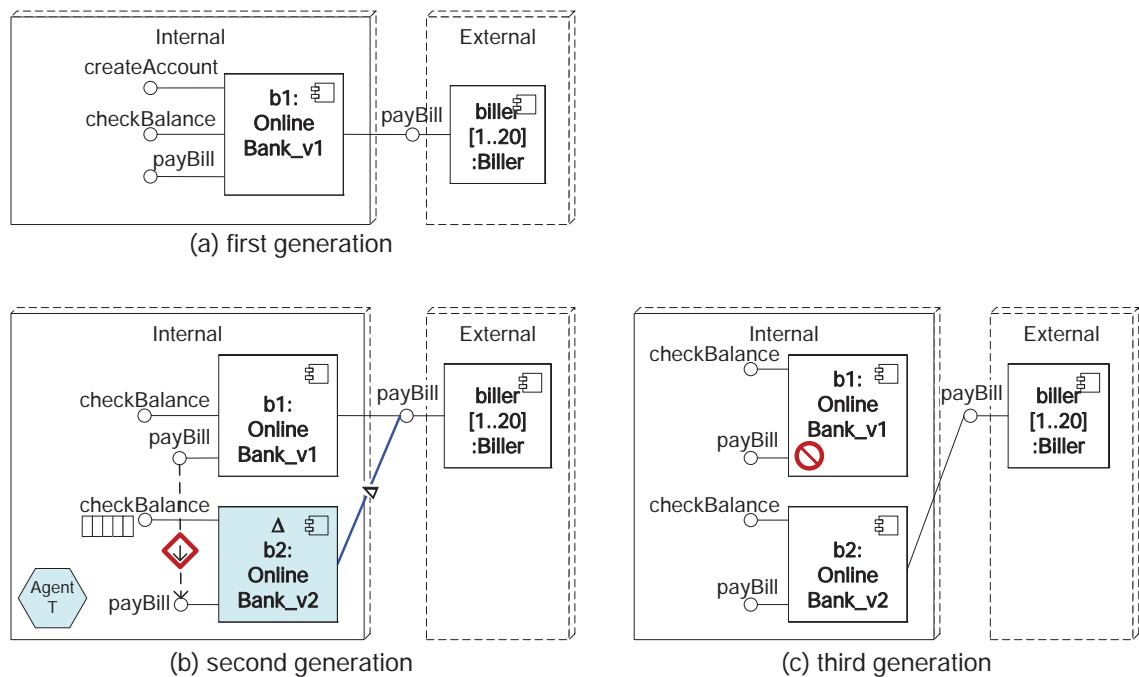
Continuum Notation	Continuum Semantics
	A Zone (Appendix C.2.1.20).
«blocked and queued» or 	Attached to a TransformableItem to indicate its Servicing Policy being set as “blocked and queued”, such as during a Transformation (Appendix C.2.1.11).
«illegal» or 	Attached to a TransformableItem to indicate its Servicing Policy being set as “illegal”, such as during a Transformation (Appendix C.2.1.11).
«delegated» or 	Attached to a TransformableItem to indicate its Servicing Policy being set as “delegated”, such as during a Transformation (Appendix C.2.1.11). The arrow points to the TransformableItem or its services being delegated with the responsibilities.
	A TransformationAgent (Appendix C.2.1.16).
Delta symbol “Δ” with additional highlighting on the parts (i.e. thickened line for a binding, shading on a TransformableItem)	Attached to a TransformableItem or a binding between TransformableItems to indicate that either it is new or it has been changed since the last generation.
	<p>A TransformableItem. Existing notations such as those from UML and BPMN can also be used instead of the one on the left. Each TransformableItem should be labelled to distinguish it from others (e.g. of the same type), with this format: <i>{instanceIds}:{TransformableItemType}</i></p> <ul style="list-style-type: none"> • {TransformableItemType} denotes the type of TransformableItem, its semantics equivalent to the notion of “class” in object-oriented concepts. • {instanceIds} specifies a set of identifiers that uniquely identify particular TransformableItems. <p>The following examples illustrate these fields:</p> <p>[a, b, c, d]:WebCrawler - a set of web crawler instances “a”, “b”, “c” and “d”</p> <p>wc[1..1]:WebCrawler or wc1:WebCrawler - a particular web crawler instance “wc1”</p> <p>wc[2..5]:WebCrawler - a set of web crawler instances numbered consecutively, which are “wc2”, “wc3”, “wc4” and “wc5”</p>

Figure Appendix C.5 shows three successive generations of an application

demonstrating use of these notations. The application is scattered in two zones: “internal” and “external”. Its components are drawn with UML’s component instance notation. In the second generation, a transformation agent “T” is added to the internal zone to handle modifications within that zone. A new version of the online bank component (i.e. “b2:OnlineBank_V2”) is also added to this zone to take over the bill payment functionality of version V1 of the online bank component (i.e. “bank:OnlineBank_V1”) (note the “delegated” servicing policy on the “payBill” function). However, access to its “checkBalance” function is not yet available (i.e. labelled with the “blocked and queued” servicing policy). In the third generation, version V1 of the online bank component is superseded by version V2, and hence version V1 is no longer accessible (i.e. labelled with the “illegal” servicing policy).



source: developed for this research





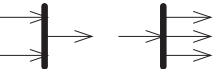


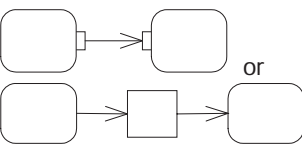
Figure Appendix C.5 Example use of Structural Configuration - Notational Extensions

C.2.2.9 Transformation Diagram and Notation

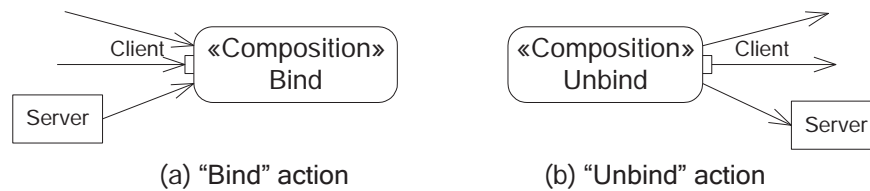
Description: A Transformation Diagram specifies the transformation actions in a transformation and the order of precedence of their execution. Several concepts and notations are adopted from UML’s behavioural modelling notations (OMG 2010b) in a transformation diagram (see Table Appendix C.17). An *UML Action* is a fundamental unit of behaviour specification and is used to represent a transformation action in a transformation. Each transformation action is labelled with the action type enclosed by

the guillemot characters «» and a short name of the behaviour for that particular type. UML's *Initial* and *Final Nodes* designate the entry and exiting points for a transformation model. UML *Control* and *Object Flows* are used to represent control and item-under-transformation (IUT) flows from one transformation action to another. A control flow in a transformation depicts the execution of an action being terminated and passed on to the next action indicated by the flow. An IUT flow represents a flow of an IUT from an output to an input action. An IUT can be a TransformableItem, a Resource, or a Zone.

Table Appendix C.17 Notations for Transformation Diagram

Continuum Notation	Continuum Semantics	Equivalent UML Semantics
	Transformation action (node). Default types are: «composition» - modification to a composition «resource» - (re)configuration to a resource «zone» - modification to a zone (e.g. resource assignment) «state» - setup of a TransformableItem's state «servicingPolicy» - setup of a TransformableItem's ServicingPolicy «custom» - any other action handled by foreign means (e.g. a tool) which is not modelled by Continuum	Action
	Start (node) of transformation	Initial node
	End (node) of transformation	Final node
	Generic flow (arc)	Flow
	Synchronisation for flow split and merge	Fork and join
	Item under transformation (IUT), including TransformableItem, Resource and Zone	Object node
	Control flow between actions	Control flow
	IUT flow between actions	Object flow

As per UML semantics, a transformation action starts execution when its incoming control(s) and IUT(s) are available to it as inputs (Bock 2003). Likewise, as a transformation action terminates, control(s) and IUT(s) are emitted from the action as outputs (Bock 2003). Figure Appendix C.6(a) denotes a Bind action being executed as soon as the Client, the Server and a designated control signal are fed to the action. Figure Appendix C.6(b) depicts the opposite action breaking the binding between the Client and the Server: both these components and a control signal emitted after execution. See Appendix C.3.2.7 for additional examples.



source: developed for this research

Figure Appendix C.6 Example transformation actions

Note that the actual order of execution of the transformation actions in a transformation model is decided by the transformation's implementation, which may choose a single-threaded execution to step through the transformation actions in the model one at a time, or run several threads to perform multiple actions in parallel where possible to maximise concurrency.

C.2.2.10 Transformation Orchestration Diagram and Notation

Description: A Transformation Orchestration Diagram specifies the steps, in terms of transformations and their sequence, to progress an application from one generation to its succeeding one. A subset of the concepts and their notations, as listed in Table Appendix C.18, are customised from the Business Process Modelling Notation (BPMN) for representing process flows (OMG 2009) to support the modelling of transformation orchestration diagrammatically. A BPMN *task* is an atomic unit of execution which represents a transformation. A BPMN *sub-process* is a hierarchical grouping of tasks and smaller sub-processes, and maps to a set of transformations to be performed during a transitional period. The BPMN *sequence flows* and *gateways* provide a means of sequencing, joining and merging transformations in an execution flow. The BPMN *start event*, *intermediate event*, *end event* and *error event* concepts identify triggers and results within the execution flow of the set of transformations. Finally, a BPMN *swimlane* identifies a set of transformations assigned to a transformation agent by enclosing them within the swimlane.

It should be noted that although UML Activity Diagrams offer features similar to BPMN for modelling transformation orchestration, BPMN is more appropriate since it is more focused and intuitive than the UML approach when representing situations where exceptions arise (White 2004).

Table Appendix C.18 Notations for Transformation Orchestration Diagram

<i>Continuum Notation</i>	<i>Continuum Semantics</i>	<i>BPMN Semantics</i>
○	The start point representing the as-is generation before a set of transformations are executed	Start event







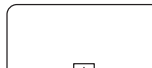








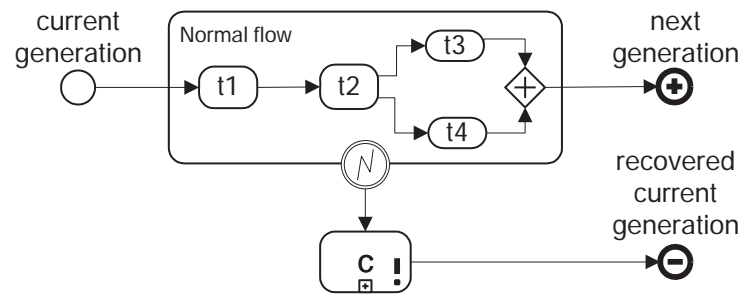
<i>Continuum Notation</i>	<i>Continuum Semantics</i>	<i>BPMN Semantics</i>
	An intermediate point representing a transformation agent is passed control of an execution flow by another agent	Intermediate event
	An end point representing the <i>expected</i> to-be generation after a transitional period	Generic end event with additional semantics
	An end point representing a <i>resolved</i> generation (e.g. because of errors) after a transitional period	Ditto
	A special resolved generation representing the <i>recovered</i> as-is generation (e.g. because of errors) after a transitional period	Ditto
	Transformation	Task
	A transformation to be implemented/handled by foreign means (e.g. using a particular third party tool)	Task
	A group of transformations	Collapsed and expanded sub-process
	A marker for a transformation to indicate that it intends to resolve a transformation exception	Including but not equivalent to compensation
	Execution flow of transformations	Sequence flow
	Synchronisation point for branching or merging alternative transformation execution flows	Exclusive gateway
	Synchronisation point for forking and joining transformations executed in parallel	Parallel gateway
	Decision point for the outgoing transformation execution flow based on the first incoming transformation execution flow from among several transformations executed in parallel	Complex gateway
	Enclosure for transformations designating the responsibility of a transformation agent	Swimlane
	Transformation exception	Error event
	Timer to set maximum duration for a transformation (or a group of)	Intermediate timer

Figure Appendix C.7 exemplifies the notations using an orchestration of transformations. A normal execution would involve four transformations, “t1”, “t2”, “t3” and “t4”, with the last two being executed in parallel. If an exception fires from any of these transformations, the transformation exception resolution “c” will attempt to revert an application to its original generation.



source: developed for this research

Figure Appendix C.7 Example transformation orchestration diagram

C.2.2.11 Zone Change Document

Description: The Zone Change Document captures changes to zones which host an application and its TransformableItems. These changes are required to synchronise the configurations of zones with the changes to the application and its TransformableItems as they evolve. A Zone Change Document records:

- addition of new zones;
- deletion of existing (empty) zones;
- addition/removal of TransformableItems to/from a zone;
- relocation of TransformableItems from one zone to another zone (considered removal followed by addition); and/or
- changes to the resources offered by zones

Table Appendix C.19 is an example of documenting the kinds of zone changes above. In general, the following notations are used for convenience:

- “+x”: either addition of transformable item “x” to the zone identified by the column heading, or introduction of new zone “x”.
- “-x”: either removal of transformable item “x” from the zone identified by the column heading, or deletion of existing zone “x”.
- “^x”: increase in resource “x” in the zone identified by the column heading.
- “Vx”: decrease in resource “x” in the zone identified by the column heading.

Table Appendix C.19 Example zone change document

Responsible Transformation(s)	+Security Zone	Web Zone	Repository Zone
Security Zone addition	^24TB shared network drive space		
Authorisation module addition	+sec:AuthModule_v1		
Web UI upgrade		+ui:WebUI_v2	
Data store upgrade			+ds:DataStore_v2

C.3 WORK UNIT FRAGMENTS

Work unit fragments concern what must be done and how to achieve given purposes during development (ISO/IEC 2007). A work unit fragment can be a *process* to follow, a *task* to complete (the “what”) in a process, or a *technique* to use (i.e. the “how”) in a task (ISO/IEC 2007). Continuum’s work unit fragments encompass all these types.

SEMDM characterises a work unit fragment with a field called the “minimum capability level” at which a project situation is expected to attain before the fragment is used. In Continuum, the minimum capability level is not individually specified for each work unit fragment. Rather, *all* Continuum work unit fragments have the minimum capability level set to two, meaning that they can only be used in a project situation that is at least “planned and tracked while it is performed; work products conform to specified standards and requirements” (ISO/IEC 2007).

C.3.1 Process and Task Fragments

Continuum’s process fragments are instances of SEMDM Process/*Kind. Each fragment is a large-grained work unit tackling a particular area of dynamic evolution during analysis and/or design.

Continuum’s task fragments are instances of SEMDM Task/*Kind, which supplement process fragments as to *what* objectives are to accomplish when those process fragments are performed (ISO/IEC 2007). Each of Continuum’s task fragments is specified with a name (as in its section heading), a purpose and a description, in accordance with SEMDM. In addition, the following fields are documented to link and relate to other fragments in Continuum:

- “Actions Performed” lists what action(s) (i.e. “Create”, “Read” and/or “Modify”) to perform on what kinds of work products, should this task be executed. This field is effectively the placeholder for specifying the connection between task fragments and work product fragments with respect to SEMDM. For convenience, however, action specifications in Continuum are derived from this field rather than explicitly specified. For instance, an action titled “‘Create’ State Map” means the associated task “creates” a work product of the type “State Map”, and instantiates an action clabject (Gonzalez-Perez & Henderson-Sellers 2005; ISO/IEC 2007) called “CreateStateMapAction” which links the task to the product “State Map”.

- “Technique(s) Used” lists the technique(s) applied in the performance of a task. Continuum uses a “deontic value” (Graham et al. 1997) to grade the degree of suitability of a technique to a task: “mandatory” (i.e. must be used in a task), “recommended” (i.e. recommended for use in a task), or “optional” (i.e. optionally used in a task).
- “Agents” lists the producers responsible for carrying out the associated task. This field is effectively the placeholder for specifying WorkPerformance clabjects (ISO/IEC 2007), each of which a connection between task and producer fragments, with respect to SEMDM. For convenience, however, the WorkPerformance specification is derived from this field rather than explicitly specified. For instance, when “Analyst” is assigned to this field of a task fragment called “ElicitRequirements”, a WorkPerformance clabject called “AnalystElicitRequirementsWorkPerformance” is instantiated, linking the producer fragment “Analyst” to the task fragment “ElicitRequirements”.

Process fragments are described next in alphabetical order. For reasons of readability and comprehensibility, task fragments intended to be used within a particular process fragment are described within the section for the process fragment. The sequence in which task fragment names are listed in each process fragment section suggests an order in which the task fragments are performed in the process fragment.

C.3.1.1 Application Lifecycle Analysis

Purpose: To define the roadmap consisting of new generations and transitional periods in the lifecycle of an application that will accommodate the changes elicited from requirements analysis.

Application Lifecycle Analysis extends the lifecycle of an application with new generations and transitional periods, with each period labelled with associated change cases identified for and implemented in the period. This process is not intended as a replacement of but rather expected to be used alongside a conventional *requirements analysis* process for eliciting requirements from various sources. It may also be beneficial to collaborate this process with *programme management* which develops a coherent framework to organise project-based change activities (Pellegrinelli 1997)²⁶. This process performs the following sequence of tasks:

²⁶ Programme management is out-of-scope for this research (see Section 1.5).

1. Identify As-Is Runtime Structure (Appendix C.3.1.1.1);
2. Derive Change Cases (Appendix C.3.1.1.2); and
3. Extend Application Lifecycle (Appendix C.3.1.1.3)

C.3.1.1.1 Identify As-Is Runtime Structure

Purpose: To produce an as-is model representing the runtime structure of an application as it appears in its zone(s).

Description: Before evolving a live application, a clear and accurate representation of the runtime structure of application *as-is* must be known in order to define dynamic changes which are specified with respect to the structure. The runtime structure is most likely to have been documented in the last project which developed the current version of the application. If so, this task reviews and updates relevant work products for the runtime structure. Otherwise, the current runtime structure is uncovered from existing work products, such as via reflection (Maes 1987). In particular, the following information is examined to identify the as-is runtime structure of an application:

- the static architectural design of the application;
- if available, the most recent version of the runtime structure of the application;
- the domain and general knowledge about the problem that the application solves and personal experience with the application (Chikofsky & Cross 1990); and
- a recent history of modifications made to the application.

The task produces the following information about the as-is runtime structure. The information is principally captured with an appropriate and existing modelling language (e.g. UML), and if necessary with additional notations from Continuum (i.e. Structural Configuration - Notational Extensions, Appendix C.2.2.8) not addressed by the language:

- transformable items in the application and their interactions (Ding & Medvidovic 2001);
- the versions of the transformable items; and
- zones in which the transformable items reside.

Technique(s) used: Runtime Structure Recovery (optional)

Actions performed: Create, Read or Modify diagrams for the as-is runtime structure.

Agents: Dynamic Evolution Designer, Runtime Application Discovery Tool

C.3.1.1.2 Derive Change Cases

Purpose: To derive and document purposeful change cases to the application from approved requirements and change requests.

Description: Requirements and change requests are elicited from people such as end-users, system administrators and the application development team on what an application is intended to become and how it will behave, and are almost invariably expressed without regard to the dynamic change aspects of a running application in mind.

This task takes the responsibility of converting new, revised and to-be-deleted requirements and change requests, to change cases that describe dynamic changes relative to the as-is generation of an application and/or its zone(s). The change cases are also described in a way to facilitate further understanding of the changes required for an application at runtime and the design of the changes. For instance, a change request “Improve the search performance of a catalogue application” can be translated to a change case “Replace the search algorithm in the search engine with a new search algorithm”.

Note that requirement elicitation is outside the scope of Continuum and is best accomplished with existing requirement engineering techniques (see Nuseibeh and Easterbrook (2000) for comprehensive references).

Technique(s) used: Change Case Modelling (mandatory)

Actions performed: *Create or Modify* Dynamic Application Change Document. *Read* existing relevant requirement work products.

Agents: Dynamic Evolution Analyst

C.3.1.1.3 Extend Application Lifecycle

Purpose: To extend an application lifecycle by adding new generations to it to progressively accommodate dynamic changes for the application.

Description: The application lifecycle identifies generations that an application advances dynamically over time. A transitional period exists between two successive generations and changes are incorporated into the application during that period. When determining the number of generations required for a set of change cases, consider how many transitional periods are required to feasibly realise those change cases in the application, and the characteristics of the change cases. To accommodate a small set of simple change cases, one transitional period might suffice. For a large set of (complex) change cases, not only may more transitional periods become desirable, but

also the sequence in which the change cases are implemented may become important. The following outcomes are produced with this task:

- change cases partitioned into groups;
- the order in which the groups of change cases, if applicable, are realised (to handle situations in which certain change cases must be realised before a peculiar change case is realised);
- a transitional period assigned to each group of change cases for their realisation; and
- an updated application lifecycle with the new transitional periods and associated generations.

The application lifecycle is a living entity; it will be continually extended and updated over time to evolve the application dynamically until its retirement.

Technique(s) used: Change Case Partitioning and Ordering (recommended)

Actions performed: Read Dynamic Application Change Document. Read and Modify Application Lifecycle Diagram.

Agents: Dynamic Evolution Analyst

C.3.1.2 Dynamic Evolution Quality Management

Purpose: To improve the quality analysis and design aspects of dynamic evolution for an application.

Dynamic Evolution Quality Management is performed in an iterative manner to continuously and progressively drive quality improvement in dynamic evolution related analysis and design work products. This process performs the following sequence of tasks:

1. Define Dynamic Evolution Quality Needs (Appendix C.3.1.2.1);
2. Assess Dynamic Evolution Quality of the dynamic evolution analysis and design artefacts (Appendix C.3.1.2.2);
3. Analyse Dynamic Evolution Quality Problems uncovered in task above (Appendix C.3.1.2.3); and
4. Improve Dynamic Evolution Quality to rectify problems (Appendix C.3.1.2.4).

Note that during analysis, the first Task “Define Dynamic Evolution Quality Needs” sets the main focus for dynamic evolution quality. As development advances to design, dynamic evolution quality related work products are incrementally developed and

refined and the development effort shifts towards the remaining three tasks to check and improve their quality. This process can be iterated a number of times, each of which focusing on improving different quality needs (e.g. security and safety).

Continuum supports a portfolio of dynamic evolution quality factors, organised in four categories as shown in Table Appendix C.20. Their definitions are attuned to dynamic evolution:

Table Appendix C.20 Summary of quality factors supported by Continuum

<i>Quality Category</i>	<i>Description</i>	<i>Quality Factors</i>
Soundness of change	The extent of dynamic changes and associated transformations being free from defects which make the application harder to evolve.	Completeness, Consistency, Correctness
Infusibility of change	The ease with which a change can be accommodated (i.e. infused) into an application at runtime.	Efficiency, Locality, Maintainability, Transparency
Changeability of application	The ease with which a distributed application can accommodate dynamic changes.	Autonomy, Coordination, Extensibility, Loose Coupling, Separation of Concerns
Robustness of application	The degree to which an application can withstand or reject invalid dynamic changes.	Fault Tolerance, Recoverability, Reliability, Safety, Security

C.3.1.2.1 Define Dynamic Evolution Quality Needs

Purpose: To define the dynamic evolution quality needs for an application.

Description: Re-evaluating the quality needs specific to dynamic evolution is essential because of issues such as the changing nature of the business, the application development team's feedback, the evolving end user needs and the changing context in which the application is used. For instance, efficiency might be regarded as important to computation intensive applications, whereas recoverability might be seen as necessary in mission-critical applications. The quality needs are determined by evaluating quality factors that are appropriate for dynamic evolution to the application. Evaluation results are documented in a Dynamic Evolution Quality Profile Report. Continuum supports a set of quality factors as summarised in Table Appendix C.21.

Table Appendix C.21 Summary of quality factors supported by Continuum

<i>Quality Category</i>	<i>Quality Factor</i>	<i>Description</i>
<i>Soundness of Change</i>	Completeness	The extent to which a transformation and its change(s) do not cause missing, broken or illegal transformable items, bindings or functions in an updated application
	Consistency	The extent to which a transformation and its change(s) do not result in an error state for an application but rather it continues processing as normal
	Correctness	The degree to which a transformation correctly and effectively applies the associated and feasible change(s) to an application

<i>Quality Category</i>	<i>Quality Factor</i>	<i>Description</i>
<i>Infusibility of Change</i>	Locality	The extent to which a change is explicitly confined within a logical boundary of an application to reduce the effort of managing and implementing change and its transformation
	Maintainability	The degree to which a change does not make an application more difficult and costly to modify, nor harder to test
	Transparency	The extent to which transformations, when being executed, are not noticeable to entities internal and external to a distributed application
<i>Changeability of Application</i>	Autonomy	The ability of transformable items to have control and governance of their own processing
	Coordination	The ability of different nodes of a distributed application to interact to deploy and install runtime changes to remote nodes or logical partitions to achieve the overall change effect
	Extensibility	The ability of application and its transformable items to be extended/upgraded with new functionality
	Loose Coupling	The degree of independence between transformable items and the freedom of having their own lifecycles and runtime environments
	Separation of Concerns	The extent to which functionality is separated from transformation, communication and security concerns, and transformable items' realisations are separated from their specifications and their clients' realisations
<i>Robustness of Application</i>	Fault Tolerance	The extent to which an application can tolerate faulty transformable items and contain their faults
	Recoverability	The ability of an application to be restored to a state in which it can continue to perform its functionality, after a failure caused by a transformation and/or its dynamic change(s)
	Reliability	The ability of an application to keep its intended functions from being compromised by ongoing changes and transformations such that it behaves in an unexpected manner
	Safety	The ability of a distributed application and its transformable items to continue operating in a safe manner, such as not leading to harm to people, properties and/or the environment, during and after a transformation
	Security	The degree to which a distributed application is protected from security threats as it undergoes dynamic evolution, and the ability of the security model of an application to adapt to the changing needs of the application

Technique(s) used: none

Actions performed: Create, or Read and Modify Dynamic Evolution Quality Profile Report.

Agents: Dynamic Evolution Analyst

C.3.1.2.2 Assess Dynamic Evolution Quality

Purpose: To assess the (current state of) quality for analysis and design work products for dynamic evolution.

Description: This assessment serves to assure that the necessary implementation for the quality factors of interest is in place, which means dynamic evolution related

analysis and design work products attain a certain level of quality anticipated for an application. Additionally, this assessment identifies quality related defects and issues in these work products. It is recommended that an investigation on identifying solutions for or resolving the found defects and issues is not carried out during this task to ensure that this task stays focused on finding defects and issues.

Typical approaches for assessment include inspection and testing²⁷. Since Continuum centres on analysis and design activities performed in software development, inspection appears to be an appropriate choice for Continuum. In this regard, this task reuses OPF's "inspection" technique²⁸ for evaluating quality (Appendix C.3.2.17.7), and supplements this technique with its own inspection checklist suitable for dynamic evolution quality (Appendix C.2.2.3).

Technique(s) used: Inspection (mandatory)

Actions performed: *Read* and *Modify* Dynamic Evolution Quality Inspection Report. *Read* Dynamic Evolution Quality Profile Report. *Read* analysis and design related work products for dynamic evolution.

Agents: Dynamic Evolution Analyst, Dynamic Evolution Designer

C.3.1.2.3 Analyse Dynamic Evolution Quality Problems

Purpose: To analyse the quality problems for analysis and design work products for dynamic evolution.

Description: With the defects and issues reported from prior assessment on quality in dynamic evolution related work products, this task investigates and establishes the root causes for the defects and issues. For instance, a transformation's failure to update an existing transformable item could be traced to incorrect steps in the transformation's design.

Technique(s) used: Root Cause Analysis.

²⁷ Testing is a complex problem which is not part of Continuum. However, studies regard runtime testing as especially important after changes have been applied to an application (e.g. Bennett & Rajlich 2000) as it is not always possible to verify the changes statically in a laboratory setting at design time (Grundy et al. 2005). Sparling (2000) distinguishes the testing of individual parts from the testing of the composite application. Thus, testing is conducted separately before and after changes are made to an application (Bennett & Rajlich 2000; Oreizy et al. 1998), and separately before and after parts are deployed into an application (Grundy et al. 2005).

²⁸ Although RUP has a number of inspection oriented tasks ("review the design" and "review the architecture" etc.). They target particular work products. Instead, OPF's "inspection" is adopted in Continuum since OPF's is generic and customisable to different kinds of work products.

Actions performed: *Read* Dynamic Evolution Quality Inspection Report. *Create*, or *Read* and *Modify* Dynamic Evolution Quality Problem Analysis Report.

Agents: Dynamic Evolution Analyst, Dynamic Evolution Designer

C.3.1.2.4 Improve Dynamic Evolution Quality

Purpose: To revise analysis and design work products for dynamic evolution to further improve their quality.

Description: This task ensues once the quality defects and issues for dynamic evolution are identified, and their root causes are established. Solutions to rectify the defects and issues found are developed and implemented in the relevant work products. A solution may be a simple correction of a design defect (e.g. filling a missing transformable item in a composition) whilst others may require specialised techniques for given purposes (e.g. improving loose coupling). Continuum offers a catalogue of *recommended* tasks and techniques to further improve particular aspects of individual quality factors some of which directly tackling particular types of quality problems, as shown in Table Appendix C.22. The table also summarises in which particular aspects of development (i.e. “A” for analysis, “D” for design) the tasks/techniques are used.

Table Appendix C.22 Work units for addressing particular aspects of quality factors

Quality Factor	Recommended Task/Technique	Purpose	Used During
<i>Soundness of Change</i>			
Consistency	Dynamic Change Impact Analysis	Identify transformable items affected by proposed changes.	D
	Start-up State Configuration	Declare resuming states to avoid progression towards error states after a transformation.	D
	Transformable Item Regression Testing	Identify and detect violations of invariants.	D
	Resource Profile Modelling	Capture adequate resources and support required for new and replacement transformable items.	D
Correctness	Change Case Modelling	Capture proposed changes explicitly.	A, D
	Define Servicing Policies	Restrict behaviour during transformations.	D
<i>Infusibility of Change</i>			
Locality	Identify Changes to Zones	Provide guidance on accommodating transformable items in zones.	D
	Dynamic Change Localisation	Confine changes to within zones.	D
Maintainability	Identify Changes to Zones	Keep the information about zones up-to-date.	D
	Identify New and Replacement Transformable Items	Keep the information about transformable items in an application up-to-date.	D

<i>Quality Factor</i>	<i>Recommended Task/Technique</i>	<i>Purpose</i>	<i>Used During</i>
	Extend Application Lifecycle	Keep the information about an application lifecycle up-to-date.	A
	Identify As-Is Runtime Structure	Keep the information about runtime structure of an application up-to-date.	A
	Define To-Be Runtime Structure		D
	Testability Analysis and Improvement	Improve testability of an application and its transformable items.	D
Transparency	Transformation Mining	Hide the effects of transformations from end users.	D
	Identify Transformations	Abstract transformations away from the business logic of an application to hide them from its programmers.	D
	Dynamic Change Localisation	Hide transformations from transformable items of an application unaffected by the transformations.	D
<i>Changeability of Application</i>			
Autonomy	Transformable Item Autonomy	Improve self-control and self-governance over transformable items.	D
Coordination	Define Transformation Orchestration	Organise transformation agents to facilitate the orchestration of transformations among multiple zones during a transitional period.	D
	Secure and Reliable Transformation Agent Coordination	Address network unreliability when transformation agents coordinate with one another during a transitional period.	D
Extensibility	Dynamic Wrapper	Add new functionality to existing compositions.	D
	Dynamic Change Localisation	Use zones to confine the scope of changes.	D
	Dynamic Variation Management	Add variation points to an architecture to plug alternative or additional transformable items to it.	D
	Identify Changes to Zones	Support multiple versions of transformable items to co-exist.	D
	Dynamic Wrapper	Add new functionality to existing transformable items.	D
Loose Coupling	Loose Coupling	Reduce coupling among transformable items.	D
Separation of Concerns	Transformable Item Mediation and Channelling	Abstract communication concerns from functionality.	D
<i>Robustness of Application</i>			
Fault Tolerance	Dynamic Wrapper	Protect an application against potential faults from new and replacement transformable items.	D
	Dynamic Wrapper	Establish barriers to contain faults from new and replacement transformable items.	D
	Dynamic Change Localisation	Confine changes to within zones to isolate faults propagating from one zone to another.	D
Recoverability	Transformation Exception Management	Manage potential exceptions raised during the execution of a transformation.	D
	Recovery Blocks	Recover an application from errors caused by new and replacement transformable items after a transformation.	D

<i>Quality Factor</i>	<i>Recommended Task/Technique</i>	<i>Purpose</i>	<i>Used During</i>
Reliability	Dynamic Wrapper	Limit the behaviour of new and replacement transformable items in case they behave unexpectedly, compromising an application.	D
	Transformable Item Regression Testing	Check whether or not new and replacement transformable items behave as expected in an application.	D
Safety	Dynamic Evolution Safety Risk Management	Identify and mitigate safety risks associated with transformations and dynamic changes.	D
Security	The security part of Secure and Reliable Transformation Agent Coordination	Provide security for transformation agent control and coordination.	D
	Dynamic Wrapper	Protect the security of an application from being compromised by new and replacement transformable items and vice versa.	D
		Provide access control, intrusion management and data input/output protection for transformable items.	D
	Dynamic Security Policy and Enforcement Management	Manage security policy and enforcement changes in response to dynamic changes in an application.	D

Note(s):

1. Each “purpose” specifies a criterion to be verified against a particular quality factor when filling out the dynamic quality inspection report (Table Appendix C.10, Appendix C.2.2.3).
2. Abbreviations in the “used during” column: “A” and “D” stand for “analysis” and “design” respectively, indicating the kinds of producers (i.e. “A” for Dynamic Evolution Analyst, “D” for Dynamic Evolution Designer) to carry out the “recommended task/technique”.

Technique(s) used: see Table Appendix C.22

Actions performed: *Read* Dynamic Evolution Quality Inspection Report. *Read* Dynamic Evolution Quality Problem Analysis Report. *Read* and *Modify* the design of runtime structures (e.g. using Structural Configuration - Notational Extensions). *Read* and *Modify* relevant dynamic evolution work products (e.g. Dynamic Application Change Document, Transformation Orchestration Diagram, Transformation Diagram).

Agents: Dynamic Evolution Analyst, Dynamic Evolution Designer

C.3.1.3 Transformation Agent Design

Purpose: To define transformation agents to progress a particular generation (i.e. as-is) of an application to the next generation (i.e. to-be) during a transitional period.

Transformation Agent Design defines the responsibilities of transformation agents in terms of how many transformation agents are required to perform a set of transformations, which transformation(s) each agent perform, and an order in which the transformations are performed by the agents. This process performs the following two

tasks:

1. Identify Transformation Agents (Appendix C.3.1.3.1); and
2. Define Transformation Orchestration (Appendix C.3.1.3.2).

C.3.1.3.1 Identify Transformation Agents

Purpose: To identify transformation agents to be responsible for executing a set of transformations during a transitional period, in order to advance an application from one generation to the next.

