UNSW
THE UNIVERSITY OF NEW SOUTH WALES
SYDNEY • AUSTRALIA

Qualitative & Semi-Quantitative Modelling and Simulation of the Software Engineering Processes

by

He Zhang

A thesis submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy

School of Computer Science and Engineering
Faculty of Engineering

2008

University of New South Wales
Faculty of Engineering
School of Computer Science and Engineering

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

Qualitative & Semi-Quantitative Modelling and Simulation of the Software Engineering Processes

by

He Zhang

Supervised by:
Prof. Ross D. Jeffery
&
Prof. Barbara A. Kitchenham

Sydney, New South Wales, Australia
September, 2008

Abstract

Software process modelling has been identified as a vehicle for understanding development processes, controlling development costs and duration, and achieving product quality. In recent years, software process simulation has become one of the essential techniques for effectively investigating and managing software development processes.

Until now, most research has focused on the quantitative aspects of process simulation and modelling. Nevertheless, purely quantitative process modelling requires a very detailed understanding and accurate measurement of the software process, which relies on reliable and precise history data. When such data are lacking or the quality is dubious, quantitative models impose constraints that restrict the model's value. Unfortunately, these data are not readily available in most cases, especially in organisations at low process maturity levels. In addition, software development is a highly complex, human-centred endeavour, which involves many uncertain factors in the course of the development process. Given this inherent uncertainty and contingent characteristics, quantitative modelling employs statistical techniques, but even with these its capability and applied conditions limit its performance on large-scale problems.

As an alternative to quantitative approaches, qualitative modelling can cope with a lack of complete knowledge, and predict qualitative process behaviours. Furthermore, semi-quantitative modelling offers the capability of handling process uncertainty with limited knowledge, and achieves a tradeoff between quantitative and qualitative approaches. Most previous research has omitted these approaches and consequently the associated methods and application of them are far from fully developed.

The main contributions of this research lies in the pioneering work on the models, methods, and applications of qualitative and semi-quantitative software process modelling and simulation in software engineering, and their relationship with

conventional quantitative modelling approaches.

This dissertation produces its novelty from two areas of research. Firstly, it explores methods and techniques to qualitatively and semi-quantitatively modelling and simulating software processes in different domains and at different granularities. Secondly, some specific applications of these modelling approaches are also developed for aspects of software engineering practice. Moreover, a proposed framework integrates these approaches with typical quantitative paradigms to guide the adoption of process simulation modelling in software organisations. As a part of this dissertation, a comprehensive software process simulation modelling state-of-the-art is reported by way of a systematic literature review.

Acknowledgements

This study would not have been possible without the encouragement and constant support of my supervisors, Professor Ross Jeffery and Professor Barbara Kitchenham. I am deeply grateful to them for their inspiration, ideas, guidance, and constructive comments.

I wish to express my warm and sincere thanks to the staff of Empirical Software Engineering at National ICT Australia and the School of Computer Science and Engineering. I appreciate the friendship, support, and collaboration of my colleagues within ESE group. In particular, I am very grateful to Ming Huo, Dr. Liming, and Dr. Jacky Keung. Associate Professor Dietmar Pfahl has also supported this work through his reviews and suggestions, particularly with regard to process simulation in software engineering.

I also wish to thank my parents for their absolute support over my life.

Contents

List of Figures

List of Tables

List of Publications

[1] He Zhang, Barbara Kitchenham, and Dietmar Pfahl. Software Process Simulation Modeling: Facts, Trends, and Directions. *Proceedings of 15th Asia-Pacific Software Engineering Conference (APSEC'08)*, Beijing, China, 2008. IEEE Computer Society.

(cf. Chapter 3: summarises the '*facts*', discovers the '*trends*', and recommends the '*directions*' of SPSM research based on a systematic literature review of ProSim publications from 1998 to 2007.)

[2] He Zhang, Barbara Kitchenham, and Dietmar Pfahl. Software Process Simulation over Decade: Trends Discovery from A Systematic Review. *Proceedings of 2nd International Symposium on Empirical Software Engineering and Measurement (ESEM'08)*, Kaiserslautern, Germany, 2008. ACM.

(cf. Chapter 3: reports the trends of software process simulation over the past ten years derived from the results of systematic review.)

[3] He Zhang, Barbara Kitchenham, and Dietmar Pfahl. Reflections on 10 Years of Software Process Simulation Modelling: A Systematic Review. *Proceedings of International Conference on Software Process (ICSP'08)*, pp.345-356, Leipzig, Germany, 2008. Springer-Verlag.

(cf. Chapter 3: reports the process and the primary results from the systematic literature review.)

[4] He Zhang, Ross Jeffery and Liming Zhu. Hybrid Modelling of Test-and-Fix Processes in Incremental Development. *Proceedings of International Conference on Software Process (ICSP'08)*, pp.333-344, Leipzig, Germany, 2008. Springer-Verlag.

(cf. Chapter 3: presents a horizontal integration scheme to build a hybrid software process simulation model.)

[5] Ming Huo, He Zhang, and Ross Jeffery. Detection of Consistent Patterns from Process Enactment Data. *Proceedings of International Conference on Software Process (ICSP'08)*, pp.174-185, Leipzig, Germany, 2008. Springer-Verlag.

(cf. Chapter 2: reports the enhancement for the systematic Approach to Process Enactment Analysis and the extended use in detecting Consistency from Process Enactment Data.)

[6] He Zhang, Ross Jeffery, and Liming Zhu. Investigating Test-and-Fix Processes of Incremental Development Using Hybrid Process Simulation. *Proceedings of 6th Workshop on Software Quality (WoSQ'08)*, Leipzig, Germany, 2008. ACM.

(cf. Chapter 3: reports a new hybrid software process model and its use to investigate the test-and-fix process of incremental development.)

[7] He Zhang, Jacky Keung, Barbara Kitchenham, and Ross Jeffery, Semi-Quantitative Modeling for Managing Software Development Processes. *Proceedings of 19th Australian Software Engineering Conference (ASWEC'08)*, pp.66-75, Perth, Australia, 2008. IEEE Computer Society.

(cf. Chapter 7 and 9: presents an enhanced semi-quantitative model of incremental development, and discusses its use in support of process/project management.)

[8] He Zhang, Barbara Kitchenham, and Ross Jeffery, A SemiQ Model of Test-and-Fix Process of Incremental Development. *Proceedings of International Workshop on Software Productivity Analysis and Cost Estimation (SPACE'07)*, pp.23-29, Nagoya, Japan, 2007. IPSJ/SIGSE.

(cf. Chapter 7: presents the initial version of semi-quantitative process model focusing test-and-fix process in incremental development.)

[9] Barbara Kitchenham, Hiyam Al-Kilidar, Muhammad Ali Babar, Mike Berry, Karl Cox, Jacky Keung, Felicia Kurniawati, Mark Staples, He Zhang, and Liming Zhu. Evaluating Guidelines for Empirical Software Engineering Studies. *Journal of Empirical Software Engineering*, vol.13(1):97-121, 2007.

(the enhanced version of ISESE'06 paper.)

[10] He Zhang, Barbara Kitchenham, and Ross Jeffery. Achieving Software Project Success: A Semi-Quantitative Approach. *Proceedings of International Conference on Software Process (ICSP'07)*, pp.332-343, Minneapolis, MN, 2007. Springer-Verlag.

(cf. Chapter 9: presents an integrated approach for software project management, i.e. planning and control, based on semi-quantitative modelling and simulation.)

[11] He Zhang, Barbara Kitchenham, and Ross Jeffery. A Framework for Adopting Software Process Simulation in CMMI Organisations. *Proceedings of International Conference on Software Process (ICSP'07)*, pp.320-331, Minneapolis, MN, 2007. Springer-Verlag.

(cf. Chapter 10: argues software process improvement framework in support of the adoption of process simulation, and presents the primary adoption framework.)

[12] He Zhang, Barbara Kitchenham, and Ross Jeffery. Planning Software Project Success with Semi-Quantitative Reasoning. *Proceedings of 18th Australian Software Engineering Conference (ASWEC'07)*, pp.369-378, Melbourne, Australia, 2007. IEEE Computer Society.

(cf. Chapter 9: presents an innovative approach for software project planning using semi-quantitative reasoning.)

[13] Ming Huo, He Zhang, and Ross Jeffery. A Systematic Approach to Process Enactment Analysis as Input to Software Process Improvement or Tailoring. *Proceedings of 13th Asian-Pacific Software Engineering Conference (APSEC'06)*, Bangalore, India, 2006. pp.401-408, IEEE Computer Society.

(cf. Chapter 2: presents a systematic approach to process enactment analysis that attempts to bridge the gap between descriptive enactment models (DEMs) and prescriptive reference models (PRMs).)

[14] Barbara Kitchenham, Hiyam Al-Kilidar, Muhammad Ali Babar, Mike Berry, Karl Cox, Jacky Keung, Felicia Kurniawati, Mark Staples, He Zhang, and Liming Zhu. Evaluating Guidelines for Empirical Software Engineering Studies. *Proceedings of International Symposium on Empirical Software Engineering (ISESE'06)*, pp.38-47, Rio de Janeiro, Brazil, 2006. ACM.

(suggests the guidelines for reporting the results of empirical research, especially for controlled experiments, and evaluating the empirical reports in software engineering from the perspectives of researchers, practitioners, and reviewers.)

[15] Ming Huo, He Zhang, and Ross Jeffery. An Exploratory Study of Process Enactment as Input to Software Process Improvement. *Proceedings of 4th Workshop on Software Quality (WoSQ'06)*, pp.39-44, Shanghai, China, 2006. ACM.

(cf. Chapter 2: reports the initial version of a software process recovery method based on mining project enactment data in support of software process improvement.)

[16] He Zhang and Barbara Kitchenham. Semi-Quantitative Simulation Modelling of Software Engineering Process. *Proceedings of International Software Process Workshop/International Workshop on Software Process Simulation and Modelling (SPW/ProSim'06)*, pp.242-253, Shanghai, China, 2006. Springer-Verlag.

(cf. Chapter 6: introduces semi-quantitative approach to model and simulate software processes, and presents the first semi-quantitative software process model.)

[17] He Zhang, Ming Huo, Barbara Kitchenham, and Ross Jeffery. Qualitative Simulation Model for Software Engineering Process. *Proceedings of 17th Australian Software Engineering Conference (ASWEC'06)*, pp.391-400, Sydney, Australia, 2006. IEEE Computer Society.

(cf. Chapter 6: presents the first structural qualitative software process model for simulation and its use for revisiting Brooks' Law.)

List of Abbreviations

Abbreviation	Description
ACD	Activity Cycle Diagram
ABS	Agent-Based Simulation
APM	Agile Process Models
ASD	Abstract Structure Diagram
ASM	Analytic Summary Model
BBNs	Bayesian Belief Nets
BPE	Business Process Engineering
CL	Capability Level of CMMI
CLD	Causal Loop Diagram
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
DEM	Descriptive Enactment Model
DES	Discrete-Event Simulation
DEVS	Discrete-Event System Specification
DTS	Discrete Time Simulation
EBSE	Evidence Based Software Engineering
ESE	Empirical Software Engineering
ESM	Explanatory Structural Model
GG	General Goal of CMMI
GSD	Global Software Development
ICSP	International Conference on Software Process
ISO	International Standardisation Organisation
KBS	Knowledge-Based Simulation
KPI	Key Performance Indicator
LPM	Life-cycle Process Model
MAS	Multi-Agent System
ML	Maturity Level of CMMI
ODE	Ordinary Differential Equation
PA	Process Area of CMMI
PEA	Process Enactment Analysis
PMBoK	Project Management Body of Knowledge

Abbreviation	Description
PRM	Prescriptive Reference Model
ProSim	International Workshop on Software Process Simulation and Modeling
Q2	Semi-quantitative extension to QSIM implemented by UTexas
Q3	Step-sized refinement to Q2 implemented by UTexas
QDE	Qualitative Differential Equation
QR	Qualitative Reasoning
QSIM	Qualitative Simulation
QSIM	QSIM algorithm and reasoner implemented by UTexas
RM	Regression Model
RPG	Role-Playing Game
SBS	State-Based Simulation
SD	System Dynamics
SE	Software Engineering
SEI	Software Engineering Institute
SemiQ	Semi-Quantitative (Modelling)
SG	Specific Goal of CMMI
SLR	Systematic Literature Review
SP	Specific Practice of CMMI
SPE	Software Process Engineering
SPI	Software Process Improvement
SPM	Software Process Modelling
SPSM	Software Process Simulation Modelling
SPW	Software Process Workshop
SQR	Semi-Quantitative Reasoning
SQSIM	Semi-Quantitative Simulation
TQM	Total Quality Management

Chapter 1

Introduction

As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.

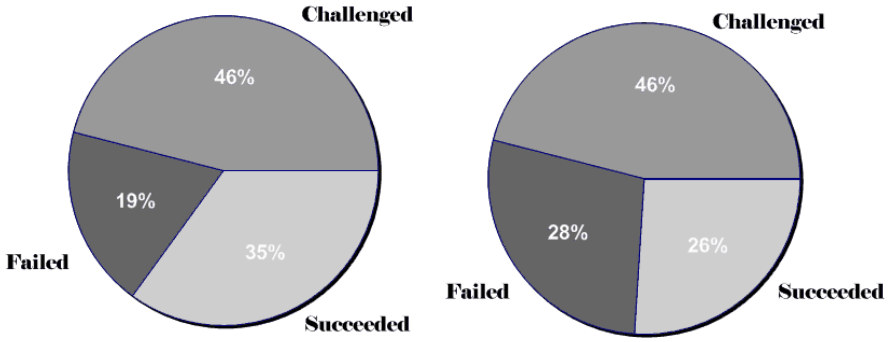
- Albert Einstein (1879-1955)

1.1 Background

Software is a large and complex artifact produced by humans and for humans. Unfortunately, the development of software is yet far from being well understood by humans. The current state of software engineering still makes most software development managers risk bearers. Much had been written about the '*software crisis*' yet the crisis remains: "*Software is a place where dreams are planted and nightmares harvested, where terrible demons compete with magical panaceas, a world of werewolves and silver bullets.*" [Cox90]

The 2006 Chaos Report by the Standish Group [SGI06] (as shown in Figure 1.1-a) reported that 35% of software projects in 2006 could be categorised as successful, meaning they were completed on time, on budget and met user requirements; 19% of projects were outright failures; and 46% of projects were described as challenged, meaning they had cost or time overruns or did not fully meet the user's needs. It illustrates that, though there have been many technical advances in software engineering, there are still critical obstacles to the real success of the majority of recent software projects.

However, compared with the report in 1998 [SGI98] (shown in Figure 1.1-b), some positive progress is displayed. The success rate of software project has been increased by 9% in the past 8 years. Meanwhile, the failure rate has been decreased by one third (9%). The report concludes three reasons for the improvement: '*better project management*', '*iterative development*', and '*the emerg-*



(a) 2006 (b) 1998

Figure 1.1: Software project resolution from SGI's Chaos Reports

ing web infrastructure', in which the first two reasons are highly relevant to the development of software process.

Moreover, by looking at Figure 1.1 again, there is no significant change over all those years in the percentage of projects with time or budget overrun. It remains at 46%. These challenges to a large extent can be attributed to the inability or immaturity of software project and process management in organisations. Therefore, improving software process research and practice is an important contribution to the success of software projects, and is a key challenge for the software engineering community.

Software process modelling is one of the important research domains for supporting software process management and improvement. State-of-the-art software process modelling strongly focuses on quantitative '*black-box*' and '*white-box*' modelling, while few studies aim at qualitative and semi-quantitative modelling, as the important but overlooked counterpart of quantitative approaches. Uncertainty permeates software development processes, yet is often absent in most quantitative software process models.

Qualitative and semi-quantitative process models are capable of modelling the absence and incompleteness of numeric information. However, most qualitative process models so far lack the power of dynamic simulation. Furthermore, until now there has been no software process model developed by applying semi-quantitative concepts and paradigm. This thesis presents the initial and explorative work for *white-box* modelling and simulation of software processes using *qualitative* and *semi-quantitative* approaches.

1.2 Research Questions

Research Premises

Two statements constitute the *premises* for this research: the existence of complexity and uncertainty in the software development process, as well as the power of qualitative and semi-quantitative approaches in analysing complex systems with incomplete knowledge.

Our initial observation is that software is extremely complex, i.e. "*software systems are perhaps the most intricate of man's handiworks*" [Bro95], as is the software development process, in which "*human interactions are complicated and never very crisp and clean in their effects, but they matter more and other aspect of the work*" [DL99]. Currently, software engineering is fraught with uncertainty, which is often used to explain the known symptoms of the software crisis, including unexpected slips in development schedule as well as unpredictable overrun on budget (cf. Section 1.1). Software development and its artifacts are typically determined by a large and complex collection of certain and uncertain factors of diverse types. This complexity and uncertainty are not only *ubiquitous*, they are also *intrinsic*.

Despite the pervasiveness of human involvement and related software uncertainty, surprisingly few attempts have been made to model this uncertainty *explicitly*. This thesis aims to remedy this situation by using the proposed modelling and simulation approaches.

Qualitative and *semi-quantitative* modelling approaches were born with the explicit representation of incomplete knowledge in modelling causal systems, which enables them to handle both complexity and uncertainty directly and intuitively. Their capability offers the possibility for coping with these difficulties in the software processes.

Research Questions

This research intends to validate the capability of the proposed qualitative and semi-quantitative modelling approaches in coping with the dynamic complexity and uncertainty of software development by investigating the following research questions:

RQ1 Feasibility: Are qualitative and semi-quantitative methods applicable in modelling and simulation of the software processes?

RQ2 Adaptability: Are Qualitative and semi-quantitative approaches able to model and simulate a variety of domains and scopes of the software process?

RQ3 Uniqueness: Can qualitative and semi-quantitative approaches provide a unique values to modelling and simulating software process? Can they

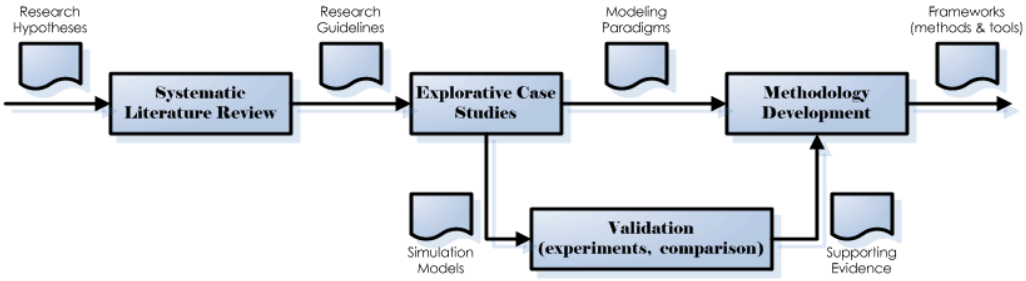


Figure 1.2: Work flow of thesis research

offer a set of methods, artifacts, and tools distinct from the conventional approaches?

RQ4 Practicability: Based on qualitative and semi-quantitative approaches, is it possible to develop any novel methodology or applications supporting diverse aspects of software engineering practice?

The sequence of these four research questions is progressive, which implies the answer to every single question is also the premise for the next one. They must be tested in the given order.

1.3 Approach & Scope

Research Approach

In the course of the research related to this thesis, the following research steps have been conducted in sequence: *1*) systematic literature review, *2*) explorative case studies, *3*) validation, and *4*) methodology development. The flow of work is summarised in Figure 1.2.

A *systematic literature review* was carried out to summarise the uses of SPSM for coping with the dynamic complexity of software development, to discover the *facts*, *trends*, and *directions* of SPSM research, and to identify the needs of this research. The guidelines for conducting the following steps in this research were also derived from the review.

Explorative case studies were applied by developing the process models for simulation in order to test the *feasibility* and *adaptability* of the proposed modelling approaches in software process research, and gain new insights into the investigated software processes.

Here, *validation* is twofold, for models and approach. First, the process models were validated by comparison with typical *quantitative* models, which had been calibrated with real cases, through the designed experiments. Moreover,

these *qualitative* and *semi-quantitative* process models were further used to validate the approaches' usefulness (in a variety of domains and scopes) in modelling software processes.

Based on the results of the modelling practice, a methodology in support of software project management and a guidance framework for adopting SPSM were developed in *methodology development*. They demonstrate the *uniqueness* and *practicability* of the proposed modelling approaches.

Research Scope

The research objects of this work are *software development processes* (defined in Chapter 2). This research focuses on '*white-box*' modelling methods. Its advantage over *black-box modelling* is that changes in the model as a result of model analysis can be directly transferred to the real world. This is due to the fact that the model structure is visible and represents entities in the real world.

The methodological scope of this research can be defined in a multi-dimension view of process modelling, including *time dependency*, *model semantics*, and *probability relevance*. Figure 1.3 shows the methodological scope of this research and its relation with other approaches. The modelling approaches related to this research fall into the shadow area, i.e. the *continuous qualitative* and *semi-quantitative* process modelling. The dashed-line boxes indicate possible future research areas.

1.4 Research Contributions

Despite the relatively young age of software process simulation modelling (SPSM) since Abdel-Hamid and others' pioneering work [AHM91] in late 1980s, there have been a number of research outcomes yielded in this field during the past two decades. Nonetheless, most of these existing studies are purely quantitative based, and the research based on the qualitative and semi-quantitative approaches, as the counterpart of quantitative approaches, was seldom found. From this point of view, this thesis is the first comprehensive endeavour in this direction.

The main contribution of this PhD research lies in the exploration of modelling and simulation of software processes using qualitative/semi-quantitative approaches for different problem domains and process scales, as well as the development and application of QSIM/SQSIM-based frameworks for supporting the paradigm adoption in software engineering practice.

The research achievements resulting from the work can be expected in the following aspects:

- Systematic literature review of the work in software process simulation mod-

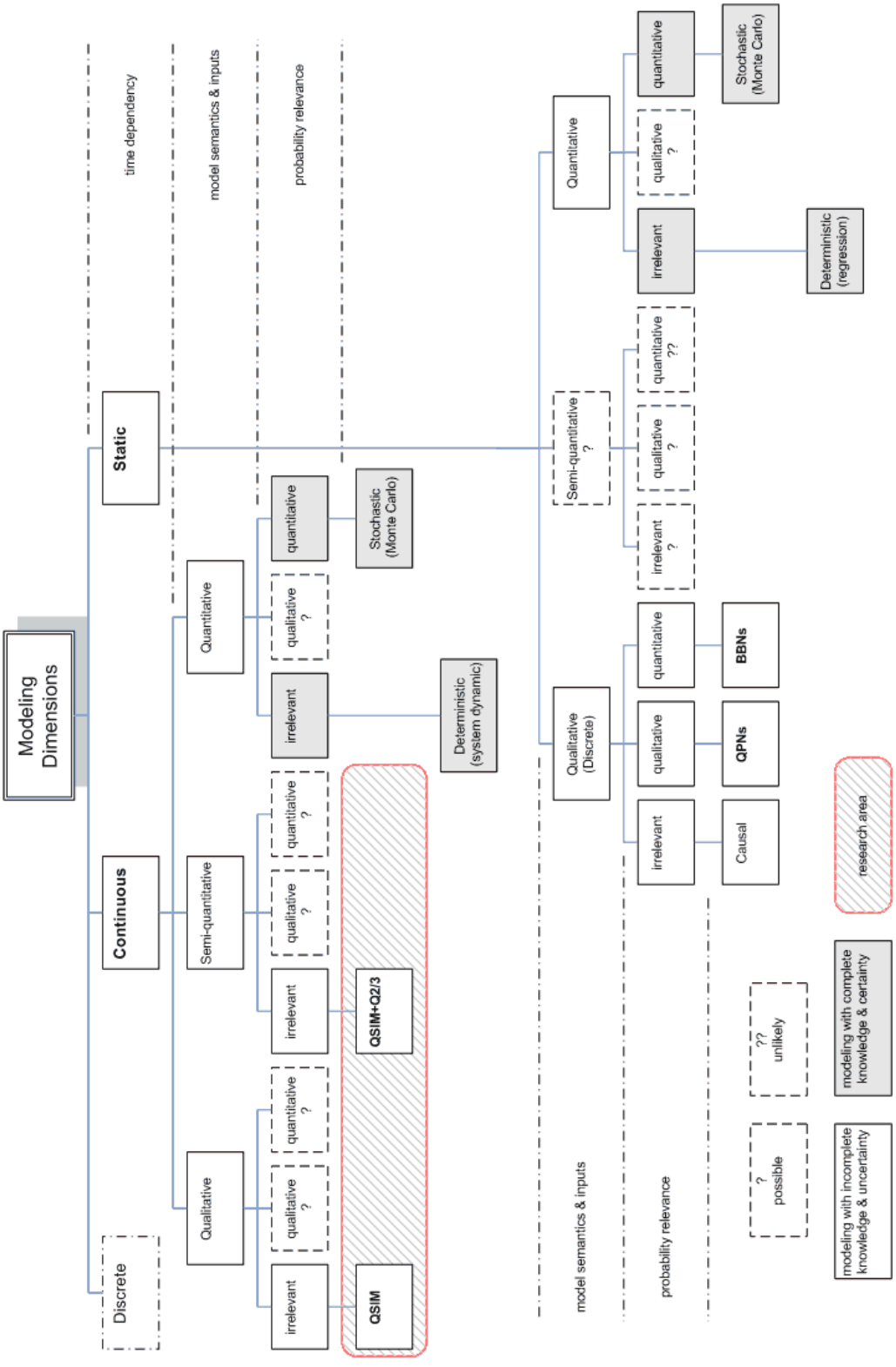


Figure 1.3: Dimensions of process modeling

elling, especially the work done in the recent 10 years. The results can be used not only as the guideline for designing this research, but also as the reflection of progress and state-of-the-art of SPSM research.

- Development of a range of qualitative and semi-quantitative software process models, differing in domain, scope, and granularity, for simulation; acquisition of new insights in the processes modelled using the proposed modelling and simulation approaches.

- Comparison between the typical purely quantitative modelling approach and qualitative/semi-quantitative approaches; development of the equivalent model conversion scheme.

- Framework for software project and process management based on the features and outcomes offered by qualitative and/or semi-quantitative modelling.

- Identification of the usefulness and unique uses of qualitative and semi-quantitative approaches in SPSM for supporting software process improvement.

1.5 Thesis Outline

The body of this research is presented in four parts in this thesis. **Part I** introduces a variety of aspects and current state of software process research, with emphasis on the process simulation modelling; **Part II** provides a concise introduction to qualitative/semi-quantitative modelling and simulation approaches as the fundamental paradigms applied in this thesis; **Part III** models and investigates a set of typical software processes differing in domains and scope; **Part IV** addresses the creative thinking in adopting the proposed approaches in software engineering practice. Each part is further composed of two or three chapters with the details as follows:

Chapter 2 provides the essential but important concepts related to this research, i.e. software engineering process, software process improvement, and especially software process modelling and simulation.

Chapter 3 describes a systematic literature review of software process simulation modelling (SPSM), which presents the state-of-the-art of SPSM during the period from 1998 to 2007. The results from the review are used to frame a solid and up-to-date overview of SPSM research, and further to support the design of this research. The underlying trends and directions are also discussed and suggested.

Chapter 4 presents a concise introduction to qualitative modelling and simulation, including its concepts, methods, notations, mechanism, and tool, as the fundamental paradigm in support of this research.

Chapter 5 extends Chapter 4 with the inclusion of incomplete quantitative knowledge and constraints for modelling and simulation research.

Chapter 6 models a typical software staffing process of a generic software development project, and further uses it to revisit Brooks' Law.

Chapter 7 investigates incremental development processes at a lower phase level. The qualitative and semi-quantitative models developed present a more complicated simulation structure with multiple transitions and iterations.

Chapter 8 enhances qualitative and semi-quantitative simulation by introducing classic delays originated in system dynamics, and provides a comprehensive comparison between the qualitative/semi-quantitative simulation and the conventional continuous approach. The models constructed for the comparison also reveal the capability of the proposed approaches in reflecting a long-term software evolution process at a high level.

Chapter 9 develops a pragmatic software project management approach based on the proposed approaches, as well as provides a set of managerial methods, tools and considerations derived from semi-quantitative simulation.

Chapter 10 argues the interactions between process simulation and process improvement, and proposes an initial version framework by integrating four typical process simulation paradigms (including qualitative/semi-quantitative simulation) and CMMI program for supporting the adoption of process simulation in a particular software organisation.

Chapter 11 summarises the capabilities and possibilities of qualitative/semi-quantitative modelling and simulation in software process research, and briefly discusses its alternative approaches. It ends at highlighting the contributions and suggesting future research work.

Part I

BACKGROUND

Chapter 2

Software Engineering Process

The research object in this thesis is software engineering process, which is the concentration of this chapter. Here, it starts with defining what a software process means, and next review the two most important themes in software process research: process improvement and process modelling. Software process simulation, as an effective research method and tool of software process modelling, is briefly introduced at the end.

2.1 Software Process

Software engineering process (also called *software process*) consists of a set of logically ordered tasks or activities in order to deliver or to maintain a software product [CKO92]. These activities involve the development of software from scratch, and enhancement of existing systems by configuring and integrating off-the-shell software or system components.

Software processes are important in software engineering practice not only because they impose consistency and structure on a set of development activities, but for enabling us to capture our experiences and pass them along to others [PA06]. In the layered framework of software engineering [Pre05], software process is the glue that holds the technology layers and quality focus together, and enables rational and timely development of software.

2.1.1 Activities in Software Process

In [FP97], Fenton and Pfleeger distinguish three classes of entities in software development: process, product, and resource entities.

- **Process**: collection of software-related activities.

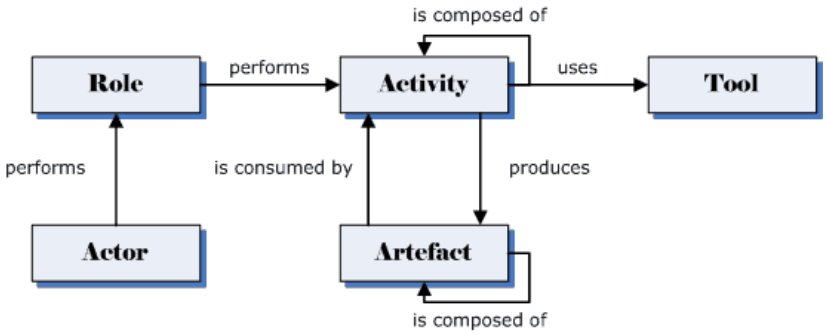


Figure 2.1: Basic entities and their interactions of process

- **Product**: any artifacts, deliverables or documents that result from a process activity.

- **Resource**: entities required by a process activity, e.g. tools, roles, and actors.

The relationships among the process entities are shown in Figure 2.1 (based on [Pfa01]). These three classes also correspond to the entities abstracted in the three dimensions of software development identified in the latest TRISO-Model [Li05, Li07], i.e. *actors* (SE Human), *activities* (SE Process), and *artifacts* (SE Technology).

Although there are a variety of software processes, some '*fundamental*' activities are common to all software processes [Som07]:

1. **Software specification** The functionality of the software and constrains on its operation must be defined.

2. **Software design and implementation** The software to meet the specification must be produced.

3. **Software validation** The software must be validated to ensure that it does what the customer wants.

4. **Software evolution** The software must evolve to meet changing customer needs.

When looking into these '*fundamental*' activities, five generic types of process activities can be identified as the '*atom*' process activities composing them. These '*atom*' activities are applicable to the vast majority of software projects, regardless of their size or complexity [Pre05]:

- **Communication** involves heavy communication and collaboration with the customer (or other stakeholders) and encompasses requirements gathering and other related activities.
- **Planning** establishes a plan for the software engineering work that describes the technical tasks to be conducted, the likely risks, the required resources, the work products to be produced, and a work schedule.
- **Modelling** encompasses the creation of models that allows the developer and the customer to better understand software requirements and design.
- **Construction** combines the activities concerned with building software, i.e. specification, design, and code generation.
- **Evaluation** covers the checking activities, i.e. validation, verification, and testing.
- **Deployment** delivers software to the customer who evaluates the released product and provides feedback.

Though the details of the software processes will be quite different in each case in software development, these processes can be decomposed into the '*fundamental*' process activities, and further into the '*atom*' activities that remain the same.

2.1.2 Macro- & Micro-Process Research

Osterweil identified two complementary types of software process definition, which can be characterised as *macro-process*, focused on phenomenological observations of external behaviours of processes, and *micro-process*, focused on the internal details and workings of processes [Ost05].

At *macro-process* level, the research is characterised as investigations of the external behaviours of processes. Much work in this direction emphasises the determination of appropriate measures of such external effects as product quality, personnel productivity, and process efficiency. Some noticeable examples include process capability determination and software cost estimation.

On the contrary, *micro-process* research focuses on investigation of the precise specification of the details of software process in order to infer how those details affect the external behaviours of the processes. The research at this level aims to provide detailed, accurate, low-level definition of processes, and reasoning capabilities that predict and explain the high level phenomena discovered by *macro-process* research [Ost05]. Some examples of work in this direction include the research on the identification of languages and semantic features in support of precise process definition.

Macro-process and micro-process denote two granularity levels of software processes. Note that *macro-process* and *micro-process* are relative concepts. For example, a software process, e.g. an iterative development, may consist of many phases. They can be sequentially or concurrently executed. If the *macro-process* research investigates the overall performance of the project, the corresponding *micro-process* research could be the investigation of each phase (its inputs, outputs, and constraints) and the relationships between them. Nevertheless, if every single phase is our research focus and viewed as '*macro-process*', then the relative '*micro-process*' zooms in to the detailed tasks and activities performed in the phase.

A natural complementarity between these two approaches is becoming an increasingly focused thread of software process research. The systematic integration of *macro-* and *micro-process* research may be carried out with the former investigating gross external effects of process behaviours, and the latter identifying the causes of these observed effects by examination of the internal structure and details of the process itself [Ost05]. They are tangent and have much to offer each other.

2.1.3 Process Engineering & Engineering Process

The term, *process engineering*, originated in the chemical industry and implies the design of a series of unit operations, actions, or activities that produce a desired product or end result. In software engineering domain, *process engineering* employs engineering technology in support of definition, development, measurement, modelling, assessment, improvement, and reuse of procedures and responsibilities in conjunction with personnel and technology capabilities with the intent to successful delivery of quality software product.

In [RV95], software processes are composed of two top subsets of processes: *engineering processes* and *non-engineering processes*. They roughly correspond to software process engineering (SPE) and business process engineering (BPE) separately. The following sections in this chapter address the two most important aspects of software process engineering, i.e. software process improvement (SPI) and software process modelling (SPM), which are the fundamental concepts related to the research reported in this thesis. Examples of the latter (BPE) include workflow research and enterprise framework modelling [ES05]. The relationship between these two categories of processes is discussed in [Jef06].

2.2 Software Process Improvement

The existence of a software process is no guarantee that software will be delivered in budget and on time, or that it will meet the customer's needs and exhibit the technical specification. *Process improvement* means understanding existing

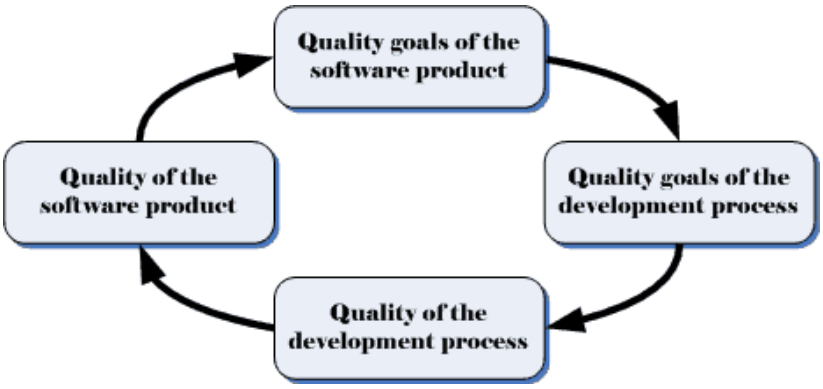


Figure 2.2: Relationship between software quality and the development process

processes and changing these processes to increase product quality and/or reduce costs and development time [Som07]. The relationship between the software development process and the resulting product, as presented by Wallmuller [Wal94], is shown in Figure 2.2. Product quality goals determine setting the goals for the development process; the achieved process quality eventually improves the quality of the resulting software, and increases project success rate.

2.2.1 Process Improvement Process

Software Process Improvement (SPI) is a process itself, including a set of cyclical activities. It is systematic approach to improving software processes in order to improve software products. It borrows basic principles from Total Quality Management (TQM), e.g. the '*plan-do-check-act*' cycle and the emphasis on long-term commitment to customer quality by the entire development organisation. It normally involves three principal stages [Som07]:

1. *Process measurement* Attributes of current project or the product are measured according to the goals of organisation involved in process improvement.

2. *Process assessment* The current process is analysed, and process weakness and bottlenecks are identified. Process models that describe the process are usually developed during this stage.

3. *Process change* Changes to the process that have been identified during *analysis* are introduced.

The steps of process improvement and the relationship between the software process and the methods applied for measurement, assessment and improvement are shown in Figure 2.3, which is revised based on [Pre05].