Description: In a distributed environment, dynamic changes may result in transformations to several parts of an application over a network. To manage this complexity, a number of transformation agents may sometimes be used. This task takes the following information to determine the transformation agents required to accomplish the dynamic changes via respective transformations:

- the intended transformations during a transitional period;
- the structures of the application before and after the intended transformations in terms of its transformable items and their locations (i.e. zones); and
- the transformable items that will be affected by the intended transformations and the zones they reside.

This task also establishes the zones that the transformation agents are stationed and their relationships to coordinate and execute the set of transformations together. Note that in a self-modifying application, its transformable items can also play the role of a transformation agent.

Technique(s) used: Transformation Agent Disposition (mandatory)

Actions performed: *Read* Dynamic Application Change Document. *Read* diagrams for the to-be runtime structure. *Read* and *Modify* diagrams for the as-is runtime structure.

Agents: Dynamic Evolution Designer

C.3.1.3.2 Define Transformation Orchestration

Purpose: To arrange transformations to be carried out during a transitional period and assign them to appropriate transformation agents.

Description: Once the transformation agents are positioned in appropriate zones in an operating environment (see Task Identify Transformation Agents, Appendix C.3.1.3.1), these agents can be assigned transformations in their respective zones, and an order

in which they carry out the transformations are determined in this task.

Technique(s) used: Transformation Orchestration and Agent Coordination (mandatory)

Actions performed: *Read* Dynamic Application Change Document. *Read* diagrams for the as-is runtime structure. *Create* Transformation Orchestration Diagram.

Agents: Dynamic Evolution Designer

C.3.1.4 Transformation Design

Purpose: To define the detailed design for a transformation to be performed during a transitional period.

This process produces the design for a transformation covering low-level modification steps for an application; changes to the zone(s) hosting the application to make the zone(s) in sync with the changes made by the transformation; and how transformable items affected by the transformation will offer their functions or services during and after the transformation. This process performs the following sequence of tasks:

1. Identify New and Replacement Transformable Items (Appendix C.3.1.4.1);
2. Identify Changes to Zones (Appendix C.3.1.4.2);
3. Define Servicing Policies (Appendix C.3.1.4.3); and
4. Develop Transformation (Appendix C.3.1.4.4).

C.3.1.4.1 Identify New and Replacement Transformable Items

Purpose: To identify transformable items, including new and replacement ones, that will be added to an application after a transformation.

Description: As new and replacement transformable items are incorporated into an application, they will need to be registered so that they are known to exist and be used. Additionally, their resource profiles should be used to verify whether zones will offer adequate resources to support these items. Likewise, their performance profiles should be used to verify whether they can meet the performance needs of the application. At other times, when incorporating a transformable item into an application, it may be necessary to start the item from a desirable rather than a default and initial state. In summary, this task determines the new and replacement transformable items to be added to an application, their operational profiles and start-up state configurations.

Technique(s) used: Resource Profile Modelling (optional), Performance Profile Modelling (optional), Start-up State Configuration (optional).

Actions performed: *Read* the design of the current and new runtime structures. *Read*

Dynamic Application Change Document and Transformation Orchestration Diagram. *Create* State Map. *Modify* New and Replacement Transformable Item Catalogue.

Agents: Dynamic Evolution Designer

C.3.1.4.2 Identify Changes to Zones

Purpose: To identify changes to be made to zones hosting transformable items in an application, to keep them in sync with changes to the transformable items.

Description: The zones hosting various parts of an application may be impacted by the changes proposed for transformable items. Correspondingly, changes are also required for the zones to support the application. For instance, additional zones may be needed to host new items, and existing zones may require updates as items are relocated to different zones. The task identifies the following information about zone changes:

- changes to the distribution of zones as a result of zone addition and removal;
- changes to the distribution of transformable items in zones as a result of addition, removal, and relocation of transformable items from one zone to another; and
- changes to resources offered by zones.

Continuum defines the following zoning policies (Appendix C.2.1.21) as guides for specifying changes to zones. These policies are intended for supporting multiple versions of the same transformable item type in the same application:

- In each zone, all transformable items of the same type must be of the same version. An upgrade to a type in the zone will cause an upgrade to all transformable items of the same type in the zone. To host transformable items of different versions, define separate zones with each hosting only one version. This policy keeps transformable items of the same type in one zone to be consistent with one another (Evans & Dickman 1999).
- To accommodate transformable items of different versions in one application, additional zones are established if necessary with each zone hosting only one version of transformable items.

Technique(s) used: none

Actions performed: *Read* Transformation Orchestration Diagram. *Read* diagrams of the current and new runtime structures. *Read* Dynamic Application Change Document. *Create* or *Modify* Zone Change Document.

Agents: Dynamic Evolution Designer

C.3.1.4.3 Define Servicing Policies

Purpose: To define rules to regulate the services or functions provided by transformable items impacted by transformations.

Description: A transformable item may be unable to fully provide its services or functions if it is affected by transformations. Thus, it may be appropriate to define how the impacted transformable item should offer its functions. For instance, when the transformable item is being replaced, its functions could be demarcated as “blocked and queued” meaning that any request to them (e.g. function invocation) is put on a queue until its replacement transformable item has been installed and resumes the processing of requests to its functions after a transformation. For convenience, Continuum prescribes some basic servicing policies and notations (Appendix C.2.2.8) to regulate access to services affected by transformations.

Technique(s) used: none

Actions performed: *Read* and *Modify* the diagram for a runtime structure.

Agents: Dynamic Evolution Designer

C.3.1.4.4 Develop Transformation

Purpose: To define the design of a transformation as a number of transformation actions, representing atomic modification steps at a fine-grained level.

Description: A transformation is performed by a transformation agent to realise a dynamic change planned for the transformation. A transformation design consists of low-level modification steps called transformation actions (Appendix C.2.1.15). Examples include:

- state configuration of new and replacement transformable items;
- addition, removal, replacement, binding and unbinding of transformable items;
- resource (re)allocation for new and replacement transformations;
- zone configuration;
- relocation of transformable items to different zones; and
- configuration of servicing policies for transformable items.

Technique(s) used: Dynamic Transformable Item Adaptation (optional), Dynamic Transformable Item (Re)binding (optional), Dynamic Transformable Item Change (optional).

Actions performed: *Read* the design of the current and new runtime structures, Dynamic Application Change Document, New and Replacement Transformable Item Catalogue, State Map, Transformation Orchestration Diagram and Zone Change Document. *Create* Transformation Diagram.

Agents: Dynamic Evolution Designer

C.3.1.5 Transformation Identification

Purpose: To identify a set of transformations to execute during a transitional period, to progress an application from an as-is generation to a to-be generation that will have accommodated the changes proposed for the as-is generation.

Transformation Identification focuses on identifying *what* transformations are required to map an application to a to-be generation. It utilises the information gathered (i.e. as change cases) during the analysis of dynamic evolution and the design for the application. The *how* and *who* aspects - i.e. how transformations will do their job, and who will execute the transformations - are handled by other processes (Transformation Design (Appendix C.3.1.4) and Transformation Agent Design (Appendix C.3.1.3)). This process performs the following sequence of tasks:

- Define To-Be Runtime Structure (Appendix C.3.1.5.1);
- Refine Change Cases (Appendix C.3.1.5.2); and
- Identify Transformations (Appendix C.3.1.5.3).

C.3.1.5.1 Define To-Be Runtime Structure

Purpose: To define what the runtime structure of an application (i.e. the new or “to-be” generation) would become after the expected changes are realised in the application.

Description: The to-be runtime structure is what an application is expected to become after a set of change cases have been realised in the application. This task takes as inputs the current or “as-is” runtime structure of the application, the new architecture of the application developed from a conventional application design activity independent of dynamic evolution, and the dynamic changes proposed for the current structure, to derive the to-be runtime structure.

Technique(s) used: Dynamic Recomposition (optional), Dynamic Refactoring (optional), Dynamic Workflow Change (optional), Dynamic Variation Management (optional), Loose Coupling (optional)

Actions performed: *Read* Dynamic Application Change Document. *Read* diagram for

the current runtime structure. *Create* diagram for the to-be runtime structure.

Agents: Dynamic Evolution Designer

C.3.1.5.2 Refine Change Cases

Purpose: To update, refine and expand the set of change cases identified for an application since analysis.

Description: Not all change cases would normally be determined during analysis until the runtime structure of the to-be generation has been defined during design. This is because, until the to-be generation is defined, the differences between the as-is and to-be generations and hence the change cases cannot be clearly identified. Moreover, once the design of the to-be generation is complete, there may be new features in the to-be generation, requiring additional change cases to capture them since they could not be accounted for during analysis when the to-be generation had not been defined. In another situation, dynamic changes proposed to one part of an application (e.g. existing transformable items) may ripple through and therefore impact other parts of the application (Bohner 2002b). Additional change cases may thus be required to represent changes to other parts of the applications. This task refines and expands the set of change cases identified so far in preparation for subsequent transformation design tasks that follow.

Technique(s) used: Change Case Modelling (mandatory), Dynamic Change Impact Analysis (mandatory)

Actions performed: *Read* diagrams of the current and new runtime structures. *Read* and *Modify* Dynamic Application Change Document.

Agents: Dynamic Evolution Analyst, Dynamic Evolution Designer

C.3.1.5.3 Identify Transformations

Purpose: To identify the transformations to realise a set of change cases by progressing an application from the as-is into the to-be generation.

Description: This task identifies the differences between the as-is and the to-be generations of an application, reviews the change cases identified, and determines the transformations, representing modification steps at a coarse-grained level, to progress the application from the as-is into the to-be generation. For a group of simple change cases targeting the same transformable item, one transformation to realise these change cases may suffice. For complex change cases, however, the number of transformations required will depend on several factors including:

- *Recipients of change cases*

Change cases affecting or impacting several parts of an application (i.e. the recipients) may require more modification steps to realise them, suggesting more transformations;

- *Transformation latency*

A long running transformation may need to be broken down into smaller transformations, to reduce the length of time during which an application's services are affected by a transformation; and

- *Zones*

Change cases might impact transformable items in multiple zones. To manage this kind of complexity, it is recommended that each transformation should not span more than one zone.

Technique(s) used: Transformation Mining (recommended)

Actions performed: *Read* and *Modify* Dynamic Application Change Document. *Read* diagrams for current and new runtime structure.

Agents: Dynamic Evolution Designer

C.3.2 Technique Fragments

Continuum's technique fragments are instances of SEMDM's Technique/*Kind, supplementing Continuum's task fragments by specifying *how* the objectives of the associated tasks are to be achieved. They are described next in alphabetical order.

C.3.2.1 Change Case Modelling

Purpose: To express change cases for an application.

Description: Change cases can be used to analyse, estimate and plan the modifications proposed for an application to implement the changes at runtime.

A change case's purpose is expressed with a "gap operator" (Salinesi et al. 2004), a recipient of the change, and any additional information characterising the purpose of the change. A gap operator identifies a type of change that can be performed on the recipient in evolution. A subset of gap operators from Salinesi et al. (2004) that are

relevant to dynamic evolution²⁹ are used for expressing change cases. These operators are listed on the left of Table Appendix C.23. A recipient is the entity receptive of the proposed change. A recipient can be a transformable item, a contract between transformable items, the structure of an application, or a zone. The notion of a recipient further distinguishes change cases from requirements and change requests which tend to be specified with respect to an application as a whole.

Table Appendix C.23 Gap operators and templates for expressing change case purposes

<i>Gap Operator</i>	<i>Recommended Template(s)</i>	<i>Example(s)</i>
Add	Add {feature} to {recipient}.	Add data caching support to online bookshop data store.
Remove	Remove {feature} from {recipient}.	Remove old currency support from online bookshop user interface.
Move	Move {feature/recipient} from {recipient1} to {recipient2}.	Move authentication functionality from online bookshop user interface to dedicated corporate authentication system. Move authentication module from business logic zone to web zone.
Merge	Merge {feature1} in {recipient1} with {feature2} in {recipient2}.	Merge authentication function in user interface with authorisation function in user interface.
Split	Split {feature} in {recipient} to {sub-feature1} and {sub-subfeature2}.	Split business logic function into business transaction function and data logic function.
Replace	Replace {feature1} in {recipient} with {feature2}.	Replace old search function in online bookshop search engine with new search function.
Modify	Modify {recipient} {property name and value description}.	Modify user interface binding with internal authentication module to corporate authentication system

Table Appendix C.23 provides templates for expressing various change case purposes using gap operators, with examples illustrating their uses. In most templates, the notion of *feature* is used. A feature refers to a function or an observable property of a recipient, such as a new function to be offered by a recipient. A feature can be directly identified from requirements and change requests, both of which written with end users and the business in mind. A “new search function” is an example feature for an existing transformable item “X” - the recipient. An expression for the purpose of such a change case can thus be written as “Add [gap operator] new search function to X [recipient]”.

Apart from its purpose, a change case can be augmented with information such as a long description about the proposed change; the application generation it targets; the transitional period during which the change is to be realised; and a transformation

²⁹ For instance, since operator “rename” (Salinesi et al. 2004) does not involve in altering an application structure or its hosting zone(s), it is not considered for dynamic evolution.

agent that will be responsible for realising the change case. A change case can also be linked to the original use cases/requirements/change requests in which the specifications may need to be updated in line with the change proposed by the change case (Ecklund et al. 1996). See the semantics of change case (Appendix C.2.1.3) and the Dynamic Application Change Document fragment (Appendix C.2.2.2) for further details.

C.3.2.2 Change Case Partitioning and Ordering

Purpose: To divide a set of change cases into groups of logical and coherent change cases, and order the groups appropriately.

Description: Partitioning of change cases into groups and ordering of the groups aid in defining (the extension of) an application lifecycle to accommodate those change cases. Partitioning uses the following strategies:

- *High cohesion*

Change cases relating to one another are clustered into one group, to stay focused on a common goal. For instance, change cases targeting the same transformable item, or transformable items within the same zone may be grouped together. Change cases linked by an impact (Appendix C.2.1.5) can also form a group.

- *Loose coupling*

Change cases in one group should be as independent as possible from change cases in other groups. For example, two change cases describing modifications to different parts of an application may be assigned to different groups. As another example, performing the Technique Dynamic Change Impact Analysis on change cases in one group does not identify change cases in another group.

- *Ease of realisation*

Complex change cases should be partitioned into smaller groups, to reduce the complexity of realising them in an application. For instance, when change cases impact a large number of transformable items in the same zone, consider organising the change cases into smaller groups to reduce the scope.

The ordering of the change case groups is derived from the ordering of the change cases among the groups:

- *Precedence*: A change case may require certain change cases to be realised in an application before it can be realised in the same application. For instance, a client cannot be upgraded to use a new function of a server until the server has been upgraded to offer the function.
- *Priority*: Critical and important change cases have precedence over others and may require immediate transitional periods to roll them out, whereas other change cases are deferred to later transitional periods. For instance, when an application is found to have defects, a priority would be given to change cases for defect correction rather than change cases for adding new features to the application.
- *Business perspective*: Consider the time, cost and resource implications for implementing particular types of change cases (Arshad et al. 2007). Some change cases, for instance, may require significant costs which may not be met in the short term and will be deferred. It should be noted that Continuum does not cover the investigation of business aspects for evolution.

C.3.2.3 Dynamic Change Impact Analysis

Purpose: To determine the potential impacts caused by proposed change cases.

Description: Change impact analysis generally concerns the identification of “potential consequences of a change, or estimating what needs to be modified to accomplish a change” (Arnold & Bohner 1996). That is because a small change to one part of an application has a tendency to ripple through and affect other parts of the application (and others outside the application); additional changes to the affected parts are required to ensure that they stay consistent with the original change (Yau et al. 1993). In particular, *dynamic* change impact analysis focuses on analysing the impact of changes to a live application at runtime (Feng & Maletic 2006). The following steps are undertaken to identify impacts and new change cases to accommodate those impacts:

1. Identify entities thought to be impacted by change cases proposed for an application. Consider the following types of entities as part of identification:
 - a transformable item internal to the application;
 - an external application; and
 - a transformable item external to the application.

Table Appendix C.24 lists the areas of impact analysis and external techniques for identifying entities to be impacted by change cases. It is instructive to note that although some of these techniques are mainly used for analysis of static

changes, their underlying concepts and approaches are sufficiently generic to be applicable to dynamic changes.

Table Appendix C.24 Areas of dynamic change impact analysis

<i>Area</i>	<i>Description</i>	<i>Analysis Technique(s)</i>
Structure	A part (i.e. transformable item) that depends on some other parts bound/connected to it to offer its functionality may be affected by changes to the other parts (Bohner 2002a).	Bohner (2002a) Mao et al. (2007)
Semantics	The semantics (i.e. type) of a change to a part determines which other parts are affected by the change (Bohner 2002a).	Feng and Maletic (2006) Mao et al. (2007) Xiao et al. (2007)
Logical ripple effect	A change to one transformable item of an application may cause incompatibility of the item with other transformable items of the application, potentially leading the other items to err (Yau et al. 1993).	Yau et al. (1993)
Performance ripple effect	A change to one transformable item of an application may affect and/or degrade the performance of other transformable items (Yau et al. 1993).	Yau et al. (1993)
Version conflict	A changes may introduce version conflicts among various transformable items of an application (Lassing et al. 2003).	Bengtsson et al. (2004)
Contract [see note 1]	<p>A change to a contract may also require transformable items bound to the contract to change.</p> <p>A change to a transformable item bound to a contract may also require the contract and/or other transformable items bound to the same contract to change to keep them in sync with the change proposed for the transformable item.</p> <p>New contracts may also be put in place.</p> <p>Existing contracts may also be decommissioned.</p>	<p>(see note 2) When a change is proposed to a contract, its transformable items become potentially impacted entities.</p> <p>When a change is proposed to a transformable item, other peers in the same contract and the contract itself become potentially impacted entities.</p>

Notes:

1. A contract epitomises some kind of mutual agreement - such as in the areas of interfaces, interaction protocols and quality of service - among transformable items for them to collaborate.
2. The analysis technique for contracts was suggested by an expert during the review of respondents' suggestions for dynamic change requirements in a survey (cf. Table 5.5).
2. For each entity identified in the last step, assess the characteristics of the impact. See Appendix C.2.1.5 for information on characterising an impact.
3. Identify new change cases required to cover unforeseen changes for entities in the impact set which are not yet covered by the original change cases.

C.3.2.4 Dynamic Evolution Safety Risk Management

Purpose: To identify and mitigate safety risks associated with transformations and dynamic changes.

Description: In the distributed system terminology, safety is a property that “something bad does not happen” (Lamport 1977). Although the state of badness depends on the context and use of an application, safety is generically defined in terms of:

- *Mishaps* (a.k.a. an accident (Firesmith 2003)), which result in harm to an asset. An asset can be one of those in Table Appendix C.25.

Table Appendix C.25 Assets and Harms

Asset Type	Asset Example	Harm Example
Human	Operators, end users	Illness, injury or death to a person
External property (i.e. outside an application)	Personal, commercial, civic, physical properties (e.g. buildings), data, money	Corruption of data, loss of money
Internal property (i.e. parts of an application)	Data, hardware and software components	Damage to components of an application or a machine
Physical environment	Rivers, roads	Fire, flooding

source: Firesmith (2003) with additional examples

- *Hazards*, which are states of an application that would lead to a mishap (Leveson 1986).
- *Safety risks*, each a function of the probability of hazards leading to a mishap and the severity of the worst mishap (Leveson 1986).

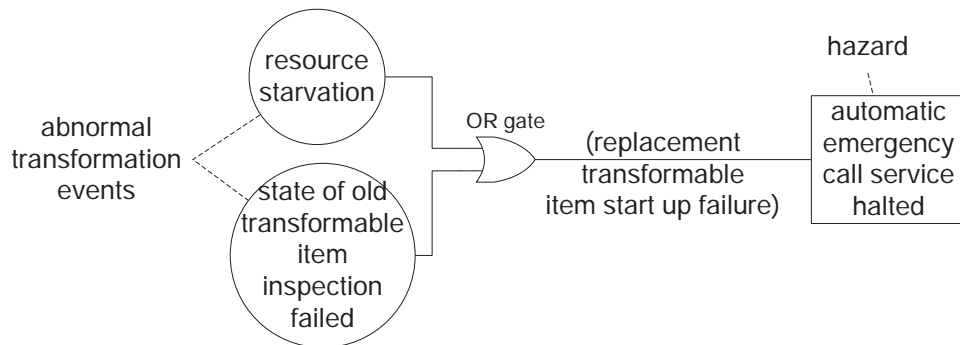
This technique is used in an overall process approach to account for dynamic evolution related safety issues. For instance, it enhances OPF’s “safety engineering” process (OPFRO 2009) by supplementing its “safety risk analysis” task with safety considerations specific to dynamic evolution. It takes a preventative approach in *anticipating* and *mitigating* safety risks by iterating the following steps. These steps are defined in accordance with the risk identification, analysis, evaluation and treatment aspects of the AS/NZS 4360 Standard for risk management (Standards Australia 2004):

1. Identify potential hazards caused by *abnormal transformation events*.

Abnormal transformation events that have the potential to trigger hazards during transformations are firstly identified. An example event is a transformation agent’s being unavailable when it is required to perform expected transformations (see taxonomy in Figure Appendix C.17). This kind of event can be identified with the Technique Transformation Exception Management (Appendix C.3.2.14). The identified events are then analysed to derive hazards. Continuum reuses *fault tree analysis* (FTA), based on combinatorial logic (e.g. AND and OR functions), for linking hazards and their causes (Andrews & Moss

2002). Each fault represents a (partial) cause for a hazard and corresponds to an abnormal transformation event.

Figure Appendix C.8 depicts an example fault tree for a new transformable item intended for replacing an existing one. The former will not work if it cannot acquire enough resource, *or* the state of the latter cannot be inspected and used to set the state of the former.



source: developed for this research

Figure Appendix C.8 Example fault tree for replacement transformable item related transformation events

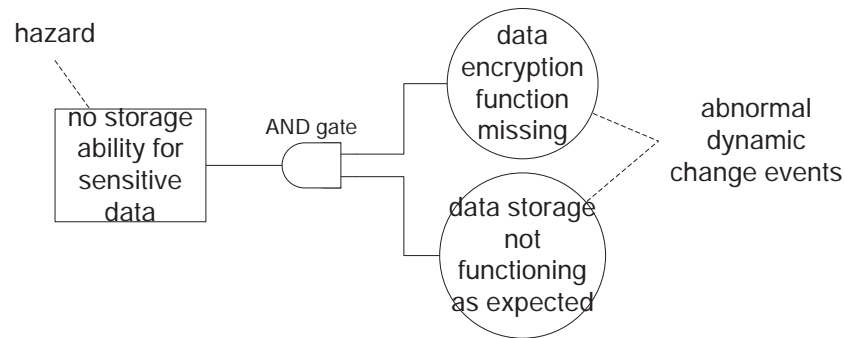
2. Identify potential hazards caused by *abnormal dynamic change events*.

This is identical to Step 1, except that it focuses on abnormal dynamic change events which may lead to hazards, *after* intended transformations have been executed to accommodate desirable dynamic changes to the application. Table Appendix C.26 suggests possible abnormal dynamic change events.

Table Appendix C.26 Abnormal dynamic change events

<i>Category</i>	<i>Example</i>
Functionality	<ul style="list-style-type: none"> • Functionality missing • Obsolete functionality still present • Transformable items/application not functioning as expected
Composition	<ul style="list-style-type: none"> • Transformable item/binding missing • Obsolete transformable item/binding still present

Figure Appendix C.9 depicts an example fault tree arising from abnormal dynamic change events. It indicates that when *all* the abnormal dynamic change events occur (i.e. all inputs to the AND gate are true), the hazard becomes consequential.

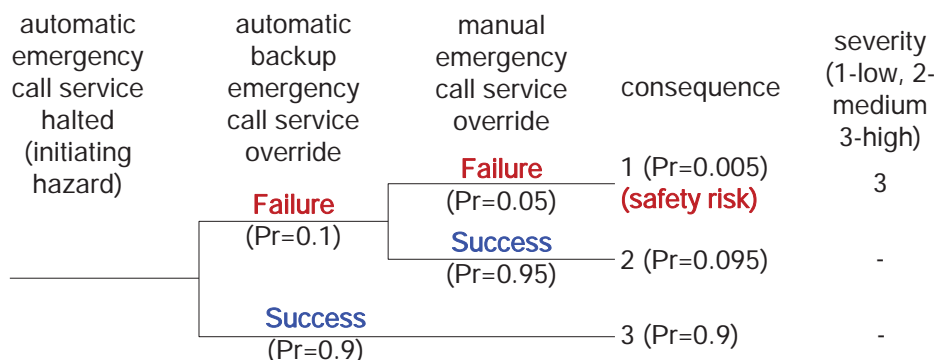


source: developed for this research

Figure Appendix C.9 Example fault tree for data related dynamic change events

3. Determine safety risks from hazards.

For each hazard identified in the previous steps, determine the safety risks associated with it, in terms of the probability and severity of the safety risks. Continuum reuses *event tree analysis* (ETA) for identifying the of those hazardous events and their probabilities (Andrews & Moss 2002). Figure Appendix C.10 illustrates such an event tree resulted from analysing the example hazard in Figure Appendix C.8 - “automatic emergency call service halted”. To cope with this hazard, the application cuts over to an in-built “automatic backup emergency call service”, to override the original emergency call service, with two possible consequences: either the backup service fails (probability $Pr=0.1$) or succeeds ($Pr=0.9$). To deal with the former, the application cuts over to the “manual emergency call service override” option with the probability of success being 0.95. Hence, the probability of the safety risk, which is the condition that both override mechanisms fail, is 0.005. The severity of the safety risk is then judged afterwards by assigning a suitable score to it.



source: developed for this research

Figure Appendix C.10 Example safety risk derived from hazard

4. Evaluate and treat safety risks.

Determine which of the safety risks identified from the previous steps are unacceptable to the business. Accordingly, hazards that lead to the unacceptable safety risks are dealt with by incorporating safety countermeasures in the design for dynamic evolution. Barbacci et al. (1995) suggest three mechanisms for addressing safety in the design: *lockin*, *lockout* and *interlock*. Their descriptions are attuned to the context of dynamic evolution, as shown in Table Appendix C.27:

Table Appendix C.27 Dynamic evolution safety design mechanisms

<i>Mechanism</i>	<i>Description</i>	<i>Example Continuum features to realise mechanism</i>
Lockin	Lock the application into safe states	<ul style="list-style-type: none"> Enforcement of operational profile (Appendix C.2.1.6) Use of Dynamic Wrapper (Appendix C.3.2.17.6)
Lockout	Lock the application out of hazardous states	<ul style="list-style-type: none"> Time limit set for transformations Transformation exception declaration and resolution Use of Recovery Blocks (Appendix C.3.2.17.10)
Interlock	Prescribe or disallow specific sequence of events	<ul style="list-style-type: none"> Explicit sequencing and ordering of transformations Explicit sequencing and ordering of transformation actions in a transformation

C.3.2.5 Dynamic Recomposition

Purpose: To compose several transformable items in a larger unit, decompose a larger transformable item into smaller units, and reconfigure the structure at runtime.

Description: Dynamic recomposition finds its presence in dynamic evolution, say, to alter the application's behaviour (e.g. with Unix pipes), recover from errors (SeCSE 2006), or provide alternative compositions for equivalent functionality (Mukhija & Glinz 2005). Dynamic recomposition can be unfolded into addition, removal and replacement of transformable items (Mukhija & Glinz 2005), each of which realised by a transformation. To determine, what transformable items need to be added, removed and replaced, the *gaps* and *similarities* between the as-is and to-be structures (i.e. before and after recomposition) are analysed (Salinesi et al. 2004). This technique considers three recomposition scenarios for the analysis:

1. Composing two composite transformable items, say, A and B, into a larger unit:
 - Identify transformable items to link A and B. This suggests additional dynamic wrappers (Appendix C.3.2.17.6) and bindings to be established.
 - Identify transformable items internal to A and B that can be shared between them (e.g. email component). This suggests redundant items to be removed.

- Identify transformable items internal to A and B which need to be upgraded in order for A and B to interact (e.g. to resolve incompatibility in communications protocol). This suggests old items to be replaced.
2. Decomposing a composite into two smaller units, A and B.
 - Identify transformable items shared between A and B. This suggests items to be replicated (and added), for them to be present in both A and B after decomposition.
 - Identify transformable items linking A and B, suggesting unused items to be removed.
 3. Reconfiguring a composition structure at runtime (more generic than the above):
 - Identify additional transformable items required to be present in the to-be structure. This suggests new items to be added.
 - Identify transformable items found in the as-is structure but absent in the to-be structure. This suggests redundant items to be removed.
 - Identify transformable items found in the to-be structure that replace their peers in the as-is structure. This suggests old items to be replaced.

C.3.2.6 Dynamic Refactoring

Purpose: To refactor a runtime structure without alternating its functionality.

Description: Refactoring aims to alter the structure of an application to make it easier to understand and cheaper to modify (Fowler et al. 1999). Dynamic refactoring is intended to achieve the same goal, by altering the structure of the application while it is running. The following steps are adapted from the work of Ebraert et al. (2004) for automatic dynamic refactoring:

1. Detect candidate design time refactoring. This step involves the determination of possible to-be static structures of the application which are alternative improvements of the as-is static structure of the application. Each static structure can be identified by iterating the following steps, as reused from the OPF's *Design Refactoring* technique (Firesmith & Henderson-Sellers 2002):
 - a. Understand a design defect to be eliminated via refactoring.
 - b. Identify a relevant refactoring design pattern.
 - c. Apply the refactoring design pattern to eliminate the defect.
 - d. Evaluate the refactored design to ensure the defect is eliminated.
2. Select the to-be static structure of best fit.

3. Derive the to-be runtime structure of the application from the selected to-be static structure and the runtime as-is structure.
4. Refine the to-be runtime structure of the application further by eliminating defects that are not apparent in the static structure.

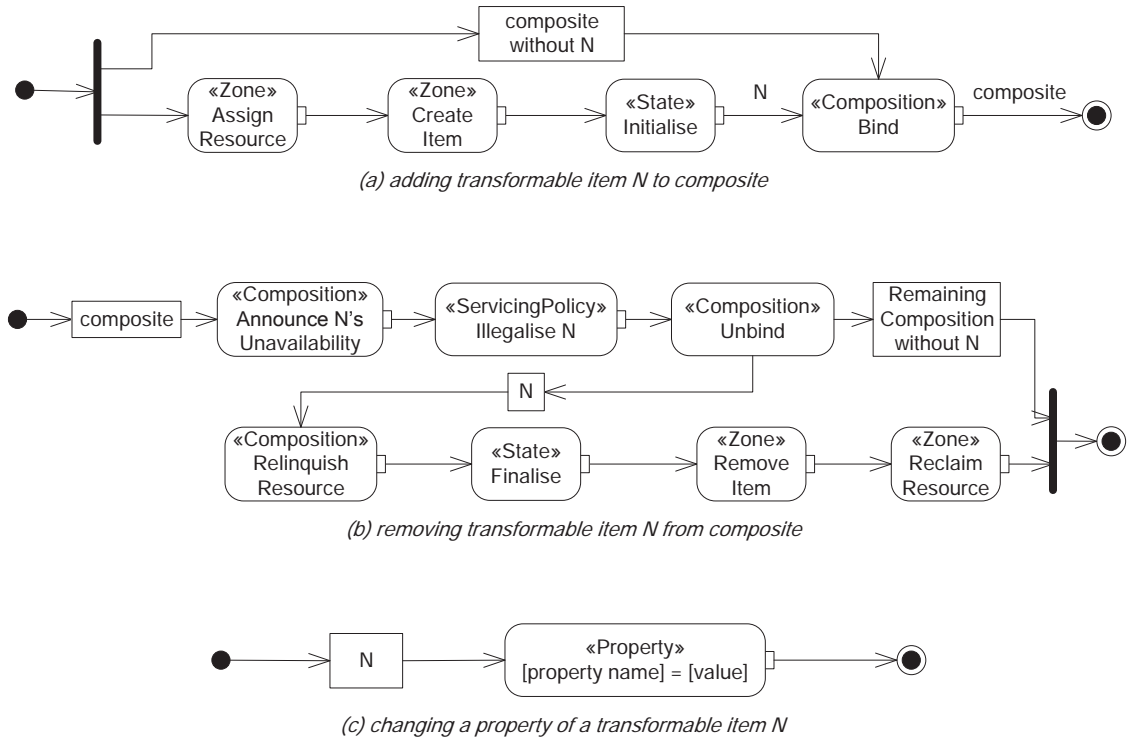
Dynamic Refactoring can be supplemented with Technique Loose Coupling (Appendix C.3.2.17.8) to further improve the structure of an application to evolve.

C.3.2.7 Dynamic Transformable Item Change

Purpose: To add, replace, remove transformable items, and/or change their properties at runtime.

Description: Addition, removal and replacement of transformable items are basic operations to an application supporting dynamic evolution (Kramer & Magee 1990; Mak et al. 2005; Oreizy et al. 1998). They represent recurrent problems to resolve and opportunities for reuse. Thus, Continuum offers a number of transformation design patterns as a guide to address these problems, and as a baseline for extension to suit more complex transformation scenarios.

Figure Appendix C.11(a) depicts a transformation pattern to *add* a new transformable item to a composition. First, the transformation assigns resources for the item. Then, the transformation creates the new transformable item in its designated zone and initialises the start-up state of the item. Finally, it binds the item to other item(s) (i.e. its containing composite) that it is going to interact with (i.e. early binding). Alternatively, the binding is not established until interactions occur (i.e. late binding).



source: developed for this research

Figure Appendix C.11 Addition, removal and property change transformation patterns

Figure Appendix C.11(b) is a transformation pattern to *remove* an existing transformable item from an application. The transformation starts with announcing or notifying to the clients of a transformable item that it will become unavailable. Then, services of the item are disabled from access. Next, the item relinquishes any resources it held. Afterwards, the unbinding of the item from the composite takes place which is followed by setting the item's state to the finalised state for graceful shutdown. The item is then removed from its hosting zones. Finally, the zone reclaims any resource previously allocated to the item.

Figure Appendix C.11(c) shows a transformation pattern to modify a property (e.g. operating parameter) of a transformable item.

A replacement can be thought of as consisting of adding a replacement transformable item and removing an existing transformable item to be replaced. Accordingly, a transformation orchestration diagram (Figure Appendix C.12) can be defined using the two transformation patterns described above, plus a rebinding transformation pattern to link these two patterns together. Rebinding is discussed in Appendix C.3.2.8.



source: developed for this research

Figure Appendix C.12 Replacement using three transformation patterns

C.3.2.8 Dynamic Transformable Item (Re)binding

Purpose: To bind or rebind a transformable item to a composition of transformable items at runtime.

Description: Binding occurs when a transformable item is to be integrated into a composite of transformable items at runtime. Rebinding refers to the situation where an existing transformable item in a composite is to be substituted with another one offering equivalent (i.e. identical or similar) functionality, such as when the existing transformable item fails to operate.