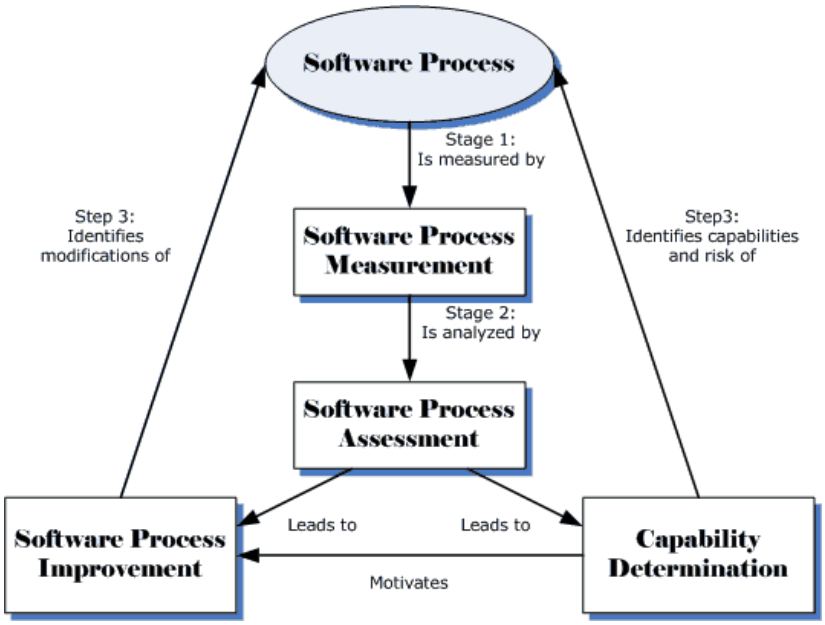


Figure 2.3: Major steps and relations involved in process improvement

2.2.2 Process Improvement Frameworks

Many so-called SPI frameworks have been proposed over the last two decades, such as the Capability Maturity Model (CMM) [PCCW93] & Capability Maturity Model Integration (CMMI) [SEI02b, SEI02a] from Software Engineering Institute (SEI), SPICE (ISO/IEC 15504), BOOTSTRAP [HMK+94], Six Sigma [HS00], and ISO-9001:2000.

CMMI integrates quality recommendations for systems engineering, software engineering, and integrated product and process development. While ISO standards are shaped for any manufacturing enterprise, CMMI is specially designed for technology companies, especially those that deal in some form or fashion with the design and development of software. More discussion on CMMI and its relationship with process simulation modelling is included in Chapter 10.

Unlike ISO and CMMI, Six Sigma is not published or regulated by a central body. It is a process improvement approach based on statistical analysis and geared towards shaping process to reflect the *voice of the customer*. This approach measures quality by measuring a company's ability to meet the needs of its customers through process refinement and improvement. Therefore, it is highly customer-focused and highly data-centric.

2.3 Software Process Modelling

A model is an abstraction of a real world object or system. Modelling a system means capturing and abstracting the system's components, relationships and behaviours, according to the model's objective(s). The modelling activity normally has the following phases: identification, definition, analysis, simulation, and validation.

Software process modelling is an active aspect of software engineering research and practice that has grown since the early 1980s. The objectives and goals of software process modelling and simulation can be concluded as three levels, i.e. cognitive level, tactical level, and strategic level, that are detailed in the next chapter.

The result of process modelling is a process model. A software process model consists of a set of elements or entities and their relationships with a well-defined goal for representing and analysing the processes utilised in developing software. The elements of a software process include activities, products, and roles. The relationships among elements determine the process behaviours over time. Software process models must have the capability of representation, comprehensive analysis, and forecasting [CKO92].

2.3.1 Model Dimensions

The diversity of software processes is large. It reflects that modelling software process is a complicated task. Depending on the modelling purposes and approaches, the process can be viewed in different dimensions. For the objective of research, software process models may be considered as *static* or *dynamic*, *qualitative* or *quantitative*, *black-box* or *white-box*, *descriptive* or *prescriptive*.

Static models refer to the number of elements of a system, and the level of detail in which these elements are described. These models consist of a set of entities and associated attributes, described in terms of variables and relations defined on these. However, static models are not able to capture and represent any sort of change over time of the system modelled. They are snapshots of the essential characteristics of a part of the real world at a certain point of time. In dynamic models, however, time plays an explicit and special role, i.e. the variables have a time index. They employ the concepts of static models but go beyond them by describing behaviour of elements or changes in the complex interactions and by allowing modelers to observe process performance over time.

The decision whether a model is qualitative or quantitative depends on the scales on which the variables contained in the model are defined [FP97], as well as the representation and completeness of knowledge. This thesis introduces semi-quantitative modelling of software process to achieve tradeoff between traditional quantitative and qualitative approaches, and emphasises on the qualitative and

semi-quantitative modelling approaches in software process research.

A *black-box* view implies that the model transforms observable inputs into observable outputs, whereas the values of internal variables and specific functions implied by the model's components (modules) are *unobservable* [Kle08]. On the contrary, *white-box* methods explicitly define the functions and relationships inside the model, where its internal structure is observable to modeler.

Software process models can also be distinguished by two modelling purposes: descriptive and prescriptive process modelling [BKHV97]. Descriptive modelling captures the current software development practices and organisational issues, i.e. process enactment. Prescriptive modelling specifies how software development practices and related organisational issues should be.

The combinations of the above dimensions present a diversity of software process models. The following subsections identify and detail four major types of software process models, which relate strongly to the research reported in this thesis.

2.3.2 Prescriptive Reference Models

Prescriptive Reference Models (PRMs) are static and generic models that can be grouped into *life cycle process models* and *agile process models*. These models are not definitive descriptions of any specific software processes. Rather, they are abstractions of the process that can be used to explain different approaches to software development.

Life-Cycle Process Models

Life-cycle process models (LPMs) prescribe software development processes in stages or phases. Typically these stages are distinguished by the activities performed during each stage. A stage is usually demarcated by entry criteria, which must be satisfied before activities of the stage undertaken, and exit criteria, which must be satisfied to complete the stage.

The life-cycle production process model first appeared in the 1950's as a result of its use in developing air defence software systems, but was not popularised until the 1970's when it became known as the "*waterfall model*" [Roy70]. This model describes the software development structured by a cascade of phases (i.e. *1*- requirements definition, *2*- system & software design, *3*- implementation & unit testing, *4*- integration & system testing, and *5*- operation & maintenance [Som07]), in which the outputs of one phase constitutes the inputs of the next phase. The inability of the waterfall model to describe relationships among activities necessary to ensuring successful project outcomes led to the development of alternative models.

The evolutionary model is characterised by a prototype that is evolved into a

product. The incremental model is featured by the development and delivery of functional increments until the product is completed for final release. The spiral model [Boe86] creates a risk-driven approach to software process. It represents a process that iterates through stages of planning, risk analysis, engineering, and evaluation. The spiral representation reflects the product functionality being increased with each iteration.

Rational Unified Process (RUP), an iterative software development process defined in [JBR99], combines the elements of all of these models. It seeks to provide quality software within time constraints and budget limitations. The object-oriented approach is used within RUP iterations and each iteration results in a prototype.

Agile Process Models

Since 1990s, many small-to-medium enterprises (SMEs) emerged to meet rapid changed market demand in software industry. However, the conventional life cycle process models imply a uniform and ordered sequence of phases in software development but lack sufficient detail to support process enactment, which is unrealistic due to the quick rhythm in change.

The overall goal of agile development is to satisfy the customer by "*early and continuous delivery of valuable software*". Agile processes are not conventional prescriptive processes; they do not describe what to do in every circumstance. Instead, agile process models (APMs) simply offers frameworks and sets of practices that emphasise maneuverability and adaptability. By building flexibility into the development process, agile methods can enable customers to add or change requirements late in the development cycle. Agile Methods contain a collection of agile process models. Some best known examples of agile process models include Dynamic System Development Method (DSDM) [Sta97], eXtreme Programming (XP) [Bec00], Crystal [Coc02], Scrum [SB01], etc.

2.3.3 Analytic Summary Models

Analytic Summary Models (ASMs) consist of relationships defined with mathematical formulae among selected model variables [Kel88]. These models normally focus on high level quantitative relationships between input and output parameters of the software development process. These models are considered to be a '*black box*' because the details of processes are not explicitly presented in the models. The input variables deal with the manageable aspects of the modelled process, such as estimated project size, product requirements, personnel skills and experience, given development environment, etc. The typical output variables, on the other hand, can be observed from external of the process, including required workforce, elapsed time, project expenditure, and expected product quality. Of-

ten these models contain empirically based relationships that can estimate the impact and predict the effect on dependent variables included.

Most of existing ASMs are static and work as black-boxes, and a large majority of them are quantitative. They are developed based on empirical data, but often need to be further calibrated by the real process they are applied in.

In practice, *summary models* can support management decision making regarding the issues relating project planning, in particular software cost estimation. Some well-known examples in this category include the regression models (RMs) and analogy models of cost estimation, such as SLIM [Put80], COCOMO [Boe81] & COCOMO II [BAB+00], and ANGEL [SS97]. However, these models are not meant to provide the manager with dynamic and in-progress feedback.

2.3.4 Explanatory Structural Models

Explanatory Structural Models (ESMs) capture the interrelations, dependencies, and structure of software development at a deeper level than ASMs. So if ASMs consider the software process as a *black box*, the ESMs consider the process as *white box*, in the other words, "*several black boxes making up a system*".

Depending on the analysis requirement, ESMs can be static or dynamic. One example of the static structural models is Bayesian Belief Nets (BBNs), which is based on Bayesian probability theory. Fenton *et al.* have modelled a variety of processes and aspects of software development by using BBNs, such as risk assessment, resource planning, and quality prediction [FMN+04, FNM+07].

A dynamic example of an ESM is the system dynamics model (SDM) developed by Abdel-Hamid and Madnick [AHM91]. This model consists of seven sectors that capture the rich relationships among workforce, productivity, quality, and activities completed. Each sector is devoted to specific high level process blocks of software development, quality assurance, testing, and management. Some continuous work using dynamic structural modelling which has been based on Abdel-Hamid and Madnick's pioneer work includes [Mad94], [Tve96], [Rus98], [Hou00], and [Pfa01].

ESMs can be quantitative or qualitative as well. However, most of previous models emphasise the *quantitative* aspect of this modelling approach. One major contribution of this thesis is to provide a comprehensive exploration of the use of qualitative dynamic models using qualitative simulation (QSIM).

2.3.5 Descriptive Enactment Models

Process models, as defined in [CKO92], provides operational guidance regarding the critical sequence of process steps, information flows, entity flows, and organisational policies and responsibilities. Thus, these models possess the capability

to check the integrity of the process, as well as support process improvement. As '*process models*', in the broad sense, include the previously mentioned models (i.e. PRM, ASM, and ESM). Here, the term, Descriptive Enactment Model (DEM) is used to refer to a fine-grained process model.

These models are distinct from ASMs and ESMs because they delve into the details of the process used to develop software. For this reason, operational modelling corresponds to the *micro-process* research.

Many tools, methods, and languages have been developed for descriptive modelling of software process, such as Little-JIL [WCL+00], Adele [EVL+03].

On the other hand, DEMs can be dynamic and able to be integrated with dynamic ESMs. One example of dynamic DEMs is discrete-event simulation (DES) model, which were comprehensively reflected by the work reported in [Raf96] and [Mar02] .

Table 2.1 concludes and highlights the advantages and limitations of these four process modelling approaches. Table 2.2 identifies the characteristics of these process models in model dimensions.

Table 2.3 reflects specific type of process model in support of the research on different software process levels. The ASMs and ESMs correspond to *macro-process* research in software engineering process domain. On the contrary, most of PRMs and DEMs emphasise *micro-process* research. The interpretation and integration between these models can offer more insights into the software processes investigated. For instance, our work reported in [HZJ06] and [HZJ08] attempts to bridge the gap between PRM and real process enactment (DEM) by conducting process enactment analysis (PEA).

The selection of process models depends primarily on the requirements of the prospective model user (e.g. describing a current or planned process, or predicting process outcomes), the process requiring modelling (e.g. micro- or macro-process), and the information available about the process (i.e. descriptive information or quantitative information).

2.4 Software Process Simulation

A simulation model can be a *static* or *dynamic* model that is meant to be solved by means of experimentation. Monte Carlo Simulation is a typical *static* simulation method. However, most process simulation models are *dynamic* and executable, which means model's behaviours vary over time. A simulation model is the only method to study new, non-existent complex dynamic systems for which analytic or static models provide at best a low fidelity model with corresponding low accuracy [Car03]. Accordingly, a good simulation model provides not only numerical measures of system performance, but insights into system behaviours.

Table 2.1: Software process modeling approaches

	Advantages	Limitations
PRMs	•Provide high level process guidance	•Lack operational detail
	•Used for reference process models	•Do not support prediction & evaluation
	•Tailorable for specific process	•Do not support sensitivity analysis
	•Easy for understanding & communication	
ASMs	•Provide system level focus	•Difficult to understanding
	•Good for sensitivity analysis	•Tie to mathematical & statistical theories employed
	•Rely on empirical data	•Difficult to handle change
	•Efficient reuse	•Do not support operational guidance
	•Capture the effects of selective relationships	•Do not support handling exceptional circumstances
	•Support external validation & benchmarking	
ESMs	•Provide high level focus on structure	•Difficult to handle exceptional circumstances
	•Capture causal-loops relationships	•Do not provide operational guidance
	•Capture chain-effected interdependencies	•May overlook specific process issues
	•Can be based on empirical data	•May contain subjective assumptions
	•Built by accumulation of analyzable formulae	•Time-consuming model development
	•Portions or relations can be reused	
	•Support external validation & benchmarking	
	•Good for in-depth understanding	
DEMs	•Provide operation level focus	•Only single process focused
	•Detail process & provide operational guidance	•Require detailed process knowledge
	•Support process definition and management	•Difficult for reuse
	•Support multiple levels of abstraction	•May contain subjective inputs
	•Make use of analytical formulae & empirical data	•May overlook system level KPIs
	•Support handling exceptional circumstances	

Table 2.2: Typical software process models

	Static	Dynamic	Qualitative	Quantitative	Black-box	White-box	Example
PRMs	●		●	●		●	LPMs APMs
ASMs	●			●	●		RMs
ESMs	●	●	●	●		●	BBNs SD QSIM
DEMs	●	●	●	●		●	DES PEA

Table 2.3: Process models on different levels

	Macro-process	Micro-process
Static	ASMs ESMs	PRMs DEMs
Dynamic	ESMs	DEMs

However, simulation is often time-consuming, and is composed of the following steps [Car03, BCNN05]:

1. **Initiation**: problem formulation, setting of objectives, and definition of model scope;
2. **Model development**: model conceptualisation, assumptions documentation, and data collection & analysis;
3. **Verification & validation**
4. **Experimentation, analysis & reporting**: experimental design, production runs & analysis, and documentation & reporting.

Software process simulation has become an increasingly active thread in software process modelling research. Basically, the simulation can be conducted at two process research levels, i.e. *macro-process* and *micro-process* levels (shown in Table 2.3). The important examples of dynamic models at these two granularity levels are ESMs and DEMs respectively.

System dynamics (SD) models are the most constructed *dynamic* ESMs. They use the widely applied *continuous simulation* paradigm which captures higher level project or product considerations and shows how feedback loops connect a variety of business characteristics.

As the common approach to build a *dynamic* DEM, discrete event simulation is the modelling of systems in which the state variable changes only at a discrete

set of points (events) in time. This approach is excellent at capturing well-defined process tasks, incorporating, queueing and scheduling considerations.

Moreover, the systematic review [ZKP08a] (cf. Chapter 3) identified 10 simulation modelling paradigms (including SD, DES, etc.) in support of software process research during the period from 1998 to 2007. Most of them are detailed in Chapter 3.

2.4.1 Important Simulation Models

Process simulation methods were introduced to software engineering by Abdel-Hamid's [AHM91] and others' pioneering efforts in the late 1980s. Their model captures the managerial aspects of a waterfall software life-cycle and was regarded as the starting point for other subsequent models of either the entire development process or parts of it. In the last two decades since then, software process simulation modelling (SPSM) has been emerging as an effective tool to help investigate and improve software processes in software engineering practice. Primarily most innovative research into SPSM models appeared in PhD theses which are summarised chronologically below.

Madachy developed a dynamic simulation model of inspection-based software life-cycles process to support quantitative process evaluation and risk assessment [Mad94]. His SD model serves to examine the effects of inspection practices on cost, schedule, and quality throughout the project life-cycle by interrelating flows of tasks, errors and personnel in each development phase.

Tvedt developed a SD model of an incremental life-cycle and investigated the impact of inspections on the production cost, the development time, and the number of defects in the delivered system [Tve96].

Raffo developed a DES model based on ISPW-6 process example [KFF+91], to quantitatively evaluate the performance of alternative software processes and process changes in terms of quality, cost and schedule [Raf96].

Rus described the use of process simulation to support software project planning with given quality requirements [Rus98]. The modelling effort focused on software reliability, but is applicable to achieving tradeoff between software quality, project cost, and delivery time. The SD model was integrated into a decision support system.

Houston developed an SD model in supporting management of particular risks to software projects [Hou00]. This model identifies the significant software development risk factors, and experiments upon their effects to software projects using stochastic simulation.

Pfahl designed an SD-based framework for Integrated Measurement, Modelling, and Simulation (IMMoS). This framework supports both strategic and project management in software organisations by providing guidance on developing and using quantitative process simulation models as a source for learning

and improvement [Pfa01].

Martin developed a hybrid simulation model of the software development process that combines the two major paradigms: SD and DES. This model allows the investigation of the impact of changes to process within the context of different assumptions about the project environment [Mar02].

The 1998 ProSim workshop* was the first major research event concentrating on SPSM. The state-of-the-art of SPSM after this event is systematically reported in Chapter 3.

2.5 Summary

This chapter introduces the modelling object in this thesis, i.e. software engineering process, as well as explains the fundamental concepts to my research, e.g. software process improvement, software process modelling and simulation.

The process simulation models mentioned in this chapter only consider the quantitative aspect of dynamic process modelling. None of them consider the possibility of the qualitative alternatives.

The next chapter introduces the software process simulation modelling (SPSM) with reference to a comprehensive systematic literature review of SPSM over the decade from 1998 to 2007.

*International Workshop on Software Process Simulation Modelling

Chapter 3

Software Process Simulation Modelling

Chapter 2 provides a concise introduction to software process and process modelling. Software Process Simulation Modelling (SPSM) has become an increasingly active research area in software process domain since it was introduced into the software engineering domain by the pioneering work summarised in [AHM91]. In the last two decades, the number of publications in this area has been growing. This approach provides effective tools to help evaluate and manage process changes made to software projects and organisations.

This chapter serves as an introduction to SPSM. First, it describes the state-of-the-art of SPSM based on the results of a well-defined and constrained secondary study, a systematic review*. Next, the outputs of the systematic review is further used to frame the introduction to SPSM, i.e. the answers to the '*why*', '*what*' and '*how*' questions. The important *trends* and *directions* are also discovered based on the review results. Finally, the results are used for design of the research reported in the chapters in Part III of this thesis.

3.1 State-of-the-Art: A Systematic Review

3.1.1 Background

As a major research event of SPSM, the ProSim workshop series has taken place since 1998, and focuses on the state-of-the-art theories and applications of SPSM research in addressing real-world problems. After that, during the last ten years, the related research community and the number of publications has been growing.

*This systematic review has been partially reported in [ZKP08a], [ZKP08b] and [ZKP08c].

In ProSim'98, Kellner, Madachy, and Raffo (KMR) discussed a variety of aspects of software process simulation in their widely-cited paper, "*Software process simulation modelling: Why? What? How?*" [KMR99], such as the reasons for undertaking simulations of software process models, and simulation approaches/paradigms. However, after almost 10-years (1998-2007) progress in software process simulation, it is appropriate to review and update the status of SPSM research, to summarise the 10-year progress, best evidence, and propose the possible directions of our future research activities in this domain.

From this point, this chapter reports the preliminary results of a systematic literature review of papers published in the proceedings and journals associated with ProSim since 1998. The review reported here is part of a larger study and presents only a subset of the research questions and research literature addressed by the larger study. As an anniversary review of the previous work, this chapter also includes part of the latest continuation and enhancement to the topics in the KMR's paper.

3.1.2 Systematic Literature Review

In 2004, Kitchenham *et al.* [KDJ04, DKJ05] suggested software engineering researchers should adopt "Evidence-Based Software Engineering" (EBSE). EBSE aims to apply an evidence-based approach, which was initially developed in medicine and is being adopted in many domains, to software engineering research and practice. In this context, evidence is defined as a synthesis of best quality scientific studies on a specific topic or research question. The main method of synthesis is a Systematic Literature Review (SLR).

In contrast to an ad-hoc literature review, a systematic literature review (also known as systematic review) is a methodologically rigorous review of research results. It is a means of identifying, evaluating and interpreting all available research relevant to a particular research questions, or topic area, or phenomenon of interest [Kit07]. A systematic review is a form of secondary study, the identified individual studies contributing to a systematic review are called primary studies.

An SLR involves several discrete activities, which can be grouped into three main phases: *1*) planning the review, *2*) conducting the review, and *3*) reporting the review. A pilot review is recommended for the reviews including multiple research questions or a large set of primary studies.

3.1.3 Methods

This study follows Kitchenham's methodological guidelines for systematic reviews [Kit04, Kit07], as adapted for PhD candidates. The primary objective of this research is to provide insights about the evolution of SPSM research during the last 10 years. This chapter reports the review process and includes the

preliminary results from the current stage. Three researchers, from three different software engineering research organisations, were involved in this research, including one principal reviewer, one secondary reviewer, plus another researcher acting as the expert panel.

Research Questions

The systematic review is intended to answer the following research questions. This chapter addresses the review and answers to question 1 to 5, which are highly related to the objective of this chapter, i.e. introduction of SPSM and research design of this thesis research.

Q1. What are the purposes or motivations for SPSM in the last decade's practice? Q1 can be split into two sub-questions: **Q1.1** How are the purposes identified by KMR supported by SPSM practice in the last ten years? **Q1.2** Are any updates required?

Q2. Which simulation paradigms have been applied in the last decade, and how popular are they in SPSM? Are there any new techniques emerging during this period?

Q3. Which simulation tools are available for SPSM and have been in use in the last decade? And how popular are they?

Q4. On model level, what are problem domains and model scopes focused on by software process simulation models?

Q5. On parameter level, what are the output variables considered when developing simulation models of software process?

Search Process

As the review of 10-years' efforts in SPSM, the time frame of sources for this study is constrained to the period from 01/01/1998 to 31/12/2007. Because the ProSim workshop series (which continued as a special track of ICSP since 2007) are regarded as the most important forum of SPSM, the sources related to ProSim (including ProSim workshop, simulation track of ICSP, and special issues of JSS and SPIP) are the primary data sources for this study. The corresponding sources and search strategy are summarised in Table 3.1.

Manual search was carried out in the ProSim conference proceedings and special issues of the journals published within the proposed time frame during the review. There are over 200 candidate papers. When there are a large number of research questions and a large set of potential primary studies, Kitchenham recommends undertaking a pilot review after the planning phase [Kit07]. The purposes of a pilot systematic review is "*to assess and refine review protocol, and*

Table 3.1: Selected sources of the systematic review

Source	Acronym	Period	Search method
Proceedings of ProSim Workshop	ProSim	'98 - '06	Manual
Proceedings of ICSP Conference	ICSP	'07	Manual
Journal of Systems & Software	JSS	'99 - '01	Manual
Journal of Software Process: Improvement & Practice	SPIP	'00 - '08	Manual

further to secure the quality of the systematic review". The pilot review chose the papers published in the special issues (SPIP) of ProSim Workshop 2005 and 2006 (10 latest journal papers available at the start of the pilot review), which reflect the current state and progress of SPSM.

Inclusion and Exclusion Criteria

There are two major steps in primary study selection: an initial selection and a final selection. The theme for this systematic review, "*software process simulation modelling*" contains two keywords: '*software process*' and '*simulation modelling*'. Therefore, as the inclusion criteria, the primary studies identified must employ simulation paradigm(s) for software process research; in the other words, the process model or modelling in the publications can be used for simulation studies.

From the candidate studies retrieved by data sources (Table 3.1), an initial selection was obtained by reviewing the title, abstract, and keywords of the publications. When an exclusion decision could not be made, the paper's structure, conclusion, and reference were also checked. Unless studies could be excluded based on the above criteria, full copies of the papers were obtained and included in the initial selection.

Next, a final selection that satisfied the selection criteria was obtained from the initially selected papers. The following articles were excluded from the primary studies:

1. Editorials, position papers and keynotes;
2. Abstracts, posters and slides alone;
3. Proposals or uncompleted work.

In addition, the review only included the most recent and comprehensive versions of duplicated papers or continued studies. For instance, some SPSM papers published in the ProSim workshop series were selected for publication in special issues of journals. To avoid duplicate aggregation, the journal articles were only

selected and reviewed as they normally enhanced the proceedings papers with more details. However, to track any trends over time, the first publication date and source of the original research paper was recorded during data extraction. The selection process was performed by the principal researcher.

Study Classification

Initially two broad categories of publications were identified by briefly reviewing the most recent papers published in SPW/ProSim 2006 and ICSP 2007 (16 papers on the special tracks of process simulation, the most recent papers by "*planning the review*" phase). One category includes the specific process simulation models or simulators, and their applications; the other discusses the methodology and guidelines for process simulation modeling. Both categories are relevant to most research questions (except Q5). Furthermore, based on the results of the pilot review, the study classification was refined as four categories as follows:

A. Software process simulation models or simulators;

B. Process simulation modelling paradigms, methodologies, and environments;

C. Applications, guidelines, and solutions for adopting process simulation in software engineering practice;

D. Experience reports of SPSM research and practice.

These four types of studies focus on different aspects of software process simulation research, and may give answers to the different research questions and from different points of view. Figure 3.1 shows the relationships between these study categories and their roles in process modelling and simulation adoption. Category B studies introduce and provide effective paradigms, methods and tools for constructing process simulation models or simulators (Category A studies). These simulation models can be further adopted in software industry by following the practical solutions or guidelines (Category C studies) in given organisation's context. The experience (Category D studies) collected from modelling and adoption can be used as feedback to improve SPSM research and practice.

Table 3.2 defines the concrete criteria (questions) to facilitate the effective identification of each study category. The categorisation was not a mutually exclusive one, i.e. it is possible that a specific study falls into more than one category. For example, one case could be that the author(s) introduced a novel simulation paradigm to software process research (Category **B**), and then described a simulation model for a specific problem domain by using this paradigm as an example (Category **A**). These studies were allowed to be mapped to multiple categories.

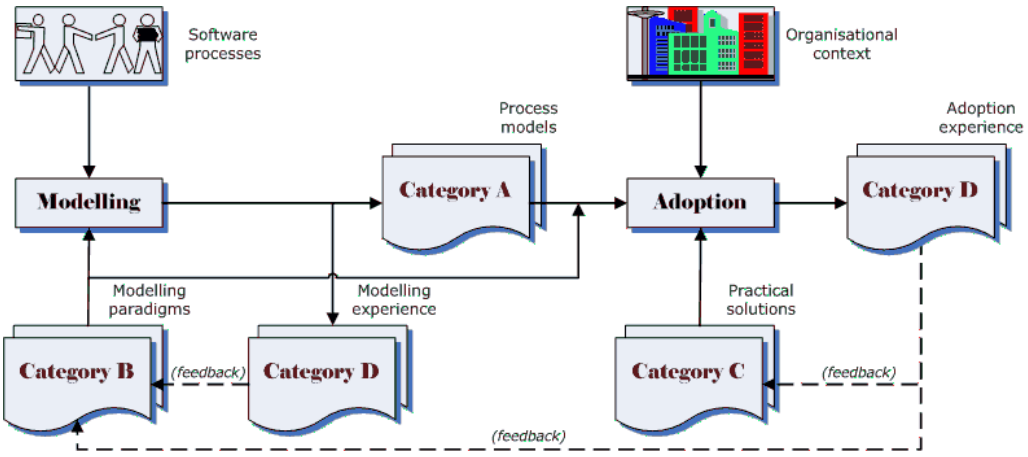


Figure 3.1: Relationships between study categories

Table 3.2: Questions for identifying study categories

Category	Question for identification
A	(a) Is a new process simulation model or simulator presented in the study? (b) Is a specific process simulation model or simulator applied in a new software engineering domain or practical context?
B	(a) Compared with previous studies, is a new simulation modelling paradigm introduced into SPSM? (b) Is a new process simulation environment or tool developed and described? (c) Is a methodology or framework proposed or developed for improving SPSM? (d) Are any factors associated to the modelling of SPSM research discussed in the study?
C	(a) Is a new application or solution based on SPSM introduced to software engineering practice? (b) Is a guideline of directing SPSM being adapted into one specific problem or context proposed or developed?
D	(a) Does the study report any experience (qualitative or quantitative) of applying SPSM in industry? (b) Does the study report best practices or lessons learnt in SPSM research? (c) Is a process simulation model or simulator built or calibrated with empirical data?

Data Extraction

The major attributes to be collected for each study through the review are listed in Table 3.3. They are grouped by the study categories. The '**Q**' column indicates which research question(s) is the attribute collected for answering.

The quality of a primary study is assessed according to a question checklist, which specifies 20 questions to study categories. To maintain the emphasis of this chapter, the detail of quality assessment is not included here, but can be found in Appendix A.

3.1.4 Results

Primary Studies

In total, 209 papers have been published in the ProSim sources, including the workshop and conference (ICSP) proceedings and the special issues of JSS/SPIP. They form a comprehensive body of knowledge of software process simulation and modelling. Unfortunately, because the electronic proceedings were not available for ProSim'98 - '00, nine papers could not be evaluated in the current stage of review. Although the author(s) for each missing paper were contacted individually to request the electronic version, only three of them had responded to us. Hence, there are around 4.3% (9 out of 209) papers missing from the review at the current stage. Nevertheless, the low proportion will not influence the review results significantly, particularly for the recent state of SPSM.

By carefully reviewing their titles, abstracts, keywords, conclusions and references, 96 articles were selected from the publications in ProSim sources and identified as the primary studies. The total number of papers per year and source are summarised in Table 3.4. The full list of primary studies and their corresponding categories are included in Appendix A.

Data extraction was performed by two researchers: the principal and secondary reviewer. The former was responsible for reviewing all primary studies, extracting data, and assessing study quality. The other reviewer selected approximately one third of the papers and performed a secondary review for validation of the extraction and assessment. When the disagreement could not be resolved, the final decision was made by the principal researcher.

Table 3.5 summarises per year the number of different countries the first authors came from. Contributions to ProSim were mainly from 13 countries. The ProSim workshop became more international since 2000, when the first authors from 7 countries were involved in ProSim 2000. After that, the number of participating countries varied between 4 and 6.

The results also indicate that USA is the leading country of SPSM research in terms of ProSim publications, where 41 (49%) studies were originated. It is followed by Germany (18%) and UK (17%).

Table 3.3: Attributes collected during data extraction

Q	Attribute	Description
Common Attributes		
1	Purpose category	The specific purpose for the simulation model or modelling. It can be one of purposes identified by KMR, or any new ones.
2	Modelling Paradigm	The paradigm used to build the simulation model. It can be one of identified by KMR, or some other approaches.
Attributes for Category **A**		
4	Problem domain	The specific problem domain in software engineering, e.g. open-source, evolution.
5	Model complexity	Including single-module model or integrate model, the number of modules and levels of the simulation model.
3	Simulation tool	The simulation tools used in executing the process model.
5	Model scope	Including the process phase(s) of life-cycle, and time span.
5	Output variables	The information produced through simulation answers the questions specified with the purpose of the model.
Attributes for Category **B**		
4/6	Study's theme	The emphasised and discussed aspects of SPSM in the study.
Attributes for Category **C**		
4/6	Focused questions	The specific questions related to SPSM raised in the study.
4/6	Proposed solution	The corresponding answers given in the study.
6	Application effects	The expected effects caused by the solution.
Attributes for Category **D**		
6	Experience source	Where does the experience come from: Industry, government, education/academia, or somewhere else.
6	Outcome of applying SPSM	The result of the application experience, i.e. positive, negative, or mixed.
1/6	Supported arguments	The arguments supported by the experience report.

Table 3.4: Sources identified as primary studies

	'98	'99	2000	'02	'03	'04	'05	'06	'07	**Total**
Proc.	15	13	21	0	32	27	24	8	8	**148**
missing	2	1	6	0	0	0	0	0	0	**9**
JSS	11	0	12	0	0	0	0	0	0	**23**
SPIP	0	10	0	7	5	7	7	2	2	**40**
selected	13	9	14	7	16	10	13	6	8	**96**

Table 3.5: Number of countries involved in ProSim series

	'98	'99	2000	'02	'03	'04	'05	'06	'07	**Total**
Number of country	3	3	7	5	5	5	6	4	6	**13**

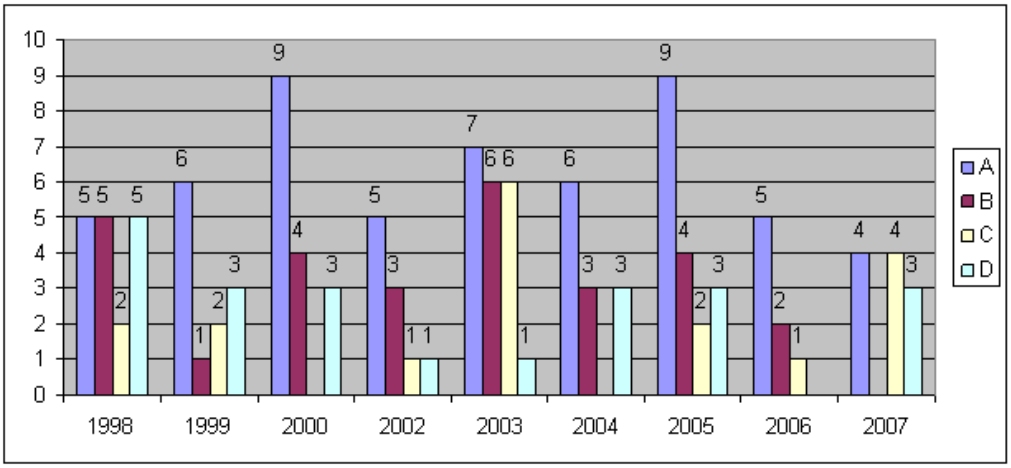


Figure 3.2: Summary of study category distribution

Classification

Four study categories were identified in the pilot review. By reviewing the full papers, all primary studies were classified into at least one category (A, B, C, and D), as defined in Section 3.1.3. Figure 3.2 shows the distribution of studies per category and year.

Most primary studies were identified as Category A, for both the decade (58%) and each year separately. In total, there have been 61 software process simulation models (simulators) developed and published in ProSim series during the last 10

years. Only 18 primary studies (19%) were identified as Category C, while 29% of primary studies were of Category B and 23% of Category D. 22 studies were classified into two categories, and 3 studies were identified as combinations of three categories.

Most studies (17 papers) of Category B discussed methods of constructing process simulation model more correctly, effectively and efficiently. Some papers introduced novel simulation paradigms (8 studies) and simulation environments (8 studies). 6 studies dealt with the strategic questions, or presented perspectives of SPSM. KMR's paper [KMR99] is the best known example among them.

The low proportion of Category C study implies that modelers may need to pay more attention to the methodologies and guidelines needed to support the application and adoption of process simulation modelling in software practice.

3.2 Why Simulation?

3.2.1 Purposes

In the first ProSim Workshop (1998), Kellner, Madachy, and Raffo presented a wide variety of reasons for undertaking simulations of software process models [KMR99]. Primarily, process simulation is an aid to decision making. They further identified six categories of purposes: *1*) strategic management; *2*) planning; *3*) control and operational management; *4*) process improvement and technology adoption; *5*) understanding; *6*) training and learning.

During the process of the review, the difficulty in clearly handling the purpose identification was gradually perceived in terms of the definitions addressed by KMR. They have two major shortcomings: *1*) ambiguity to some extent exists among their 6 purposes; *2*) since the examples given in KMR's paper were mainly derived from the publications in ProSim'98 and literature prior to the event, the scope of their purposes is limited; *3*) their research and arguments were not based on a systematic and rigorous review methodology, like the systematic literature review.

For instance, in terms of their categorisation, '*planning*' is different from '*strategic management*'. But the latter usually consists of the former at the organisational level or on a long-term scale. As another example, process simulation can help predict software size in open source development. However, according to the definition in PMBoK [PMI04], '*planning*' is the process that contains the activities of '*define scope*', '*develop management plan*', '*identify and schedule activities and resources*', which are not the cases in open-source development. Therefore, '*planning*' needs to be refined as '*prediction and planning*' to fit such change.

To clearly present the purposes for SPSM research identified from the primary studies, they were grouped at three levels:

Table 3.6: SPSM purposes, levels, and model scopes

Purposes	Scope	single phase	multi-phase	project	multi-project	product	evolution	long-term org.
	Cognitive		**Tactical**			**Strategic**		
Understanding	●							
Communication	●							
Process investigation	●							
Training & learning	●							
Prediction & planning				●			●	
Control & operational mgmt				●			●	
Risk management				●			●	
Process improvement				●			●	
Technology adoption				●			●	
Tradeoff analysis & optimising				●			●	

1. **Cognitive level**
2. **Tactical level**
3. **Strategic level**

They can be further detailed as 10 purposes. Table 3.6 shows the their ascription and relationship with model scope (see Section 3.3). The ***cognitive*** level contains the purposes of *1*) understanding, *2*) communication, *3*) process investigation, *4*) training and learning. On the ***tactical*** and ***strategic*** levels purposes are similar. They are *5*) prediction and planning, *6*) control and operational management, *7*) risk management, *8*) process improvement, *9*) technology adoption, *10*) tradeoff analysis and optimising. They differ in scope and impact between the two levels.