This technique starts with the “binding and re-binding” technique³⁰ from the SeCSE methodology (SeCSE 2007) to look for an equivalent transformable item using the following steps:

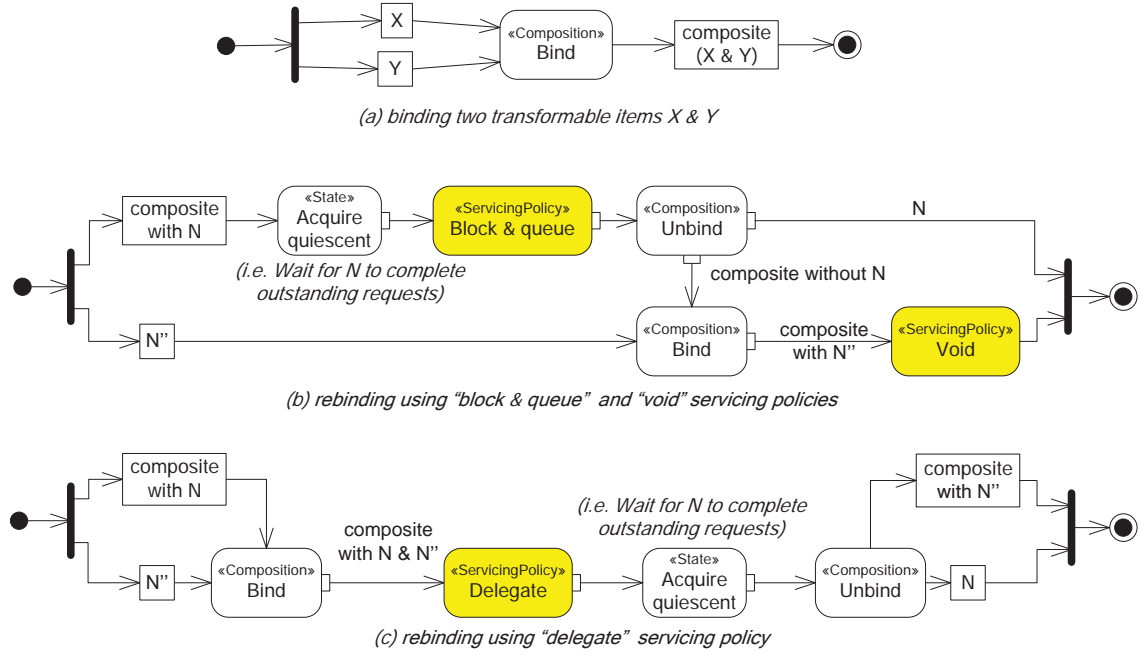
1. Gather the following information:
 - the transformable item’s functional and non-functional characteristics; and
 - monitored data about the transformable item to further determine the specific features that the replacement transformable item should have. This information is necessary in case the transformable item fails.
2. Obtain the list of candidate (replacement) transformable items.
3. Select the most appropriate (replacement) transformable item.
4. Map the operations and parameters between the selected transformable item and the composite.

These steps can be carried out at design time and runtime. In fact, SeCSE automates these steps in an application and the platform supporting the application. To automatically trigger these steps, SeCSE proposes the detection of transformable item failures as well as of the arrival of new transformable items as replacement for their older counterparts.

Once the transformable item is identified from the steps described above, a

³⁰ Strictly speaking, Binding and Re-Binding should be classified a technique as opposed to a process by SeCSE, as it describes *how* to bind or rebind a composition to a service, in terms of steps.

transformation design takes place to (re)establish the actual binding in order for the application to continue to operate as normal. Continuum enhances SeCSE by offering several designs to accomplish this. First, the binding case is the most straightforward as illustrated in Figure Appendix C.13(a).



source: developed for this research

Figure Appendix C.13 (Re)binding transformation patterns

For rebinding, there are two options. The transformation pattern in Figure Appendix C.13(b) utilises the "blocked and queued" servicing policy to temporarily buffer input requests to the composite while establishing the binding between the composite and N'', the new transformable item. In Figure Appendix C.13(c), both N and N'' are bound to the composite first. Then the "delegate" servicing policy is used to forward all new requests to N'', while leaving N to fulfil all pending and existing requests. After all existing requests have been processed by N (i.e. a particular quiescent state reached), N is no longer required and is thus unbound from the composite.

C.3.2.9 Resource Profile Modelling

Purpose: To analyse and predict the resource needs of transformable items.

Description: A new and replacement transformable item may require resources, dedicated or shared, in order to function. At other times, an existing transformable item may increase/decrease its resource consumption to adapt to the changing pattern of its usage. For instance, a new search engine would need a dedicated disk space to buffer search data. Without sufficient disk space, the engine would not be able to operate

after it has been deployed live to an application. On the other hand, resource contention arises when there are insufficient resources to meet the demands from two transformable items sharing the resources.

Continuum prescribes the following basic steps which will help one to estimate the resource needs for each transformable item:

1. Identify resources required by the transformable item (i.e. the consumer).
2. For each resource identified above, assess if the consumer will share the resource with others, or require exclusive use of the resource.
3. For each resource identified above which is also quantifiable, estimate the *static* and *dynamic* usages (OMG 2005) in appropriate units. Static usage refers to the use of the resource that does not vary with time or load (e.g. when the transformable item is idle). Dynamic usage accounts for situations where the use of the resource varies on-demand (e.g. when a transformable item is being used). For the latter, Muskens and Chaudron (2004) proposed an estimation technique for resource consumption during the invocation of each service of a component (i.e. transformable item).
4. In a composition paradigm, a transformable item can potentially invoke services provided by other transformable items to deliver its own services. Accordingly, the resources consumed by the latter items must also be estimated (de Jonge et al. 2003). In this situation, repeat Steps 1-3 for these additional items.

For advanced resource modelling, readers can refer to the *General Resource Modelling Framework* (OMG 2005) which provides comprehensive constructs for modelling resource usage patterns (static or varying loads on resources), resource types, resource management and so forth. They are beyond the scope of Continuum.

C.3.2.10 Root Cause Analysis

Purpose: To determine the root causes which led to dynamic evolution quality problems, including defects and issues.

Description: Root causes are analysed in this technique after which appropriate actions can be planned to rectify them later on. This technique consists of two steps to analyse defects and issues for their root causes:

1. Classify defects/issues. Leszak et al. (2002) propose a classification scheme for analysing defects along the following dimensions:

- *Phase detected*: the development phase (e.g. analysis, design, testing) in which a defect or an issue was detected;
- *Defect type*: the type of the defect or issue found;
- *Defect location*: the work product in which the defect or issue has actually been found (e.g. code, documentation); and
- *Root cause(s)*: the root cause(s) or trigger(s) which led to the defect or issue. The can be more than one root cause for each defect or issue.

Leszak et al. (2002) also define a set of make attribute values for the above. Some of these values are substituted, with respect to dynamic evolution and Continuum's features, to make them suitable for use in Continuum.

Table Appendix C.28 Suggested values for dynamic evolution quality defect/issue attributes

Name		Value
Defect/Issue Type		An instance of Continuum's model unit fragments: "Application", "ApplicationLifecycle", "ChangeCase", "Generation", "Impact", "OperationalProfile", "Resource", "ServicingPolicy", "TransformableItem", "Transformation", "TransformationAction", "TransformationAgent", "TransformationException", "TransformationExceptionResolution", "TransitionalPeriod", "Zone", "ZoningPolicy"
Defect/Issue Location		An instance of Continuum's diagram and document fragments: "Application Lifecycle Diagram", "Dynamic Application Change Document", "Dynamic Evolution Quality Inspection Report", "Dynamic Evolution Quality Problem Analysis Report", "Dynamic Evolution Quality Profile Report", "New and Replacement Transformable Item Catalogue", "State Map", "Structural Configuration - Notational Extensions", "Transformation Diagram", "Transformation Orchestration Diagram", "Zone Change Document"
Root Cause	Phase Related	An instance of Continuum's process fragments: "Application Lifecycle Analysis", "Transformation Identification", "Transformation Agent Design", "Transformation Design", "Dynamic Evolution Quality Management" and its outcomes further qualified by the nature of the root cause (reused from (Leszak et al. 2002)): "incorrect", "incomplete", "ambiguous", "changed/revised/evolved", "not aligned with customer needs"
	Human Related	(reused from (Leszak et al. 2002)): "change coordination", "lack of domain knowledge", "lack of system knowledge", "lack of tools knowledge", "lack of process knowledge", "individual mistake", "introduced with other repair", "communications problem", "missing awareness of need for documentation"
	Project Related	(reused from (Leszak et al. 2002)): "time pressure", "management mistake", "caused by other product"
	Review Related	(reused from (Leszak et al. 2002)): "no review", "incomplete review", "not enough preparation", "inadequate participation"
	Other	description for an ad hoc root cause

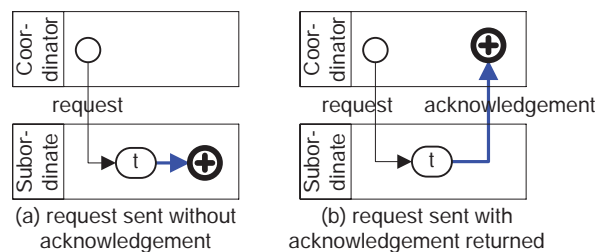
2. Group root causes according to defect/issue locations, defect/issue types and root causes. This is to facilitate defects/issues to be resolved. For example, defects found from the same Transformable Item may be dealt with together.

C.3.2.11 Secure and Reliable Transformation Agent Coordination

Purpose: To support security and reliability of the coordination among transformation agents in a distributed environment for the performance of a set of transformations.

Description: A distributed environment opens the door to security threats and network failures. This means the communication and thus the coordination among transformation agents are susceptible to security and reliability issues when they are performing a set of transformations in concert. To address the unreliability of the network impacting the coordination of transformation agents, Continuum suggests the following enhancements:

- *Reliable network communications protocol:* For instance, Georgiadis et al. (2002) applied a reliable broadcast protocol which distributes reconfiguration information and commands to distributed component managers (i.e. subordinate agents) to coordinate their actions.
- *Explicit request and acknowledgement:* For every request a coordinator sends to a subordinate agent, an acknowledgement of the completion of the request should be explicitly captured (Endler 1993). Figure Appendix C.14(a) depicts that after a coordinator issues a subordinate agent to perform its transformation “t”, it is unable to determine whether “t” completes or not. Figure Appendix C.14(b) shows a slight improvement; an acknowledgement is sent back to the coordinator to say that “t” has indeed been executed.

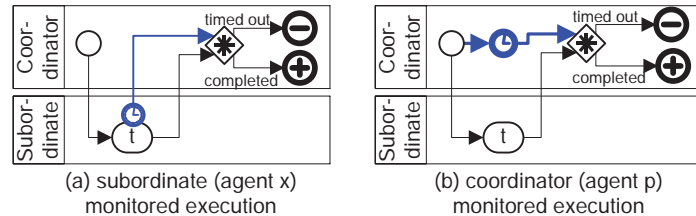


source: developed for this research

Figure Appendix C.14 Example uses of request and acknowledgement between two transformation agents

- *Deadline:* In the presence of network failure, a coordinator agent is unlikely to receive acknowledgement messages for requests it sent. Imposing a time limit on such messages and handling the situation on message expiration is one way

to deal with latency in network communications. An expiration should be handled by the coordinator (Figure Appendix C.15(b)) rather than its subordinates (Figure Appendix C.15(a)), for the coordinator needs to retain control of the execution flow.



source: developed for this research

Figure Appendix C.15 Example of timing out a subordinate agent's transformation

To provide security, Kon et al. (2000) identified three common security supports used together for secure transformation agent control and interactions:

- *Secure communications*, messages exchanged among transformation agents, and commands from an agent administrator to the agents are encrypted so that a third party cannot eavesdrop on their content and resend them;
- *Authentication*, transformation agents and their administrator establish the identities of one another for which they wish to trust; and
- *Access control*, access to transformation agents in terms of issuing them transformation commands is controlled. Furthermore, in a role-based access control, what a coordinator agent is permitted to instruct a transformation agent to perform is determined by its role. For instance, a coordinator agent has permissions to instruct a subordinate agent to relocate transformable items from one zone to another zone only, whereas another coordinator agent has full control of the subordinate.

C.3.2.12 Start-up State Configuration

Purpose: To determine the states from which new and replacement transformable items start to operate after they have been placed into their application.

Description: In some situations, transformable items are designed to have states (as opposed to “stateless” ones (e.g. Milanovic & Malek 2004)) and state configuration may be required for these transformable items before they start operating. For instance, it is sometimes desirable to make a (new) transformable item operate from a non-initial state right after a transformation to reduce the latency caused by the transformable item’s initialisation. A more complex scenario is when a replacement transformable

item's state must be derived from the state of an existing item being replaced before operation. For example, when replacing a live database connection pool, it is desirable to transfer its "ready" state to the new pool by instantiating connection objects in the new pool such that it is ready to offer its connection objects to its clients.

Start-up State Configuration involves two steps.

1. Identify all resuming states of a new and replacement transformable item. A resume state is one that a transformable item can resume or commence its operation (cf. Appendix C.2.1.13.1). Sometimes the initial state of a transformable item can also be, but not always, the resuming state.
2. Determine from which resuming states the transformable item can commence operation. If it is intended to replace an existing transformable item, its state will likely be derived from the latter and a State Map (cf. Appendix C.2.2.7) will be specified.

While the steps presented above are elementary, state configuration can be highly complex and existing theoretical approaches for in-depth state configuration design can be consulted (e.g. Gupta et al. 1996; Kramer & Magee 1990). For instance, Vandewoude and Berbers (2002) discuss implementation issues of state transfer and offer an algorithm to partially automate it.

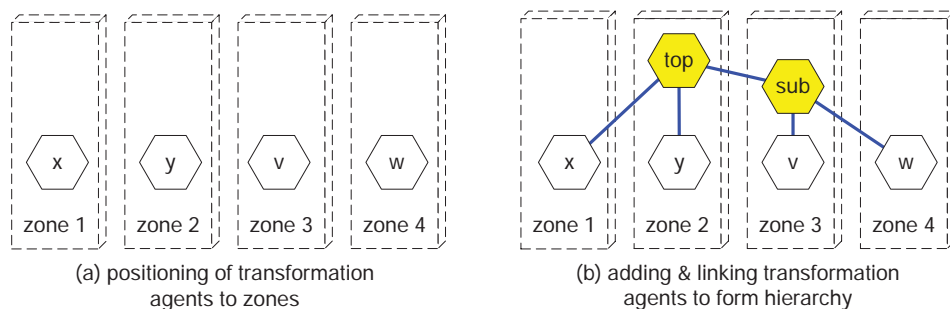
C.3.2.13 Transformation Agent Disposition

Purpose: To station transformation agents in zones ready for transformations.

Description: When a set of related transformations to occur during a transitional period are intended for different parts of an application in a distributed environment, the control of the associated transformations can be *centralised* to a transformation agent or *decentralised/distributed* to several agents, the latter being more scalable for large applications (Bradbury et al. 2004). Endler (1993) recommends that agents should be arranged *hierarchically* - each having a parent (the coordinator), and one or more children (the subordinates) - to avoid potential interaction deadlocks if transformations were concurrently executed. Inspired by Endler's hierarchical and decentralised approach, the following steps are proposed to identify transformation agents:

1. Identify the zones in which transformations are performed. This defines the domain of responsibilities for transformation agents.
2. Assign a transformation agent to each identified zone. See Figure Appendix C.16(a) for illustration.

3. Define a hierarchy of transformation agents in identified zones by linking the agents to form a tree-like structure. If appropriate, introduce additional agents that will help subordinated agents to manage their workload. An example is illustrated in Figure Appendix C.16(b) where agent “sub” is placed into zone 3 to explicitly manage subordinate agents “v” and “w”, and agent “top” is responsible for coordinating all transformations handled by agents “x”, “y” and “sub” in the four zones.



source: developed for this research

Figure Appendix C.16 Example transformation agent hierarchy

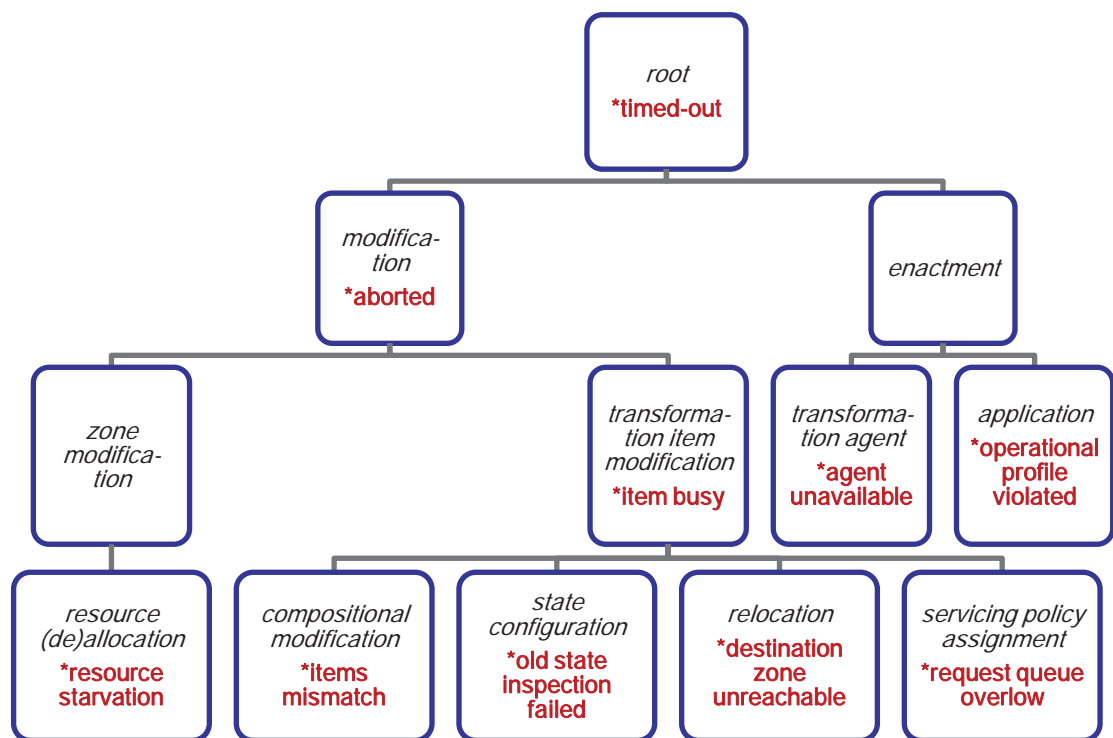
C.3.2.14 Transformation Exception Management

Purpose: To manage exceptions as a result of the enactment of a transformation.

Description: To improve the resilience of an application against failures anticipated from transformations, appropriate exceptions should be declared for transformation failures and handled accordingly. This technique has two steps simplified from Dellarocas et al.'s (1998) approach for managing exceptions in an evolving system, and customised to specifically address transformation failures:

1. *Anticipate and detect* transformation exceptions. Given the design of a transformation (see the task Develop Transformation, Appendix C.3.1.4.4), identify a list of transformation exceptions (Appendix C.2.1.17) known to occur from the transformation. Klein and Dellarocas (2000) proposed an identification approach that uses a taxonomy of events representing what can occur during normal circumstances, and associates the events with the kinds of exceptions anticipated. A taxonomy is analogous to a class hierarchy; events towards the top of the taxonomy are more generic than those towards the bottom of the taxonomy which are increasingly specialised from the former. Each event inherits all exceptions from its ancestors in the hierarchy and may contain additional exceptions that are specific to it.

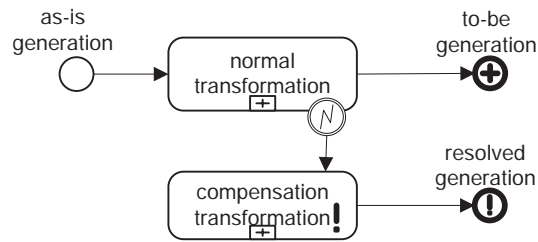
Figure Appendix C.17 shows an example event taxonomy (using notations from Klein & Dellarocas 2000). Events on the “modification” branch are derived from the types of transformation actions supported in Continuum (Appendix C.2.1.15.1) whereas events on the “enactment” branch cover the aspect of carrying out transformations. With respect to the taxonomy, if a transformation intends to relocate transformable items to another zone (i.e. “relocation” event), possible transformation exceptions include “destination zone unreachable”, “item busy”, “aborted” and “timed-out”.



source: developed for this research

*Figure Appendix C.17 Example taxonomy of abnormal transformation events (exceptions labelled with *)*

Transformation exceptions identified are then incorporated into the transformation design. Using Continuum’s notation (Appendix C.2.2.8), Figure Appendix C.18 declares a transformation annotated with an exception to trap the failure condition of that transformation.



source: developed for this research

Figure Appendix C.18 Example transformation with exception declaration and rollback

2. *Resolve* transformation exceptions. Based on the diagnosis of each transformation exception, a suitable *strategy* (or more) is determined to resolve it. Continuum adopts Laprie's (1995) three error processing approaches (as below) as generic and basic means of resolving transformation exceptions:
 - a. *Backward recovery*: Bring the application back to the state prior to the transformation. This means a *rollback* to its structure and configuration before the transformation commences (e.g. Figure Appendix C.18).
 - b. *Forward recovery*: Promote the application forward to a state from which it can still operate. One example is to re-execute the failed transformation if possible.
 - c. *Failure compensation*: The transformation failure is masked out or compensated to let the application continue to operate. The actual compensation depends on the type of a transformation exception, the design of the transformation and the context in which the transformation occurred. For instance, if there are insufficient resources to operate a new transformable item, a possible solution is to negotiate a lower resource requirement for the item.

As in the case for transformation, exception handling is undertaken at runtime. Thus, the information produced from the steps above would normally be fed to exception handling infrastructures to facilitate automation. Since infrastructure and tool aspects are not part of Continuum, they are not elaborated further.

C.3.2.15 Transformation Mining

Purpose: To extract transformations from change cases, and refine transformations such that they can be realised during a transitional period.

Description: Composition-based distributed applications facilitate dynamic evolution in a certain way and hence it is appropriate to leverage this capability when defining

transformations for this kind of application. This technique defines the following steps to identify transformations from change cases:

1. For each change case targeting more than one transformable item or zone, treat it as if there were several identical change cases, each targeting an individual transformable item/zone.
2. For each transformable item or zone targeted by one or more change cases, perform the following sub-steps:
 - a. Group change cases that target the same transformable item/zone.
 - b. For each group identified in the last step, identify modifications to realise change cases in the group. In a composition-based distributed application, the following modifications are common and hence can be used as a baseline:
 - *addition* of a new transformable item/zone;
 - *removal* of an existing transformable item/zone;
 - *replacement* of an existing transformable item, which can be thought of as an addition, followed by a rebinding and a removal (cf. Appendix C.3.2.7);
 - *binding* of a transformable item to another one;
 - *rebinding* of a transformable item from one transformable item to a different one; and
 - *reconfiguration* of a transformable item/zone's parameters/settings.

Note that one modification can realise multiple change cases. For instance, a modification can both add a new function and upgrade an existing function of a transformable item at the same time.

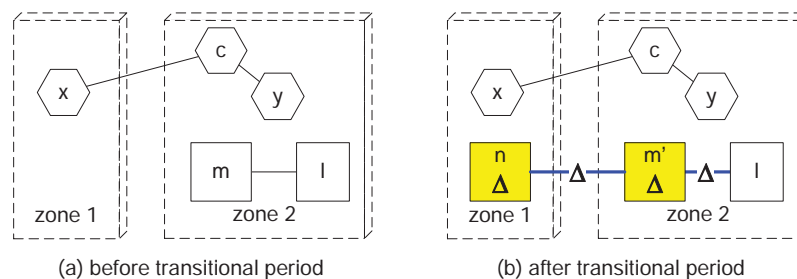
- c. For each modification identified above, match it with a transformation pattern. Continuum offers addition, removal, property change, replacement, binding and rebinding transformation patterns (Appendices C.3.2.7 and C.3.2.8). If no patterns are suitable for the modification, a separate transformation will be individually designed for it.
- d. Break up a transformation into smaller transformations if necessary. The break up might be beneficial if the transformable item being targeted is a composite of other transformable items such that the transformation affects many of these items. Smaller transformations are also likely to have shorter execution times and to cause smaller interruptions to the

application. A note of caution is that transformations should not be very small as this would explode the number of transformations to cover all change cases to be realised during a transitional period. A resulting overhead could be an increase in the number of times an application waits for a *quiescent* state (Appendix C.2.1.13.1) as the number of transformations increases.

C.3.2.16 Transformation Orchestration and Agent Coordination

Purpose: To design an orchestration of transformations and assign them to transformation agents which will coordinate with one another to carry out those transformations during a transitional period.

Description: In a distributed environment, transformations scattered in different zones of an application should be managed together to accomplish desirable effects. To guide the use of this technique, consider a simple hypothetical application consisting of two zones, 1 and 2, with transformable items “m” and “l” held in zone 2 (Figure Appendix C.19(a)).



source: developed for this research

Figure Appendix C.19 Example application used to illustrate Transformation Orchestration and Agent Coordination

Based on the Technique Transformation Agent Disposition (Appendix C.3.2.13), transformation agents “x” and “y” have been identified and assigned to zones 1 and 2 respectively to deal with transformations in these zones. Additionally, transformation agent “c” is defined to coordinate the transformations of agents “x” and “y”.

Suppose that a new transformable item “n” is to be deployed to zone 1 and integrated with “m” (Figure Appendix C.19(b)). For this to work, “m” must also be upgraded to expose a new interface used for integration with “n”. This requirement translates to the following transformations:

- “m’.add”, “l.rebind” and “m.remove”, which add the new version of “m” (i.e. “m’”) to zone 2, rebind “l” from “m” to “m’”, and remove the old version of “m”.

- “n.add” and “n.bind”, which add “n” to zone 1 and integrate “n” with “m”.

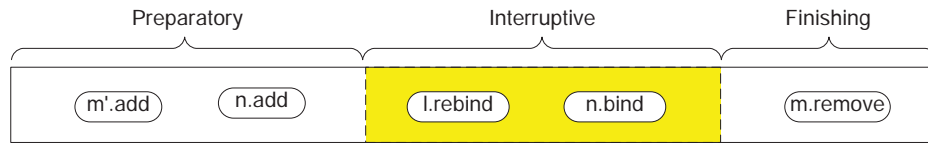
To design the orchestration of transformations and the coordination of transformation agents to perform these transformations, the following steps are executed:

1. Allocate transformations to particular phases of a transitional period.

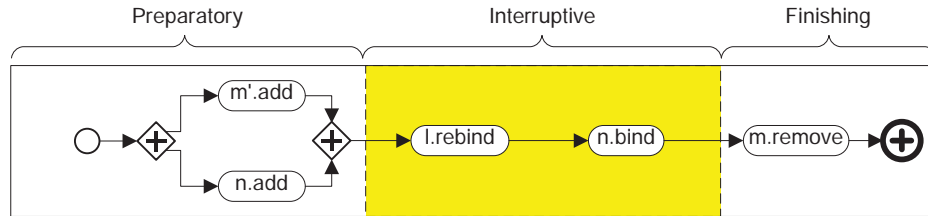
Transformations almost never complete instantly; some might cause interruptions whilst others are invisible to end users and the rest of the application. To confine the window of interruption, this step allocates each transformation to one of the three phases, to distinguish between transformations that will interrupt an application and those that will not do so:

- a. Transformations in the *preparatory* phase do not interrupt an application. They aim to perform preparatory work ready for transformations that will interrupt the application. In the example shown in Figure Appendix C.19, as transformations “m’.add” and “n.add” will not impact the application, they can be performed first and thus assigned to the preparatory phase.
- b. Transformations in the *interruptive* phase interfere with the normal running of an application, its transformable items and/or services being offered (e.g. “l.rebind” and “n.bind”). For instance, a transformable item or an application as a whole may need to be temporarily out of service during a transformation. In the Figure Appendix C.19 example, both “l.rebind” and “n.bind” are allocated to the interruptive phase since they will cause some degree of interruption to the application.
- c. Transformations in the *finishing* phase aim to perform housekeeping work after transformations in the interruptive phase, such as the removal of transformable items that are no longer used (e.g. “m.remove”).

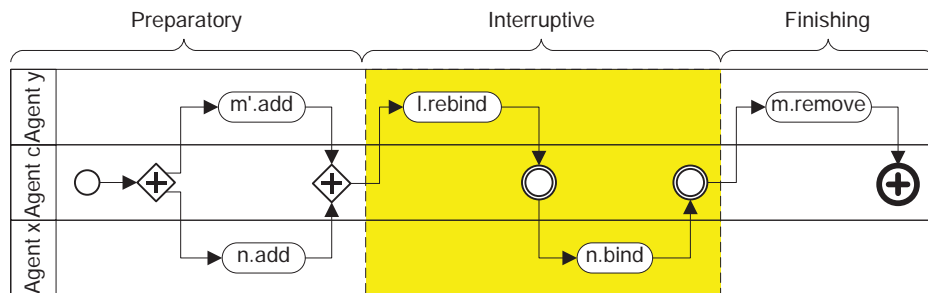
The allocation of transformations into different phases using sub-steps 1.a to 1.c above results in an initial version of the transformation orchestration diagram (Appendix C.2.2.10) as shown in Figure Appendix C.20(a).



(a) allocation of transformations to phases



(b) orchestration of transformations



(c) assignment & coordination of transformation agents

source: developed for this research

Figure Appendix C.20 Example development of transformation orchestration

2. Arrange and link transformations into an orchestration.

Link transformations to form an orchestration depicting a workflow of transformations. In doing so, consider the nature of the transformations and take advantage of parallelism to shorten the transitional period. It is because certain transformations may be executed in parallel whereas some transformations must be completed prior to others. For instance, in Figure Appendix C.20(b), transformations “m’.add” and “n.add” in the preparatory phase can be executed concurrently since they do not interfere with each other. In the interruptive phase, however, transformation “l.rebind” should precede transformation “n.bind” since “m” should be integrated into the application before “n” is integrated with “m”.

3. Assign transformations to transformation agents and define their coordination.

To illustrate this, consider Figure Appendix C.20(c). First, add a swimlane to the transformation orchestration diagram for each transformation agent identified (i.e. agents “x”, “y” and “c”). Next, transformations to appear within a particular

zone are placed in the swimlane of the agent responsible for the zone (i.e. transformations “n.add” and “n.bind” to agent “x”, and transformations “m’.add”, “l.rebind” and “m.add” to agent “y”). Appropriate control flows (Table Appendix C.18) are then inserted into the workflow to define the coordination of transformations. In Figure Appendix C.20(c), for example, start (○) intermediate (⊙) and end points (⊕) as well as gateways (⊞) are assigned to the swimlane for agent “c” as it will coordinate agents “x” and “y” to perform their specified transformations.

C.3.2.17 Reused Technique Fragments

Continuum incorporates several existing techniques into its set of technique method fragments. These techniques - wrapped as technique fragments below - have been reviewed as suitable for dynamic evolution. Although they may have been developed for particular types of composition-based distributed applications, they are sufficiently generic to be useful for other types of composition-based distributed applications. Interested readers should consult their respective documentation for further details.

C.3.2.17.1 Dynamic Change Localisation

Purpose: To localise of changes to within a logical boundary of an application.

Description: Evans and Dickman (1999) describe an approach to confine the scope of dynamic changes to within a logical boundary of an application to manage their complexity. Their approach is underpinned by three abstractions: zones, contracts between zones, and change absorbers. Their notion of zone is equivalent to Continuum’s zone which defines logical partitions in an application (Appendix C.2.1.20). Contracts explicitly capture the dependencies between zones, by way of specifying their communications and invocations between them. To confine a change to within a zone, change absorbers with respect to the change are inserted at the boundary of the zone to perform data translation, interface mapping etc. Change absorbers ensure that the transformable items in the zone, even after the change, still appear to fulfil any contract associated with the zone as if they were unaffected by the change. Their approach confines not only dynamic changes to an application but also errors to within the zone itself, should the changes fail.

See the related Technique Identify Changes to Zones (Appendix C.3.2.17.1) about zoning policies recommended for an application.

C.3.2.17.2 Dynamic Security Policy and Enforcement Management

Purpose: To manage security policy and enforcement changes in response to dynamic changes in an application.

Description: Grimm and Bershad (2001) present a generic access control model which separates security policies, security enforcement and application functionality from one another. The model adapts to dynamic security policy changes as extensions (i.e. transformable items) are added to and removed from an application. It explicitly defines a security policy manager and a security enforcement manager as entities which are separated from an application. The security policy manager keeps security policies for the application (i.e. which subjects are granted what kinds of privileges to access protected objects) up-to-date whenever extensions are added to or removed from an application. The enforcement manager intercepts messages or invocation requests to protected extensions, and checks the access rights of the clients which sent the messages or invocation requests against their granted privileges for the extensions by interrogating the security policy manager.

C.3.2.17.3 Dynamic Transformable Item Adaptation

Purpose: To wrap and plug transformable items into an architecture at runtime, while resolving their mismatches.

Description: The Rational Unified Process (RUP) (Kruchten 2003) defines the “service mediation” guideline which uses mediation to resolve communication incompatibilities between incompatible services (which are transformable items) and their consumers, by way of transforming consumer requests or protocols into formats that services can understand. It offers three forms of mediation: interface-, protocol- and operation-based.

C.3.2.17.4 Dynamic Variation Management

Purpose: To define customisation points in a structure, to plug in or swap different transformable items, to support limited variations in functionality.

Description: SeCSE (2006) offers the “Variations Points Management” technique³¹ to describe at design time which parts of a composition are variable which means that they can be changed at runtime. It offers steps to analyse a composition model and add variation points to it.

³¹ Strictly speaking, Variations Point Management should be classified as a technique as opposed to a task by SeCSE, as it describes *how* to produce variation points in a composition.

C.3.2.17.5 Dynamic Workflow Change

Purpose: To dynamically change a workflow while it is operational.

Description: Workflow evolution improves the flexibility of a workflow in adapting to the changing needs of an application. In particular, dynamic workflow change extends this flexibility further by supporting workflows in execution to be changed dynamically.

Research on evolving workflows at runtime has been extensive, some of which can be useful for Continuum. For instance, Casati et al. (1998) divide the problem of workflow evolution into static and dynamic facets. They proposed a language to address both facets of workflow evolution: a set of primitives to modify the definitions of the workflow schema (i.e. the static support), and mechanisms to migrate existing running workflow to conform to the new schema (i.e. the dynamic support). Tomic et al. (2007) proposed a language for the specification of policy assertions in business processes (i.e. workflows) implemented with Web Services. One of the discussed policy assertion types concerns the structural adaptation for processes, supporting addition, removal, and replacement of activities (i.e. sub-processes) in a process. Ellis and Keddara (2000) proposed a modelling language for specifying dynamic workflow changes, intended to be generic enough for several domains: organisational, manufacturing, software etc. Their approach characterises changes along six modalities, such as whether a change is applied instantaneously or gradually on a workflow.

C.3.2.17.6 Dynamic Wrapper

Purpose: To dynamically enclose a transformable item (or more) to modify its input/output characteristics for desired results.

Description: The origin of wrapper (a.k.a. adapter) is traced to a design pattern for object-oriented development, which aims to convert an object or an interface of a class into another form compatible with its clients (Gamma et al. 1995; Graham 1991). It is utilised in different contexts for various purposes. In dynamic evolution, wrappers are specifically applied at runtime to achieve the following results:

- *Input/output confinement:* Voas (1998) defines two types of wrappers used together to limit what a distrusted component (i.e. transformable items) can do in an application, to guard the application against any unexpected behaviour from the component. *Input wrappers* limit the range of inputs to the component to filter illegal inputs and prevent the component from invocations with inputs that could lead the component or its application to produce out-of-range or

illegal outputs. *Output wrappers* intercept outputs from the component, ensure they meet certain constraints, and pass outputs to the receiving component.

- *Functionality extension*: Truyen et al. (2001) describe an architectural approach for supporting customisation to an application by accommodating extensions into an application dynamically. In their approach, an application is regarded as consisting of core “components” (i.e. transformable items) offering minimal functionality, and extensions to be integrated with the core ones. An extension holds refinements of an existing functionality or new functionality, and is encapsulated into one or more wrappers. To extend a core component, one or more wrappers dynamically enclose the component using design patterns (Gamma et al. 1995) to alter its observable behaviour. Their approach also supports extensions to existing functionality at the application level (i.e. functionality realised by a collaboration of several components). This is accomplished using a combination of wrappers and an interceptor for adjusting the message flow among the components.
- *Fault protection*: Wrappers have also been employed to protect an application against faults anticipated from wrapped transformable items (Torres-Pomales 2000) and against erroneous requests from the application (Meadows & McLean 1998). de Castro Guerra et al. (2003) proposed *protectors*, a special kind of wrappers, to serve these dual roles. Each protector is specified with constraints on the interaction between its wrapped component and the rest of the application. The constraints are used by both the protector for runtime detection of constraint violation and exception handlers to recover the application from such violations. Constraints are derived from information such as the behaviour of the wrapped components (i.e. transformable items) viewed from the application. For arbitrary (both anticipated and unanticipated) faults up to a certain number as determined by the size of a distributed application (i.e. Byzantine faults, Lamport et al. 1982), several fault-tolerant infrastructures have been investigated to operate potentially faulty web services (e.g. Merideth et al. 2005; Zhao 2009).
- *Fault containment*: Gama and Donsez (2010) proposed a special kind of wrapper called “sandbox”, to execute potentially and/or not trusted faulty transformable items. A sandbox continuously predicts and monitors known types of faults from these transformable items. If such faults occur, the sandbox contains them by avoiding them from propagating to transformable items

outside the sandbox. In a more severe case, if the sandbox crashes because of such faults, the sandbox automatically recovers without restarting the application. It minimises failure impacts on and disruptions to the application.

- *Security confinement*: Herrmann and Krumm (2001) employ *security wrappers* to secure:
 - an application and its environment against wrapped transformable items not trusted to the full extent, for the application and its environment may be compromised by the transformable items; and
 - wrapped transformable items against misuse from others, the application and its environment.

Security wrappers support these protections by enforcing the contracts for transformable items, the application and its environment. Each contract specifies the security behaviour constraints on all these parties. To reduce runtime overheads, the level of security checks against the contracts is dynamically adjusted, depending on prior knowledge of trust already established among the parties.

Fraser et al. (1999) identify and offer support for other uses of security wrappers:

- enforcement of customised access control to transformable items;
- auditing and intrusion detection of access to and parameters passed to transformable items; and
- security enhancements, such as encryption and decryption of sensitive data passed in and out of a transformable item.

C.3.2.17.7 Inspections

Purpose: To evaluate one or more work products against a checklist in order to identify areas for resolution and improvement.

Description: OPF's "inspections" (Henderson-Sellers et al. 1998) is a quality engineering technique for conducting an evaluation on work products against a checklist in order to identify defects and issues. It defines pre-conditions (i.e. inputs) for an inspection, steps to perform when preparing and during an inspection, and completion criteria (i.e. outputs) after an inspection. Its steps are generic in that they are not specifically defined for particular kinds of checklists or work products.

It should be noted that defect correction or issue improvement is not covered in this

technique to ensure the performance of this technique stays focused on identifying defects and issues to be resolved.

C.3.2.17.8 Loose Coupling

Purpose: To de-couple independent transformable items in an application.

Description: Papazoglou and van den Heuvel (2006) proposed the “service coupling” principle to make services (i.e. transformable items) and business processes (i.e. workflows) as independent from one another as possible. Their principle centres on reducing the number of bindings among the services and business processes.

C.3.2.17.9 Performance Profile Modelling

Purpose: To analyse and predict the performance characteristics of new and replacement transformable items.

Description: ASG (Lehner et al. 2006) offers the “performance engineering methodology”, a process which is used alongside its development process to specifically deal with performance objectives of a system that comprises an application, its parts (i.e. transformable items) and its platform (Kempton et al. 2007). In this regard, for each development, it offers steps and techniques to ensure the system is performing and scalable. During the requirements phase, performance requirements are specified. During the design phase, these requirements are mapped to the system’s design from which the performance models and performance predictions are created. In the implementation phase, these models and predictions are refined as the system is implemented and instrumented for collecting performance data. In the testing phase, performance testing is evaluated to obtain the actual performance of the system.

Of the guidance and techniques specified in the performance engineering methodology, Continuum reuses its technique “instant performance prototyping”. In summary, this technique performs the following steps to analyse and predict the performance characteristics of a system, its transformable items and its platforms:

1. Create a system model (or transformable items).
2. Define load tests.
3. Perform experiments in sub-steps:
 - a. Create a performance prototype of the system model.
 - b. Run load tests to measure performance of the prototype.
 - c. Analyse performance results.

C.3.2.17.10 Recovery Blocks

Purpose: To recover an application from errors brought by new and replacement transformable items after a transformation.

Description: A recovery block basically encompasses a primary block, zero or more alternative blocks, and a mechanism to verify and select a block (Horning et al. 1974). In a normal operating mode, a recovery block relies on its *primary block* to deliver its services. When errors are detected in the primary block, an *alternative block* is selected to substitute the services of the primary block to recover the operations provided by the recovery block as a whole. Recovery blocks can be composite, i.e. one block being made of smaller ones.

The recovery block technique can be applied to handle errors brought into an application by new and replacement transformable items. For new transformable items, the structure of an application with the new items and the one without them are mapped to the primary and alternative blocks respectively. For replacement items, the new and older versions of transformable items correspond to the primary block and alternative blocks respectively. The replacement case is illustrated by HERCULES, a framework for component upgrade (Cook & Dage 1999).

C.3.2.17.11 Runtime Structure Recovery

Purpose: To recover the information about the structure of a running application.

Description: When the as-is architecture of an application deviates from its *as-documented* form over time, implementing changes to an application poses real challenges such as an incorrect implementation of new functionality (Ding & Medvidovic 2001). Note that the name of this technique “Runtime Structure Recovery” is synonymous with “Runtime Structure Discovery” since the literature uses both recovery and discovery in describing different approaches for the same intent.

Schmerl et al. (2006) proposed a technique to observe and construct an architectural view of a running application. Their technique essentially covers three steps: monitor low-level runtime events in an application (e.g. method invocations); map them into architecturally related events (component creations); and build architectural models from those events. This approach requires certain assumptions about the architecture to perform the last two steps. It expects the architecture to follow a known architectural style and coding conventions in its implementation. To enable the application to generate events, its compiled code is instrumented.

Huang et al. (2006) utilise the reflection ability of a component framework used for building an application to recover the architecture of an application. As an application runs, a set of runtime entities is automatically created as basic elements are created in the application at runtime. As soon as changes to the application are made, the recovered architecture comprising these entities is kept up-to-date as changes to the elements are reflected in changes to these entities. Different views of the architecture are then constructed for the recovered architecture according to some principles (e.g. composite components) to increase the level of abstraction for the architecture and make it easier to understand.

C.3.2.17.12 Testability Analysis and Improvement

Purpose: To analyse and improve the testability of an application and its transformable items.

Description: Freedman (1991) proposed an approach to analyse and improve testability of an application. Testability is defined as the ease with which the input and data characteristics, and states of a computing unit (a transformable item or an application etc.) are identified. Two measures reflect testability: *controllability*, for ease of producing specified outputs from specified inputs, and *observability*, for ease of determining if specified inputs affect outputs. In summary, testability is improved with:

- proper documentation for input and output characteristics and states of transformable items;
- observability and controllability for the application, stated as functional requirements, as modification to an application may be required to make hidden input/output characteristics and states accessible; and
- implementation of observability and controllability in the application.

C.3.2.17.13 Transformable Item Autonomy

Purpose: To facilitate transformable items to have control and governance over their own processing.

Description: Erl (2005) describes in his methodology the “service autonomy” principle, offering guidance on dividing a business logic into services (i.e. transformable items) and defining their boundaries to improve the ability of the services to have control and governance of their own processing. Erl also notes that *service loose coupling* not only reduces service dependencies but also enhances their autonomy.

C.3.2.17.14 Transformable Item Mediation and Channelling

Purpose: To abstract communication concerns from transformable items.

Description: RUP's "service mediation" guideline (Kruchten 2003) describes three ways to handle mediation between incompatible service consumers and providers. One of these addresses the mediation of protocols between service consumers and providers using in a "service channel" representing an abstraction for communication between service providers and consumers. A service channel is defined separately from the specification of services offered by the service provider.

C.3.2.17.15 Transformable Item Regression Testing

Purpose: To ensure that a transformable item is accessible correctly, performs its operations, and behaves as expected, both functionally and non-functionally.

Description: SeCSE proposed a technique called "regression testing" to validate that services (i.e. transformable items) after deployment are in line with their expectations as they evolve (SeCSE 2008). In short, services are tested on two fronts. First, test cases are created and updated as services evolve or are added to an application, and executed against these services on demand. Second, normal invocations of the services are monitored in place to ensure that they produce the desired outputs for the invocations. Monitoring and verification also reduce the costs associated with conducting validation with the test cases on a live application, such as an increase in consumption of computing resources.

To address potential violations of invariants, SeCSE supports two ways of identifying invariants, and defines test cases to regression test an application and its services. First, for *known* invariants, they are explicitly specified upfront along with pre- and post-conditions for services and the application. Test cases are then specified and used in regression testing. Otherwise, *likely* invariants are profiled using the following steps:

1. Determine the possible ranges of inputs to services.
2. Randomly generate a large set of test cases simulating inputs to services within these ranges.
3. Execute the test cases on the services.
4. Analyse the inputs/outputs and derive the likely invariants (e.g. using a tool called Daikon (Ernst et al. 2007)).
5. Generate the test cases for the derived invariants.

Appendix D. CASE STUDY RESULTS OF APPLYING CONTINUUM

This Appendix describes the outcomes of using Continuum - the analysis and design artefacts produced with Continuum - in a case study application (cf. Section 7.2). It is split into the following sections, each of which documenting the artefacts produced with the designated Continuum process and its associated method fragments:

- Appendix D.1 for Process “Application Lifecycle Analysis”;
- Appendix D.2 for Process “Transformation Identification”;
- Appendix D.3 for Process “Transformation Agent Design”;
- Appendix D.4 for Process “Transformation Design”; and
- Appendix D.5 for Process “Dynamic Evolution Quality Management”.

For convenience, the artefacts presented in Appendices D.1 to D.4 also include the enhancements made to them to resolve the quality defects/issues found from their earlier versions. The defects/issues were identified and resolved by applying Process “Dynamic Evolution Quality Management” (cf. Appendix D.5). See Table Appendix D.14 for a description of the quality defects/issues and the respective improvements to rectify the defects/issues.

D.1 APPLICATION LIFECYCLE ANALYSIS OUTCOMES

The Application Lifecycle Analysis process extended the lifecycle of the application used in the case study to accommodate the changes to the application at runtime which were elicited from requirements analysis. It produced three key artefacts:

- from Task “Identify As-Is Runtime Structure”, the runtime structure of generation V1 of the application (Appendix D.1.1);
- from Task “Derive Change Cases”, the change cases proposed for the application (Appendix D.1.2); and
- from Task “Extend Application Lifecycle”, the extended application lifecycle to realise the change cases above (Appendix D.1.3).

D.1.1 Distributed Property Valuation - Generation V1

The Distributed Property Valuation system (or “DPV” in short) is a commercial and distributed application for issuing requests for property valuations and collecting property valuations. (See Section 7.2.1 for an introduction to DPV and its change

history.) DPV uses a variety of technologies (Java™ and Microsoft .NET framework™) and implementation models (Web Services plus components) in its architecture. In generation V1 of DPV, its elements (which are transformable items) are distributed in the following zones:

- Internet zone, the domain in which individual property valuers and staff of valuation firms (collectively called “valuers”) use a mobile device application called “Mobile Job Application” or “MJA” to obtain property valuation bookings or appointments, to fill out property valuation reports, and to upload the completed reports;
- Web zone, hosting Internet facing sub-systems;
- Internal zone, hosting DPV’s business related sub-systems and other applications used by internal business and administration staff; and
- Repository zone, in which valuation related data are kept.

The runtime structure of generation V1 of DPV is illustrated in Figure Appendix D.1. The transformable items in each zone are briefly described in Table Appendix D.1.

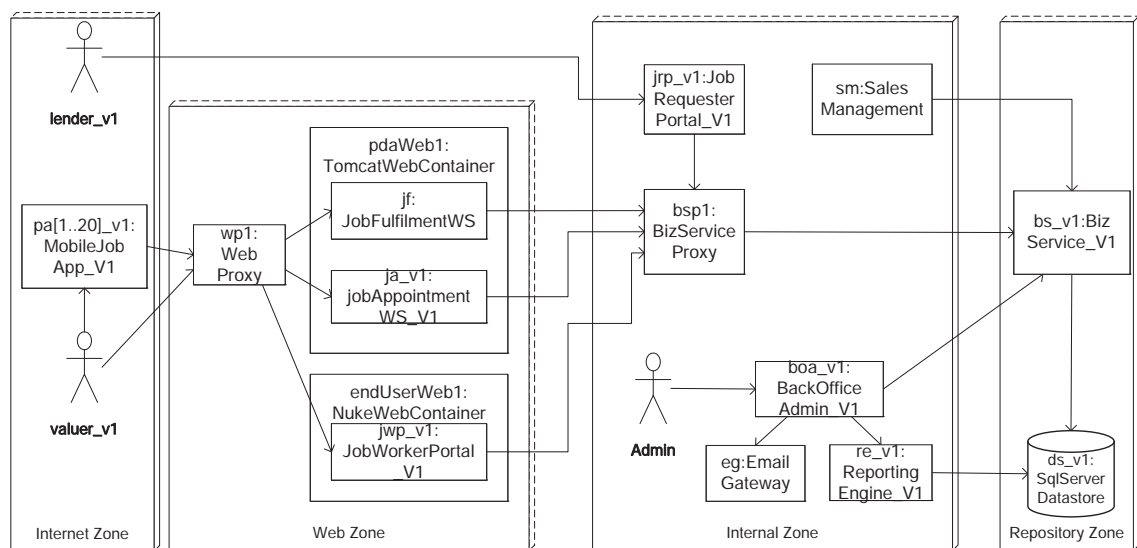


Figure Appendix D.1 DPV: generation V1

Table Appendix D.1 DPV: transformable items in generation V1

Zone	Transformable Item	Description
Internet	pa_v1:MobileJobApp_V1	An application (MJA) running on a personal digital assistant device (PDA) and enabling valuers to perform tasks remotely, including downloading valuation appointments, recording valuation data on-site and submitting the data. In generation V1, there were twenty PDAs with MJA installed.
Web	pdaWeb1:TomcatWebContainer	A web container hosting a web-based interface for MJA to communicate with DPV. It uses Apache Tomcat, an open source HTTP framework for Java.

<i>Zone</i>	<i>Transformable Item</i>	<i>Description</i>
	jf:JobFulfilmentWS	A Web Service for MJA to submit completed valuations.
	ja_v1:JobAppointmentWS_V1	A Web Service to download valuation bookings to MJA.
	endUserWeb1:Nuke WebContainer	A web container hosting all valuer facing applications via the Internet. It uses DotNetNuke, an open source web content management system, and Microsoft .NET framework™ for application development.
	jwp:v1:JobWorkerPortal_V1	A web application for valuers to view valuation bookings and the information required to conduct a property valuation, such as the location of a property.
	wp1:webProxy	An access point to transformable items in the Web zone from the Internet.
Internal	jrj_v1:JobRequesterPortal_V1	A propriety application accessed via Microsoft Windows Remote Desktop to let clients submit valuation requests to be performed for properties of interest.
	sm:SalesManagement	An internal application, external to DPV, to collect data from DPV for invoicing lenders and paying valuers.
	boa_v1:BackOfficeAdmin_V1	An internal application for internal staff to privileged access and administer valuation data such as valuation quotes, valuation requests, valuation bookings, valuations, progress reports, invoices, accounts etc.
	eg:EmailGateway	A system interface, external to DPV, for DPV to send emails to clients (i.e. lenders) and valuers.
	re_v1:ReportingEngine_V1	An application which runs periodically and on-demand to generate valuation and business activity reports from the data store. It uses Crystal Reports (a commercial database tool).
	bsp1:BizServiceProxy	An access point from the Web zone to transformable items in the Internal zone.
Repository	bs_v1:BizService_V1	A service to act as a facade to access to valuation related data held in the data store.
	ds_v1:SqlServerDatastore_V1	A Microsoft® SQL Server database or data store storing valuation related data.

Notes:

1. “Job” denotes “property valuation”.
2. “WS” stands for “Web Service”.
3. Transformable Items suffixed with “_V1” designate that they were specific to version V1 and were to be upgraded with a new version, all of which suffixed with “_V2”.

D.1.2 Change Cases

Change cases were derived from the business requirements specified for V2, the runtime structure of generation V1 of DPV and the following business constraints:

1. *Outstanding valuations-in-progress*: In the business domain, the period of time that begins when a valuation request has been created and ends when an associated valuation report is completed can last from a day to a fortnight. Thus, a valuation can be analogous to a long running transaction (Curbera et al. 2003). At any time during the original roll-out there were likely to be valuations booked

in DPV that were yet to be completed. Furthermore, valuations managed by DPV use different data structures between V1 and V2. Hence, when V2 was released, DPV should retain the V1 functionality to let valuers complete all outstanding valuations created with V1 of DPV.

2. *Latency in training completion*: Existing valuers of version V1 of DPV were handed out training materials to learn to use the V2 version of DPV at their own pace. The sponsor observed that it was inappropriate to roll out version V2 of DPV before all these valuers completed the training. Thus, DPV should handle a period during which both V1 and V2 functionality co-exist to support both untrained and trained users.

The business constraints were especially imposed by the sponsor that they should be addressed in the case study since they were encountered but not resolved satisfactorily in the original V2 upgrade project.

In Task “Derive Change Cases” (Process “Application Lifecycle Analysis”), the initial set of change cases were directly derived from the business requirements and constraints. The derived change cases are described in Table Appendix D.2 and grouped according to the generations targeted by the change cases. Note that the original requirements from which the change cases were derived are not shown in Table Appendix D.2 for reasons of confidentiality and anonymity.

Additional change cases were then identified by performing Task “Refine Change Cases” in Process “Transformation Identification” (cf. Appendix D.2) and subsequently incorporated to the initial set. For completeness, these change cases are also shown in Table Appendix D.2 (in *italicised* font). They were determined from:

- the differences between successive generations (i.e. V1 and V1.1beta, V1.1beta and V1.1, V1.1 and V1.2, V1.2 and V1.3, and V1.3 and V2) after the structures of the generations had been defined (in Process “Transformation Identification”); and
- Technique “Dynamic Change Impact Analysis”. (Change cases identified from this technique (e.g. “CC4003”) are highlighted in the “ID” column in Table Appendix D.2.)

To illustrate, “CC3011” is a supplementary change case for “CC4001”, which means when the data store is migrated to a different database platform (i.e. “CC4001”), the database driver of the reporting engine must also be changed (i.e. “CC3011”) to make

the reporting engine compatible with the new database platform. An example of the latter situation (i.e. additional change case) is when the old web container is replaced with one using a new technology platform (i.e. “CC2000”); not only must the new web container be installed in the web zone (i.e. “CC2005” being an additional change case) but also any existing web application running on the old web container needs to be relocated to the new web container (i.e. “CC2006” being the additional change case).

Table Appendix D.2 DPV: dynamic application change document

Change Case			Impact			
ID	Purpose	Description	Type	Disruption Level	Target	Supplementary Change Case ID
<i>Applied to generation V1</i>						
CC3005	Replace clientManagement function in boa_v1:BackOfficeAdmin_V1 with new version.	Bug fixes for clientManagement	direct	high	[self]	
CC3006	Replace jobReporting function in boa_v1:BackOfficeAdmin_V1 with new version.	Bug fixes for jobReporting	direct	high	[self]	
CC3007	Replace jobBookingAssignment function in boa_v1:BackofficeAdmin_V1 with new version.	Bug fixes for jobBookingAssignment	direct	high	[self]	
CC4001	Replace SqlServer persistence support in Repository zone with MySql persistence support.	Microsoft® SQL Server replacement with MySQL, plus refactoring and redesign of existing data schema. This change case is split into CC4001a and CC4001b.	See CC4001a and CC4001b			
CC4001a	Add MySql persistence support to Repository zone.	MySQL with refactoring and redesign of existing data schema.	indirect	high	sm:SalesManagement	CC3011
			indirect	high	re_v1:ReportingEngine_V1	CC4003
			indirect	high	bs_v1:Business_V1	CC4002
CC4002	Replace old bizService function with new version in Repository Zone.	Microsoft® SQL Server replacement with MySQL. This change case is split into CC4002a and CC4002b.	See CC4002a and CC4002b.			
CC4002a	Add new version of bizService function to Repository zone.	New version	indirect	high	boa_v1:backOfficeAdmin_V1	CC3010
			indirect	high	jf:JobFulfillmentWS	CC3008
			indirect	high	ja_v1:JobAppointmentWS_V1	CC3008
			indirect	high	jwp_v1:JobWorkerPortal_V1	CC3008

Change Case			Impact			
ID	Purpose	Description	Type	Disruption Level	Target	Supplementary Change Case ID
CC4003	Replace <i>SqlServer</i> database driver in <i>re_v1:ReportingEngine_V1</i> with <i>MySql</i> database driver.	<i>re_v1:ReportingEngine_V1</i> to communicate with new data store type (<i>MySQL</i>) and instance (<i>ds_v2:MySqlDatastore</i>)	direct	high	[self]	
CC4005	Add <i>bizServiceRecovery</i> function to Repository zone.	<i>rsrb:BizServiceRecoveryBlock</i> to ensure that if <i>bsf:BizService1to2Facade</i> fails to operate as expected, it will fall back to <i>bs_v1:BizService_V1</i> to continue to provide its functions.	indirect	high	<i>bsp1:BizServiceProxy</i>	CC3009
			indirect	high	<i>sm:SalesManagement</i>	CC3011
CC3008	Add facade for new version of <i>bizService</i> function to Internal zone.	Providing a facade to <i>bs_v2:BizService_V2</i> to make the functions provided by <i>bs_v2:BizService_V2</i> backwards compatible with those of <i>bs_v1:BizService_V1</i> .	indirect	high	<i>bsp1:BizServiceProxy</i>	CC3009
			indirect	high	<i>sm:SalesManagement</i>	CC3011
CC3009	Modify <i>bsp1:BizServiceProxy</i> 's binding with old <i>bizService</i> function to <i>bizServiceRecovery</i> function.	<i>bsp1:BizServiceProxy</i> no longer accessing <i>bs_v1:BizService_V1</i> directly. Instead, <i>bsp1:BizServiceProxy</i> accesses <i>bs_v1:BizService_V1</i> functions via <i>rsrb:BizServiceRecoveryBlock</i> .	direct	low	[self]	
CC3010	Replace <i>SqlServer</i> database driver in <i>boa_v1:BackOfficeAdmin_V1</i> with that for <i>MySql</i> database.	Change to <i>boa_v1:BackOfficeAdmin_V1</i> required for it to communicate with new data store type (<i>MySQL</i>) and instance (<i>ds_v2:MySqlDatastore</i>)	direct	high	[self]	
CC3011	Modify <i>sm:SalesManagement</i> 's binding with old <i>bizService</i> function to <i>bizServiceRecovery</i> function.	<i>sm:SalesManagement</i> no longer accessing <i>bs_v1:BizService_V1</i> directly. Instead, <i>sm:SalesManagement</i> accesses <i>bs_v1:BizService_V1</i> functions via <i>rsrb:BizServiceRecoveryBlock</i> .	direct	low	[self]	
Applied to generation V1.1beta						
CC4001b	Remove <i>SqlServer</i> persistence support from Repository zone.	<i>ds_v1:SqlServerDatastore</i> no longer required	direct	high	<i>bs_v1:BizService_V1</i>	CC4002b
CC4002b	Remove old version of <i>bizService</i> function from Repository zone.	<i>bs_v1:BizService_V1</i> no longer required	direct	high	<i>rsrb:BizServiceRecoveryBlock</i>	CC4006
CC4006	Remove <i>bizServiceRecovery</i> function from Repository zone.	<i>bsf:BizService1to2Facade</i> working and <i>rsrb:BizServiceRecoveryBlock</i> no longer required	direct	high	<i>bsp1:BizServiceProxy</i>	CC3012
			direct	high	<i>sm:SalesManagement</i>	CC3013

Change Case			Impact			
ID	Purpose	Description	Type	Disruption Level	Target	Supplementary Change Case ID
CC3012	Modify <i>bsp1:BizServiceProxy</i> 's binding with <i>bizServiceRecovery</i> function to facade for new version of <i>bizService</i> function.	<i>rsrb:BizServiceRecoveryBlock</i> no longer needed, <i>bsp1:BizServiceProxy</i> to directly access <i>bsf:BizService1to2Facade</i>	direct	low	[self]	
CC3013	Modify <i>sm:SalesManagement</i> 's binding with <i>bizServiceRecovery</i> function to facade for new version of <i>bizService</i> function.	<i>rsrb:BizServiceRecoveryBlock</i> no longer needed, <i>sm:SalesManagement</i> to directly access <i>bsf:BizService1to2Facade</i>	direct	low	[self]	
Applied to generation V1.1						
CC2000	Replace DotNetNuke valuer web site in Web zone with Tomcat valuer web site.	Web container platform migration from DotNetNuke to Tomcat	direct	high	[self]	
CC2001	Move <i>jobRequester</i> function from Internal zone to Web zone.	<i>jobRequester</i> relocation from Windows Remote Desktop to the Web zone	direct	low	[self]	
CC2002	Move <i>jobBookingView</i> function from Internal zone to Web zone.	<i>jobBookingView</i> relocation from Windows Remote Desktop to the Web zone	direct	low	[self]	
CC2003	Move <i>jobManagement</i> function from Internal zone to Web zone.	<i>jobManagement</i> relocation from Windows Remote Desktop to the Web zone	direct	low	[self]	
CC2004	Move notes function from Internal zone to Web zone.	Notes relocation from Windows Remote Desktop to the Web zone	direct	low	[self]	
CC1001a	Replace old version of MJA in 20 PDAs (<i>pa[1..20]_v1:MobileJobApp_V1</i>) with new version.	bug fixes for MJA	direct	high	[self]	
CC1001b	Add new version of MJA to 80 PDAs.	More valuers for the new version of MJA	direct	high	[self]	
CC1002	Add <i>jobHistory</i> function to MJA in 20 PDAs (<i>pa[1..20]_v1:MobileJobApp_V1</i>).	New feature <i>jobHistory</i> to add to MJA	direct	high	[self]	
CC2005	Add Tomcat valuer web site to Web zone.	See CC2000	direct	low	[self]	
CC2006	Add <i>jobWorker</i> function to Tomcat valuer web site (<i>endUserWeb2:TomcatWebContainer</i>).	See CC2000	direct	low	[self]	
CC2007	Add <i>jobRequester</i> function to Tomcat valuer web site (<i>endUserWeb2:TomcatWebContainer</i>).	See CC2001, CC2002, CC2003 and CC2004	direct	low	[self]	

Change Case			Impact			
ID	Purpose	Description	Type	Disruption Level	Target	Supplementary Change Case ID
CC2008	Add new PDA facing web site to Web zone.	New web container to host pdaWeb1: such that versions V1 and V2 of ja_v1:JobAppointmentWS_V1 can run in parallel (i.e. in pdaWeb1: and pdaWeb2:)	direct	low	[self]	
CC2010	Add new version of jobAppointment function to new PDA facing web site (pdaWeb2:TomcatWebContainer).	New version	direct	low	[self]	
CC2009	Move jobFulfilment function from old PDA facing web site (pdaWeb1:TomcatWebContainer) to new PDA facing web site (pdaWeb2:TomcatWebContainer).	jf:JobFulfilmentWS to stay, ja_v1:JobAppointmentWS_V1 to be removed together with pdaWeb1: later on	indirect	low	pdaWeb1	CC2013
			indirect	low	pdaWeb2	CC2014
			indirect	medium	wp1	CC2011
CC2011	Modify wp1:WebProxy's binding with jobFulfilment function in old PDA facing web site (pdaWeb1:TomcatWebContainer) to jobFulfilment function in new PDA facing web site (pdaWeb2:TomcatWebContainer).	jf:JobFulfilmentWS relocated from pdaWeb1: to pdaWeb2:	direct	low	[self]	
CC2012	Add new webProxy function to Web zone.	New	direct	low	[self]	
CC2013	Remove jobFulfilment function from old PDA facing web site (pdaWeb1:TomcatWebContainer).	See CC2009	direct	high	[self]	
CC2014	Add jobFulfilment function to new PDA facing web site (pdaWeb2:TomcatWebContainer).	See CC2009	direct	low	[self]	
CC4004	Add new version of bizServiceProxy function to Internal zone.	New	direct	low	[self]	
Applied to generation V1.2						
CC3001	Remove old version of jobRequester function from Internal zone.	See CC2001, CC2002, CC2003 and CC2004	direct	high	[self]	
CC3002	Replace old version of jobAppointment function in DotNetNuke valuer web site (endUserWeb1:NukeWebContainer) with stub function.	Temporary stub to always return nil job appointments to valuers	direct	high	[self]	

<i>Change Case</i>			<i>Impact</i>			
<i>ID</i>	<i>Purpose</i>	<i>Description</i>	<i>Type</i>	<i>Disruption Level</i>	<i>Target</i>	<i>Supplementary Change Case ID</i>
<i>Applied to generation v1.3</i>						
<i>CC2015</i>	<i>Remove old version of webProxy function from Web zone.</i>	<i>Obsolete</i>	<i>direct</i>	<i>low</i>	<i>[self]</i>	
CC2016	Remove old PDA facing web site (which also removes jobAppointment stub function) from Web zone.	See CC1001a, CC1001b and CC1002	indirect	medium	wp1:WebProxy	CC2015
CC2017	Remove DotNetNuke valuer web site (which also removes old version of jobWorker function) from Web zone.	See CC2000	indirect	medium	wp1:WebProxy	CC2015
CC2018	Remove old version of MJA (pa[1..20]_v1:MobileJobAp p_V1) from 20 PDAs.	Instances of old version of MJA to be removed	indirect	medium	wp1:WebProxy	

Note: *Italicised* change cases were identified in Task “Refine Change Cases” in Process “Transformation Identification”.

D.1.3 Application Lifecycle

It was decided in the case study meetings that in order to cope with the business constraints stated in Appendix D.1.2, DPV will have to pass through a few temporary generations (V1.1beta, V1.1, V1.2 and V1.3) when it progresses from V1 to V2. The initial version of the resultant application lifecycle produced from Process “Application Lifecycle Analysis” thus becomes what is depicted in Figure Appendix D.2.

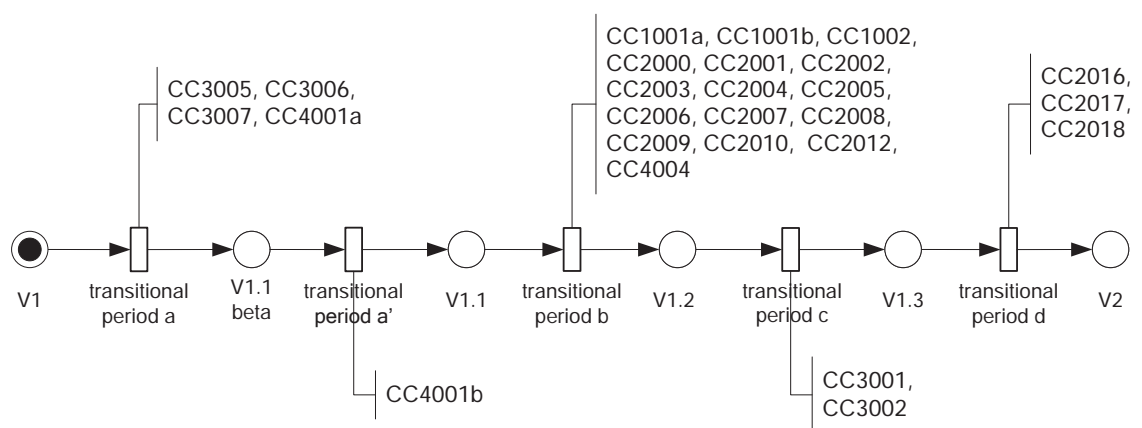
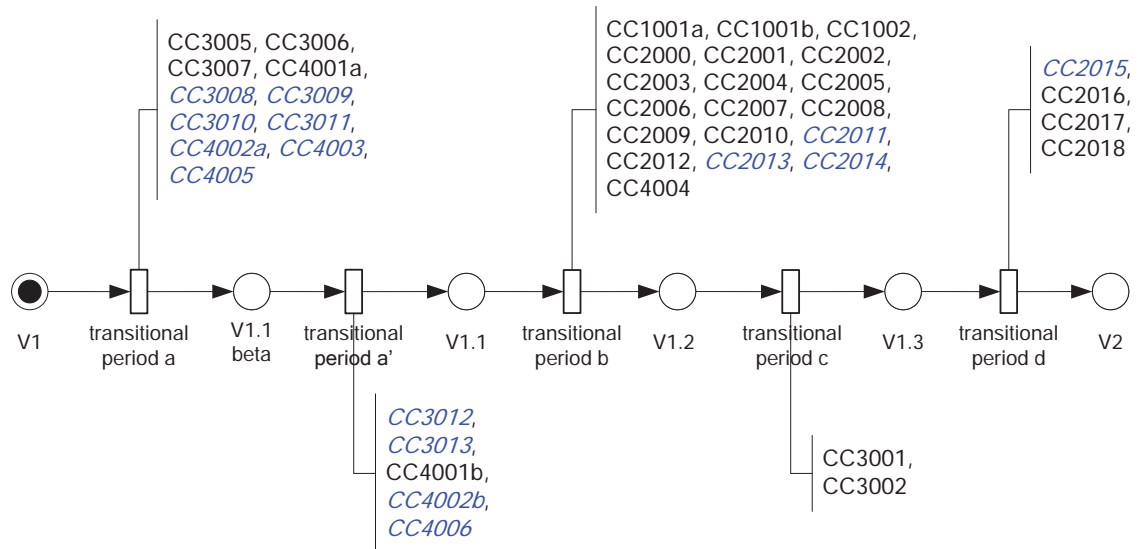


Figure Appendix D.2 DPV: application lifecycle diagram before process “Transformation Identification”

The resultant application lifecycle was subsequently updated to accommodate the new change cases identified in Process “Transformation Identification” (cf. Appendix D.2). The lifecycle is depicted in Figure Appendix D.3.