3.2.2 Benefits

Software process simulation has several advantages over the other methods of modelling and analysing software development processes, such as:

- Dynamic simulations model the structure and behaviour of the development process in varying levels of degree using flows and/or discrete events. As a result, a simulation can address the dynamic and human factors of development processes.

- Feedback loops enable a dynamic simulation to model the nonlinear behaviours exhibited in a development process.
- Though a simulation uses calculations, it does not require that an underlying complicated mathematical model of the system has been identified. Therefore, it avoids factor independence required in regression analysis [Boe81].

3.3 What to Simulate?

Many aspects of what to simulate are inter-related and driven based on the model purposes and key questions described in Section 3.2. This section reflects three aspects of *what to simulate* derived from the systematic literature review, in terms of *1*) problem domains, *2*) model scopes, and *3*) output variables, which are later used for the research design of this thesis work in Section 3.6.

3.3.1 Problem Domains

Problem domain is the primary aspect to answer what to simulate. It identifies the problem(s) in software engineering that the simulation model investigates. It further determines the model's structure, input parameters, and output variables. The systematic review extracted 19 problem domains from Category A studies (shown in the leftmost column of Table 3.7). '*Generic development*' models the normal development process of software project. Among other domains, '*software evolution*' has been the most frequently modelled area.

3.3.2 Model Scopes

Model scope specifies the boundary of a simulation model in two dimensions: time span and organisational breadth. To more properly review and classify the model scopes of the published simulation models, the scopes were extended from 5 (defined by KMR) to 7:

- **single phase** (e.g. some or all of design or testing phase)
- **multi-phase** (more than one single phase in project life cycle)
- **project** (single software project life cycle)
- **multi-project** (program life cycle, including multiple, successive or concurrent projects)
- **product** (software product life cycle, including development, deployment, and maintenance.)

Table 3.7: Modelling problem domains vs. model scopes

Domain	Scope	**single phase**	**multi-phase**	**project**	multi-project	product	**evolution**	long-term org.	unknown or n/a	**Total**
generic development				9					1	**10**
software evolution				1			7			**8**
software process improvement		1	1					1	3	**6**
incremental development		1	2	1					1	**5**
requirement		2		1			1		1	**5**
open-source development		1			1		2			**4**
global development				1			3			**4**
software economics		1		1		1				**3**
software product-line						1		1		**2**
agile process				1					1	**2**
quality assurance			1						1	**2**
acquisition/outsourcing			1						1	**2**
software engineering education				2						**2**
software test		1								**1**
software design		1								**1**
software services						1				**1**
productivity analysis				1						**1**
risk management				1						**1**
software reliability									1	**1**
subtotals		7	4	19	1	2	8	2	9	

- **evolution** (long-term product evolution, including successive releases of software product, i.e. software product line)

- **long-term organisation** (strategic considerations or planning spanning releases of multiple products over a substantial time span)

Table 3.7 shows the relations between modelled domains and scope extracted from Category A studies. Notice that the number in the 'subtotal' row is not always the exact sum of the column. This is because some studies modelled multiple domains, e.g. a combination of global development and evolution.

'*project*' is the most frequently modelled study scope, particularly for '*generic development*'. '*evolution*' is the next most studied process scope.

Table 3.8: Summary of simulation outputs

Output	Description	Number of studies	Percentage
Time	Project schedule or elapsed time	20	35.7%
Effort	Effort or cost	16	28.6%
Quality	Product quality or defect level	11	19.6%
Size	Requirement size or functionality	11	19.6%
Resource	Resource or staffing level	7	12.5%
Plan	Project or development plan (e.g. task allocation)	3	5.4%
ROI	Return on investment or cost/benefit analysis	2	3.6%
Productivity	Development productivity	1	1.8%
Market share	Product market share	1	1.8%
Index	Nominal index	1	1.8%
Behaviour	Behaviour patterns	1	1.8%
Flow	Process flow	1	1.8%

3.3.3 Output Variables

The systematic review identified 12 model variable as simulation outputs from the Category A studies (shown in Table 3.8). The third column indicates the number of studies including the leftmost output variable, and the fourth column shows the corresponding percentage in Category A studies (divided by 56, the number of Category A studies). Note that there are simulation studies with multiple outputs.

In terms of Table 3.8, it is evident that *time*, *effort*, *quality*, and *size* are most common drivers for simulation study. There are 62.5% studies (35 out of 56) including any one of them or their combination as model outputs.

3.4 How to Simulate?

3.4.1 Simulation Paradigms

The diversity and complexity of software processes and the richness of research question (concluded into simulation purposes in Section 3.2) determine the different capabilities of simulation paradigms needed.

Overall, 10 simulation modelling paradigms were found in the current stage of the review. Figure 3.3 shows the paradigms with the applied study number more than one. System dynamics (SD, 49%) and Discrete-event simulation (DES, 31%) are the most widely used techniques in SPSM. Other paradigms

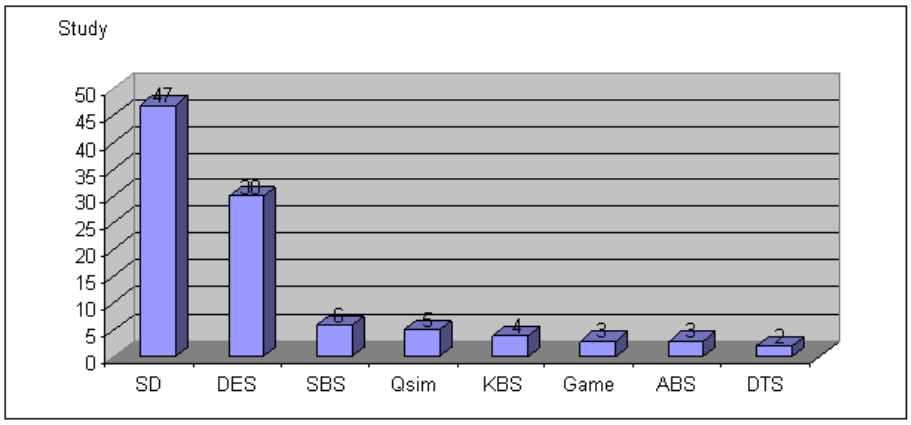


Figure 3.3: Study distribution by simulation paradigms

include state-based simulation (SBS), qualitative(or semi-quantitative) simulation (QSIM), knowledge(rule)-based simulation (KBS), role-playing game (RPG), agent-based simulation (ABS), and discrete-time simulation (DTS). However, only SD, DES, SBS and KBS were discussed by KMR.

QSIM and ABS are paradigms that are relatively new to software process research and will be further described in Section 3.4.3 and Section 3.4.4. As a special case, DTS is classified as one type of SBS by some studies. In addition, three games (role-playing simulators) were developed with focus on training and learning purposes.

During the last decade, hybrid simulation has been one of the most frequent research themes in the ProSim community. Most of these studies (10 papers) presenting the hybrid simulation focus on the combination of continuous (SD) and discrete-event (DES) simulations. Considering the importance in the findings of this review and relevance of these paradigms to this thesis, the following subsections briefly introduce process simulation paradigms with emphasis on SD, DES, QSIM, ABS, and RPG. The descriptions of Other paradigms identified in the review can be found in the corresponding primary studies listed in Appendix A.

3.4.2 Continuous vs. Discrete Simulation

In the first ProSim workshop, KMR identified 8 software process modelling approaches and languages [KMR99], 6 of them are relevant to dynamic simulation, but they can be grouped into four simulation paradigms:

- **System dynamics**
- **Discrete-event simulation**, including *queueing models*.
- **State-based simulation**, including *Petri-net models*.

- **Knowledge-based simulation**, including *rule-based languages*.

They can be further classified into *continuous* and *discrete* simulation approaches. System dynamics and discrete-event simulation are the most typical paradigms for each category.

System Dynamics

System dynamics (SD), introduced by Forrester's pioneer work [For69], is used for modelling complex dynamic systems with continuous changes. SD is capable to "*deal with the time-dependent behaviour of managed systems with the aim of describing the system and understanding, through qualitative and quantitative models, how information feedback governs its behaviour, and designing robust information feedback structures and control policies through simulation and optimisation*" [Coy96].

These models are formulated using continuous quantities interconnected in causal-effect relationships and feedback loops. The quantities are expressed as *levels* (or *stocks*), which are the current values of variables that have resulted from the accumulated difference between inflows and outflows and represent the dynamic system's current state, and *rates* (or *flows*), which are considered as control variables that represent the activity in dynamic system and determine the *levels*. The directed information links between *rates* and *levels* represent the feedback loops. These levels are affected by the flow rates, and the flow rates may be affected by the levels as well.

SD models are quantitative. The basic mathematical representation of an SD model is a system of coupled, nonlinear, first-order differential equations,

$$x'(t) = f(x(t), p) \tag{3.1}$$

where x is a vector of levels, p is a set of parameters, and f is a nonlinear vector valued function. State variables of the modelled systems are represented by the *levels*.

The construction of an SD model can be implemented in five distinct stages [Coy96]: *1*) problem recognition, *2*) problem understanding and system description (drawing causal-loop diagram), *3*) qualitative analysis, *4*) simulation modelling (developing and testing model), and *5*) policy testing and design. Causal Loop Diagram (CLD) is an indispensable tool facilitating the Stage 1 to 3 in SD modelling.

System dynamics has been successfully adopted and continuously centred in SPSM research for the past two decades, since Abdel-Hamid and Madnick's pioneer work [AHM91]. The power of modelling a software development process using SD lies in its ability to take into account a number of product and process factors that affect reliability, cycle time, and cost to determine the global impact

of their interactions. Most of important process simulation models discussed in Section 2.4.1 were SD based as well.

Though '*qualitative analysis*' is one important step during the SD modelling, the simulation model is solely used for *quantitative* study. More detail of SD modelling approach is also included in Chapter 8 for the comparison with qualitative and semi-quantitative simulation modelling.

Discrete Event Simulation

Discrete-event simulation (DES), also called discrete-event system specification (DEVS), is concerned with the modelling of *discrete system* that can be represented by a series of *events*. The state variables of a discrete system change only at *discrete* set of points in time [BCNN05]. The simulation of the system describes each discrete event, moving from one to the next in the right order as time progresses. Activity Cycle Diagram (ACD), an essential part of DES, provides the means of describing the logic of a simulation model and represents the interactions among system objects (entities) [Pid04].

One typical *discrete system* is a queueing system that is described by its calling population, the nature of the arrivals, the service mechanism, the system capability, and the queueing discipline. The *state* of the system is the number of units in the system and the status of the *server* (busy or idle). An *event* is a set of circumstances that causes an instantaneous change in the state of the system.

The measurement of time in a simulation corresponds to appropriate units of time in the real system. The simulation clock indicates the time of the next event to be performed. A simulation, starting at time zero, performs all events in the order in which they occur, and runs until either there are no more events to perform, or the clock time exceeds the given duration, or some interrupt is triggered.

In software development, developers are represented as *servers* in a DES model. They perform development *activities*, e.g. coding, inspection, testing, and reworking, in given *delay*. The common *entities* are the artifacts in development processes, such as documents, function points or modules, defects. Some DES models of software processes include [Raf96], [HRD+01], [Pad02], [CGC06], and [AEPR08].

Comparison between SD and DES

These two classic and popular process simulation are different from each other in the many aspects, such as:

View of System DES tends to focus more on individual elements than SD. The results from DES are sometimes difficult to interpret as a whole due to the high degree of interaction between entities. It is therefore corresponding to

the *micro-process* research and not the most appropriate for constructing complex models. SD on the contrary tends to portray the system as a whole. Its feedback structure can be used to study the interaction of control policies and the dynamic behaviour of the system. Thus, it is more appropriate for the *macro-process* research than DES.

Loop Structure A discrete system more often has an open-loop structure than a continuous system or system dynamic approach, which tends to have a closed-loop structure. In an open-loop system, the output arises from the input but often has no effect on the input.

Time Span It is probably the most distinctive difference between these two approaches. In DES, time may advance in large jumps until the next event. Its span between events is not the important concern of the model. In contrast, time is considered as a specific variable of direct interest in SD modelling. It is divided into many equally small steps in computer simulation, where the system state is evaluated.

Precision In DES modelling, each entity in the system is monitored during simulation. All decision rules are sampled from the predefined distributions. The entities are moved from one queue to another. In SD modelling, approximation is usually used instead of the exact numbers of entities. The average value of the variables (*levels* and *rates*) is sufficient.

Deterministic DES is a typical method employed in non-deterministic or stochastic models, in other words, containing one or more random variables. Though Monte Carlo method can be applied in continuous simulation, for every single run of simulation, most SD simulation is deterministic.

In addition, Mak investigated the possibility of combining DES and SD simulation modelling, and developed a set of rules for converting an ACD into a SD representation [Mak92].

In the real world, "*few systems are wholly discrete or continuous, but since one type of change predominates for most systems, it will usually be possible to classify a system as being either discrete or continuous*" [LK00]. The selection of *continuous* or *discrete* modelling approach is determined by the nature of problem and the purpose of simulation study. Recently, hybrid simulation modelling has become an active attempt to combine the advantages of both (Section 3.5.3).

3.4.3 Quantitative vs. Qualitative Simulation

System dynamics and discrete-event simulation are both quantitative modelling and simulation approaches. Here, qualitative simulation (QSIM) acts as the counter-part of quantitative continuous simulation, e.g. system dynamics.

Qualitative simulation modelling reflects the systems in the real world at an abstract level. Fewer assumptions are required than for purely quantitative approaches. The outputs generated by qualitative simulation are all the possible behaviours of the system, whose states are described by qualitative landmarks, instead of numeric values.

Some initial ideas regarding the application of qualitative modelling to software engineering were discussed by Suarez *et al.* [SAGO02]. However, their attempt was limited in qualitative assignment of model variables, instead of model construction. Neither conceptual nor executable qualitative model was reported in their paper. Almost simultaneously, the first example of QSIM use was developed by Ramil and Smith [RS02]. They investigated the software evolution processes by simulating the qualitative version of previous discrete-time models. Compared with Suarez *et al.*'s work, Ramil and Smith constructed a series of executable QSIM models of evolution processes. Nevertheless, these models were simply converted from a set of mathematical analytic equations, rather than dynamic structural (causal) models. Thus, their work did not completely present the capability and characteristics of qualitative modelling and simulation.

As an extension of qualitative simulation, *semi-quantitative simulation* (SQSIM) focuses on the use of bounding intervals to represent partial quantitative knowledge. This paradigm provides a seamless transition between purely qualitative and quantitative approaches (cf. Chapter 8).

Part II (Chapter 4 and 5) introduce the basic mechanisms and notations of qualitative and semi-quantitative simulation respectively with more detail and a simple example as the foundation of my research.

3.4.4 Emerging Simulation Paradigms

Besides QSIM/SQSIM, agent-based simulation (ABS) and role-playing game (RPG) are dominant in the emerging simulation paradigms in SPSM during the last 10 years (Figure 3.3).

Agent-Based Simulation

An agent-based simulation (ABS) model is regarded as a multi-agent system (MAS), which is a system composed of multiple interacting intelligent agents. Multi-agent systems can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to solve.

Here, *agents* are autonomous computer programs, capable of independent action in environments that are typically dynamic and unpredictable. These agents in a MAS have several important characteristics [Woo02]: *1*) *autonomy*: the agents are at least partially autonomous, and able to interact with each other; *2*) *localisation*: no agent has a full global view of the system, or they have only

local knowledge about their surroundings; *3*) *decentralisation*: there is no one controlling agent (or the system is effectively reduced to a monolithic system); and *4*) *adaptability*: agents are responsible for maintaining their own state.

In an ABS, the overall behaviour of the system is an emergent property of the individual, independent interactions of the agents. It seems a promising approach to many problems involving simulation of complex systems of interacting entities [DHK+07].

In SPSM, software developers are normally represented by agents in an ABS model. They walk randomly around the software processes, executing the tasks and changing the code as they go. Till now, this modelling approach exhibits its capability of simulating long-term, large scale, and interaction-enriched software process, such as open source development [SCR06], agile processes [YP06] and software evolution [SCR06, SC06].

Role-Playing Games

A role-playing game (RPG) is different from the simulation models developed by using the other modelling approaches. The '*story*' or '*scenarios*' have to be predefined prior to simulation. However, the results of simulation highly depend on the interactions, the player's decisions, on the fly.

Often RPG simulator requires the support of other simulation paradigm(s), such as SD. However, it enriches the model's graphic user interface (GUI) and allows a conventional simulator to support game-like interactions between (human) player and model. RPG is good at supporting *cognitive* level purposes of SPSM, particularly *training and learning*.

Some typical examples of RPG include Navarro and Hoek's SimSE game [NH05] and Barros *et al.*'s Incredible Manager [BDVW06], which were both based on SD modelling mechanism.

3.4.5 Simulation Tools

With respect to the simulation models/simulators presented in Category A studies, 13 tools were explicitly identified. Their application numbers are shown in Table 3.9. Because some authors did not explicitly mention the simulation tools in their papers, the total application number (38) is less than the number of Category A studies (56).

Due to a large number of SD models developed and published, Vensim™ (from Ventana Systems, Inc.) is the most popular tool for continuous simulation. The review recognised 6 tools that offer the capability of building both continuous and discrete simulation model. Four of them (SmallTalk, DEVSim++, DSOL, and DEVSJava) are implemented as simulation formalism frameworks using object-oriented languages as the modelling languages, which provide more

Table 3.9: Summary of simulation tools

Tool	SD	DES	SBS	QSIM	KBS	ABS	DTS	Number of studies
Vensim	●							12
Extend	●	●						11
iThink	●							3
QSIM			❍	●				2
NetLogo						●		1
RePast						●		1
DSOL	❍	●						1
SmallTalk	●	●						1
DEVSim++	❍	●					●	1
DEVSJava	❍	●						1
QNAP2	●	●						1
PML					●			1
SESAM					●			1

●: fully support ❍: partially (but not inherently) support

flexibility in model development. Because of the excellence of the graphic modelling workbench, Extend™ (from ImagineThat, Inc.) has been the first choice by the discrete-event and hybrid modelers in SPSM.

Note that no tools listed in Table 3.9 fully support SBS. It is because all three models published (Table 3.10) did not explicitly mention the tools they used.

3.5 Trends & Directions

Trend means "*the general movement over time of a statistically detectable change*" (Merriam-Webster dictionary). This section first attempts to detect the important '*movement*' or '*change*' derived from the review results over the decade.