Note: Change cases identified in the "Transformation Identification" process are *italicised*.

Figure Appendix D.3 DPV: updated application lifecycle diagram after new change cases identified in Process "Transformation Identification"

The generations of DPV are described in Table Appendix D.3.

Table Appendix D.3 DPV: identified generations

Generation	Description
V1	The "as-is" version of DPV.
V1.1beta	The new database platform and its associated transformable items <i>released</i> . The old database platform and its associated transformable items on standby as a fall back. <ul style="list-style-type: none"> All V1 features still available to end users (i.e. no noticeable changes or differences to valuers).
V1.1	The old database platform and its associated transformable items <i>removed</i> , given that the new database platform and its associated transformable items operate satisfactorily. <ul style="list-style-type: none"> All V1 features still available to end users (i.e. no noticeable changes or differences to valuers).
V1.2	All new features (i.e. V2, including bug fixes) <i>released</i> . Existing V1 features still remain in DPV. <ul style="list-style-type: none"> Both V1 and V2 features available to end users. Trained valuers starting to use V2 features if desired.
V1.3	Old version (i.e. V1) of valuation request features <i>removed</i> . <ul style="list-style-type: none"> Lenders to use the new features in V2 to submit valuation requests via the Internet. Old version (i.e. V1) of valuation booking features <i>removed</i> . <ul style="list-style-type: none"> Valuers to use V2 part of DPV to download valuation booking information. However, they continue to use the V1 part of DPV to submit existing valuations from which the bookings were created with the V1 part of DPV.
V2	All V1 parts of DPV <i>removed</i> since they are no longer required. <ul style="list-style-type: none"> All valuation requests booked with the V1 part of DPV fulfilled.

D.2 TRANSFORMATION IDENTIFICATION OUTCOMES

The Transformation Identification process identified the transformations required to realise the change cases during the five transitional periods between generations V1

and V2 of DPV. It produced the following artefacts:

- from Task “Define To-Be Runtime Structure”, the runtime structures of the generations based on the initial set of identified change cases (i.e. those shown in non-italic font in Table Appendix D.2) and the purposes of the generations (cf. Table Appendix D.3);
- from Task “Refine Change Cases”, new change cases derived from the differences between two successive generations (i.e. “as-is” and “to-be”). These change cases were in addition to those identified in Process “Application Lifecycle Analysis”; and
- from Task “Identify Transformations”, the transformations for producing the runtime structure of each “to-be” generation from its “as-is” generation.

The additional change cases are already described in Table Appendix D.2. The runtime structures and transformations are presented in the rest of this section.

After transitional period a, when DPV has progressed from V1 to V1.1beta:

The resulting generation V1.1beta is depicted in Figure Appendix D.4. In the Repository zone, a new data store “ds_v2:MySqlDatastore” and a number of transformable items giving access to the data store are installed. Particular attention should be given to “bsf:BizService1to2Facade” which resides in the Repository zone. It aims to keep the services provided by the new “bs_v2:BizService_V2” backwards compatible with existing transformable items (e.g. “jf:JobFulfilmentWS” in the Web zone), and enables DPV to offer both the old (V1) and new (V2) functionality to users from generation V1.2 later on (cf. Table Appendix D.3).

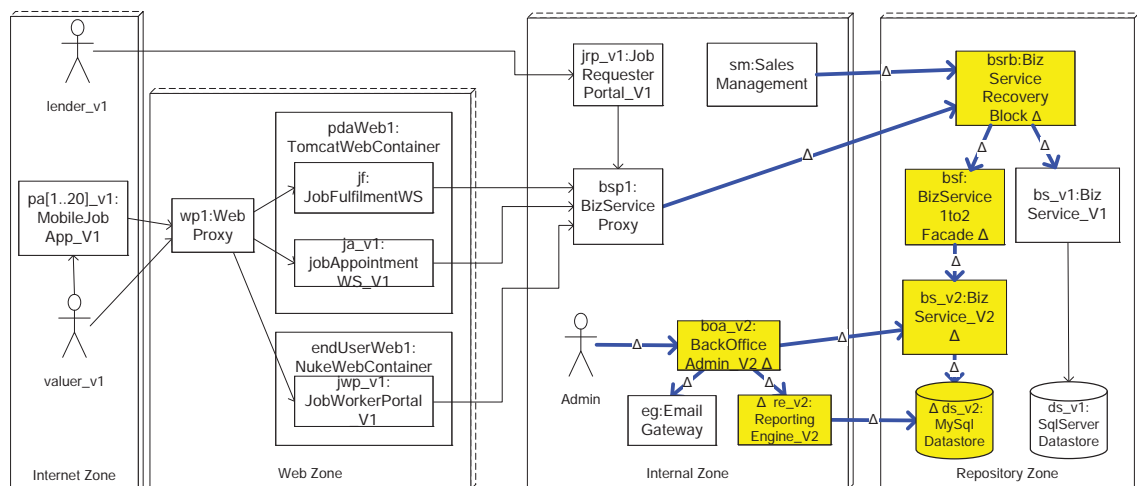


Figure Appendix D.4 DPV: generation V1.1beta

Since it is critical for DPV to continue to offer V1 functionality, “bsrb:BizServiceRecoveryBlock” will ensure that if “bsf:BizService1to2Facade” cannot correctly provide V1 functionality through its interfaces, “bsrb:BizServiceRecoveryBlock” will have the opportunity to switch to and use “bs_v1:BizService_V1” as a backup for DPV to continue to operate. In the Internal zone, “BackOfficeAdmin” and “ReportingEngine” are upgraded with new features and the ability to use the new data store in the Repository zone.

The transformations identified to bring DPV to the V1.1beta generation are listed in Table Appendix D.4. Notice the evidence of the many-to-many relationship between change cases and transformations: one transformation “boa_v1:BackOfficeAdmin_V1 upgrade” to three change cases CC3005, CC3006 and CC3007, and three transformations “ds_v2:MySqlDatastore deployment”, “ds_v1: to ds_v2: data replication” and “ds_v1: to ds_v2: data sync” to one change case CC4001a.

Table Appendix D.4 DPV: change cases for progressing V1 to V1.1beta

<i>Change Case ID</i>	<i>Change Case Purpose</i>	<i>Enactment (i.e. Responsible Transformation(s))</i>
CC3005	Replace clientManagement function in boa_v1:BackOfficeAdmin_V1 with new version.	boa_v1:BackOfficeAdmin_V1 upgrade
CC3006	Replace jobReporting function in boa_v1:BackOfficeAdmin_V1 with new version.	
CC3007	Replace jobBookingAssignment function in boa_v1:BackOfficeAdmin_V1 with new version.	
CC4001a	Add MySql persistence support to Repository zone.	ds_v2:MySqlDatastore deployment, ds_v1: to ds_v2: data replication, ds_v1: to ds_v2: data sync. Removal of ds_v1:SqlServerDatastore is dealt with in CC4001b, Table Appendix D.5
CC4002a	Add new version of bizService function to Repository zone.	bs_v2:BizService_V2 deployment. Removal of bs_v1:BizService_V1 is dealt with in CC4002b, Table Appendix D.5
CC4003	Replace SqlServer database driver in re_v1:ReportingEngine_V1 with MySql database driver.	re_v2:ReportingEngine_V2 deployment, re_v1:ReportingEngine_V1 removal
CC4005	Add bizServiceRecovery function to Repository zone.	bsrb:BizServiceRecoveryBlock deployment
CC3008	Add facade for new version of bizService function to Internal zone.	bsf:BizService1to2Facade deployment
CC3009	Modify bsp1:BizServiceProxy's binding with old bizService function to bizServiceRecovery function.	bsp1:BizServiceProxy rebinding1
CC3010	Replace SqlServer database driver in boa_v1:BackOfficeAdmin_V1 with MySql database driver.	boa_v1:BackOfficeAdmin_V1 upgrade

Change Case ID	Change Case Purpose	Enactment (i.e. Responsible Transformation(s))
CC3011	<i>Modify sm:SalesManagement's binding with old bizService function to bizServiceRecovery function.</i>	<i>sm:SalesManagement rebinding1</i>

Note: Change cases identified from Task "Refine Change Cases" are *italicised*.

After transitional period a', when DPV has progressed from V1.1beta to V1.1:

The resulting generation V1.1 is depicted in Figure Appendix D.5. The transformations to appear in transitional period a' remove Microsoft® SQL Server specific parts (i.e. "ds_v1:SqlServerDatastore" and "bs_v1:BizService_V1") plus "bsrb:BizServiceRecoveryBlock" from the Repository zone, after ensuring that "bsf:BizService1to2Facade" has been successful in providing the old (V1) functionality in generation V1.1beta. See Table Appendix D.5 for the transformations identified to promote DPV to generation V1.1.

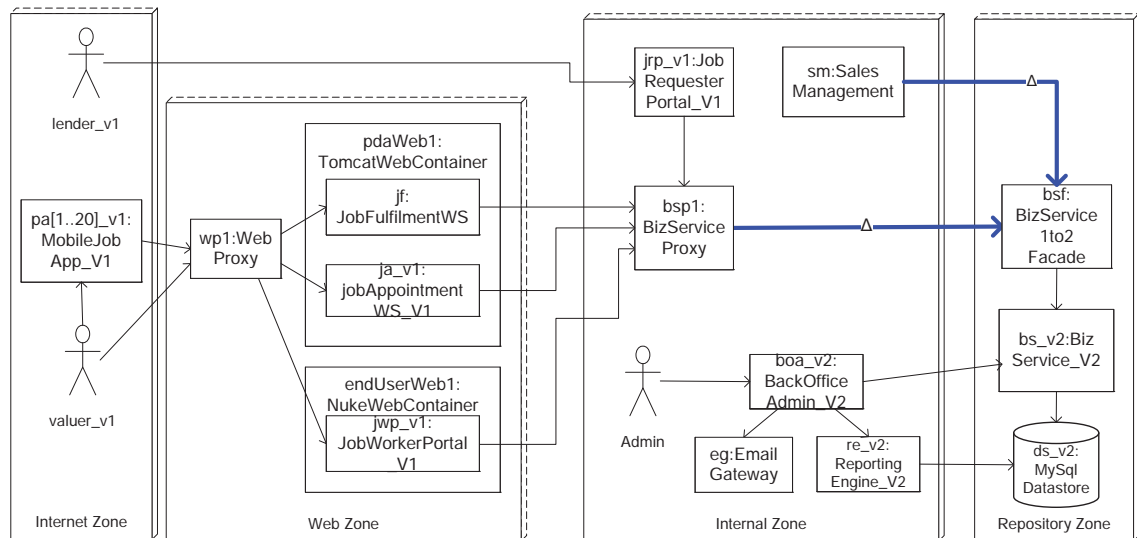


Figure Appendix D.5 DPV: generation V1.1

Table Appendix D.5 DPV: change cases for progressing V1.1beta to V1.1

Change Case ID	Change Case Purpose	Enactment (i.e. Responsible Transformation(s))
CC4001b	Remove SqlServer persistence support from Repository zone.	ds_v1:SqlServerDatastore removal
CC4002b	<i>Remove old version of bizService function from Repository zone.</i>	<i>bs_v1:BizService_V1 removal</i>
CC4006	<i>Remove bizServiceRecovery function from Repository zone.</i>	<i>rsrb:BizServiceRecoveryBlock removal</i>
CC3012	<i>Modify bsp1:BizServiceProxy's binding with bizServiceRecovery function to facade for new version of bizService function.</i>	<i>bsp1:BizServiceProxy rebinding2</i>
CC3013	<i>Modify sm:SalesManagement's binding with bizServiceRecovery function to facade for new version of bizService function.</i>	<i>sm:SalesManagement rebinding2</i>

Note: Change cases identified from Task "Refine Change Cases" are *italicised*.

After transitional period b, when DPV has progressed from V1.1 to V1.2:

The resulting generation V1.2 is depicted in Figure Appendix D.6. The transformations to appear in transitional period b update transformable items in the Web zone to offer the new (V2) functionality of DPV to valuers, while keeping the old (V1) functionality available (cf. Table Appendix D.3). They are listed in Table Appendix D.6.

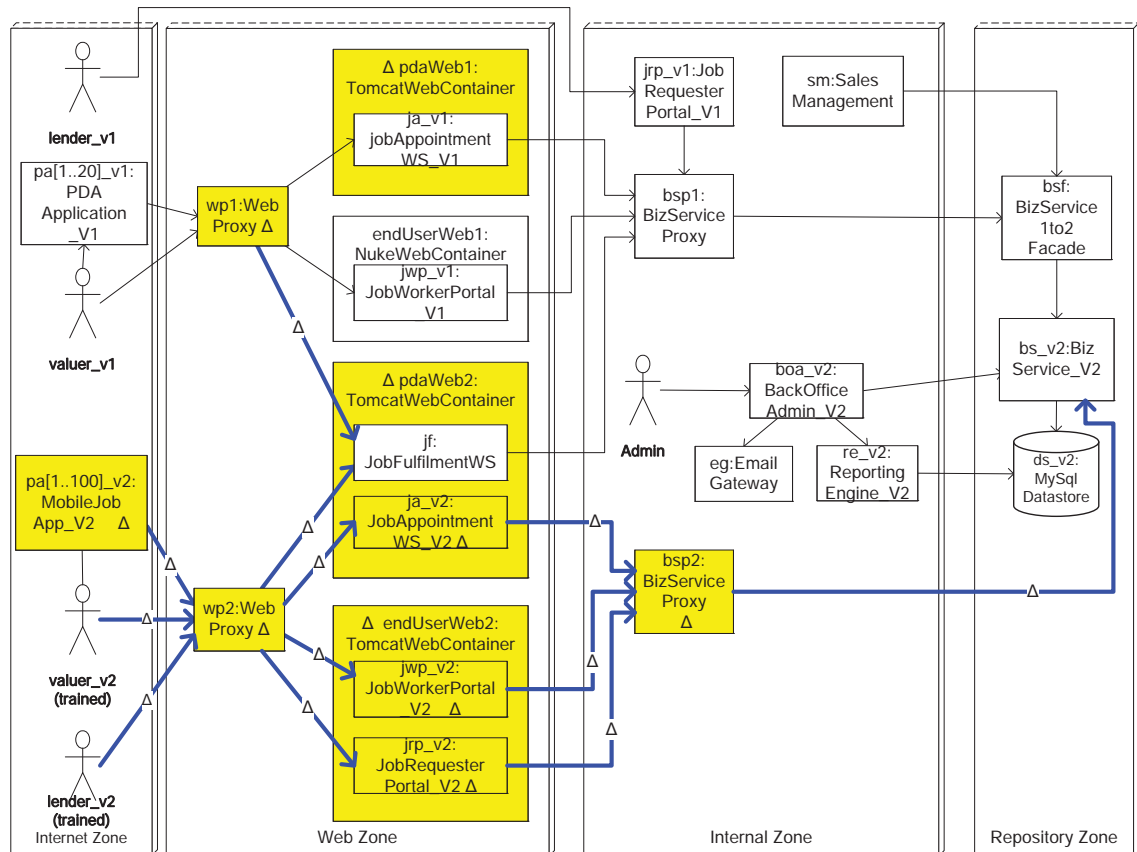


Figure Appendix D.6 DPV: generation V1.2

Table Appendix D.6 DPV: change cases for progressing V1.1 to V1.2

Change Case ID	Change Case Purpose	Enactment (i.e. Responsible Transformation(s))
CC2000	Replace DotNetNuke valuer web site in Web zone with Tomcat valuer web site.	endUserWeb2:TomcatWebContainer deployment (removal of endUserWeb1:NukeWebContainer is dealt with in CC2012)
CC2001	Move jobRequester function from Internal zone to Web zone.	jrp_v2:JobRequesterPortal_V2 deployment (removal of jrp_v1:JobRequesterPortal_V1 is dealt with in CC3001)
CC2002	Move jobBookingView function from Internal zone to Web zone.	
CC2003	Move jobManagement function from Internal zone to Web zone.	
CC2004	Move notes function from Internal zone to Web zone.	

<i>Change Case ID</i>	<i>Change Case Purpose</i>	<i>Enactment (i.e. Responsible Transformation(s))</i>
CC1001a	Replace old version of MJA in 20 PDAs with new version.	pa[1..100]_v2:MobileJobApp_V2 deployment. (pa[1..20]_v1:MobileJobApp_V1 removal dealt with in CC2013)
CC1001b	Add new version of MJA to 80 PDAs.	
CC1002	Add jobHistory function to MJA in 20 PDAs.	pa[1..100]_v2:MobileJobApp_V2 deployment
CC2005	Add Tomcat valuer web site to Web zone.	endUserWeb2:TomcatWebContainer deployment
CC2006	Add jobWorker function to Tomcat valuer web site.	jwp_v2:JobWorkerPortal_V2 deployment
CC2007	Add jobRequester function to Tomcat valuer web site.	jrp_v2:JobRequesterPortal_V2 deployment
CC2008	Add new PDA facing web site to Web zone.	pdaWeb2:TomcatWebContainer deployment
CC2010	Add new version of jobAppointment function to new PDA facing web site.	ja_v2:JobAppointmentWS_V2 deployment
CC2009	Move jobFulfilment function from old PDA facing web site to new PDA facing web site.	jf:JobFulfilmentWS removal (from pdaWeb1); jf:JobFulfilmentWS deployment (to pdaWeb2)
<i>CC2011</i>	<i>Modify wp1:WebProxy's binding with jobFulfilment function in old PDA facing web site to jobFulfilment function in new PDA facing web site.</i>	<i>wp1:WebProxy reconfiguration</i>
CC2012	Add new webProxy function to Web zone.	wp2:WebProxy deployment
<i>CC2013</i>	<i>Remove jobFulfilment function from old PDA facing web site.</i>	<i>see CC2009</i>
<i>CC2014</i>	<i>Add jobFulfilment function to new PDA facing web site.</i>	<i>see CC2009</i>
CC4004	Add new version of bizSeviceProxy function to Internal zone.	bsp2:BizServiceProxy deployment

Note: Change cases identified from Task "Refine Change Cases" are *italicised*.

After transitional period c, when DPV has progressed from V1.2 to V1.3:

The resulting generation V1.3 is depicted in Figure Appendix D.7. The transformations to appear in transitional period c aim to stop lenders and valuers from creating and managing valuations using the old (V1) user interfaces and functionality (cf. Table Appendix D.3). The transformations are listed in Table Appendix D.7.

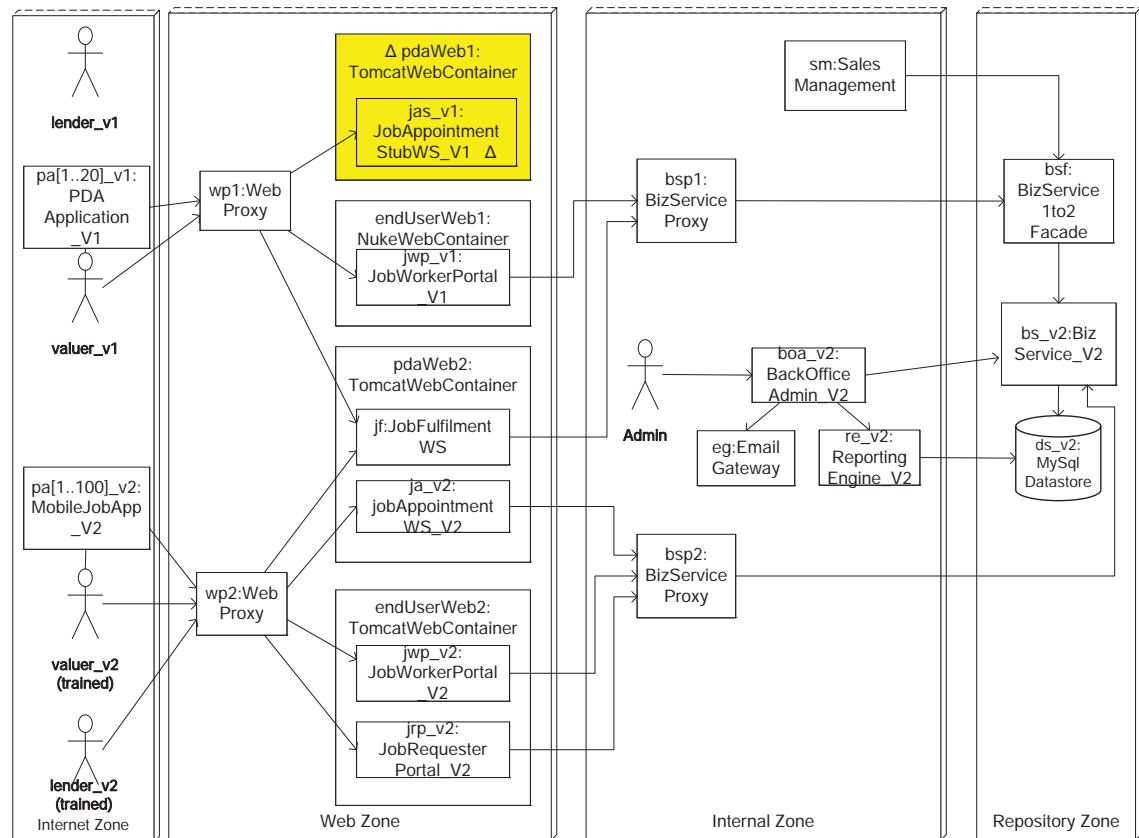


Figure Appendix D.7 DPV: generation V1.3

Table Appendix D.7 DPV: change cases for progressing V1.2 to V1.3

Change Case ID	Change Case Purpose	Enactment (i.e. Responsible Transformation(s))
CC3001	Remove old version of jobRequester function from Internal zone.	jrp_v1:JobRequesterPortal_V1 removal
CC3002	Replace old version of jobAppointment function in DotNetNuke valuer web site with stub function.	ja_v1:JobAppointmentWS_V1 replacement with jas_v1:JobAppointmentStubWS_V1

After transitional period d, when DPV progressed from generation V1.3 to V2

The resulting generation V2 is depicted in Figure Appendix D.8. The transformations to appear in transitional period c remove transformable items which provide the old (V1) functionality to valuers (cf. Table Appendix D.3). The transformations are listed in Table Appendix D.8.

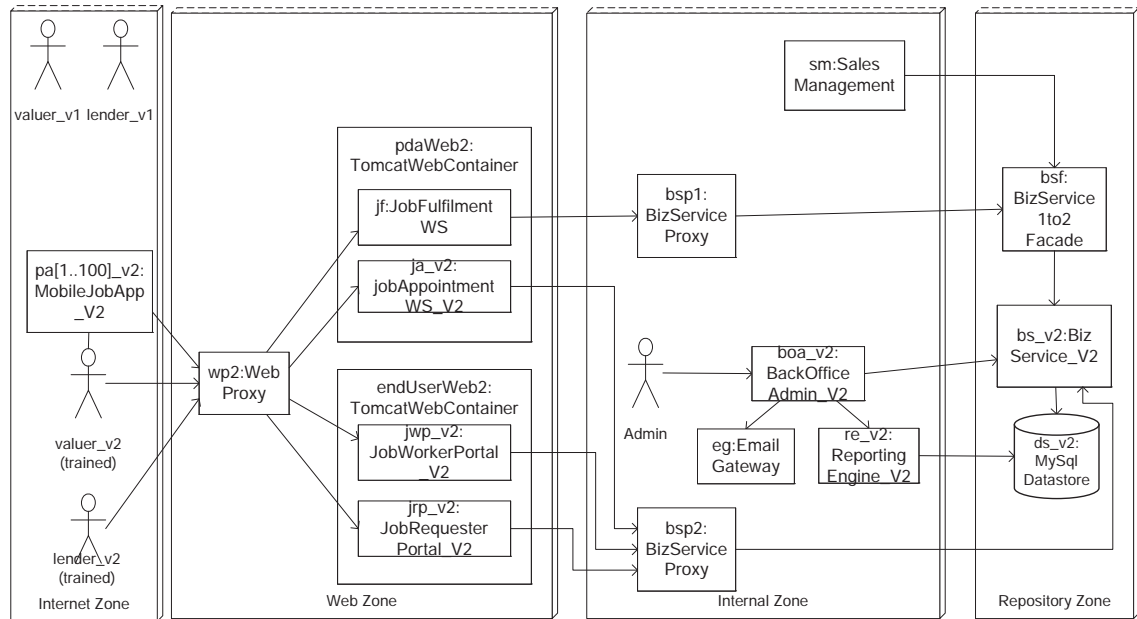


Figure Appendix D.8 DPV: generation V2

Table Appendix D.8 DPV: change cases for progressing V1.3 to V2

Change Case ID	Change Case Purpose	Enactment (i.e. Responsible Transformation(s))
CC2015	Remove old version of webProxy function from Web zone.	wp1:WebProxy removal
CC2016	Remove old PDA facing web site (which also removes jobAppointment stub function) from Web zone.	pdaWeb1:TomcatWebContainer removal
CC2017	Remove DotNetNuke valuer web site (which also removes old version of jobWorker function) from Web zone.	endUserWeb1:NukeWebContainer removal
CC2018	Remove old version of MJA from 20 PDAs.	pa[1..20]_v1:MobileJobApp_V1 removal

Note: Change cases identified from Task "Refine Change Cases" are *italicised*.

D.3 TRANSFORMATION AGENT DESIGN OUTCOMES

The Transformation Agent Design process identified the transformation agents required to perform a set of transformations for DPV and determine how they will collaborate with one another to perform the transformations. With respect to the agent identification in the case study (i.e. Task "Identify Transformation Agents"), Figure Appendix D.9 depicts the disposition of transformation agents in various zones where DPV's transformable items reside.

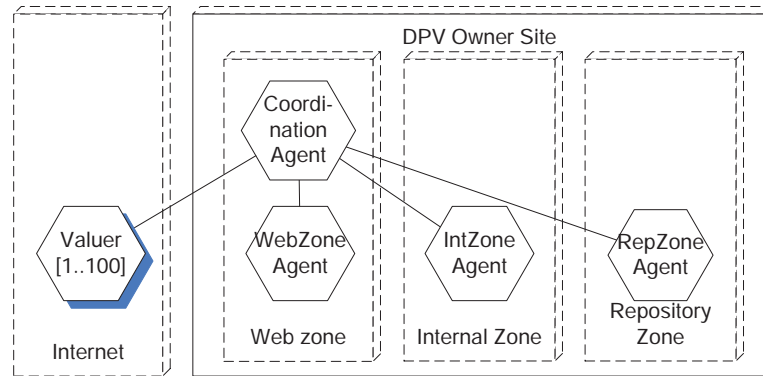


Figure Appendix D.9 DPV: disposition of transformation agents in different zones

Each of the Web, Internal and Repository Zones is assigned a dedicated agent responsible for transformations within that zone. In the Internet zone, there are two options to upgrade MJA running on valuers' PDAs:

1. Each valuer manually installs version V2 of MJA to his/her PDA, and removes version V1 of MJA, if any, from his/her PDA.
2. Each valuer installs a transformation agent, a custom built software, on his/her PDA and let it automatically upgrade MJA from V1 to V2.

The case study sponsor opted for option 1 since it was more economical and feasible. This option makes each valuer play the role of a transformation agent. Hence, the transformation agents in the Internet zone are labelled "Valuer" in Figure Appendix D.9. Finally, a "Coordination Agent" is placed in the Web zone to coordinate transformation activities among the agents in the four zones.

With respect to the second outcome of the Transformation Agent Design process (i.e. using Task "Define Transformation Orchestration"), Figure Appendix D.10 to Figure Appendix D.14 depict the collaboration of transformation agents to perform the set of transformations, in terms of the orchestration of transformations during the transitional periods $a(V1 \rightarrow V1.1\text{beta})$, $a'(V1.1\text{beta} \rightarrow V1.1)$, $b(V1 \rightarrow V1.2)$, $c(V1.2 \rightarrow V1.3)$, and $d(V1.3 \rightarrow V2)$ respectively.

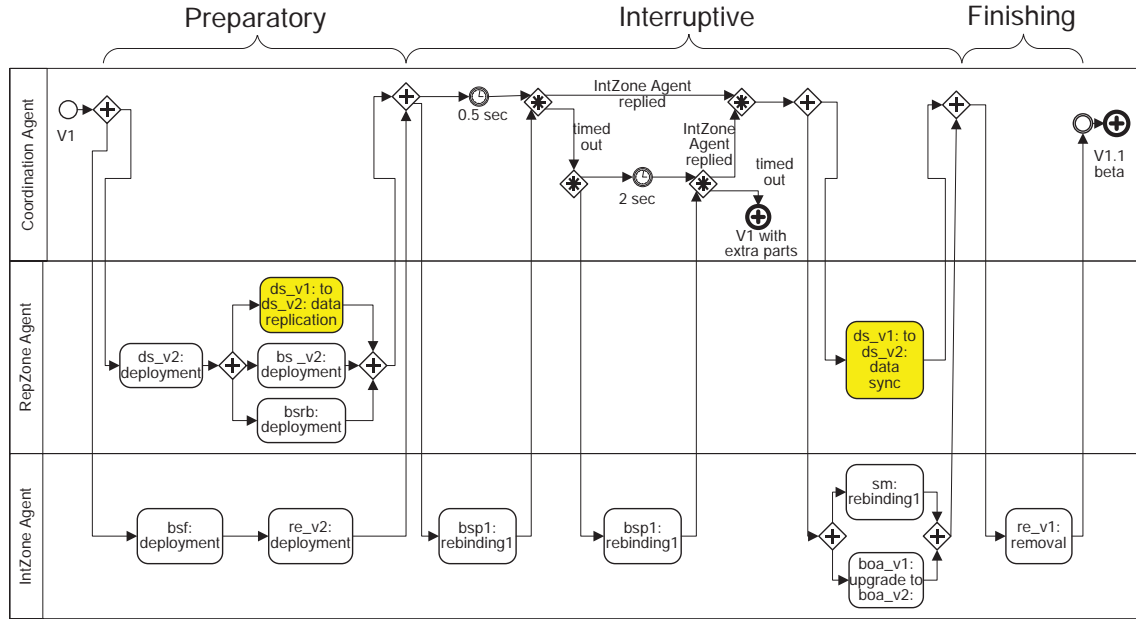


Figure Appendix D.10 DPV: transformation orchestration diagram (V1 to V1.1beta)

Notes for Figure Appendix D.10: Highlighted transformations are product specific. The transformation “bsp1: rebinding1” issued from the Coordination Agent to the IntZone Agent will be *retried* once if the latter does not respond within a specified time (0.5 second). If still unsuccessful, this transitional period will terminate DPV in an error state and DPV will have extra transformable items (which are “ds_v2:MySQLDatastore”, “bs_v2:BizService_V2” and “bsrb:BizServiceRecoveryBlock”) installed but not used. The error will be logged and investigated. However, manual handling of the error is deemed out of the scope for this case study.

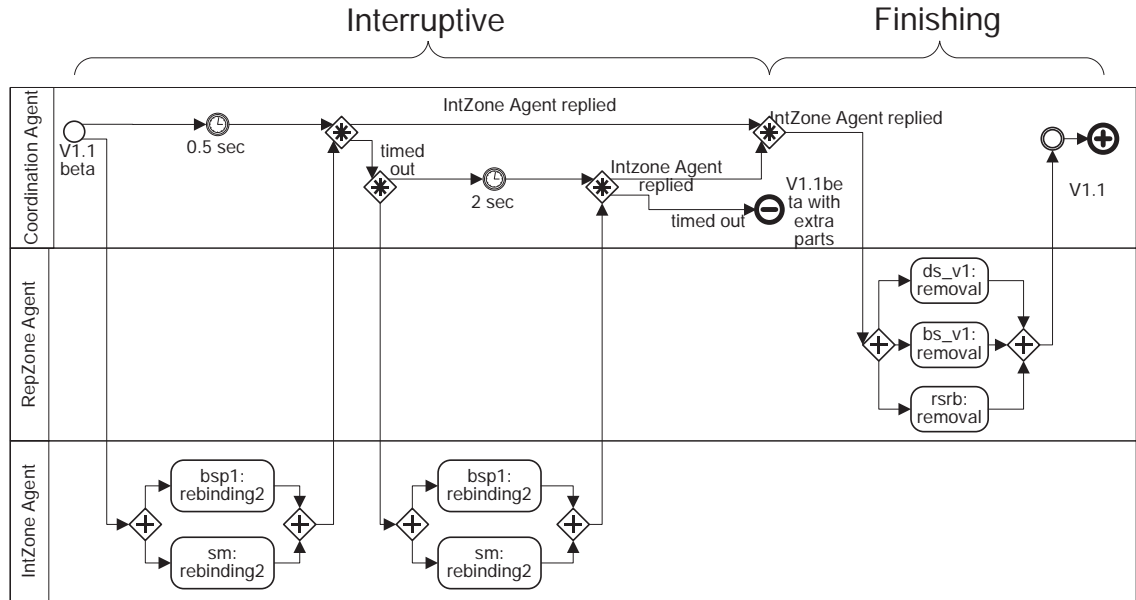


Figure Appendix D.11 DPV: transformation orchestration diagram (V1.1beta to V1.1)

Notes for Figure Appendix D.11: A retry similar to Figure Appendix D.10 may occur between the Coordination and the IntZone Agents to re-perform the critical transformation “bsp1: rebinding2”.