3.5.1 New Paradigms

In 1998, there were only three simulation paradigms employed by the Category A studies (models) published in the first ProSim workshop (see Table 3.10). They were system dynamics (SD), discrete-event simulation (DES), and knowledge-base simulation (KBS). As the seminal paper in ProSim community, KMR discussed four types of simulation in [KMR99] then. However, the systematic review found ten simulation paradigms from the last 10 years' publications.

Trend 1 System dynamics and discrete-event simulation form the main stream of SPSM paradigms.

Table 3.10: Paradigms applied in simulation models over years

	1998	'99	2000	'02	'03	'04	'05	'06	'07	**Total**
SD	4	5	7	1	4	3	5	2	1	**32**
DES	1	2	3	1	2	2	4	1	2	**18**
SBS				1	1				1	**3**
KBS	1		1	1						**3**
QSIM				1				1		**2**
RPG						1	1			**2**
ABS							1	2		**3**
DTS						1				**1**
Stigmergy							1			**1**
Hybrid	*1*	*1*	*2*		*1*	*1*	*3*	*1*		***10***

Table 3.10 shows the number of Category A studies per paradigm and per year. Note that some studies employed more than one paradigm in their simulation research. The bottom line indicates the number of hybrid models each year, which integrate two or more techniques above it. Any approach or their combination at the top two rows (SD and DES) had one or more applications in building a majority of simulation models each year.

Trend 2 New simulation paradigms continue to be introduced into SPSM research.

Table 3.11 presents a more comprehensive view of all primary studies, including Category A, B, C, and D studies. Unlike Table 3.10, the study numbers are replaced with the solid dots, which make it clear which paradigms were applied each year.

The four rows at the top of Table 3.11 are the paradigms discussed in KMR's paper in 1998, which are separated from others by a solid line. It helps the new emerging paradigms to be easily observed. The new simulation approaches were introduced to SPSM research since 2000, and their application became dominant at 2004. As the bottom line of Table 3.11 concludes, the annual number of active simulation approaches in ProSim community is stable between 4 and 5.

Trend 3 Most of newly introduced simulation paradigms emphasise the purely quantitative software process research.

According to Table 3.10 and Table 3.11, there are no studies introducing and employing the modelling paradigm supporting qualitative or semi-quantitative simulation in recent years, except the studies using QSIM/SQSIM (including my work published in ProSim/ICSP relevant to this thesis).

Table 3.11: Paradigms applied in primary studies over years

	1998	'99	2000	'02	'03	'04	'05	'06	'07
SD	●	●	●	●	●	●	●	●	●
DES	●	●	●	●	●	●	●	●	●
SBS	●		●	●					●
KBS	●		●	●					
QSIM				●		●		●	●
RPG					●	●	●		
ABS							●	●	
DTS						●			
Stigmergy							●		
Emergent			●						
Number	**4**	**2**	**5**	**5**	**4**	**5**	**5**	**4**	**4**

3.5.2 Finer Granularity

As two complementary types of software process, *macro-process* and *micro-process* research (Section 2.1.2), identified by Osterweil [Ost05], are applicable to SPSM research as well. Based on the nature of each modelling paradigm and their corresponding level, they can be mapped to two of the four categories of software process models discussed in Section 2.3, i.e. dynamic *explanatory structural model* (ESM), and dynamic *descriptive enactment model* (DEM). However, the systematic review reflects that there are three different granularity levels that process simulation model can focus on: namely *system* level, *process* level, and *entity* level. They are supported by different simulation paradigms.

At ***system*** level, a software process is modelled as an overall system. The behaviour of the process modelled is described by a set of external parameters that continuously vary over time. This granularity corresponds to the *macro-process* research. The conventional *continuous* simulation supports the modelling on this level.

Software processes are modelled and observed with more details on ***process*** level, where the executed tasks with the resources required and their sequential relationships, rather than causal relationships alone on *system* level, can be tracked during simulation. The modelling on this level is supported by traditional *discrete (event)* simulation.

Trend 4 *Continuous* modelling gradually lost its dominant position in SPSM research in comparison with *discrete* approaches during the decade.

When revisiting the top two rows of Table 3.10, both SD and DES supported simulation studies of software process each year. Although the total number of

SD models is larger than any others' (including DES), it is not difficult to observe a decreasing trend of SD application in the time frame, by absolute number or percentage. The studies using SD dominated in the early years (prior to 2000). Recently, the number of published studies using SD has decreased and approached the number of studies using DES. It implies that simulation research has become more interested in *micro-process* modelling.

Process simulation models focusing on *process* level are based on the conventional *life-cycle* or sequential process modelling. However, software development, like other sectors of society, requires synergistic collaboration among many diverse contributors, such as the careful coordination among designers, programmers, testers, managers, and so on [Ost07]. For instance, agile process models simply offer a framework and set of practices, rather than describe what to do next. The real process executed often depends on the practitioners' *on-the-spot* adjustments [SB01]. Moreover, as a human activity, software development's performance is influenced to a large extent by team coordination, which means "*the articulation of the individual actions accomplished by each of the agents in such as way that the whole ends up being a coherent and high-performance operation*" [Woo02]. It is difficult to simulate the team effects by *process* level modelling through aggregating individuals. This becomes an issue when modelling a process with many participants, such as an open source developer community. For example, as a typical modelling paradigm on *process* level, DES is able to model entities individually. However, it does not allow entities to be autonomous agents with (individual) emergent behaviours.

Accordingly, the modelling and simulation on this ***entity*** level require characteristics quite different from the technology on *process* level. Normally, the simulation on *entity* level consumes more resource to track all entities and their relationships individually than the approaches using aggregation alone. Thanks to the advance in computer and electronic engineering, this approach has recently become affordable.

Table 3.12 indicates each simulation paradigm found from the review with their inherent supporting research granularity level(s). Here, RPG is a special case, which usually need to be combined with other approach(es) to construct a simulator. Thus, its supporting granularity level often depends on its companion.

Trend 5 Most of newly introduced simulation approaches enhanced the research capability on *micro-process* level.

All newly introduced paradigms are listed at the right side of the vertical line (between KBS and QSIM) in Table 3.12 (compared with the four paradigms discussed by KMR at leftmost). If we extend the scope of *micro-process* by covering *entity* level, then most of the new paradigms are capable of the *process* or *entity* level research, which correspond to the refined *micro-processes*.

Table 3.12: Simulation paradigms in support of different granularity research

	SD	DES	SBS	KBS	QSIM	RPG	ABS	DTS	Stig	Emrg
System	●				●	❍				
Process		●	●	●		❍		●		
Entity						❍	●		●	●

● - inherently support ❍ - condition applied

Table 3.13: Granularity level of simulation studies over years

	1998	'99	2000	'02	'03	'04	'05	'06	'07	**Total**
System	4	5	7	2	4	3	5	3	1	**34**
Process	2	2	4	3	3	3	4	1	3	**25**
Entity							2	2		**4**

Trend 6 In recent years, *micro-processes* have been attracting more simulation research.

In terms of the mapping in Table 3.12, Table 3.13 records the number of published studies on each granularity level over years. Some hybrid simulation models are counted into more than one level. When applying the extended concept of *micro-process*, we found since 2004 the number of *micro-process* simulation models has been no less than *macro-process* models. It is also an enhanced statement of Trend 4.

3.5.3 Hybrid Modelling

Similar to the interaction between *macro-process* and *micro-process* (Chapter 2), process modelling research on the three granularity levels are not mutually exclusive, and sometimes combined together.

Hybrid modelling employs more than one simulation paradigms in developing a process simulation model. The systematic review concludes that hybrid process simulation models have attracted interest as a possibility to avoid the limitations of applying single modelling method, and more realistically capture complex real world software processes.

Trend 7 System dynamics and discrete-event simulation are the most common combination for constructing hybrid simulation models.

The bottom line of Table 3.10 shows the number of hybrid simulation models published per year. All these ten hybrid models were at least based on the com-

bination of SD and DES, or even more.

Generally speaking, there are two broad approaches for developing a hybrid process model: ***vertical integration***, which primarily implements discrete modelling at the lower *process* level, then continuously calculates the process factors and incorporates the feedback loops at *system* level; and ***horizontal integration***, in which the sub-processes or phases within a large scale and/or complex software process may be modelled using different approaches respectively and sequentially, and the data flow has to be converted at the interface between them for process transition.

Trend 8 Hybrid simulation modelling emphasised on *vertical integration*.

All the studies published in ProSim series conferences have tried to construct the hybrid models using *vertical integration* of continuous (SD) and discrete (DES) approaches, such as the recent work published in [CBK06] and [SWR07]. On the other side, there have been no attempts on *horizontal integration* until the recent work reported in [ZJZ08].

3.5.4 Possible Directions

According to Merriam-Webster dictionary, there are two meanings of '*direction*' relevant to what is applied in this subsection: "*the line or course on which something is moving or is aimed to move or along which something is pointing or facing*", as well as "*assistance in pointing out the proper route*". It is impossible to point out the *exact 'line or course'* that SPSM is moving towards in the future. Instead, this subsection here tries to point out what SPSM faces, and present the consideration threads of the '*proper routes*' raised from the abovementioned *facts* (answers to "*why? what? and how?*") and *trends* derived from the review.

Direction 1 More recent modern software development processes need to be further investigated in SPSM research.

Some examples in this direction include agile processes, open-source development, and global development. These modern software processes are quite distinct from the traditional processes discussed by KMR.

Direction 2 More new simulation paradigms need to be experimented and introduced into SPSM community.

The recent changes of software processes require the simulation modelling capable of coping with higher complexity, scalability, uncertainty, and agility in the process, especially for modelling and simulating lower granularity level processes, e.g. *process* and *entity* levels.

Direction 3 More attempts are needed to effectively tackle the uncertainty of software process in practice.

The lack of complete knowledge or data of software processes is one of the most common problems obstructing the effective use and adoption of SPSM in practice. However, most purely quantitative simulation modelling approaches rely on the traditional probability based methods and require very specific and precise assumptions. On the other hand, few studies focus on the intuitive and novel approaches for modelling the uncertainty involved (Trend 3).

Direction 4 Hybrid simulation models should address more than SD and DES in vertical integration.

The interest in micro-process and the increasing use of different simulation paradigms for micro-processes suggest the need for different hybrid models to integrate *macro-* and *micro-processes* including entity level models, to cater for new paradigms capable of modelling modern software processes. The integration is not limited by the combination of SD and DES, continuous and discrete, or *system* and *process* levels.

Direction 5 Process simulation models should become more reusable, which makes them easier to build.

Simulation modelling is often time-consuming and requires expertise and experience, especially building the model from scratch. However, the review found that model reuse was often omitted by many SPSM studies. Most Category A studies developed one-off models for simulating the specific software processes.

3.6 Design of Research

The systematic review produces rich evidence to support the design of this research. In order to effectively answer the research questions raised in Chapter 1, particularly for the *feasibility* and *adaptability* of qualitative and semi-quantitative approaches in software process research, the design of research needs to answer "*why and what to model and simulate?*" by considering the dominant *facts* derived from this review.

3.6.1 Purposes, Domains, Scopes & Outputs

As the first consideration, the research of this thesis should be able to test the modelling and simulation for supporting the three levels of motivations (*why?*) identified in the systematic review (cf. Section 3.2).

Table 3.14: Software processes and supporting purpose levels

	staffing process	*incremental process*	*evolution process*
Cognitive level	●		
Tactical level		●	
Strategic level			●

According to Section 3.3, the most common characteristics (*what?*) of SPSM research are extracted. Here, the top 4 (most investigated) items from *problem domains* (cf. Table 3.7), *model scopes* (cf. Table 3.7), and *output variables* (cf. Table 3.8) are chosen as the options for research design:

- **Purposes**: *cognitive level*, *tactical level*, and *strategic level*.
- **Domains**: *generic development*, *software evolution*, *software process improvement*, and *incremental development*.
- **Scopes**: *project*, *single phase*, *evolution*, and *multi-phase*.
- **Outputs**: *time*, *effort*, *quality*, and *size*

These *common* features are dominant in the number of studies from the others. Their combinations were used in development of the most process simulation models ever, and are the top considerations when constructing typical software process models for simulation. The successful development of the process models using the proposed modelling approaches and investigation of these aspects of the modelled processes can be used as the rigorous and convictive evidence for answering the research questions.

3.6.2 Selection of Software Processes

The examples used to demonstrate qualitative and semi-quantitative modelling in this thesis (Part III) were selected to span various domains and levels of granularity. The results of the SLR confirm that the chosen examples i.e. *software staffing process* (Chapter 6), *incremental development process* (Chapter 7), and *software evolution process* (Chapter 8) provide good coverage of different *purpose levels*, *domains*, *scopes*, and *output variables*.

The supporting research purpose levels of these nominated software processes are indicated in Table 3.14. Table 3.15 maps these software processes into the cells in the matrix of problem domain and model scope. Note that *software process improvement* is not included in the matrix, but the discussion on this topic (in

Table 3.15: Domain and scope of the software processes for research

	single phase	multi-phase	project	evolution
generic development			*staffing process*	
software evolution				*evolution process*
incremental development	*incremental process*	*incremental process*		

Table 3.16: Software processes and corresponding output variables

	staffing process	*incremental process*	*evolution process*
Time	●		
Effort	●		
Quality		●	
Size			●

relation of SPSM) is included later in this thesis (Chapter 10). Table 3.16 shows the output variables related to each process for investigation.

Although there are only three software processes selected for modelling and simulation in this thesis, they cover the most investigated aspects of software process for simulation study. Therefore, the successful modelling and simulating these processes (using the proposed qualitative and semi-quantitative approaches) with the predefined focuses can provide strong evidence for answering the research questions of this thesis, i.e. *feasibility* and *adaptability*. Furthermore, the findings from the studies may also answer the research question of *uniqueness*.

3.7 Summary

This chapter presents the state-of-the-art of simulation modelling research in software process domain. The first systematic literature review conducted in SPSM intends to provide insights about the evolution of SPSM research from 1998 to 2007. This chapter presents the preliminary results of the current stage of the review that is exclusively focusing on a core set of publication sources. More than 200 relevant publications were analysed in order to find answers to the research questions, including the purposes and paradigms of SPSM, the studies'

domains, scopes and outputs, and other predominant research issues.

The results from this systematic review are also used to serve as the introduction to SPSM research (by answering the '*why*', '*what*', and '*how*' questions), and to provide a *solid* state-of-the-art. They further provide the evidence for supporting the research design in this thesis. The important aspects this chapter achieves can be highlighted as follows:

1. Categories for classifying software process simulation models are proposed to better capture the diversity of published models.
2. Purposes for driving SPSM research as originally suggested by the seminal publication in ProSim'98 have been restructured and updated.
3. *Facts* of SPSM, including paradigms, domains, scopes, outputs and tools, over the ten years have been summarised and reported in a '*snap-shot*' style.
4. Important and dominant simulation modelling paradigms relevant to the review and this thesis are introduced and compared.
5. Underlying *trends* over the time frame are analysed and discovered.
6. New insights about SPSM research on modelling granularity and model integration are first presented.
7. Potential research *directions* for SPSM community are suggested.
8. Typical characteristics of SPSM research are derived from the *facts* of the review, and provide evidence in support of the research design in this thesis.

In addition, the outcomes produced from this chapter can help both insiders' and outsiders' observation and understanding of SPSM research. The study categorisation can be used for future systematic review and meta-analysis with more specific focuses. Moreover, the decoded *trends* and suggested *directions* may help practitioners and modelers in SPSM arena (not limited to ProSim community) realistically and effectively capturing complex real-world software processes.

Some limitations still exist in the current study and need further improvement: 1) the study categorisation was mainly determined by the principal reviewer's judgement based on the author's knowledge in SPSM , which needs other researchers' further examination; 2) the missing papers may influence the integrity of this review and need to be included in the near future; 3) the impact of study quality needs to be considered in data analysis, especially for the inclusion of low quality studies.

In next part, the technical foundations of this thesis, qualitative & semi-quantitative modelling and simulation, are introduced and described in greater detail.

Part II

FOUNDATION

Chapter 4

Qualitative Modelling & Simulation

Qualitative simulation predicts the set of possible behaviours consistent with a qualitative differential equation (QDE) model of the world. Its value comes from the ability to express natural types of incomplete knowledge of the world, and the ability to derive a provably complete set of possible behaviours in spite of the incompleteness of the model [Kui01].

Qualitative modelling and simulation are key inference methods for problem-solvers for major tasks of model-based reasoning: monitoring, diagnosis, design, planning, and explanation. Although the qualitative simulation (reasoning) field has addressed diverse problem areas and developed a variety of theories and systems, there are a number of prominent features that are typical for many of the approaches. Some of the most important ones include *ontologies, causality, compositional modelling, inference of behaviour*, and *qualitativeness* [BS03]. Among these works, Kuipers *et al.*'s work [Kui94] is a comprehensive treatment of qualitative simulation, and a seamless framework to integrate qualitative and semi-quantitative simulation. Therefore, in this thesis, it is selected as the major fundamental modelling and simulation theories used in software process research.

This chapter aims to provide an initial description of the fundamental mechanism of qualitative modelling and simulation, and its model representation, which are used in other parts of this thesis. Thus, the detailed algorithm implementation and some advanced topics, such as higher-order derivatives and global dynamical constraints, are omitted here.

4.1 Incomplete Knowledge Representation

A quantity is a real-valued attribute of a physical object. However, human knowledge is finite, so our knowledge of real numbers describing the physical world must be incomplete. There are many ways to represent incomplete knowledge in a description of a quantity and its value.

4.1.1 Quantity

Interval Arithmetic In interval arithmetic, incomplete knowledge of the real value of a quantity x is expressed as a closed interval $[b_l, b_u]$ representing the knowledge that $b_l \leq x \leq b_u$. The theory shows that many functions defined on real numbers have natural extensions to the domain of interval values. Interval arithmetic is used to express semi-quantitative annotations on qualitative behaviour predictions (cf. Chapter 5).

Nominal, Ordinal, Interval, Ratio Statisticians distinguish among different types of knowledge that can be embodied in real-valued data according to the operations that can legitimately be applied to that data.

- Nominal data can only be compared for equality or inequality.
- Ordinal data can be compared for order as well as equality.
- Interval data can be subtracted to determine the difference between values. Conversely, a difference can be added to one value to get another.
- Ratio data can be added, subtracted, multiplied, or divided.

This classification implies that different descriptions and inference methods require different kinds of data. For example, the median can be computed from *ordinal* data, but the arithmetic mean requires *interval* data, and the geometric mean requires *ratio* data.

Qualitative simulation is based primarily on *ordinal* knowledge of real-valued quantities, because human perception and memory seem to be particularly sensitive to ordinal relationships, especially with "*landmark*" values.

Landmark Values Landmark values are the "natural joints" that break a continuous set of values into qualitatively distinct regions. A landmark value is a symbolic name for a particular real number, whose numerical value may or may not be known. It serves as a precise boundary for a qualitative region. The qualitative properties of a value in the set depend primarily on its ordinal relations with the landmarks.