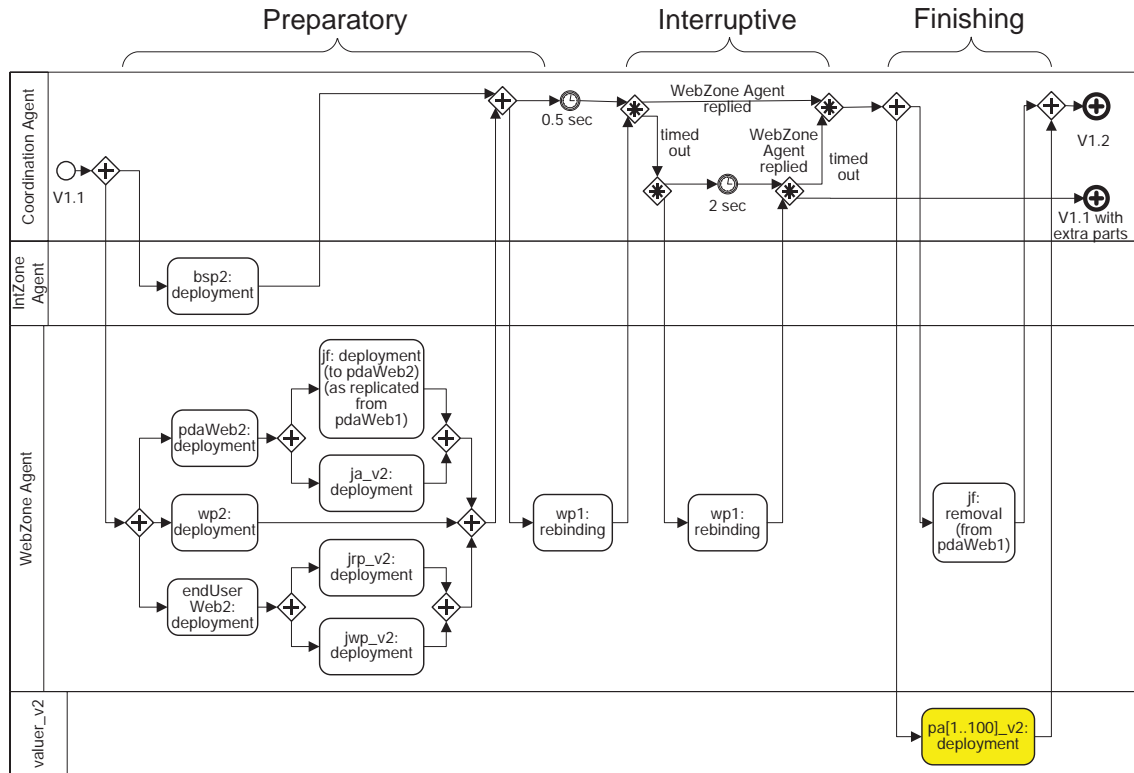


Figure Appendix D.12 DPV: transformation orchestration diagram (V1.1 to V1.2)

Notes for Figure Appendix D.12:

1. The highlighted transformation will be performed by each of the one hundred valuers. This means each one will install version V2 of MJA on his/her PDA.
2. A retry similar to Figure Appendix D.10 may occur between the Coordination and the WebZone Agents to re-perform the critical transformation “wp1: rebinding”.

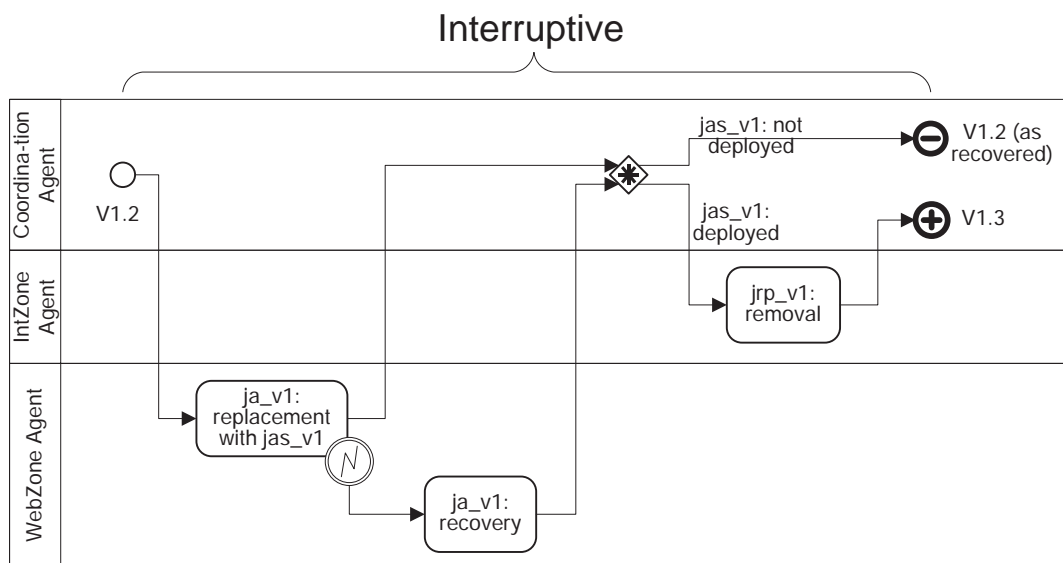


Figure Appendix D.13 DPV: transformation orchestration diagram (V1.2 to V1.3)

Notes for Figure Appendix D.13: The transformation “ja_v1:recovery” is specifically added to this design

to handle the case where the preceding transformation “ja_v1: replacement with jas_v1” fails to complete. This addition was done during dynamic evolution quality inspection (cf. Table Appendix D.14), and not directly derived from change cases (cf. Table Appendix D.7). If this failure happens, DPV will be rolled back to V1.2 and the root cause of this failure will be manually investigated and dealt with.

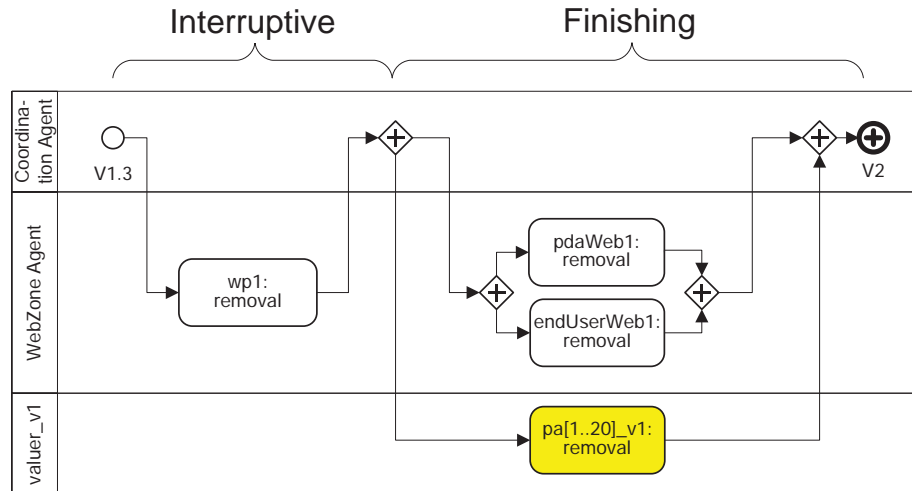


Figure Appendix D.14 DPV: transformation orchestration diagram (V1.3 to V2)

Notes for Figure Appendix D.14: The highlighted transformation is manually performed by individual valuers. In fact, it will be executed once by each of the twenty valuers whose PDA has the old version (V1) of MJA installed.

D.4 TRANSFORMATION DESIGN OUTCOMES

The Transformation Design process carried out the designing of individual transformations, producing the following outcomes for DPV:

- from Task “Identify New and Replacement Transformable Items”, the new and replacement transformable items to be added to DPV and their configurations (Appendix D.4.1);
- from Task “Identify Changes to Zones”, changes to zones hosting of transformable items of DPV (Appendix D.4.2);
- from Task “Develop Transformation”, the detailed design for each transformation (Appendix D.4.3); and
- from Task “Define Servicing Policies”, the policies to specify services provided by transformable items that will be affected by a transformation when it operates (Appendix D.4.3).

D.4.1 New and Replacement Transformable Items

Table Appendix D.9 summarises the new and replacement transformable items to be added to DPV, including:

- their new resource profiles (specifying resources that should be offered by their hosting environment) vs. those of the existing transformable items they intend to replace in DPV; and
- their new performance profiles vs. those of the existing transformable items they intend to replace in DPV; and
- the states from which they start to operate.

Table Appendix D.9 DPV: new and replacement transformable item catalogue

New and Replacement Transformable Items				
Responsible Transformation(s)	Name	Change in Resource Profile (current vs. new)	Change in Performance Profile (current vs. new)	Start-up State Configuration (see note 1)
<i>progressing from V1beta to V1.1</i>				
bsf:BizService1to2Facade deployment	bsf:BizService1to2Facade			N/A
bs_v2:BizService_V2 deployment	bs_v2:BizService_V2			Connection pool (holding 16 connections to ds_v2:MySQLDatastore) established
ds_v2:MySQLDatastore deployment	ds_v2:MySQLDatastore	Exclusive disk space: 500GB/10TB (current/new)		Loaded with up-to-date data obtained from ds_v1:SqlServerDatastore
bsrb:BizServiceRecoveryBlock deployment	bsrb:BizServiceRecoveryBlock			N/A
re_v1:ReportingEngine_V2 deployment	re_v1:ReportingEngine_V2			N/A
boa_v1:BackOfficeAdmin_V1 upgrade	boa_v2:BackOfficeAdmin_V2			N/A
<i>progressing from V1.1beta to V1.1</i>				
(none)				
<i>progressing from V1.1 to V1.2</i>				
endUserWeb2:TomcatWebContainer deployment	endUserWeb2:TomcatWebContainer	Exclusive memory: 512MB/4GB (current/new)	Minimum number of users: 20/100 (current/new)	N/A
jwp_v2:JobWorkerPortal_V2 deployment	jwp_v2:JobWorkerPortal_V2			N/A
jrp_v2:JobRequesterPortal_V2 deployment	jrp_v2:JobRequesterPortal_V2			N/A

<i>Responsible Transformation(s)</i>	<i>New and Replacement Transformable Items</i>			
	<i>Name</i>	<i>Change in Resource Profile (current vs. new)</i>	<i>Change in Performance Profile (current vs. new)</i>	<i>Start-up State Configuration (see note 1)</i>
pdaWeb2:TomcatWebContainer deployment	pdaWeb2:TomcatWebContainer	Exclusive memory: 256MB/2GB (current/new)	Minimum number of users: 20/100 (current/new)	N/A
ja_v2:JobAppointmentWS_V2 deployment	ja_v2:JobAppointmentWS_V2			N/A
jf:jobFulfilmentWS deployment (to pdaWeb2)	jf2:JobFulfilmentWS			N/A
wp2:WebProxy deployment	wp2:WebProxy			N/A
bsp2:BizServiceProxy deployment	bsp2:BizServiceProxy			N/A
pa[1..100]_v2:MobileJobApp_V2 deployment	pa[1..100]_v2:MobileJobApp_V2			N/A
<i>progressing from V1.2 to V1.3</i>				
ja_v1:JobAppointmentWS_V1 replacement with jas_v1:JobAppointmentStu bWS_V1	ja_v1:JobAppointmentStu bWS_V1 (temporary, to be removed in V2)			States mapped from ja_v1:JobAppointmentWS_V1 (see Table Appendix D.10)
<i>progressing from V1.3 to V2</i>				
(none)				

Notes:

1. "N/A" stands for "not applicable", which means the transformable item is stateless and/or no specific start-up state for the transformable item is needed.
2. Existing data need to be copied from the old instance (ds_v1:SqlServerDatastore) to the new instance (ds_v2:MySQLDatastore) before the new instance commences its operation.

As can be seen from Table Appendix D.9, there are three transformable items in which their start-up states must be defined:

- "bs_v2:BizService_V2", which provides streamlined access to the business functions, needs to set up the connections to the data store in its connection pool before it is ready for use, to reduce the latency caused by the set up. This is abstracted as a particular transformation action "«State»initialise" (cf. Figure Appendix C.11(a)) to be performed in the respective transformation (i.e. "bs_v2:BizService_V2 deployment").
- "ds_v2:MySQLDatastore", requires its data to be copied from the part it is replacing (i.e. "ds_v1:SqlServerDatastore"). Two transformations - "ds_v1: to ds_v2: data

replication” and “ds_v1: to ds_v2: data sync” - are dedicated to accomplish this (see Figure Appendix D.10).

- The case for replacing the “JobAppointmentWS” Web Service (i.e. “ja_v1:JobAppointmentWS_V1”) with its stub (i.e. “jas_v1:JobAppointmentStubWS_V1”) to deprecate its functions (cf. Table Appendix D.2) requires a State Map. More specifically, DPV is allowed to switch “JobAppointmentWS” to its stub only when “JobAppointmentWS” is in one of the two permissible states (“Ready_empty” and “Ready_pending”) during which it does not have any incoming requests for processing. The State Map is documented in Table Appendix D.10.

Table Appendix D.10 DPV: state map from JobAppointmentWS_V1 to its stub

Resolved State		Replacement State (of jas_v1:JobAppointmentStubWS_V1)			
		Startup	Ready_empty [M]	Ready_pending - [M]	Servicing
Old state (of ja_v1:JobAppointmentWS_V1)	Startup (i.e. not initialised):				
	Ready_empty - [QV] (i.e. without pending requests)		Ready_empty		
	Ready_pending - [QV] (i.e. with pending requests not yet served)			Ready_pending	
	Servicing - [QV] (i.e. requests being served)				

Note: Q/V/M: abbreviations for quiescent/resolved/resuming states.

D.4.2 Zone Changes

Table Appendix D.11 summarises changes to the respective zones as transformable items are added to and removed from DPV when it progresses from generation V1 to V2.

Table Appendix D.11 DPV: zone change document

Responsible Transformation(s)	External zone	Web zone	Internal zone	Repository zone
Progressing from V1 to V1.1beta				
bsf:BizService1to2Facade deployment				+bsf:BizService1to2Facade
bs_v2:BizService_V2 deployment				+bs_v2:BizService_V2

<i>Responsible Transformation(s)</i>	<i>External zone</i>	<i>Web zone</i>	<i>Internal zone</i>	<i>Repository zone</i>
ds_v2:MySQLDatastore deployment				+ds_v2:MySQLDatastore, allocation of additional disk space (cf. Table Appendix D.9)
bsrb:BizServiceRecoveryBlock deployment				+bsrb:BizServiceRecoveryBlock
re_v1:ReportingEngine_V2 deployment			+re_v1:ReportingEngine_V2	
boa_v1:BackOfficeAdmin_V1 upgrade			+boa_v2:BackOfficeAdmin_V2 - boa_v1:BackOfficeAdmin_V1	
re_v1:ReportingEngine_V1 removal	-		re_v1:ReportingEngine_V1	
<i>Progressing from V1.1beta to V1.1</i>				
ds_v1:SqlServerDatastore removal				- ds_v1:SqlServerDatastore
bs_v1:BizService_V1 removal				- bs_v1:BizService_V1
bsrb:BizServiceRecoveryBlock removal				- bsrb:BizServiceRecoveryBlock
<i>Progressing V1.1 to V1.2</i>				
endUserWeb2:TomcatWebContainer deployment		+endUserWeb2:TomcatWebContainer, allocation of additional memory (cf. Table Appendix D.9)		
jwp_v2:JobWorkerPortal_V2 deployment		+jwp_v2:JobWorkerPortal_V2		
jrp_v2:JobRequesterPortal_V2 deployment		+jrp_v2:JobRequesterPortal_V2		
pdaWeb2:TomcatWebContainer deployment		+pdaWeb2:TomcatWebContainer, allocation of additional memory (cf. Table Appendix D.9)		
ja_v2:JobAppointmentWS_V2 deployment		+ja_v2:JobAppointmentWS_V2		
jf:jobFulfilmentWS deployment (to pdaWeb2)		+jf2:JobFulfilmentWS		
wp2:WebProxy deployment		+wp2:WebProxy		
bsp2:BizServiceProxy deployment			+bsp2:BizServiceProxy	

<i>Responsible Transformation(s)</i>	<i>External zone</i>	<i>Web zone</i>	<i>Internal zone</i>	<i>Repository zone</i>
pa[1..100]_v2:MobileJobApp_V2 deployment	+pa[1..100]_v2:MobileJobApp_V2			
jf:JobFulfilmentWS removal (from pdaWeb1)		-jf:JobFulfilmentWS from pdaWeb1		
<i>Progressing V1.2 to V1.3</i>				
jrj_v1:JobRequesterPortal_V1 removal		-jrj_v1:JobRequesterPortal_V1		
ja_v1:JobAppointmentWS_V1 replacement with jas_v1:JobAppointmentStubWS_V1		-ja_v1:JobAppointmentWS_V1, +ja_v1:JobAppointmentStubWS_V1		
ja_v1:JobAppointmentWS_V1 recovery				
<i>Progressing V1.3 to V2</i>				
wp1:WebProxy removal		-wp1:WebProxy		
pdaWeb1:TomcatWebContainer removal		-pdaWeb1:TomcatWebContainer (and containing parts)		
endUserWeb1:NukeWebContainer removal		-endUserWeb1:NukeWebContainer (and containing parts)		
pa[1..20]_v1:MobileJobApp_V1 removal	-pa[1..20]_v1:MobileJobApp_V1			

Notations:

“+x”: add transformable item “x” to the zone identified by the column.

“-x”: remove transformable item “x” from the zone identified by the column.

D.4.3 Detailed Design of Transformations

When designing a transformation, it is useful to check if a transformation design pattern already exists as a solution to the design. Accordingly, Table Appendix D.12 documents the outcomes of the transformation pattern matching effort which links the transformations identified for DPV to the transformation patterns prescribed by Continuum (addition, removal, replacement etc., Appendices C.3.2.7 and C.3.2.8). Furthermore, a new transformation pattern called “tomcat addition” (cf. Figure Appendix D.15) is specifically designed to perform changes in a particular web container platform (i.e. Apache Tomcat). Indeed, “tomcat addition” is qualified as a pattern because it can be used for a few transformations in Table Appendix D.12.

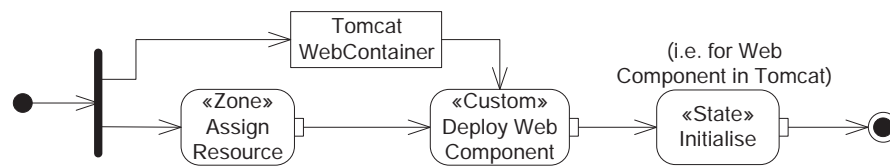


Figure Appendix D.15 DPV: transformation pattern for “tomcat addition”

Table Appendix D.12 DPV: applying transformation patterns to transformation design

Transitional Period	Responsible Transformation(s)	Transformation Pattern	Comments ²
V1 to V1.1beta	bsf: BizService1to2Facade deployment	addition	
	bs_v2: BizService_V2 deployment	addition	
	ds_v1: to ds_v2: data replication (i.e. from ds_v1:SqlServerDatastore to ds_v2:MySqlDatastore)	N/A ¹	
	ds_v2: MySqlDatastore deployment	addition	
	re_v1: ReportingEngine_V2 deployment	addition	
	boa_v1: BackOfficeAdmin_V1 upgrade	replacement	
	sm: SalesManagement rebinding1	rebinding	Rebinding with “blocked and queued” and “void” servicing policies (See Figure Appendix C.13(b))
	bsp1: BizServiceProxy rebinding1	rebinding	Ditto
	ds_v1: to ds_v2: data sync (i.e. from ds_v1:SqlServerDatastore to ds_v2:MySqlDatastore)	N/A	
	re_v1: ReportingEngine_V1 removal	removal	
V1.1beta to V1.1	ds_v1: SqlServerDatastore removal	removal	
	bs_v1: BizService_V1 removal	removal	
	bsrb: BizServiceRecoveryBlock removal	removal	
	sm: SalesManagement rebinding2	rebinding	Rebinding with “blocked and queued” and “void” servicing policies (See Figure Appendix C.13(b))
	bsp1: BizServiceProxy rebinding2	rebinding	Ditto
V1.1 to V1.2	endUserWeb2: TomcatWebContainer deployment	addition	
	jwp_v2: JobWorkerPortal_V2 deployment	tomcat addition	
	jrp_v2: JobRequesterPortal_V2 deployment	tomcat addition	
	pdaWeb2: TomcatWebContainer deployment	addition	
	ja_v2: JobAppointmentWS_V2 deployment	tomcat addition	
	jobFulfilmentWS deployment (to pdaWeb2)	addition	
	wp2: WebProxy deployment	addition	

<i>Transitional Period</i>	<i>Responsible Transformation(s)</i>	<i>Transformation Pattern</i>	<i>Comments²</i>
	bsp2: BizServiceProxy deployment	addition	
	wp1: WebProxy rebinding	rebinding	Rebinding with “blocked and queued” and “void” servicing policies (See Figure Appendix C.13(b))
	pa[1..100]_v2: MobileJobApp_V2 deployment	N/A	
	jf: JobFulfilmentWS removal (from pdaWeb1)	removal	
V1.2 to V1.3	jrp_v1: JobRequesterPortal_V1 removal	removal	Unavailability to be announced to users with “lender1” role
	ja_v1: JobAppointmentWS_V1 replacement with jas_v1: JobAppointmentStubWS_V1	addition, then rebinding, then removal (cf. Figure Appendix C.12)	Rebinding with “delegate” servicing policy (See Figure Appendix C.13(c))
V1.3 to V2	wp1: WebProxy removal	removal	Unavailability to be announced to users with “lender1” or “valuer1” roles
	pdaWeb1: TomcatWebContainer removal	removal	
	endUserWeb1: NukeWebContainer removal	removal	
	pa[1..20]_v1: MobileJobApp_V1 removal	N/A	

Notes:

1. “N/A” stands for “not applicable”.
2. The “Comments” column also records the servicing policies used in particular transformations with respect to the applied transformation patterns.

It should be noted from Table Appendix D.12 that there are a number of transformations for which no transformation design patterns are suitable. These transformations are individually discussed below:

- *ds_v1: to ds_v2: data replication* (cf. Figure Appendix D.10)

This transformation initialises “ds_v2:MySQLDatastore” with the data held in “ds_v1:SqlServerDatastore”. Although data evolution issues are not in scope for this research (Section 1.3), for completeness the following steps were determined during the case study to complete its design, which is shown in Figure Appendix D.16:

1. Set a checkpoint in “ds_v1:SqlServerDatastore”. A checkpoint is a point in time from which all changes to a database is logged. A follow-up transformation called “ds_v1: to ds_v2: data sync” is used to synchronise

“ds_v2:MySqlDatastore” with all updates to “ds_v1:SqlServerDatastore” since this checkpoint.

2. Export data from “ds_v1:SqlServerDatastore” before the check point.
3. Convert exported data to a format compatible with “ds_v2:MySqlDatastore”.
4. Import data to “ds_v2:MySqlDatastore”.

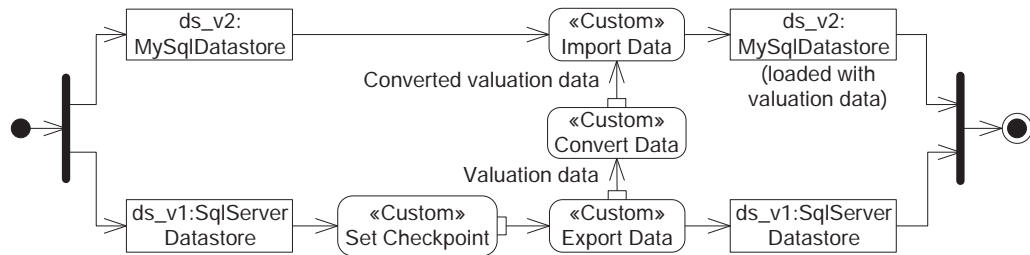


Figure Appendix D.16 DPV: transformation design for “ds_v1: to ds_v2: data replication”

- ds_v1: to ds_v2: data sync (cf. Figure Appendix D.10)

This transformation synchronises “ds_v2:MySqlDatastore” with all updates to “ds_v1:SqlServerDatastore” since the transformation “ds_v1: to ds_v2: data replication” was completed, before “ds_v2:MySqlDatastore” takes over the role of “ds_v1:SqlServerDatastore” in DPV. Similar to “ds_v1: to ds_v2: data replication”, data evolution issues are not in scope but for completeness the following steps were determined. The transformation design is shown in Figure Appendix D.17:

1. Export changes to data from “ds_v1:SqlServerDatastore” since the checkpoint set in “ds_v1:SqlServerDatastore”.
2. Convert exported data to a format compatible with “ds_v2:MySqlDatastore”.
3. Import data to “ds_v2:MySqlDatastore”.

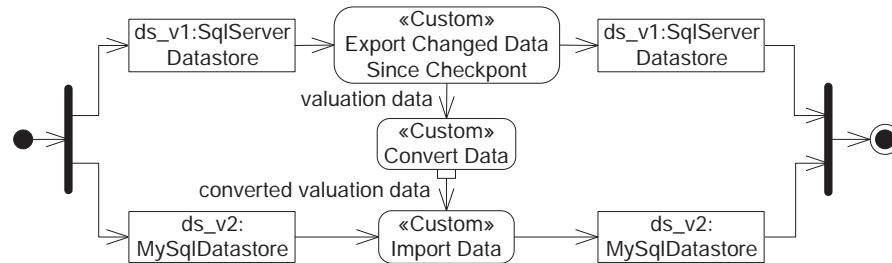


Figure Appendix D.17 DPV: transformation design for “ds_v1: to ds_v2: data sync”

- pa[1..100]_v2:MobileJobApp_V2 deployment (cf. Figure Appendix D.12)

This transformation installs the new version (V2) of MJA in one hundred PDAs. This falls into operational aspects and hence is not in scope for this research

(Section 1.3). However, the following steps were suggested by one case study participant:

1. Set up “MobileJobApp_V2” accessible for download from the Internet.
 2. Send email one hundred valuers to notify them that “MobileJobApp_V2” is available for download, and the instructions to install it.
 3. Each valuer downloads and installs “MobileJobApp_V2” to his/her PDA.
- *ja_v1:JobAppointmentWS_V1 recovery* (cf. Figure Appendix D.13)

This recovery transformation reverts DPV to generation V1.2 in case the transformation “ja_v1:JobAppointmentWS_V1 replacement with ja_v1:JobAppointmentStubWS_V1” fails. It has the following steps (as shown in Figure Appendix D.18) and is a variation of the “tomcat addition” pattern in Figure Appendix D.15:

1. Redeploy “ja_v1:JobAppointmentWS_V1” to “pdaWeb1:TomcatContainer” if the former has already been removed from the latter.
2. Reinstate the state of “ja_v1:JobAppointmentWS_V1” to one of its resolved states (as defined in the state map in Table Appendix D.10) prior to the transformation “ja_v1:JobAppointmentWS_V1 replacement with ja_v1:JobAppointmentStubWS_V1” was executed (cf. Figure Appendix D.13).



Figure Appendix D.18 DPV: transformation design for “ja_v1:JobAppointmentWS_V1 recovery”

- *pa[1..20]_v1:MobileJobApp_V1 removal* (cf. Figure Appendix D.14)

This transformation removes the old version (V1) of MJA from twenty valuers’ PDAs. This falls into operational aspects and is not in scope for this research (Section 1.3). However, for completeness the following steps were determined:

1. Inform the twenty valuers that the old version of MJA is obsolete, and send them instructions to uninstall it.
2. The valuers manually uninstall this version of MJA from their own PDAs.

A note about transformations “pa[1..100]_v2:MobileJobApp_V2 deployment” and “pa[1..20]_v1:MobileJobApp_V1 removal” - The number of valuers signed-up to use DPV

was twenty when V1 was released. In the original deployment project (i.e. upgrading DPV from V1 to V2), however, the number of valuers who signed-up to use the new version of DPV (i.e. V2) increased to one hundred but the project did not have time to address this growth. Therefore, the sponsor explicitly requested the case study to address this growth requirement as a business requirement. From the design perspective, the twenty valuers who already have the old version of MJA installed on their PDAs will have to perform “pa[1..100]_v2:MobileJobApp_V2 deployment” transformation to install the new version of MJA, and afterwards the “pa[1..20]_v1:MobileJobApp_V1 removal” transformation to rid the old version of MJA. For the eighty new valuers who signed-up to use V2 of DPV, they will simply install the new version of MJS by performing the “pa[1..100]_v2:MobileJobApp_V2 deployment” transformation.

D.5 DYNAMIC EVOLUTION QUALITY MANAGEMENT OUTCOMES

The Dynamic Evolution Quality Management process improved the quality of the dynamic evolution analysis and design artefacts produced for DPV. As described in Appendix C.3.1.2, this process iterates over four tasks: dynamic evolution quality definition, assessment, analysis and improvement. The execution of the definition task in the case study produced the rankings of dynamic evolution quality factors for DPV, in terms of their levels of importance as perceived by the case study participants. This is reported in Table Appendix D.13. Although the rankings could be used to prioritise in subsequent tasks which quality factors should be targeted for DPV, the sponsor endeavoured to investigate *all* quality factors in order to evaluate Continuum’s support for all these factors.

Table Appendix D.13 DPV: dynamic evolution quality profile

Quality Factor	Importance Score (1-lowest, 7-highest)	Rank
Recoverability	6.00	1
Completeness, Fault Tolerance, Reliability	5.50	2
Correctness	5.38	5
Coordination	5.25	6
Consistency	4.91	7
Separation of Concerns	4.90	8
Extensibility	4.83	9
Maintainability	4.63	10
Locality, Loose Coupling, Security	4.50	11
Transparency	3.63	14
Autonomy, Safety	3.50	15

After the definition task, the inspection task was carried out to identify defects/issues in the artefacts with respect to the dynamic evolution quality factors. The defects/issues were assessed in the assessment task to reveal their root causes, and then fixed, corrected and improved as per the assessment results in the improvement task. The defects/issues in the artefacts, their root causes and the changes carried out to resolve those defects/issues are documented in Table Appendix D.14.

Table Appendix D.14 DPV: dynamic evolution quality inspection, assessment and improvement results for defects/issues

Inspection		Assessment		Improvement
Artefact Type(s) to Inspect	Inspection Question(s) (all answered with "No")	If "No", describe all defects/issues	If "No", describe root cause for each defect/issue	If "No", propose a solution(s) to resolve the defect/issues
Consistency				
D	Protocols: Do transformable items use compatible protocols when communicating with one another after a transformation?	re_v1:ReportingEngine_V1 would not handle long report generation with MySQLDatastore.	The database driver used in re_v1:ReportingEngine_V1 (i.e. CrystalReport) is incompatible with the database version of MySQLDatastore.	Upgrade re_v1:ReportingEngine_V1 to re_v2:ReportingEngine_V2 which uses the correct driver version.
Locality				
D	Zoning and change localisation: Are changes local to a set of transformable items (e.g. in a zone) rather than global to an application?	Changes to the Repository zone are unlikely localised to within the Repository zone.	bsf:BizService1to2Facade, which is intended to encapsulate updates to the Repository zone, is hosted in the wrong zone (i.e. Internal).	Relocate bsf:BizService1to2Facade to the Repository zone. sm:SalesManagement should also directly bind to bsf:BizService1to2Facade instead of to ds_v2:MySQLDatastore.

<i>Inspection</i>		<i>Assessment</i>		<i>Improvement</i>
<i>Artefact Type(s) to Inspect</i>	<i>Inspection Question(s) (all answered with "No")</i>	<i>If "No", describe all defects/issues</i>	<i>If "No", describe root cause for each defect/issue</i>	<i>If "No", propose a solution(s) to resolve the defect/issues</i>
Coordination				
D	<i>Network:</i> Do transformation agents have a means of tolerating network unreliability when they coordinate with one another during a transitional period?	Disturbance of transformations in an interruptive phase may lead DPV to an undesirable state.	wp1: WebProxy binding, bsp1: BizServiceProxy rebinding1 and bsp1: BizServiceProxy rebinding2 are critical transformations but none deal with network unreliability.	Impose a time limit on the agents responsible for these transformations and retry the issuing of these transformations to their agents at least once.
Loose Coupling				
D	<i>Transformable items:</i> Are transformable items loosely coupled from one another?	Couplings between the Internet and Web zones, and between the Web and Internal zones need further improvement.	There are many bindings between the Internet and Web zones, and between the Web and Internal zones. This makes changes to parts in one zone hard to be isolated from their clients in another zone.	Add "proxy" elements (Gamma et al. 1995) between adjacent zones (i.e. wp1: webProxy, wp2: webProxy, bsp1: BizServiceProxy, bsp2: bizServiceProxy)
Fault Tolerance				
D	<i>New and replacement transformable items:</i> Is an application protected from potential faults from new and replacement transformable items?	It is critical that bsf: BizService1to2Facade is fully backwards compatible to bs_v1: BizService_V1 otherwise in Internal and Web zones transformable items unchanged from V1 will be impacted.	bsf: BizService1to2Facade and bs_v2: BizService_V2 may introduce faults to DPV.	(1) Add a wrapper to bsf: BizService1to2Facade to regulate input/output ranges as per bs_v1: BizService_V1. (2) Add a recovery block (see issues under Recoverability.) Option (2) was preferred by the sponsor.
Recoverability				
D	<i>Failure caused by transformation:</i> Are transformations declared with exceptions and exception handlers where appropriate to restore an application to continue to perform its functionality, for failures caused by transformations?	If the transformation "ja_v1: JobAppointmentStubWS_V1 replacement with jas_v1: JobAppointmentStubWS_V1" fails, DPV is not restored to a state to continue to offer web access to valuers.	No transformation exception handling is defined for the transformation "ja_v1: JobAppointmentStubWS_V1 replacement with jas_v1: JobAppointmentStubWS_V1".	Declare a transformation exception encapsulating the replacement transformation and resolve it with a recovery transformation "ja_v1: JobAppointmentStubWS_V1 recovery".

<i>Inspection</i>		<i>Assessment</i>		<i>Improvement</i>
<i>Artefact Type(s) to Inspect</i>	<i>Inspection Question(s) (all answered with "No")</i>	<i>If "No", describe all defects/issues</i>	<i>If "No", describe root cause for each defect/issue</i>	<i>If "No", propose a solution(s) to resolve the defect/issues</i>
D	<p><i>Failure caused by dynamic change:</i></p> <p>Is an application restored to a state to continue to operate, for failures caused by dynamic changes (i.e. after a transformation)?</p>	If bsf:BizService1to2Facade as a critical component fails, DPV will stop working.	Current design does not handle the case in which bsf:BizService1to2Facade does not provide functionality backwards compatible with what was offered in V1.	Run bs_v1:BizService_V1 and ds_v1:SqlServerDatasore in parallel with bsf:BizService1to2Facade. Define bsrb:BizServiceRecoveryBlock in front of bsf and bs_v1. By default, bsrb uses bsf. If bsf has errors/debugs, bsrb falls back to bs_v1. Define a temporary generation (V1.1 beta) to keep bsrb in the architecture until bsf is fully tested, after which bsrb is removed.
Reliability				
D	<p><i>New and replacement transformable items:</i></p> <p>Do new and replacement transformable items behave as expected, both functionally and non-functionally, in an application?</p>	re_v1:ReportingEngine_V1 would not handle long report generation with MySqlDatastore.	The database driver used in re_v1:ReportingEngine_V1 (i.e. CrystalReport) is incompatible with the database version of MySqlDatastore.	Upgrade re_v1:ReportingEngine_V1 to re_v2:ReportingEngine_V2 which uses the correct driver version.

Note: "Artefact Type(s) to Inspect": "A" and "D" stand for analysis and design artefacts respectively.