For example, a natural set of qualitative regions for the temperature of water is defined by the following landmarks: $absolute\_zero \ldots freezing \ldots boiling \ldots \infty$. For a model with constant (though possibly unknown) pressure, the qualitatively distinct values for temperature of water are the landmark values and the open intervals between them.

Fuzzy Values '*Fuzzy*' values are qualitative descriptions without precise boundaries. For example, when describing values of a continuous scalar quantity such as the amount of water in a tank, there are no meaningful landmark values representing the boundaries between *low* and *normal*, or between *normal* and *high*. Linguistic terms such as these refer to *fuzzy sets* of numbers. More discussion between fuzzy sets and qualitative/semi-quantitative modelling can be found in Chapter 11.

4.1.2 Continuous Change

Discrete State Graphs The changing world is often described in discrete terms, such as events, actions, states, and state-transitions. A variety of representations have been developed for describing action and change in discrete terms, including finite-state causal graphs, situation calculus, and temporal logic. Discrete state graphs are useful at a level of abstraction where the continuity of change and the continuous dynamics of behaviour are not critical. Because they do not exploit the properties of continuity, they have difficulty reasoning about variables with values moving towards limits, dynamic phenomena involving a balance of forces, or the effect of perturbations on feedback systems.

Qualitative modelling and simulation provides a level of description between discrete state graphs and the continuous world: continuous change is described symbolically, but in a way that obeys the constraints of continuity.

Differential Equations The physicist uses the language of differential equations for describing a *system* and drawing inferences about it. A differential equation represents the structure of the system by selecting certain continuous variables that characterise the state of the system, and certain mathematical constraints on the values those variables can take on. One important use of a differential equation description is to predict the behaviour of the system over time: a set of continuous functions of time that describe the way the variables evolve over time starting from a given initial state. Its strength comes from the *expressive power* to state models that capture the dynamic character of the world, and the *inferential power* to derive predictions from those models.

The qualitative model representation (described in Section 4.3) is closely related to differential equation as a language for describing aspects of the world.

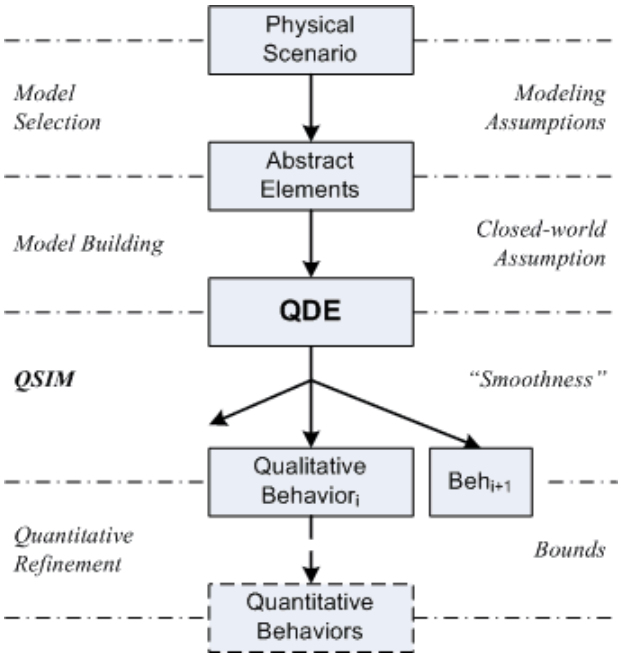


Figure 4.1: Qualitative modelling and simulation framework

4.2 Modelling & Simulation Framework

A model is a finite (small) description of an infinitely complex reality, constructed for the purpose of answering particular questions. The process of reasoning with models breaks down into two major subproblems [Kui94]:

1. Out of all the possible ways of describing the world, select an appropriate model or combination of models to answer a particular question.

2. Given a model, simulate it or otherwise analyse it to make explicit some facts about the world that are implicit in that description of the world (e.g. its predicted behaviours).

Qualitative modelling and simulation address both of these problems. The *model-building* problem is the more open-ended of them, while given the model's representation, the *model-simulation* problem is more tightly constrained (a more technical progress). The whole procedure breaks into four important steps (as shown in Figure 4.1).

1. Model building requires modelling assumptions that specify which aspects of the world are negligible and which should be included in the model, and the closed-world assumption to transform a collection of fragments

into a model. These assumptions are defined qualitatively to reflect the incomplete knowledge of continuous quantities.

2. The QSIM graphic representation (qualitative abstract structure) for QDEs (see Section 4.3.2) is used to formally define the qualitative assumptions. It is then converted into pseudo code description of QSIM algorithm for qualitative simulation.

3. Given QDE models and initial state, the qualitative reasoning algorithm (QSIM applied in this thesis) simulate with the steady-state assumption and across region transitions. Simulation generates all behaviours consistent with the QDE models that are covered by the qualitative prediction.

4. The quantitative and semi-quantitative information can be unified with the behaviours predicted by qualitative simulation to produce stronger, more precise predictions without losing guarantee that all behaviours are covered (cf. Chapter 5).

The qualitative modelling and simulation framework in Figure 4.1 provides two benefits over traditional engineering approaches to modelling and simulation. First, the representations are designed to express states of incomplete knowledge that are common in human knowledge but are hard to express using traditional methods. Second, the inference methods are designed to be essentially deductive, so that assumptions are explicit and guarantees are provided that every step is sound.

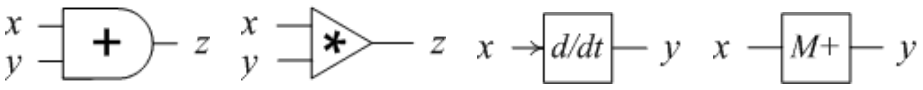
4.3 Qualitative Model Representation

4.3.1 Abstract Structure Diagram

An abstract structure diagram (ASD) reflects the assumptions for qualitative modelling within a graphic format. It explicitly visualises the structure of qualitative model, and bridges the qualitative assumptions (described in natural language) abstracted from physical scenario and the formal QDE model for qualitative simulation. The basic graphic notations used for building an ASD are summarised in Figure 4.2. Each of them exclusively corresponds to one type of qualitative constraint in QDE. In the plot (d), y is monotonically increasing function of x.

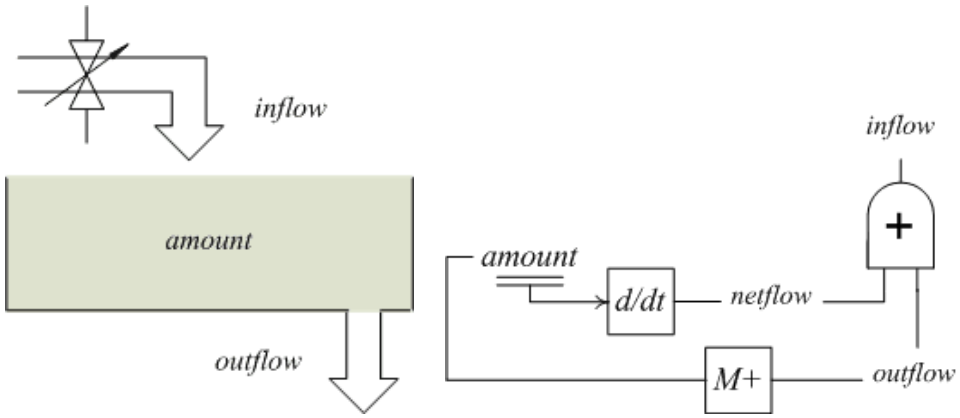
Example: Bathtub

To explore the basic concepts and results of qualitative and semi-quantitative modelling and simulation, a simple example exhibiting first-order (differential) dynamic behaviour is needed and used continuously in this chapter and next



(a) $x + y = z$ (b) $x * y = z$ (c) $\frac{dx}{dt} = y$ (d) $y = M^{+}(x)$

Figure 4.2: Typical notations of abstract structure diagram



(a) A simple bathtub (b) Abstract structure

Figure 4.3: Qualitative bathtub example model

chapter. The structure of a bathtub example is simple but clear, and its behaviours are easily deduced. Nonetheless, it is enough to motivate the basic features of qualitative modelling and simulation.

A simplified view of a bathtub (Figure 4.3-a) consists of a tank with two flows. The *inflow* and *outflow* represent the flows of water into and out of the bathtub. The bathtub's behaviour is determined by the amount of water in it, and the difference between *inflow* and *outflow* (i.e. *netflow*). The *outflow* is determined by the pressure (amount) of water. The corresponding abstract structure diagram is shown in Figure 4.3-b.

4.3.2 Qualitative Differential Equation

A real world system is normally modelled as a set of ordinary differential equations (ODEs). Qualitative modelling makes explicit and precise the abstraction relationship between the qualitative partial knowledge representations and the theory of differential equations. Accordingly, qualitative models can be called *qualitative differential equations* or QDEs. If a mechanism can be described by an ODE meeting certain restrictions, there is a corresponding but weaker QDE describing the same mechanism. '*Weaker*' here means that any behaviour that satisfies the ODE must satisfy the QDE, but not necessarily vice versa.

There are two fundamental improvements in expressive power as abstracted from ordinary to qualitative differential equations.

1. A functional relationship between two variables may be incompletely known, specified only as being in the class of monotonically increasing (or decreasing) functions.

2. The real number line in which variables take their values is described in terms of a finite set of qualitatively significant '*landmark values*' and the intervals between them.

A QDE is a tuple of four elements,

$$QDE = \langle V, Q, C, T \rangle \tag{4.1}$$

where each of them will be defined below.

- V is a set of *variables*, each of which is a '*reasonable*' function related to time.
- Q is a set of *quantity spaces*, one for each variable in V.
- C is a set of *constraints* applying to the variables in V, each variable must appear in some constraint(s).
- T is a set of *transitions*, which are rules defining the boundary of the domain of applicability of the QDE.

In qualitative modelling, QDE is represented in Common Lisp format, which enables the execution by QSIM reasoning engine.

The remaining of this section introduces the basic concepts of qualitative modelling and simulation: *quantity space*, *qualitative constraint*, and *region transition*, which are used as the elements in building a QDE.

4.3.3 Quantity Space & Qualitative Value

A quantity space defines a low-resolution qualitative description for incompletely known quantities. It only consists of a few qualitatively important *landmark values*. Formally speaking, a *quantity space* is a finite, totally ordered set of symbols, the *landmark values*, $l_1 < l_2 < \cdots < l_k$.

Each landmark (l_k) is a symbolic name, representing a particular value in $\Re$ whose actual quantitative value is often unknown. A quantity space normally contains the typical landmarks, i.e. $-\infty$, 0, 1, and $+\infty$. Time is also represented by the qualitative variable, say *time*, that has the quantity space $t_0 < t_1 < \cdots < t_n < \infty$.

The quantity space representing the range for a reasonable function $f(t)$ must include a landmark value corresponding to each *critical value* of $f(t)$, for example the value of $f(t)$ when $f'(t) = 0$. New landmarks may need to be introduced

```
(quantity-spaces
  (amount   (     0 FULL inf))
  (outflow  (     0      inf))
  (inflow   (     0 INFL inf))
  (netflow  (minf 0      inf)))
```

Figure 4.4: Quantity spaces of bathtub example

during simulation if critical points are discovered that do not have corresponding landmarks in the predefined quantity space of a QDE.

A qualitative variable v, and its quantity space $l_1 < l_2 < \cdots < l_k$, define a symbolic language, a finite set of meaningful distinctions, for describing the values of a reasonable function $f(t)$. At any time, the qualitative value of $f(t)$ is described in terms of its ordinal relationships with landmarks in its quantity space, and its direction of change.

The *qualitative value* of $f(t)$ with respect to the quantity space $l_1 < l_2 < \cdots < l_k$, is the pair $\langle qmag, qdir \rangle$, defined as

$$qmag = \begin{cases} l_j & \text{if } f(t) = l_j, \text{ a landmark value,} \\ (l_j, l_{j+1}) & \text{if } f(t) \in (l_j, l_{j+1}). \end{cases} \tag{4.2}$$

$$qdir = \begin{cases} inc & \text{if } f'(t) > 0, \\ std & \text{if } f'(t) = 0, \\ dec & \text{if } f'(t) < 0. \end{cases} \tag{4.3}$$

where '*qmag*' and '*qdir*' stand for *qualitative magnitude* and *qualitative direction* respectively.

Figure 4.4 shows the quantity spaces of the bathtub example, where `minf` and `inf` present $-\infty$ and $+\infty$ separately. `FULL` and `INFL` are the preset qualitative landmarks when the actual numeric values are unavailable in qualitative modelling.

4.3.4 Qualitative Constraints

The state of a system at a time t is described in terms of the values of some set of variables $x, y, \ldots$, each of which is a reasonable function of time. The relationships among these variables are expressed by qualitative constraints. The QDEs that make up a model include combinations of these constraints including auxiliary variables when needed.

Table 4.1 lists the typical constraints used for building QDEs. Most of the constraints (`add, mult, minus, d/dt`, and `constant`) are straightforward equiva-

Table 4.1: Typical qualitative constraints

Quantitative relations	Qualitative constraints
$x(t) + y(t) = z(t)$	`(add x y z)`
$x(t) * y(t) = z(t)$	`(mult x y z)`
$y(t) = -x(t)$	`(minus x y)`
$\frac{d}{dt}x(t) = y(t)$	`(d/dt x y)`
$\frac{d}{dt}x(t) = 0$	`(constant x)`
$y(t) = f(x(t)), \dot{f} > 0$	`(M+ x y)`
$y(t) = f(x(t)), \dot{f} < 0$	`(M- x y)`

add	[+]	[0]	[−]	[?]
[+]	[+]	[+]	[?]	[?]
[0]	[+]	[0]	[−]	[?]
[−]	[?]	[−]	[−]	[?]
[?]	[?]	[?]	[?]	[?]

mult	[+]	[0]	[−]	[?]
[+]	[+]	[0]	[−]	[?]
[0]	[0]	[0]	[0]	[0]
[−]	[−]	[0]	[+]	[?]
[?]	[?]	[0]	[?]	[?]

Figure 4.5: Arithmetic operations of qualitative addition and multiplication

lents to their quantitative counterparts. The functional constraints M^+, S^+ (and their negative versions M^-, S^-) will be explained later.

Qualitative Addition & Multiplication

Addition over the real numbers is a *function* that takes two values and returns a third. This is not possible, however, to define addition over the signs in the same way, since the sum of the qualitative values [+] and [−] is ambiguous, so addition function cannot be single-valued. To represent addition as a function, it must be defined over the *extended* signs. Here the sign [?] is used to represent the ambiguity among [+]/[0]/[−].

Figure 4.5 gives the definitions of the qualitative addition and multiplication functions, for which the `add` and `mult` constraints hold over the signs.

Monotonic Functions

The qualitative monotonic constraint

```
((M+ x y) (x0 y0) ... (xi yi) ...)
```

represents the assertion that $y = f(x)$ for some $f \in M^+$, and that $f(x_i) = y_i$ for each corresponding value pair (x_i, y_i). This constraint implies the following conditions, which are easily evaluated on qualitative descriptions of values of x and y.

1. $[\dot{x}] = [\dot{y}]$, unless x or y is at the endpoint of its range. That is, the directions of change of x and y must be the same in the *interior* of the range $[a, b]$ of the monotonic function f, since the definition of $f \in M^+$ requires only that $f' > 0$ on the open interval (a, b).

2. If (x_i, y_i) is a pair of corresponding values, then $[x]_{x_i} = [y]_{y_i}$. That is, the two values must be on the same side of the landmarks in each pair of corresponding values.

Similarly, the other monotonic constraint

```
((M- x y) (x0 y0) ... (xi yi) ...)
```

implies that $[\dot{x}] = -[\dot{y}]$ and $[x]_{x_i} = -[y]_{y_i}$ for every pair of corresponding values (x_i, y_i).

Multivariate Constraints

While the basic qualitative constraints have fix numbers of arguments, there are several useful constraints that have arbitrary numbers of arguments. This section only introduces three often used multivariate constraints.

Multivariate Monotonic Function Constraint

```
(((M s1 ... sn) x1 ... xn y) (a1 ... an a0) ... )
```

This constraint generalises the definition of monotonic function constraint to handle functions of several variables,

$$y = f(x_1, \ldots x_n) \tag{4.4}$$

where $\frac{\partial f}{\partial x_i} = s_i$, and qualitative values for $x_1, \ldots x_n$ refers to y, and a_0 stands for a landmark of y. It provides generality required in a variety of contexts.

Sum Constraint

```
((SUM x1 ... xn y) (a1 ... an a0) ... )
```

This constraint represents the relation $x_1 + \cdots + x_n = y$ with corresponding values $(a_1, \ldots a_n, a_0) \ldots$.

```
(constraints
  ((M+       amount outflow)          (0 0) (inf inf))
  ((add      netflow outflow inflow))
  ((d/dt     amount netflow))
  ((constant inflow)))
```

Figure 4.6: Constraints of bathtub example

Signed-Sum Constraint

`(((SSUM s1 ... sn) x1 ... xn y) (a1 ... an a0) ... )`

This constraint is more specified than the monotonic functions, since the arguments combine strictly additively:

$$\bar{s}_1 x_1 + \cdots + \bar{s}_n x_n = y, \text{where } \bar{s}_i = \begin{cases} +1 & \text{if } s_i = [+] \\ 0 & \text{if } s_i = [0] \\ -1 & \text{if } s_i = [-] \end{cases} \tag{4.5}$$

`SSUM` constraint helps the modeler avoid irrelevant and under-constrained intermediate variables that are required if only the basic constraints are available (more discussion in [Kui94]).

The relations among the elements of example model now can be defined with the above qualitative constraints. Figure 4.6 is the code clip of constraints applied for bathtub example, where `(0 0)` means that $outflow = 0$ iff $amount = 0$ (implies monotonic function must pass through $(0, 0)$, and `(inf inf)` constrains the monotonic function to eliminate the possibility of horizontal or vertical asymptote.

4.3.5 Region Transitions

The transitions associated with a QDE define the limits of the region of applicability of the QDE, and optionally specify a transition to a new QDE if that limit is reached. A *transition* can be defined as a rule of the form

$$(condition \rightarrow transition\_function) \tag{4.6}$$

where

- The *condition* is a pattern of the form $(\langle variable \rangle \langle qmag, qdir \rangle)$, or a boolean combination of such patterns. It becomes true at a state when the values of the specified variables match the corresponding description.

```
(define-QDE Bathtub
  (quantity-spaces
    (amount   (     0 FULL inf))
    (outflow  (     0      inf))
    (inflow   (     0 INFL inf))
    (netflow  (minf 0      inf)))
  (constraints
    ((M+       amount outflow)         (0 0) (inf inf))
    ((add      netflow outflow inflow))
    ((d/dt     amount netflow))
    ((constant inflow)))
  (transitions
    ((amount (FULL inc)) -> t)))
```

Figure 4.7: QDE of bathtub example

- The *transition function* is applied to the current state if the condition becomes true. It returns a new qualitative state, perhaps defined with respect to a new QDE, from which simulation can resume.