The Continuum techniques actually used for improving the quality of the artefacts during the improvement task are shown in Table Appendix D.15.

Table Appendix D.15 DPV: use of Continuum techniques in the Task Improve Dynamic Evolution Quality

<i>Dynamic Evolution Quality Factor</i>	<i>New/Enhanced Technique(s) developed for Continuum</i>		<i>External Technique(s) reused in Continuum</i>	
	<i>Used in Case Study</i>	<i>Not Used in Case Study</i>	<i>Used in Case Study</i>	<i>Not Used in Case Study</i>
<i>Recoverability</i>	Transformation Exception Management		Recovery Blocks	
<i>Completeness</i>	[no specific techniques for Completeness]			
<i>Fault Tolerance</i>			Dynamic Wrapper (for fault containment and protection)	

<i>Dynamic Evolution Quality Factor</i>	<i>New/Enhanced Technique(s) developed for Continuum</i>		<i>External Technique(s) reused in Continuum</i>	
	<i>Used in Case Study</i>	<i>Not Used in Case Study</i>	<i>Used in Case Study</i>	<i>Not Used in Case Study</i>
<i>Reliability</i>				Dynamic Wrapper (for input/output confinement), Transformable Item Regression Testing
<i>Correctness</i>	Change Case Modelling			
<i>Coordination</i>	Secure and Reliable Transformation Agent Coordination			
<i>Consistency</i>	Dynamic Change Impact Analysis, Start-up State Configuration, Resource Profile Modelling			Transformable Item Regression Testing
<i>Separation of Concerns</i>				Transformable Item Mediation and Channelling
<i>Extensibility</i>			Dynamic Change Localisation	Dynamic Wrapper (for functional extension), Dynamic Variation Management
<i>Maintainability</i>				Testability Analysis and Improvement
<i>Locality</i>			Dynamic Change Localisation	
<i>Loose Coupling</i>			Loose Coupling	
<i>Security</i>	Secure and Reliable Transformation Agent Coordination			Dynamic Wrapper (for security containment), Dynamic Security Policy and Enforcement Management
<i>Transparency</i>	Transformation Mining (formerly called "Transformation Sizing")		Dynamic Change Localisation	
<i>Autonomy</i>				Transformable Item Autonomy
<i>Safety</i>		Dynamic Evolution Safety Risk Management		

With regard to the techniques specifically developed in this research (i.e. new and enhanced from others), all except Technique "Dynamic Evolution Safety Risk

Management” were applied to resolve the defects/issues. An explanation for why this technique was not used is that the participants did not regard Safety as relevant in DPV (cf. Table Appendix D.13) and therefore no Safety defect/issue was reported (cf. Table Appendix D.14). For completeness of the case study’s evaluation of Continuum, however, they reviewed its content and steps (cf. Appendix C.3.2.4) as if it were to be used, and provided feedback for it (cf. Appendix F).

In contrast to the techniques specifically developed in this research, only a subset of the techniques reused from the literature and methodologies to resolve the defects/issues had been used by the participants (right of Table Appendix D.15). The fact that some of the reused techniques were not used in the case study is less of an issue in evaluating Continuum because the reused techniques would have been evaluated by their respective authors to a certain degree. In fact, they had already been reviewed by the experts before the case study (Section 7.1) and again by the case study participants (Appendix 7.2).

Appendix E. QUESTIONNAIRE FORMS

This Appendix documents the questionnaire forms used in this research:

- Appendix E.1, for the form “Survey on Dynamic Evolution Quality Factors” used in Task 1.1 (Section 4.2)
- Appendix E.2, for the form “Survey on Dynamic Change Requirements” used in Task 1.2 (Section 5.2)
- Appendix E.3, for the form “Expert Review of Development Methodology Extension to Support Dynamic Evolution” used in Task 3.1 (Section 7.1)
- Appendix E.4, for the form “Case Study Evaluation on Developing Dynamic Evolution for a Software Prototype” used in Task 3.2 (Section 7.2)

E.1 SURVEY ON DYNAMIC EVOLUTION QUALITY FACTORS

E.1.1 Participant Information Sheet

Survey on the Characteristics of Dynamic Evolution in Distributed Applications

Dear respondent,

Thank you for your participation in this survey which evaluates the characteristics of dynamic evolution in distributed applications. Dynamic evolution is a phenomenon in which a distributed application can undergo continuous changes without the need for shutdown. Example applications include global financial systems, mobile phone networks etc. Dynamic evolution is evident in systems built with current technologies, such as updating Enterprise Java Beans™ (EJB™) components in a Java™ 2 Enterprise Edition (J2EE™) environment, and assembly components used in Microsoft® .NET platforms. The result of this survey will provide us information to develop better techniques and models for this type of system development.

This survey has two parts.

The first part consists of completing the questionnaire about your opinions on, and experience with, the characteristics of dynamic evolution in distributed applications.

The second part, if you agree, is a follow-up interview, if necessary, to clarify your questionnaire responses. The interview will be:

- email exchanges, and/or
- phone calls, and/or
- face-to-face conversations

Contacts

<i>Survey Organisers</i>	
Mr. Kam Hay Fung School of Information Systems, Technology and Management, The University of New South Wales, SYDNEY NSW 2052 AUSTRALIA email: k.h.fung@student.unsw.edu.au	Prof. Graham Low School of Information Systems, Technology and Management, The University of New South Wales, SYDNEY NSW 2052 AUSTRALIA email: g.low@unsw.edu.au

Complaints

Complaints may be directed to:

Ethics Secretariat,
The University of New South Wales,
SYDNEY NSW 2052 AUSTRALIA
phone +61 2 93854234
fax +61 2 93856648
email: ethics.sec@unsw.edu.au

Confidentiality

The information you provide is strictly confidential except as required by law, and no personal identification is permanently stored on computer media after the completion of this research. Your contact details will be used for administration purposes only: to let us conduct the follow-up interview and email you the summary result.

How to proceed

Please click “Start” to begin.

Or “Return to Survey” to continue with your last incomplete survey.

Your voluntary participation in this survey is greatly appreciated. In return, we provide you a summary result of this survey if you request and fill out your contact details. You are permitted to withdraw from this survey at any time before the completion of this research without penalty or prejudice.

E.1.2 Survey Web Pages

The content of the web pages as viewed by end users is presented here. Note that these pages do not reproduce the exact look-and-feel as seen by end users through an HTML browser.

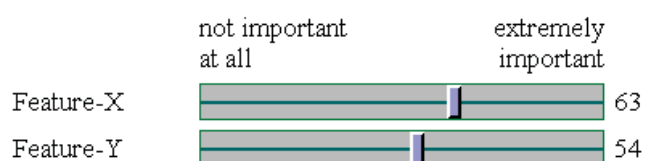
E.1.2.1 Instructions

How to complete the rating questions

The rating questions in this questionnaire ask you about your perception on the importance of each characteristic of dynamic evolution on distributed applications, broken down into four categories. Each question consists of a description about a feature in dynamic evolution, and a scroll bar for you to rate the importance of the feature. The bar is arranged in an increasing order of importance from left to right, representing a score from 0 to 100. The score reading is shown on the right hand side of the bar.



To assign a rating, please move the bar's handle to a desired position. If you feel that feature X is relatively more important than feature Y, please make sure you assign a higher rating to X than Y, e.g.:



There are no right or wrong answers; just answer as accurately as possible. Please complete all

questions.

E.1.2.2 Terminology

Let's introduce the terminology for the rest of this survey.

Component

Components are individual units of logical computation designed to be integrated into larger applications at design time or while they are running. They also represent explicit units of distribution and can be deployed into systems at runtime.

Port

Components connect to one another via their ports. A component's port defines a point of interaction between its component and others, as well as the data they exchange. A port is more than just an interface; it describes the interfaces used by and provided by a component.

Resources and Services

When a component is added to a distributed application, it needs to be allocated with sufficient resources (e.g. CPU time). It must also be granted the use of certain services (e.g. file read/write access) before it can operate.

Resources and services are also temporarily needed by a reconfigurator in order to execute transformations.

Dynamic Distributed Application (also referred to as 'Distributed Application')

A dynamic distributed application is one that can be built from components (i.e. parts) and does not need to be shut down completely for maintenance and/or enhancement. Examples are long-running (e.g. operating systems), and mission critical applications (e.g. emergency hotline). These components are scattered on more than one physical node in a network. Usually one or more components are allocated to an operating system process, and multiple processes are running in each node.

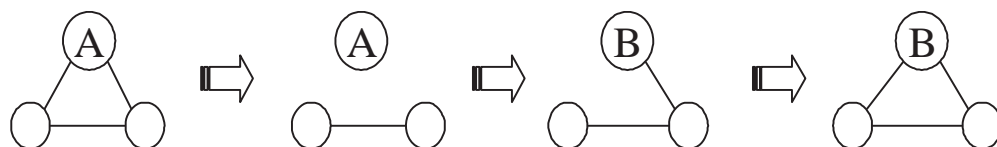
Change

A change refers to the modification required to such an application to make it behave as desired.

Transformation

A transformation means the act of updating an application with the change in a finite and reasonable time bound. In the following diagram, suppose that we wish to upgrade in an application component A to component B which performs better than A. That is, we need to reach the system structure to the right end of the diagram. In this scenario, the change refers to B, and the transformation to the progression of the structure alternation from left to right. Once B is in place, it needs to be activated in the application so that the change becomes effective in the application.

Figure 1 - example transformation



Reconfigurator

Reconfigurators are software processes responsible for making changes and updates (i.e. applying transformations) to a dynamic distributed application at runtime without requiring it to shutdown. They are scattered throughout different physical nodes of the distributed application. For instance, when a component is to be added to an application, a reconfigurator will create an instance of the component, initialize its runtime parameters, deploy it to the node where it will

operate, and activate it in the application.

E.1.2.3 Questionnaire overview

This questionnaire asks your opinion on dynamic evolution, your software development experience, your organisation and its experience in distributed application development.

Before commencing this questionnaire, please:







- read the instructions by clicking the Instructions link in the navigation bar at the top of this page, and
- familiarise yourself with our terminology by clicking the Terminology link in the navigation bar at the top of this page

















If you need to go back to the Instructions and the Terminology pages while filling out the questionnaire, simply click on the link found in the blue navigation bar at the top or the bottom of each questionnaire page.

Should you need to leave while filling out the questionnaire and come back to complete it later, simply click the "Save and Quit" button at the bottom right hand corner of a questionnaire page to save your data entered so far. A token is then displayed and you must use it when returning to complete your questionnaire.

(The identification code for each rating question shown next is also displayed for convenience.)

E.1.2.4 Rating questions - page 1

	not important at all	extremely important	
How important do you believe that prior to a transformation,			
• [A3] adequate resources and services are allocated at remote physical nodes for new and changed components?			50
• [A15] all components involved in a runtime change are identified?			50
• [A19] no critical functions are being executed by components that are affected by the transformation?			50
• [A20] there are no unprocessed messages, unfinished interactions, or incomplete transactions in a distributed application?			50
How important do you believe that during a transformation,			
• [A12] a component to be replaced is not removed until its replacement component fully satisfies its roles?			50
How important do you believe that after a transformation,			
• [A5] all services required by every component are provided by the rest of the application?			50

• [A6] there are no missing components in the distributed application?		50
• [A8] there are no missing, illegal, or broken connections among components in the distributed application?		50
• [A10] intended functionality is not compromised?		50
• [A16] a distributed application does not progress towards an error state (e.g. a deadlock)?		50
• [A17] the state of a component added to a distributed application is set properly to synchronise with the state of the application?		50
• [A18] a changed distributed application continues to execute from a valid state in the changed application as if it had been executing from an initial state of the changed application?		50
How important do you believe that a change to a distributed application		
• [A1] is checked to ensure that the change is non-arbitrary and can be put into the application?		50
• [A9] meets the assumptions and properties of the application and its components?		50
• [A22] does not make the application harder and more costly to change in the future?		50
• [A23] does not make the application harder to test in the future?		50
Miscellaneous: how important do you believe that		
• [A4] a distributed application does not behave in an unintentional manner during and after a transformation?		50
• [A7] all components are clearly defined in interaction specifications?		50
• [A11] a distributed application is tolerant of faulty new and/or changed components?		50
• [A13] the types and directions of connected ports match?		50
• [A14] protocols of communication among components are compatible in order for them to interact correctly?		50
• [A21] system invariants (e.g. no read access allowed on deleted files) are unaffected by a transformation?		50

E.1.2.5 Rating questions - page 2

	not important at all	extremely important	
How important do you believe that the reconfigurators of a distributed application			
• [B1] coordinate among multiple nodes to apply a change to distributed application at runtime?			50
• [B2] are able to cope with unreliability in the communication network during a transformation?			50
How important do you believe that a transformation			
• [B3] can be executed easily?			50
• [B4] can be executed quickly?			50
• [B5] is efficient in resource use?			50
• [B6] causes minimal disruptions to services provided by a distributed application and to the parts using these services?			50
• [B7] causes minimal degradation to the performance of a distributed application?			50
• [B8] inserts components into and removes components from a distributed application in a correct order?			50
• [B9] is executed at the right time (e.g. when the application is not busy)?			50
[B10] How important do you believe that a distributed application is partitioned and a runtime change is confined to an identifiable partition of one or more components?			50
How important do you believe that the separation of			
• [B14] the functional behaviour from issues regarding runtime changes makes it easier to alter one without affecting the other?			50
• [B15] the communication concerns (i.e. mechanisms and strategies) from functionality implementation allows them to evolve independently of each other?			50
• [B16] security support from functionality implementation permits changes to security policies without requiring access to the implementation code?			50
• [B17] security policies from the implementation for their enforcement permits changes to security policies without requiring changes to the implementation?			50

E.1.2.6 Rating questions - page 3

	not important at all	extremely important	
How important do you believe that the reconfigurators of a distributed application			
• [C1] are secured from unauthorised access?			50
• [C9] are transparent to the environment in which the application operates?			50
How important do you believe that new and replacement components to a distributed application			
• [C2] are confined so that they can only perform permitted operations without causing security problems?			50
• [C3] impose restrictions on how other components of the application can access their services, and vice versa after a transformation?			50
[C5] Security policies for a distributed application are updated as components are added to and removed from the application?			50
How important do you believe that a transformation occurring in a distributed application is transparent to			
• [C6] users of the application?			50
• [C8] other components of the application unaffected by the transformation?			50
How important do you believe that during a transformation,			
• [C12] a distributed application and all its components continue to operate in a safe manner?			50
Miscellaneous: how important do you believe that			
• [C7] the design and implementation for transformation is transparent to application programmers who implement the functionality of a distributed application?			50
• [C11] an interaction among components is stated with sufficient details (e.g. remote procedure calls versus shared memory) to make a change easier to apply to the interaction?			50

E.1.2.7 Respondent profile

Which best describes your principal responsibility? (Please mark as many boxes as apply)

- ☐ Technical Manager
- ☐ Project Manager
- ☐ Technical Team Leader
- ☐ Architect
- ☐ Senior Developer / Programmer
- ☐ Research Scientist
- ☐ Consultant
- ☐ Other (Please specify) _____

What is your software development experience in: (Please specify for each category)

Your present organisation? _____ Years

Your professional career(s) in software development? _____ Years

Using software development methodologies? (e.g. Rational Unified Process, in-house) _____ Years

Developing distributed applications (both dynamic and non-dynamic)? _____ Years

For your software development experience in dynamic distributed applications (DDA), if any:

How many DDAs have you been involved in development for, including the ones you are currently working on?

How long have you worked/been working on these DDAs?

_____ months

What roles did/do you play in the development of these DDAs?

E.1.2.8 Software practices

About your organisation at your location, please fill out the following information:

Country of your location _____

Primary line of business

- ☐ Consulting
- ☐ Hardware (e.g. mobile phone)
- ☐ Research
- ☐ Services
- ☐ Education (e.g. university)
- ☐ Software Product
- ☐ Government
- ☐ Other (Please specify) _____

What software development methodologies are currently adopted?

- ☐ None
- ☐ In-house
- ☐ Don't know
- ☐ Other (Please specify, e.g. Rational Unified Process) _____

What certification/accreditation(s) does your organisation currently hold?

- ☐ None, and not planned within the next six months
- ☐ None, but planned within the next six months
- ☐ Other (Please specify, e.g. ISO 9001, CMM level 2) _____

E.1.2.9 Project information

For each completed distributed application development project (especially applications that can be changed at runtime) at your location in the past five years, please provide the following information:

<i>Application</i>	<i>Number of releases</i>	<i>Project start date (mm/yyyy)</i>	<i>Average duration per release (months)</i>	<i>Average effort per release (person-months)</i>	<i>Change activation ("manual restart" or "none")</i>	<i>Industry (e.g. finance, telecommunication, transport, healthcare)</i>	<i>Main software technologies used (e.g. J2EE, Web services, Microsoft® .NET, CORBA™)</i>
1							
2							
...
10							

A *release* corresponds to an installation and acceptance of an application's version to its operating environment.

Change activation refers to the action performed to an application after changes are applied to make the changes active in the application. There are two options:

- *manual restart* means the application must be manually restarted for the changes to be active.
- *none* means nothing needs to be done as the effect of the changes is automatically activated by the application.

E.1.2.10 Software development activities

Please select the appropriate radio button for each activity listed below used while developing the applications listed above:

<i>Activity</i>	<i>Does the activity explicitly address dynamic evolution and transformation concerns?</i>		
	<i>Don't know</i>	<i>Yes: Does the activity work well for dynamic evolution and transformations?</i>	<i>No</i>
Software requirements analysis	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>
Software modification analysis (see example 1 below)	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>
Software architectural design	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>
Software detailed design	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>
Software coding and testing	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>
Integration with existing application (see example 2 below)	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>
Software acceptance testing	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>
Transforming application to the new version in the operating environment	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>
Retirement of removed software parts (see example 3 below)	<input type="radio"/>	<input type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Don't know	<input type="radio"/>

Examples

1. Software modification analysis - What is the impact of a modification to the application?
2. Integration with existing application - Will the change be compatible with existing parts of the application? If not, what needs be done?
3. Retirement of removed software parts - What needs to be done with removed parts? Is backing up of data using old database schema required?

E.1.2.11 Questionnaire submission

If you have any comment about this survey, and suggestions of quality issues for consideration, please write it down here.

Please choose the following options by clicking the check boxes:

- ☐ consent to a possible follow up interview
- ☐ receive a summary of the results of this survey

If you choose either of the above options, please fill out your contact details below:

Name

Job Title

Email

Work Tel. No.

If complete, please click the “Submit” button to send your response to us. Thank you for taking time to fill out this questionnaire.

Submit

E.2 SURVEY ON DYNAMIC CHANGE REQUIREMENTS

E.2.1 Participant Information Sheet

Participant selection and purpose of study

You, the research participant, are invited to participate in a study about the desirable technical features in a software development methodology to support dynamic evolution in a service-oriented architecture (SOA). Dynamic evolution enables the application to undergo changes without the need for shutdown and restart. We, the investigators, hope to learn from this research to develop a methodology to support for dynamic evolution in an SOA environment. You were selected as a possible participant in this research because of your technical knowledge and/or expertise in this area.

Description of study and risks

If you decide to participate, we will ask you complete a questionnaire which seeks your technical views about a list of desirable features of a methodology to support dynamic evolution, and asks you to suggest features missing from the list. The questionnaire is expected to be completed in 15 minutes.

This research does not solicit information about yourself, the work or role you perform at your organisation(s), or the organisation(s) for which you work. There are no known risks from taking part in this research. To participate we expect that you have sound technical knowledge of SOA. If not, please do not to proceed with this research.

Confidentiality and disclosure of information

Any information that is obtained in connection with this research and that can be identified with you will remain confidential and will be disclosed only with your permission, except as required by law. We plan to publish the results in an academic journal or periodical, and in a PhD thesis.

In any publication, information will be provided in such a way that you cannot be identified.

We will provide you a summary result of this research if you request it and fill out your contact details.

Complaints

Complaints may be directed to the Ethics Secretariat, The University of New South Wales, SYDNEY 2052 AUSTRALIA (phone 9385 4234, fax 9385 6648, email ethics.sec@unsw.edu.au). Any complaint you make will be investigated promptly and you will be informed of the outcome.

Your consent

Your decision whether or not to participate will not prejudice your future relations with the University of New South Wales. If you decide to participate, you are free to withdraw your consent and to discontinue participation at any time without prejudice before the end of this research.

Your completion and return of the questionnaire will be regarded as your consent to participate in this research.

If you have any questions, please feel free to ask us. If you have any additional questions later, we will be happy to answer them. Our contact details can be found below:

You will be given a copy of this form to keep.

Investigators' contact details

<i>Investigator-in-Chief</i>	<i>Investigator</i>
Mr. Kam Hay Fung School of Information Systems, Technology and Management, The University of New South Wales, SYDNEY NSW 2052 AUSTRALIA email: k.h.fung@student.unsw.edu.au	Prof. Graham Low School of Information Systems, Technology and Management, The University of New South Wales, SYDNEY NSW 2052 AUSTRALIA email: g.low@unsw.edu.au

E.2.2 Questionnaire

E.2.2.1 Key terminology

Please read the following terms which assist you in understanding and completing this questionnaire.

Runtime or Dynamic

Refer to something happening to a running application without requiring it to shutdown and restart.

Runtime change

The *intended result* of a modification to the structure, elements and/or configuration of a running application at *runtime*. An example is to *add an extra step* to the end of a banking transaction business process to log each completed transaction record.

Transformation

The *act* of performing modifications to a running application at *runtime*. *Hot swapping* a running service with a newer version to fix a bug is an example of a transformation.

Transformation Agent

The entity responsible for coordinating with one another and executing transformations in a distributed SOA application.

Generation

A *stable version* of an application at a particular time instant.

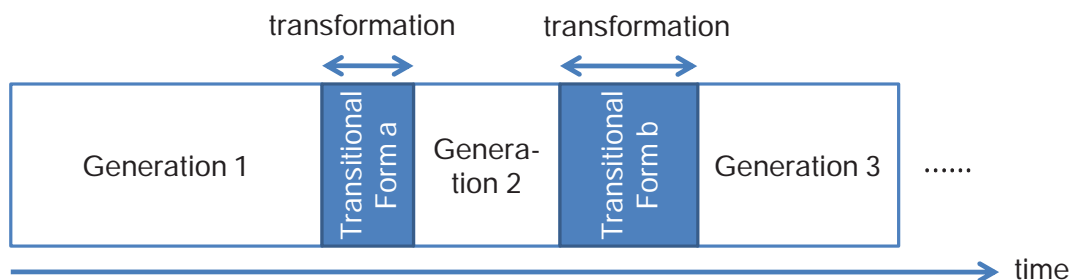
Transitional Form

A *temporary* form and structure of an application that exists between two generations as the application progressively evolves from one generation to the next in a finite amount of time. For instance, during hot swapping, a service becomes temporarily unavailable in an application and cannot be used.

Dynamic evolution

A phenomenon describing an application undergoing evolution, in which planned transformations occur regularly to adapt to changing business needs, without requiring it to shutdown and restart.

The following diagram shows the timeline of an application undergoing dynamic evolution. It also illustrates the relationships of generation, transitional form and transformation.



Feature (of interest)

A feature is a characteristic support offered by a methodology for a particular aspect of software development. In this questionnaire, the features of interest are limited to:

- work to be carried out during software development; and
- modelling concepts or models used and/or created in software development; and
- features specific to runtime changes. For instance, a “*requirement*” is not considered as a feature but a “*change*” is in this research.

Quality related features (e.g. Completeness, Consistency, Coordination, Correctness, Efficiency, Fault Tolerance, Locality, Maintainability, Reliability, Safety, Security, Separation of Concerns, and Transparency) have been considered in another study and are excluded from this questionnaire.

E.2.2.2 Modelling features

Modelling features concern the **modelling concepts**, **notations** and **models** used/produced in SOA development. Please indicate how much you agree with the following modelling features as important in a methodology to support dynamic evolution, by ticking the appropriate box.

DESIRABLE MODELLING FEATURE	Not Important At All	Slightly Important	Moderately Important	Very Important	Extremely Important
Service Level					
Define support for <i>multiple versions</i> present in various parts of an application as the service evolves over time.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the changing <i>resource needs</i> of a service as it evolves.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the changing <i>performance characteristics</i> of a service as it evolves.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the ability to <i>block access</i> to a service while it is being updated or replaced.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

<i>DESIRABLE MODELLING FEATURE</i>	<i>Not Important At All</i>	<i>Slightly Important</i>	<i>Moderately Important</i>	<i>Very Important</i>	<i>Extremely Important</i>
Application Level					
Model the notion of a <i>runtime change</i> for an application (as defined in the Terminology Section).	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the notion of a <i>transformation</i> to an application (as defined in the Terminology Section).	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the presence of a <i>transitional form</i> in an application (as defined in the Terminology Section).	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the notion of a <i>generation</i> of an application (as defined in the Terminology Section).	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the notion of an <i>application lifecycle</i> which organises application evolution as a series of generations over time.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the ability of an application to <i>continuously offer some functionality</i> during a transformation.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Others					
Model the notion of a <i>transformation agent</i> responsible for performing transformations.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the <i>steps</i> undertaken in a transformation, e.g. adding a new service, followed by configuring the service.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the notion of an <i>error condition</i> occurred during a transformation, e.g. because of faulty changes.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Model the notion of a <i>compensation</i> - e.g. a rollback - to revert errors caused by aborted or faulty changes.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please list any other **modelling** feature(s) missing in the previous table that you believe should be supported in a methodology for SOA in a SOA environment.

--

E.2.2.3 Work related features

Work related features concern what must be done, and how to achieve given purposes in SOA development. A piece of work can be a **process** to follow, a **task** to complete, or a **technique** to use. Please indicate how much you agree with the following work related features as important in a methodology to support dynamic evolution, by ticking the appropriate box.

<i>DESIRABLE WORK RELATED FEATURE</i>	<i>Not Important At All</i>	<i>Slightly Important</i>	<i>Moderately Important</i>	<i>Very Important</i>	<i>Extremely Important</i>
Service Level					
Define support to <i>add, replace and remove individual services</i> , and <i>modify their parameters</i> at runtime.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define <i>adapters</i> for wrapping and plugging services into an architecture at runtime, and <i>resolving their mismatches</i> .	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define housekeeping support to <i>retire services</i> after they are no longer needed and have been removed from an application.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define support to <i>(re)bind</i> one service consumer to a new service provider at runtime as the original provider is replaced or upgraded.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define support to <i>predict the resource needs</i> for a new service.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define support to <i>predict the performance characteristics</i> for a new service.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

<i>DESIRABLE WORK RELATED FEATURE</i>	<i>Not Important At All</i>	<i>Slightly Important</i>	<i>Moderately Important</i>	<i>Very Important</i>	<i>Extremely Important</i>
Application Level					
Define support to <i>evolve a protocol definition</i> on which services base their interactions while the protocol is being used.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define support to <i>evolve a business process</i> definition while the business process is operational.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define support to <i>combine</i> several services/ business processes into a larger unit, <i>split</i> a larger service/business process into smaller units or <i>reconfigure</i> the architectural structure at runtime.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define support to <i>refactor</i> an application structure without functional changes at runtime, say, to reduce its complexity and improve its performance.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define customisation points in an architecture to plug in or swap different services at runtime to support limited <i>variations</i> in functionality (e.g. using credit card vs. account debit for payment).	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define support to <i>transfer the state</i> from an instance of an old implementation of a business process to an instance of the new implementation of the business process as the business process evolves.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Analyse the <i>impact of dynamic changes</i> to an application to determine which services/business processes will need to be affected and updated to accommodate the changes.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Others					
Define support to <i>relocate services</i> to a hosting environment as needed.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please list any other **work related** feature(s) missing in the previous table that you believe should be supported in a methodology for SOA in a SOA environment.

--

Please fill out your contact details below to receive a summary of the results of this research:

Name	
Email	

When complete, please return the questionnaire to:

Mr. Kam Hay Fung
 School of Information Systems, Technology and Management,
 The Australian School of Business,
 The University of New South Wales,
 SYDNEY NSW 2052 AUSTRALIA
 email: k.h.fung@student.unsw.edu.au

E.3 EXPERT REVIEW OF DEVELOPMENT METHODOLOGY EXTENSION TO SUPPORT DYNAMIC EVOLUTION

E.3.1 Participant Information Sheet

Participant selection and purpose of study

You are invited to participate in a study to review a development methodology extension to support dynamic evolution in distributed applications which undergo changes without the need for shutdown and restart. We hope to use this study to improve and enhance the methodology extension. You were selected as a possible participant because of your technical knowledge and/or expertise in this area.

Description of study and risks

If you decide to participate, we will ask you complete a questionnaire which asks you to read and provide feedback on the methodology extension documentation about 130 pages. The questionnaire is expected to be completed in 4-5 days and you can spread the time over several days as you go through the document. Afterwards, a face-to-face meeting will be held with you to clarify and avoid possible misinterpretation of your feedback. You will then be communicated via email to confirm refinements to the documentation in accordance with suggestions in the feedback.

There is no risk to you if you participate. You may learn something new when reading the methodology extension. However, we cannot and do not guarantee or promise that you will receive any benefits from this study.

This study does not solicit information about yourself, the work or role you perform at your organisation(s), or the organisation(s) for which you work. There are no known risks from taking part in this study.

Confidentiality and disclosure of information

Any information that is obtained in connection with this study and that can be identified with you will remain confidential and will be disclosed only with your permission, except as required by law. We plan to publish the results in an academic journal or periodical, and in a PhD thesis. In any publication, information will be provided in such a way that you cannot be identified.

Complaints

Complaints may be directed to the Ethics Secretariat, The University of New South Wales, SYDNEY 2052 AUSTRALIA (phone 9385 4234, fax 9385 6648, email ethics.sec@unsw.edu.au). Any complaint you make will be investigated promptly and you will be informed out the outcome.

Your consent

Your decision whether or not to participate will not prejudice your future relations with the University of New South Wales. If you decide to participate, you are free to withdraw your consent and to discontinue participation at any time without prejudice before the end of this study. Your completion and return of the questionnaire will be regarded as your consent to participate in this study.

If you have any questions, please feel free to ask us. If you have any additional questions later, we will be happy to answer them. Our contact details can be found below:

<i>Investigator</i>	<i>Supervisor</i>
Mr. Kam Hay Fung School of Information Systems, Technology and Management, The University of New South Wales, SYDNEY NSW 2052 AUSTRALIA email: k.h.fung@student.unsw.edu.au	Prof. Graham Low School of Information Systems, Technology and Management, The University of New South Wales, SYDNEY NSW 2052 AUSTRALIA email: g.low@unsw.edu.au

E.3.2 Questionnaire

E.3.2.1 Key terminology

Please read the following terms which assist you in understanding and completing this questionnaire.

Methodology

The specification of a process to follow together with the work products to be used and generated during a development effort.

Method(ology) Fragments

A generic term to refer to elements that can be assembled into a methodology.

Work Unit

A specific type of method fragment referring to what must be done and how to achieve given purposes during a development effort.

Task

A type of work unit specifying *what* objectives are to accomplish.

Technique

A type of work unit specifying *how* the objectives of certain tasks are accomplished.

Process

A kind of work unit referring to the systematic application of tasks and techniques through stages in a life cycle.

Work Product

A specific type of method fragment, referring to anything of value used and/or generated during a development effort.

Metamodel

A model of models, i.e. a definition of constructs and rules at a higher level of abstraction for specifying models.

E.3.2.2 Instructions for completing the questionnaire

The questionnaire asks you to read the documentation of a software development methodology extension, called Continuum, and write your comments on it. The documentation has two parts, the main part (Section 1) and an appendix. The former gives an in-depth introduction to Continuum and its method fragments, and how they all fit together. The appendix provides full descriptions of individual fragments in Continuum, organised according to the fragment types, to complement the former. As such, some level of descriptions may repeat in both parts of the documentation.

We suggest the following procedure to read, comprehend and write your comments on Continuum:

1. Do a first read of Section 1 (i.e. the Introduction part) to gain a fair understanding of what Continuum is about.
2. Read Section 1 again and write comments as you go. Use the appendix as guidance for details about the fragments referenced in the main part.
3. Read and comment on individual task and technique definitions in the appendix (i.e. the full specifications part).
4. (Optional) Read and comment on the work product definitions in the appendix.

On the following pages, we solicit comments from you about Continuum.

When complete, please contact Mr. Kam Hay Fung (email: k.h.fung@student.unsw.edu.au) to collect the questionnaire and the documentation, and schedule a face-to-face meeting.

E.3.2.3 Strengths of Continuum

Summarise the **strengths** of Continuum. Here is a sample of hypothetical strengths:

- Continuum offers broad aspects of coffee making tasks and techniques, from beans preparation to coffee decoration.
- Continuum is flexible for making a variety of coffee types (e.g. mocha, espresso, latte).
- Continuum is suitable for baristas in training who are new to the process of coffee making.

Write your summary below in point form.

1.
2.

E.3.2.4 Suggested improvements for Continuum

Identify in Continuum **areas for improvement** and **suggestions for improvement**. An improvement is not limited to a new feature, or a fix or an enhancement to an existing feature. Here is a sample of short and hypothetical suggestions:

<i>Areas for improvements</i>	<i>Suggestions for improvement</i>
The tasks "Make cappuccino" and "Make espresso" overlap when addressing the problem of warming up the milk.	Consider factoring the milk preparation as a separate reusable task or technique.
For Task "Make coffee", the cup size has not been accounted for.	Add the parameter "cup size" as input to the task.
For Task "Make coffee", there can be various ways to add milk to coffee.	Add a technique "Making milk froth pattern" [citing reference] which enhances the presentation of the coffee produced by this task.
A "Person" may drink more than a cup of "Coffee" but the cardinality from "Person" to "Coffee" is shown to be "0..1".	Correct the cardinality as "**".
The concept "Bike" is confusing as it has more than one meaning (motorbike and bicycle).	Rename the concept "Bike" to "Bicycle".

Write your comments on areas of improvement and suggestions for improvement in these areas. You can annotate your comments **directly adjacent to the areas within the document**, or write your comments **below in point form**, whichever is most convenient. Use additional pages if necessary.

<i>Areas for improvements</i>	<i>Suggestions for improvement</i>

E.4 CASE STUDY EVALUATION ON DEVELOPING DYNAMIC EVOLUTION FOR A SOFTWARE PROTOTYPE

E.4.1 Participant Information Statement

Participant selection and purpose of study

You are invited to participate in a study to evaluate a software development methodology

extension support dynamic evolution in distributed applications which undergo changes without the need for shutdown and restart. We hope to use this study to improve and enhance the methodology extension. You were selected as a possible participant because of your technical knowledge and/or expertise in this area.

Description of study and risks

If you decide to participate, we will ask you use a methodology extension, called Continuum, as a guide (about 80 pages) to design a software prototype, to demonstrate dynamic adaptation capabilities using your organisation's software components. During this case study, you'll be asked to fill out a questionnaire, with evaluation results and suggestions on improvements on Continuum, as you use it. Overall, the questionnaire is expected to be completed in 2 days. However, the development will last about 4 months (exact duration to be determined by your organisation). Weekly meetings will be held with you to guide the use of Continuum. When the questionnaire is completed, a meeting will be held with you to clarify and avoid possible misinterpretation of your questionnaire data. Another meeting will be held afterwards to and confirm refinements to the documentation of Continuum in accordance with your feedback and suggestions.

There is no risk to you if you participate. You may learn something new when reading Continuum. However, we cannot and do not guarantee or promise that you will receive any benefits from this study.

This study does not solicit information about yourself, the work or role you perform at your organisation, or your organisation for which you work. There are no known risks from taking part in this study.

Confidentiality and disclosure of information

Any information that is obtained in connection with this study and that can be identified with you will remain confidential and will be disclosed only with your permission, except as required by law. We plan to publish the results in an academic journal or periodical, and in a PhD thesis. In any publication, information will be provided in such a way that you cannot be identified.

Complaints

Complaints may be directed to the Ethics Secretariat, The University of New South Wales, SYDNEY 2052 AUSTRALIA (phone 9385 4234, fax 9385 6648, email ethics.sec@unsw.edu.au). Any complaint you make will be investigated promptly and you will be informed of the outcome.

Your consent

Your decision whether or not to participate will not prejudice your future relations with the University of New South Wales and your organisation. If you decide to participate, you are free to withdraw your consent and to discontinue participation at any time without prejudice before the end of this study. Your completion and return of the questionnaire will be regarded as your consent to participate in this study.

If you have any questions, please feel free to ask us. If you have any additional questions later, we will be happy to answer them. Our contact details can be found below:

<i>Investigator</i>	<i>Supervisor</i>
Mr. Kam Hay Fung School of Information Systems, Technology and Management, The University of New South Wales, SYDNEY NSW 2052 AUSTRALIA email: k.h.fung@student.unsw.edu.au	Prof. Graham Low School of Information Systems, Technology and Management, The University of New South Wales, SYDNEY NSW 2052 AUSTRALIA email: g.low@unsw.edu.au

You will be given a copy of this form to keep.

E.4.2 Questionnaire

E.4.2.1 Instructions for completing the questionnaire

The questionnaire asks you to give comments on various parts of Continuum. It is provided to you as a documentation of two parts, the main part (Section 1) and the appendix. The former gives an in-depth introduction to Continuum and its method fragments, and how they fit together. The appendix provides a full description of individual fragments in Continuum, organised according to the fragment types, to complement the former. As such, some level of description may repeat in both parts of the documentation.

We suggest the following procedure to read, comprehend and write your comments on Continuum:

- Do a first read of Section 1 (i.e. the main part) to gain an understanding of what Continuum is about.
- Use Read Section 1 again. This time, follow up on reading the details of the fragments in the appendix as they are referenced in the main part.
- Use and comment on Continuum method fragments as you go, and after designing the software prototype.

When complete, please email this questionnaire to Mr. Kam Hay Fung (email: k.h.fung@student.unsw.edu.au), and schedule a face-to-face meeting.

E.4.2.2 Usefulness

If you have been involved using your in-house methodology to address dynamic evolution in the last project, complete this section. Otherwise, skip to the next section.

For each significant analysis/design dynamic evolution issue from the last project that is RECURRING in this case study, please describe the issue in the table below and:

- if the issue was addressed by Continuum in this case study, choose one of the following:
 - Continuum was better as in this case study,
 - your in-house methodology was better as in the last project, or
 - about the same
- otherwise (i.e. issue not addressed by Continuum in this case study) describe how the issue was addressed and why Continuum was not used.

	<i>Issue addressed with Continuum in case study</i>			<i>Issue not addressed with Continuum in case study</i>
<i>Description of Issue encountered</i>	<i>better with Continuum</i>	<i>better with your in-house methodology</i>	<i>about the same</i>	<i>Description of how the issue was addressed and the rationale for not using Continuum</i>
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

E.4.2.3 Dynamic evolution issues encountered

For each significant analysis/design issue in dynamic evolution encountered in this case study, describe the issue and rate the extent to which Continuum addressed it, with one of the following 7 values on the Likert Scale:

1 extremely badly	2 very badly	3 badly	4 neutral	5 well	6 very well	7 extremely well
-------------------------	-----------------	------------	--------------	-----------	----------------	------------------------

<i>Description of issue encountered</i>	<i>How well did Continuum address the issue?</i>	
	<i>extremely badly</i>	<i>extremely well</i>
	1 ○ ○ ○ ○ ○ ○ ○ 7	
	1 ○ ○ ○ ○ ○ ○ ○ 7	
	1 ○ ○ ○ ○ ○ ○ ○ 7	

E.4.2.4 Completeness

With respect to this case study, please write down in the space below:

- any suggestion for new features for Continuum; and
- any suggestion for improvement of existing features for Continuum

that should be considered to better address dynamic evolution analysis/design concerns in this case study.

E.4.2.5 Usability

The following pages ask you to rate and write your comments on various parts of Continuum.

The rating scale is a 7-point Likert scale looking like 1 ○ ○ ○ ○ ○ ○ ○ 7, with “1” on the left most being the lowest (strongly DISAGREE) and “7” on the right most being the highest (strongly AGREE).

1 strongly DISAGREE	2 disagree	3 disagree somewhat	4 neutral (i.e. neither disagree or agree)	5 agree somewhat	6 agree	7 strongly AGREE
---------------------------	---------------	---------------------------	--	------------------------	------------	------------------------

Tasks/Techniques labelled with “(reused)” are reused as-is from existing methodologies and approaches.

E.4.2.5.1 Evaluation of metamodel

<i>Area</i>	<i>Related Model Units</i>	<i>Easy to Understand</i>
Structural Foundation	Application, Resource OperationalProfile, TransformableItem, and Zone	1 ○ ○ ○ ○ ○ ○ ○ 7
Application Lifecycle	Application, ApplicationLifecycle, ChangeCase, Generation, Impact, Stage, TransformableItem, and TransitionalPeriod	1 ○ ○ ○ ○ ○ ○ ○ 7
Transitional Period	TransitionalPeriod, TransformationAgent, ChangeCase, Transformation, TransformationException, and TransformationExceptionResolution	1 ○ ○ ○ ○ ○ ○ ○ 7

Transformation	Resource, TransformableItem, Transformation, TransformationAction, and Zone	1 ○ ○ ○ ○ ○ ○ ○ ○ 7
Policy	Policy, ServicingPolicy, and ZoningPolicy	1 ○ ○ ○ ○ ○ ○ ○ ○ 7
<i>Suggestions for improvement and any other comments on Metamodel</i>		

E.4.2.5.2 Evaluation of Application Lifecycle Analysis process and related fragments

<i>Task</i>	<i>Easy to Understand</i>	<i>Easy to Use³²</i>	
Identify As-Is Runtime Structure	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Derive Change Cases	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Extend Application Lifecycle	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
<i>Technique</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	
Change Case Modelling	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Change Case Partitioning and Ordering	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Runtime Structure Recovery (reused)	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
<i>Work Product</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	<i>Produced models/ diagrams/documents Easy to Understand³³</i>
Application Lifecycle Diagram	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7
Dynamic Application Change Document	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7
Structural Configuration - Notational Extensions	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7
<i>Suggestions for improvement and any other comments on Application Lifecycle Analysis Related Fragments. For instance, if a task/technique/work product exists as a better alternative for the ones above, please describe it below.</i>			

³² “Easy to use” means, “Is ‘X’ easy to use, such as to produce a relevant diagram?”

³³ “Produced models/diagrams/documents easy to understand” means “Are the models/diagrams/documents produced from ‘X’ easy to understand?”

E.4.2.5.3 Evaluation of Transformation Identification process and related fragments

<i>Task</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	
Define To-Be Runtime Structure	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Refine Change Cases	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Identify Transformations	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
<i>Technique</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	
Dynamic Variation Management (reused)	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Dynamic Workflow Change (reused)	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Dynamic Refactoring	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Dynamic Recomposition	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Change Case Modelling	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Dynamic Change Impact Analysis	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Transformation Sizing	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
<i>Work Product</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	<i>Produced models/ diagrams/documents Easy to Understand</i>
Dynamic Application Change Document	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7
Structural Configuration - Notational Extensions	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7
<i>Suggestions for improvement and any other comments on Transformation Identification Related Fragments. For instance, if a task/technique/work product exists as a better alternative for the ones above, please describe it below.</i>			

E.4.2.5.4 Evaluation of Transformation Agent Design Process and related fragments

<i>Task</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	
Identify Transformation Agents	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
Define Transformation Orchestration	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	
<i>Work Product</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	<i>Produced models/ diagrams/documents Easy to Understand</i>
Transformation Orchestration Diagram	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ ○ 7

--

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523</
--	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-------

<i>Task</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	
Identify New and Replacement Transformable Items	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Identify Changes to Zones	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Define Servicing Policies	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Develop Transformation	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
<i>Technique</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	
Resource Profile Modelling	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Start-up State Configuration	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Performance Profile Modelling	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Dynamic Transformable Item Adaptation	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Dynamic Transformable Item (Re)binding	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Transformable Item Change	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
<i>Work Product</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	<i>Produced models/ diagrams/documents Easy to Understand</i>
Transformation Diagram	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7
State Map	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7
New and Replacement Transformable Item Catalogue	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7
Zone Change Document	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7

--

<i>Task</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	
Define Dynamic Evolution Quality Needs	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Assess Dynamic Evolution Quality	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Analyse Dynamic Evolution Quality Problems	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Improve Dynamic Evolution Quality	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
<i>Technique</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	
Dynamic Change Localisation (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Dynamic Evolution Safety Risk Management	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Dynamic Security Policy and Enforcement Management (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Dynamic Wrapper (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Inspection (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Loose Coupling (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Recovery Blocks (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Secure and Reliable Transformation Agent Coordination	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Start-up State Configuration	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Testability Analysis and Improvement (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Transformable Item Autonomy (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Transformable Item Mediation and Channelling (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Transformable Item Regression Testing (reused)	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Transformation Exception Management	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	
Transformation Sizing	1 0 0 0 0 0 0 0 7	1 0 0 0 0 0 0 0 7	

<i>Work Product</i>	<i>Easy to Understand</i>	<i>Easy to Use</i>	<i>Produced models/ diagrams/documents Easy to Understand</i>
Dynamic Evolution Quality Profile Report	1 ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ 7
Dynamic Evolution Quality Inspection Report	1 ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ 7	1 ○ ○ ○ ○ ○ ○ ○ 7
<p><i>Suggestions for improvement and any other comments on Dynamic Evolution Quality Management Related Fragments. For instance, if a task/technique/work product exists as a better alternative for the ones above, please describe it below.</i></p>			

[This page is intentionally left blank.]

Appendix F. REFINEMENTS TO CONTINUUM

This Appendix documents the refinements made to Continuum based on the evaluation results from Phase 3 of this research. They are presented in two tables:

- Table Appendix F.1, for the expert review results and subsequent refinements (Section 7.1)
- Table Appendix F.2, for the results of applying Continuum in a case study and subsequent refinements (Section 7.2)





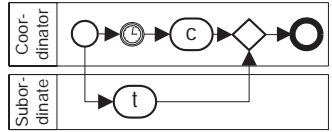


In each table, the refinements are categorised as “metamodel”, “work unit”, “work product” and “others”. The first three target specific types of method fragments in Continuum whereas the last category accounts for all other cases (e.g. structure of documentation).

Table Appendix F.1 Suggested improvements from expert review and subsequent refinements to Continuum

Area for Improvement		Suggestion for Improvement	Actual Refinement	
Type	Method Fragment and Issue		Description	Reference
Expert 1				
Metamodel	The association between “Application” and “TransformableItem” does not reflect the whole-part relationship between them.	Change it to an UML aggregation (i.e. the hollow diamond shape).	Changed as per suggestion.	Figure 6.5, Figure 6.6, Figure 6.7
	“Change impact” is discussed in Continuum but not apparent in the metamodel.	Consider including the notion change impact in the metamodel.	New model unit “Impact” added to the metamodel.	Appendix C.2.1.5
	“Workflow” is discussed in Continuum but not apparent in the metamodel.	Consider workflow as a special case of TransformableItem.	Definition of TransformableItem expanded to include a workflow as a special case for TransformableItem.	Appendix C.2.1.13

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
	“Compensation” has its peculiar meaning and excludes certain types of handlers for transformation exceptions. In BPEL / BPMN, it refers to “undoing” of a <i>completed</i> activity to deal with undesirable outcomes from the activity. So, compensation does not account for cases such as when a transformation aborts half-way through.	The new term should accommodate both compensation and recovery scenarios, the latter referring to getting an application back to a known state and form (or the one prior to a transformation) to continue to run.	“Compensation” renamed to “TransformationExceptionResolution” in the metamodel, and its definition expanded to cover both compensation and recovery.	Appendix C.2.1.18
Work Unit	In the introduction section of Continuum, the process diagrams for the Application Lifecycle Analysis, Transition Design and Transformation Design processes lack sequence information about the tasks to perform in these process. This is somewhat confusing until the textual description is read and followed.	Add sequence numbers for the tasks in these diagrams.	Changed as per suggestion.	Figure 6.11, Figure 6.13, Figure 6.16, Figure 6.21, Figure 6.23
	In the introduction section of Continuum, the definitions for process, task and technique, and their notations in process diagrams are missing. This is confusing to readers unfamiliar with these concepts.	Add definitions for process, task and technique fragments, and example uses of their notations in the introduction section of Continuum.	Descriptions of notations for process, task and technique fragments added to the introduction section of Continuum.	Section 6.3.2
	In Technique “Start-up State Configuration Modelling”, a TransformableItem can be stateless which makes the technique superfluous.	Clarify when Technique “Start-up State Configuration Modelling” is appropriate with respect to the use of states in TransformableItems.	Descriptions for Technique “Start-up State Configuration” updated to confine it to TransformableItems that are designed to have states.	Appendix C.3.2.12

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
	Task “Develop Transition” and Technique “Secure Transformation Agent Coordination” overlap in terms of assigning responsibilities to transformation agents.	Describe the assignment of transformation agent responsibilities within one work unit.	New Process “Transformation Agent Design” and associated tasks added to Continuum to specifically address transformation agent concerns.	Appendix C.3.1.3
			Task “Identify Transition” renamed to “Define Transformation Orchestration”.	Appendix C.3.1.3.2
			Technique “Secure Transformation Agent Coordination” renamed to “Secure and Reliable Transformation Agent Coordination” and simplified because of the new process “Transformation Agent Design”.	Appendix C.3.2.11
	In Task “Define Dynamic Evolution Quality Needs”, although the definitions of quality factors are given (in the embedded table), they are a bit abstract for one to perform assessment on the dynamic evolution quality needs for a situation (i.e. an application).	Consider providing more details on the quality factors and/or breaking them down into smaller units, along with examples of each to further explain their meanings, to facilitate assessment.	New Work Product “Dynamic Evolution Quality Profile Report” added to Continuum, offering instructions on assessing dynamic evolution quality needs for a situation.	Appendix C.2.2.5
	Process “Dynamic Evolution Quality Management” aims to improve quality of the analysis and design aspects of dynamic evolution but it is unclear which work units/products focus on analysis whilst the others on design aspects.	Link the quality work units/products to analysis and/or design aspects of dynamic evolution.	Work Product “Dynamic Evolution Quality Inspection Report” updated to indicate which quality factors are used in analysis and which ones are used in design.	Table Appendix C.10
	In Task “Extend Application Lifecycle”, there may be situations in which change cases do not need to be ordered for implementation (think about change cases which are all independent from one another).	Tone down the sentence, “... the order in which the groups of change cases are realised is determined ...”	Changed as per suggestion, “... the order in which the groups of change cases, if applicable, are realised ...”	Appendix C.3.1.1.3

Area for Improvement		Suggestion for Improvement	Actual Refinement	
Type	Method Fragment and Issue		Description	Reference
Work Product	In the Transformation Orchestration Diagram, the reuse of BPMN's "end event" notation  cannot distinguish among possible outcomes each of which representing a unique generation. The presence of multiple resulting generations after a transition is due to transformation exceptions which lead to different generations when resolved by a generation.	Refine notations to distinguish various resulting generations.	New notations added to Transformation Orchestration Diagram (  ) to indicate different outcomes (i.e. generations) of transformation orchestrations.	Appendix C.2.2.10
	In the example diagram for "timing out a subordinate agent's transformation" as repeated  next, the BPMN's "parallel gateway"  cannot distinguish or choose between the case when the timer expires and the case when the subordinate agent completes transformation "t", as both flows will eventually arrive at the gateway.	Consider modifying this example using BPMN's "complex gateway" to handle this kind of situations.	The "complex gateway" notation  added to Transformation Orchestration Diagram.	Appendix C.2.2.10
			The example diagram updated to use the "complex gateway".	Figure Appendix C.15(b)
	"State Map" may be inappropriate for stateless TransformableItems.	Exclude "State Map" for stateless TransformableItems.	Changed as per suggestion.	Appendix C.2.2.7
Others	The scope of Continuum - with the focus on analysis and design aspects - is reiterated in several parts of Continuum except at the beginning of the introduction section of Continuum.	Relocate the analysis and design focus of Continuum to the beginning of the introduction section of Continuum.	Changed as per suggestion.	Section 6.3.2

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
	Continuum is not developed from ground up. It reuses and enhances materials from other methodologies.	Highlight the extensional and incremental nature of Continuum.	Citations for methodologies and the literature, noted in the introduction section of Continuum, for method fragments that are reused and enhanced.	Table 6.5, Table 6.7, Table 6.10, Table 6.12, Table 6.15
	Because of its comprehensiveness, Continuum can be overkill for small dynamic evolution projects.	Offer guidelines as to what work units are mandatory/optional.	Suitability of a technique to each task (mandatory, recommended, optional) added to high-level diagrams for process fragments.	Figure 6.11, Figure 6.13, Figure 6.16, Figure 6.21, Figure 6.23
			Usage guidelines extended to provide steps to adopt Continuum to an endeavour and link its process fragments.	Section 6.3.7
	In the full specifications section of Continuum, processes and tasks are documented in separate sub-sections. It makes the processes harder to navigate and follow without the tasks.	Consider relocating the specification of tasks to their respective process fragment headings in the full specifications section of Continuum.	Task fragments relocated and documented under their respective process fragment headings in the full specifications section of Continuum.	Appendix C.3.1
<i>Expert 2</i>				
Metamodel	In the dynamic evolution metamodel in the introduction section of Continuum, the discussions on Application Lifecycle, Transition and Transformation appear after the metamodel class diagram is presented.	Relocate the discussions on these concepts to the beginning of the metamodel in the introduction section of Continuum. This gives a better understanding of the metamodel which is underpinned by these key concepts.	Changed as per suggestion.	Section 6.3.3

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
Work Unit	In Task “Identify As-Is Runtime Structure”, Yan et al. (2004)’s paper titled “DiscoTect: a system for discovering architectures from running systems” is also a popular reference to obtain a runtime structure.	Add this reference to the list of references for recovering runtime structures.	Discussions on recovery removed from Task “Identify As-Is Runtime Structure”. New Technique “Runtime Structure Recovery” added to Continuum to explicitly deal with recovery. A citation to Schmerl et al. (2006) which is a more recent version of Yan et al. (2004) also added to this technique.	Appendix C.3.2.17.11
	In Task “Extend Application Lifecycle”, there is no provision for determining of the number of transitional periods required for a set of known change cases.	Introduce some guidelines to determine the number of transitional periods required.	Task “Extend Application Lifecycle” updated to identify transitional periods.	Appendix C.3.1.1.3
			New Technique “Change Case Partitioning and Ordering” added to Continuum to organise change cases and to help to determine the number of transitional periods.	Appendix C.3.2.2
Work Product	In “State Map”, the role of the backward mapping part of a double-ended arrow dashed arc is unclear.	Clarify how the backward mapping part of a double-ended arrow dashed arc helps in defining a State Map.	Clarification on backward mapping added to “State Map”.	Appendix C.2.2.7
	In “State Map” - example state map for two UML state machines, the role of state “p” as a resolved state is unclear.	Explain the role of a resolved state in the State map example.	Clarification on a resolved state added to “State Map”.	Appendix C.2.2.7
Others	It is unclear which part of Continuum is new contribution and which part of Continuum is reused/enhanced from existing work.	Organise the text to distinguish the new contribution from Continuum from the reused/enhanced parts of Continuum.	Highlighting added to summary tables in the introduction section of Continuum to distinguish reused and enhanced method fragments from those developed for Continuum.	Table 6.5, Table 6.7, Table 6.10, Table 6.12, Table 6.15
	Continuum remains (too) abstract in some places and opens to interpretation and ambiguity.	Provide a full example (Continuum has partly) and use it to illustrate important new modelling elements.	New examples added in the introduction section of Continuum. All examples now linked to the same illustrative application for completeness.	Section 6.3

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
	SEMDM also highlights the role of people in addition to process and product.	More elaboration on the people aspect of the SEMDM in the context of dynamic change management would help.	Although people issues are not a focus of Continuum, for completeness a number of basic producer method fragments are added to Continuum.	Section 6.3.6, Appendix C.1
	In the example for “EPCP: orchestration designs for transformation agents”, it is unclear how the service continuity is achieved in the service blocking time (is service blocking the same as downtime?).	Clarify the issue of downtime in relation to the “search service blocking” mode of a servicing policy.	Clarifications added to the definitions for Model Unit “ServicingPolicyType” to highlight that a sufficiently long delay caused by a service being “blocked and queued” may regard the service as unavailable, in which case the “delegated” ServicingPolicy should also be considered.	Appendix C.2.1.11
	There is also a change management process in ITIL best practices framework (see e.g., http://www.itil-officialsite.com/home/home.asp and http://www.infra-corp.com/solutions/ITIL/change-management-workflow.asp).	Investigate how ITIL is related to this research.	None. ITIL has a process called Change Management (CHM) for managing the full lifecycle of changes, from change identification to change implementation. CHM lacks emphasis on evolution and is not specific to a particular type of application development (e.g. composition-based). Hence, CHM is reviewed in this research but not considered for reuse opportunities in Continuum.	Not applicable
Expert 1 (comments on refinements made to Continuum based on Expert 2’s feedback)				
Work Unit	In Task “Runtime Structure Recovery”, the term “discovery” rather than “recovery” is also used, even though the approach for discovery may be through recovery.	Make “recovery” synonymous with “discovery”.	Synonymy between the “Runtime Structure Recovery” and “Runtime Structure Discovery” noted in Task “Runtime Structure Recovery”.	Appendix C.3.2.17.11

Table Appendix F.2 Suggested improvements from case study and actual refinements to Continuum

Area for Improvement		Suggestion for Improvement	Actual Refinement	
Type	Method Fragment and Issue		Description	Reference
Participant 1				
Metamodel	As I understand, the concept of “Transition” is to be replaced as it is confusing.	“Transition” has dual meanings: 1) the process of changing from one state or condition to another; and 2) a period of such change. Continuum refers to “transition” as 2. However, it can be misinterpreted as 1).	“Transition” renamed to “transitional period” to make Continuum clearer about the notion of a period during which transformations are executed.	Changes made throughout Continuum
	The concept and effectiveness of using ChangeCases is a bit confusing. Some ChangeCases are read like Transformations.	Semantics for ChangeCases and Transformations are overlapping and should be separated.	ChangeCase definitions updated to focus it on functional and property changes from the requirement perspective, whereas transformation definitions remain focused on structural and configuration changes from the architectural perspective.	Sections 6.3.3, 6.3.4.1 and 6.3.4.2; Appendices C.3.1.1.2, C.3.1.5.2, C.3.2.1, C.3.2.2, C.3.2.3, C.2.1.3 and C.2.2.2
	In ChangeCase, the "target" association of ChangeCase is limited to TransformableItem.	It should be an attribute to refer to anything that can be reconfigured (e.g. Zone).	The “target” association changed to an attribute in the ChangeCase model unit.	Figure 6.5, Figure 6.7, Appendix C.2.1.3
Work Unit	In Technique “Transformation Sizing”, the description is not very clear how Transformation Sizing works. Is it measured by effort or calendar time?	Clarify Technique “Transformation Sizing”.	Technique “Transformation Sizing” replaced with “Transformation Mining” in which the name reflects the original intent of the technique “Transformation Sizing”. “Transformation Mining” made clearer in terms of spotting Transformations from change cases.	Appendix C.3.2.15

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
	In Process “Dynamic Evolution Quality Management” and related tasks/techniques, Safety remains too abstract. It has different meanings in different domains. And it seems to be overlapping with some of the other quality such as consistency, integrity and security.	More concrete descriptions of safety in relation to dynamic evolution should be provided. This would also help distinguish safety from other quality factors.	In Technique “Dynamic Evolution Safety Risk Management”, the notion of safety split into hazards, mishaps, assets and safety risks, with new examples provided. Steps and associated examples refined to be more relevant to dynamic evolution.	Appendix C.3.2.4
	In the “addition” transformation pattern of Technique “Dynamic Transformable Item Change”, the TransformationAction “«zone» create item” appears before “«zone» assign resource”. This may be problematic as the transformable item will need resources when it is instantiated/created in a zone.	Change the transformation pattern such that “«zone» assign resource” appears before “«zone» create item”.	Changed as per suggestion.	Figure Appendix C.11
	In the “removal” transformation pattern offered by Technique “Dynamic Transformable Item Change”, the resources actually attained by a transformable item itself at runtime should be relinquished by the item before it is removed, as these resources may not be those originally pre-allocated to it when it was started.	Resource management in this pattern should be handled in two steps. The transformable item firstly relinquishes the resources allocated to it. Then, the resources are claimed back by the zone.	Two TransformationActions explicitly defined in the “removal” transformation pattern, “«composition» relinquish resource” and “«zone» reclaim resource”, the latter appearing at the end of the pattern.	Figure Appendix C.11
	In the “removal” transformation pattern of technique “Dynamic Transformable Item Change”, before removing a transformable item, it would be more appropriate to notify consumers of the item.	Add a notification TransformationAction at/near the beginning of the “removal” transformation pattern. It is used to notify all “consumers” of the item.	A TransformationAction “«composition» announce N’s unavailability” added to the beginning of the “removal” transformation pattern.	Figure Appendix C.11
	Task “Define To-Be Runtime Structure” may utilise “Loose Coupling”.	Technique “Loose Coupling” not only is useful to “Define To-Be Runtime Structure” but also complements Technique “Dynamic Refactoring” when defining a to-be runtime structure.	Links to “Loose Coupling” added in Task “Define To-Be Runtime Structure” and Technique “Dynamic Refactoring”.	Appendix C.3.1.5.1, Appendix C.3.2.6

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
	In the EPCP example and Technique “Change Case Modelling”, the discussions about change case identification appears to be a bit repetitive and mix different levels of abstractions across requirements, high level designs and realisation details in the final deployment. Its effectiveness is not very obvious.	Confine change cases to relate change cases closely to requirements, and transformations to relate closely to design. This will decouple change cases from design which should otherwise be addressed with transformations as it is right now.	Technique “Change Case Modelling” updated to incorporate the notion of a feature, a function or property directly derived from requirements, into a change case expression.	Appendix C.3.2.1
			Change case examples in EPCP and Technique “Change Case Modelling” updated to relate change cases more closely to requirements.	Table 6.6, Table 6.9, Table Appendix C.23
	In Technique “Secure and Reliable Transformation Agent Coordination”, it seems understanding the parallelism nature of all the transformations is essential. The coordination could be non-trivial and require computer-aided optimisation.	More guidance could be provided on coordination among transformation agents.	New Technique “Transformation Orchestration and Agent Coordination” added to Continuum to 1) explicitly address coordination; 2) provide an example as a guide; and 3) explicitly assign transformations to different phases to reduce overall interruption time.	Appendix C.3.2.16
	In Task “Define Transformation Orchestration”, during an orchestration some transformations may be executed without any impact on an application, whilst some others interrupt the normal running of the application.	Non-interruptive transformations might be better performed outside of an interruption period.	In a subsequent expert review, Expert 1 commented on this new technique and further improvement was made accordingly.	
	In Techniques “Dynamic Transformable Item Change” and “Dynamic Transformable Item (Re)binding”, more transformation patterns could be included over time.	For instance, consider the “rebinding” transformation as a candidate pattern as it has been used a number of times in the case study.	New transformation pattern “Rebinding” added to Continuum.	Appendix C.3.2.8

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
	In Task “Develop Transformation”, a Transformation Diagram with TransformationActions is one way of documenting the design of a transformation. There are situations in which a transformation design is not modelled with transformation actions, but as a sequence of steps, or as an external proprietary technique. A similar argument is made for TransformationAction which may not account for all possible steps in a transformation.	The approach of documenting a transformation should be flexible enough to accommodate this variation.	The tag “«custom»” added to Continuum to label Transformations and TransformationActions that are handled by foreign means which is not supported by Continuum.	Table Appendix C.17, Table Appendix C.18
Work Product	“Structural Configuration - Notational Extensions” does not provide support to highlight differences between two successive generations.	Notations about “change” and “removal” between two successive generations could be improved, to contrast differences between the two successive generations.	Highlighting and the delta symbol Δ in a generation added to Continuum to contrast generation differences. (Caution: Removed parts cannot be highlighted since they no longer appear in a diagram.)	Appendix C.2.2.8
	“Structural Configuration - Notational Extensions” does not provide support to label individual Transformable Item instances of the same type.	As Continuum deals with runtime instances instead of static designs, it is appropriate to define notations to label individual instances.	Added notations to uniquely label individual instances of transformable items (e.g. “robot[1]” and “robot[2]” for two instances of software robots).	Appendix C.2.2.8
<i>Participant 2</i>				
Metamodel	In Zone, it is unclear what a “disjoint controlled environment” is with respect to the model unit Zone. Citing a reference it is not enough.	An example and more explanations would be more helpful.	The notion of a Zone simplified in the introduction section of Continuum.	Section 6.3.3
			The specification for Zone expanded with further details.	Appendix C.2.1.20
	OperationalProfile seems to represent resource needs.	Clarifications should be given as to whether there is any relationship between OperationalProfile and Resource.	New Model Unit ResourceProfile - inherited from OperationalProfile and associated with Resource - added to Continuum.	Figure 6.5, Figure 6.6, Appendix C.2.1.10

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
	TransformationAgent is abstract. It is unclear if it refers to humans or computers.	TransformationAgent is not limited to software distribution units. TransformationAgent may also be tools, scripts, practitioners, software components etc. The notion of a TransformationAgent should be expanded to accommodate this variation.	The definition of TransformationAgent rephrased to emphasise it as a role which can be played by software components, tools, practitioners etc.	Appendix C.2.1.16
	The association between TransformationAgent and TransformationAction is incorrect.	TransformationAgent should be directly related to Transformation as a whole rather than to individual TransformationActions.	Corrected as per suggestion.	Figure 6.5, Figure 6.8, Appendix C.2.1.14, Appendix C.2.1.16
Work Unit	In Task “Define Servicing Policies”, the concept of “service levels” is ambiguous.	More clarification should be provided for “service levels”, e.g. how many service levels there are.	“service levels” replaced with “services” and “functions”.	Appendix C.3.1.4.3
	Technique “Performance Profile Modelling” is a bit short as to the description of the “Performance Engineering Methodology” that this technique reuses from ASG (Lehner et al. 2006).	More descriptions should be provided for “Performance Engineering Methodology” to help readers to understand its basic ideas.	Model Unit PerformanceProfile added to Continuum.	Appendix C.2.1.8
			Work Product New and Replacement Transformable Item Catalogue changed to utilise PerformanceProfile.	Appendix C.2.2.6
			More descriptions added to Technique Performance Profile Modelling.	Appendix C.3.2.17.9
	Task “Identify Transformation Agents” lacks guidance.	More guidance on how to identify Transformation Agents should be provided.	New Technique “Transformation Agent Disposition” added to Continuum to specifically deal with identifying of Transformation Agents.	Appendix C.3.2.13

<i>Area for Improvement</i>		<i>Suggestion for Improvement</i>	<i>Actual Refinement</i>	
<i>Type</i>	<i>Method Fragment and Issue</i>		<i>Description</i>	<i>Reference</i>
Work Product	Discrepancies are found between the “New and Replacement Transformable Item Catalogue” produced from the case study and “Zone Change Document” filled out for this case study vs. their templates specified in Continuum documentation.	The templates should be rectified in this regard. (Note: the documents produced with the “New and Replacement Transformable Item Catalogue” and “Zone Change Document” templates deviated from the templates to better record data from the case study. Their templates, however, had not been updated accordingly.)	Templates for “New and Replacement Transformable Item Catalogue” and “Zone Change Document” updated accordingly.	Appendix C.2.2.6, Appendix C.2.2.11
	In “Structural Configuration - Notational Extensions”, why only seven notations are defined in Continuum? What are the advantages and disadvantages of current notations?	More explanations should be given on why current and existing notations are extended in this regard.	Clarified the role of Structural Configuration - Notational Extensions as complements and extensions to existing languages to help to express and depict dynamic evolution.	Appendix C.2.2.8
	The example in “Structural Configuration - Notational Extensions” is a bit short.	Consider extending the example structural configuration diagram to include most if not all notations from Structural Configuration - Notational Extensions, to illustrate their usefulness.	Example expanded to show most if not all the notations.	Appendix C.2.2.8
Others	In the introduction section of Continuum, the Section reference for “Structural Configuration - Notational Extensions” is missing when it first appears. It is hard for readers to learn more details about it.	The reference to Structural Configuration - Notational Extensions should be added when it first appears in the introduction section of Continuum.	Changed as per suggestion.	Section 6.3.1
	In the introduction section of Continuum, there is no explanation on what the Structural Foundation aspect is in regard to the metamodel. This concept is a little bit hard to understand.	More description on the notion of Structural Foundation should be given.	Additional descriptions incorporated into the dynamic evolution metamodel in the introduction section of Continuum.	Section 6.3.3

Area for Improvement		Suggestion for Improvement	Actual Refinement	
Type	Method Fragment and Issue		Description	Reference
	The overall metamodel diagram in the introduction section of Continuum does not highlight key dynamic evolution concepts.	If possible, highlight the key dynamic evolution concepts (ApplicationLifecycle, Transformation and TransitionalPeriod) in the overall metamodel.	Key concepts highlighted in the diagram depicting the whole dynamic evolution metamodel.	Figure 6.5
		It would also be clearer to draw five boundary boxes in the overall metamodel if possible, corresponding to the five related aspects of the metamodel.	None. An attempt was made to add boundary boxes to the metamodel diagram but it led to visual cluttering found. This change was dropped.	Not applicable
Expert 1 (comments on refinements made to Continuum based on case study participants' feedback)				
Work Unit	Technique "Transformation Orchestration and Agent Coordination" is a bit complicated. I had to read it a few times to understand it.	Simplify it.	Descriptions and steps shortened and simplified.	Appendix C.3.2.16