The transition function is responsible for establishing the correspondence between the pre- and post-transition states. It can be used to represent known transitions between QDEs or as an escape to a model-defined process. If no transition function is provided, simulation stops along the current behaviour.

Figure 4.7 shows the complete QDE of bathtub example, which includes the `transitions` section. It constrains the simulation to be terminated (denoted by `t`) when `amount` reaches `FULL` and its direction is increasing (`inc`).

4.4 Qualitative Simulation

Qualitative simulation starts with a QDE and a qualitative description of an initial state. Given a qualitative description of a state (called a *qstate*), it predicts the qualitative state descriptions that can possibly be direct successors of the current state description. Repeating this process produces a sequence of qualitative state descriptions, in which the paths starting from the root are the possible qualitative behaviours. The graph of qualitative states is pruned according to criteria derived from the theory of ordinary differential equations, in order to preserve the guarantee that all possible behaviours are predicted.

4.4.1 QSIM: Algorithm & Tool

The availability of algorithms and tools to construct and simulate QDE models is limited. QPE is a reasoning engine implements Qualitative Process Theory

(QPT). QSIM [Kui94] is the implementation of the constraint-based approach. Both of them require programming skills in LISP. After them, easy-to-use learning environments for qualitative modelling and simulation have been developed, notably Betty's Brain [BSB01] and Vmodel [BF04]. Since these packages were implemented for teaching, some essential features of qualitative simulation are missing. Recently Garp3 [BSBL05], implemented in Prolog, has been developed to provide a seamless workbench for building, simulating, and inspecting qualitative models. Here, QSIM was selected as the simulation package to be used in this thesis because it supports both qualitative and semi-quantitative simulation.

The term QSIM is used in two ways in this thesis. First, it is the abbreviation of *qualitative simulation* (QSIM); second, it denotes the reasoning algorithm or reasoner (simulator) developed by Kuipers *et al.* [Kui94] for performing qualitative simulation. To differentiate them in this thesis, QSIM is used for the second meaning.

The QSIM algorithm [QRG, FKRT94] performs qualitative simulation by deriving the immediate successors of each qualitative state, and repeating this process to grow the behaviours. To start simulation, a set of QDEs is passed to QSIM, together with initial values for the known variables. QSIM then completes the initial state by assigning values for the other variables in the model. If there is more than one possible set of assignments, QSIM will simultaneously generate several behaviours. In the bathtub model for instance, the water level may increase, decrease or remain constant, depending on the initial level and inflow rate.

4.4.2 Outputs

Behaviour Tree

Rather than finding a single behaviour for each model by ODE-based simulators, QSIM generates a tree consisting of all possible behaviours, branching when there is more than one possible evolution of the system. A tree of behaviours for the bathtub model is shown in Figure 4.8. The key to the symbols of the '*tree*' is shown in Table 4.2. When QSIM is applied to the bathtub model, with the bathtub initially part full, it finds three initial states, which reflect whether the part-full bathtub is emptying (1), filling (3, 4, 5) or in equilibrium (2).

QSIM then finds the possible consequent behaviour(s) of each state. Five distinct qualitative behaviours for the bathtub are predicted (Figure 4.8). Note that Behaviour 3 finishes at a transition point (⓵), whose condition is specified in `transitions` section in QDE (Figure 4.7).

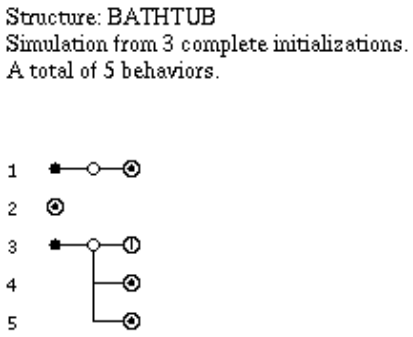


Figure 4.8: Behaviour tree from qualitative simulation for bathtub example

Table 4.2: Symbols used in qualitative behaviour trees

⊙	Final state
⏀	Transition point
•	Intermediate critical time-point
○	Intermediate time-interval

Variable's Behaviour Trend

QSIM allows the behaviour trend of each variable to be shown graphically (cf. Figure 4.9). One possible behaviour is composed by variables' qualitative trends. While each variable's trend depicts its varying direction (increasing, decreasing, or static) with arrows, and marks the corresponding qualitative value (i.e. landmarks) at the important time-points of simulation (including final state, transition points, and critical time-points). The dotted lines connecting the time-points are simply visual aids and have no significance.

For the bathtub example, QSIM finds five distinct qualitative behaviours over time (shown in Figure 4.9): *1*) an emptying bath reaches equilibrium before becoming empty (Figure 4.9-a); *2*) bathtub's water level remains constant (b); *3*) a filling bath can overflow (c); *4*) reach equilibrium before it overflows (d), or *5*) reach equilibrium just as the water reaches the brim (e). Each of them corresponds to one particular branch of the behaviour tree (Figure 4.8). This set of behaviours confirms that the bathtub cannot reach the empty state.

Phase-Space View

Apart from the *behaviour tree* and *variable's behaviour trend*, QSIM can generate a variety of graphic outputs assisting in analysis of the simulated process, such as *phase-space viewer* and *numeric viewer*.

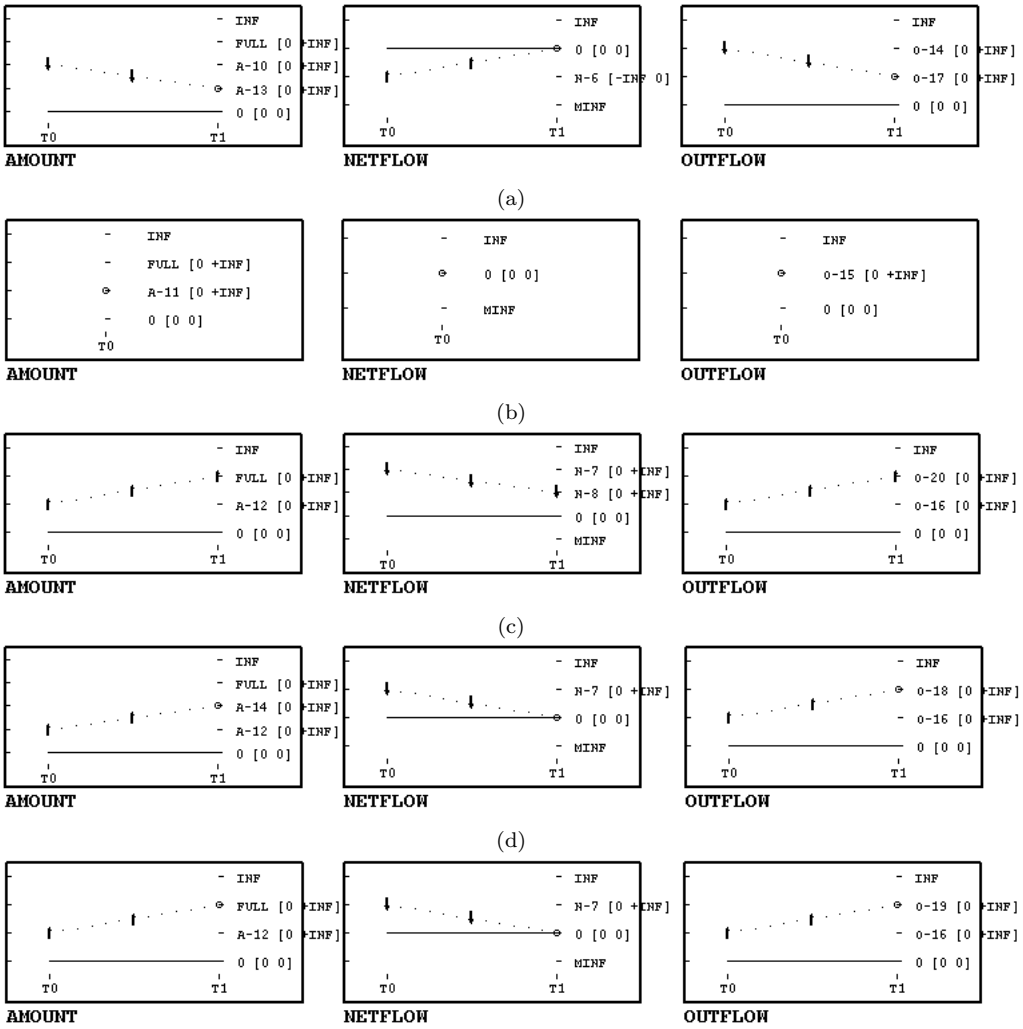


Figure 4.9: Qualitative behaviors of bathtub example

The *phase-space* is a standard mathematical representation for dynamic systems. It is the Cartesian product of a set of independent state variables that fully describe the system; for two variables, this corresponds to a *phase plane*. A point in the phase space represents a state of the system. Thus, the dynamics of the system correspond to a trajectory through the phase space. A geometrical representation of the phase space is called a *phase portrait*. The *phase-space viewer* displays the *phase portrait* of any two variables of a QDE.

Figure 4.10 shows the examination of some of the behaviours using the *phase-space viewer*. In each plot, one axis represents `amount` and the other axis represents `netflow`.

While the *phase-space viewer* is often useful in understanding the relationship between two variables, and is especially useful for analysing oscillatory systems,

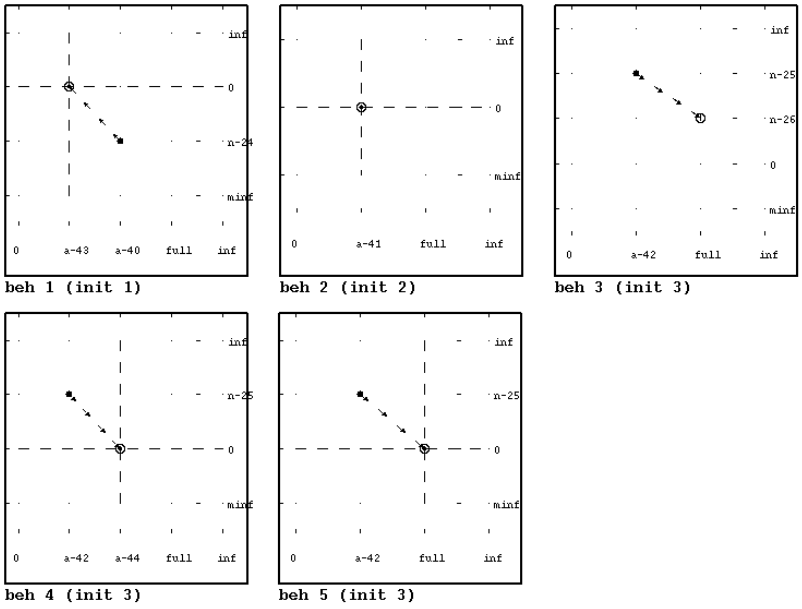


Figure 4.10: Phase view for bathtub example

its main use is in conjunction with the non-intersection constraint. Chapter 6 includes its use in analysis of software staffing process.

4.5 Summary

This chapter introduces qualitative process modelling and simulation, presents the basic concepts of qualitative model presentation and mechanism of simulation with a simple example. This paradigm is used as the fundamental modelling theory and simulation technique for software process research in the following chapters.

In the next chapter, the semi-quantitative simulation is introduced as a powerful extension of qualitative modelling and simulation for bridging traditional quantitative and qualitative approaches.

Chapter 5

Semi-Quantitative Modelling & Simulation

Semi-quantitative simulation (also known as semi-quantitative reasoning) is the task of combining incomplete quantitative and qualitative knowledge. It is an extension of qualitative simulation, involving quantitative constraints.

The previous chapter and this chapter introduce several acronyms relevant to qualitative and semi-quantitative simulation. Table 5.1 summarises these abbreviations and their representing meanings.

5.1 Semi-Quantitative Extension

Q2 (for Qualitative + Quantitative) is the basic semi-quantitative reasoner implemented as an extension to QSIM. It acts as a global filter on behaviour, inferring bounds on value-denoting terms and refuting behaviours when possible. Q2 uses constraint propagation and interval arithmetic to tighten interval bounds on value-denoting terms. Figure 5.1 shows the representation used in QSIM+Q2 for qualitative and quantitative information, and the process that generates each from its predecessors.

QSIM predicts one or more qualitative behaviours from a given QDE and its

Table 5.1: Acronyms in qualitative and semi-quantitative simulation

	Modelling	Simulation (Reasoning)	Algorithm or Tool
Qualitative		QSIM (QR)	QSIM
Semi-quantitative	SemiQ	SQSIM (SQR)	Q2

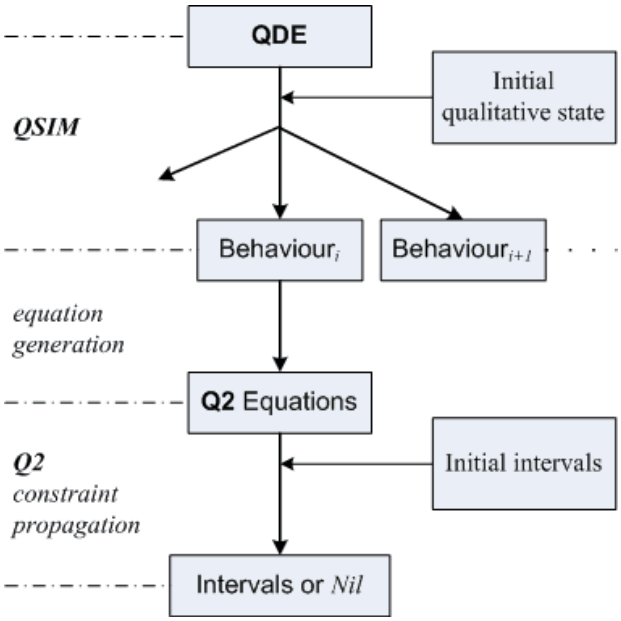


Figure 5.1: Steps in semi-quantitative (qualitative & quantitative) modelling and simulation

initial state. Each behaviour implies a set of algebraic equations relating value- and set-denoting terms to each other. Incomplete quantitative knowledge about values associated with these terms can be propagated across the equations, either refuting the behaviour or producing a stronger semi-quantitative description.

Similar to QSIM, Q2 does conservative inference, ruling out values or behaviours only when they are genuinely inconsistent. This extends QSIM's guaranteed coverage to semi-quantitative simulation.

5.2 Interval Constraints

There are a number of different representations for incomplete knowledge of quantities, including bounding intervals, probability distribution functions, fuzzy set, and order-of-magnitude relations (some of which were briefly addressed in Section 4.1). This chapter focuses on the use of bounding intervals, i.e. value ranges and envelop functions, to represent partial quantitative knowledge in semi-quantitative modelling.

5.2.1 Value Ranges

A landmark value is a symbol representing a unique but unknown real number. The incompletely known value of a landmark value p can be quantified by a closed value range (i.e. interval):

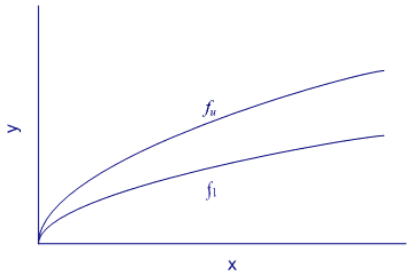


Figure 5.2: Example of envelope functions

$$range(p) = [b_l, b_u] = \{x \in \Re | b_l \leq x \leq b_u\} \tag{5.1}$$

Quantitative knowledge expressed as intervals complements qualitative knowledge expressed as ordinal relations, each description providing information difficult or impossible to express in the other form. The value ranges for the landmarks of bathtub model are defined in `initial-ranges` section in Figure 5.3.

Although value range is simple, this representation offers a number of important benefits. Its simplicity means that descriptions of incomplete knowledge of quantities in the form of bounding intervals are easy to obtain. Furthermore, the consequence of two interval descriptions of the same quantity is easy to compute by intersecting the intervals.

5.2.2 Envelope Functions

Describing an unknown monotonic function $f \in M^+$ is more complex than describing a single value. By applying the same philosophy, however, '*envelope*' functions can be specified to bound the corresponding monotonic functions. In addition to using envelopes to constrain the values a function f can take on, it is possible to assert bounds on its derivatives f' and f''.

Envelopes are numerically computable partial functions f_l and f_u, with numerically computable partial inverses f_l^{-1} and f_u^{-1}, such that

$$\forall x, f_l(x) \leq f(x) \leq f_u(x) \tag{5.2}$$

When $f_l = f_u$, the envelope functions turn to be an `exact` function, which is equivalent to its corresponding ordinary version. Figure 5.2 shows an example envelopes with bounds of function f_l and f_u.

The updated QDE of bathtub model is shown in Figure 5.3 with the envelope function defined in `envelopes` section.

```
(define-QDE bathtub
  (quantity-spaces
    (amount   (     0 FULL inf))
    (outflow  (     0      inf))
    (inflow   (     0 INFL inf))
    (netflow  (minf 0      inf)))
  (constraints
    ((M+       amount outflow)         (0 0) (inf inf))
    ((add      netflow outflow inflow))
    ((d/dt     amount netflow))
    ((constant inflow)))
  (transitions
    ((amount (FULL inc)) -> t))
  (envelopes
    ((M+ amount outflow) (upper ue)
                         (u-inv ui)
                         (lower le)
                         (l-inv li)))
  (initial-ranges
    ((amount FULL) (80 100))
    ((inflow INFL) (4 8))
    ((time t0)     (0 0))))
```

`ue, ui, le, li` are the names of the envelope functions.

Figure 5.3: QDE of semi-quantitative bathtub example model

5.3 Semi-Quantitative Propagation

5.3.1 Q2: QSIM Extension

Q2 combines interval arithmetic and qualitative behaviour refutation to perform the propagation. One well-known limitation of interval-based representation (value ranges and envelope functions) might be the uncertainty explosion that results from repeated operations on interval values. Q2 avoids the uncertainty explosion because there is a fixed set of terms with value ranges, determined by qualitative behaviour, rather than an indefinitely increasing set of terms as in a traditional time-stepped quantitative simulation.

Interval propagation in semi-quantitative simulation can be implemented as Q2 algorithm (cf. [Kui94, FKRT94]). The conclusions derived by Q2 can be guaranteed, because if Q2 derives a contradiction, then the qualitative behaviour is inconsistent with the given QDE, initial qualitative state, and its associated quantitative ranges. Therefore, every real solution to an ODE consistent with this scenario must be included among the predictions from simulation (using

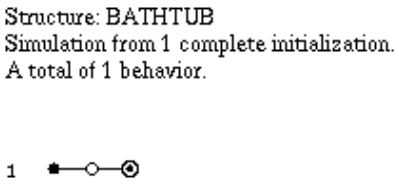


Figure 5.4: Behaviour tree from semi-quantitative simulation for bathtub example

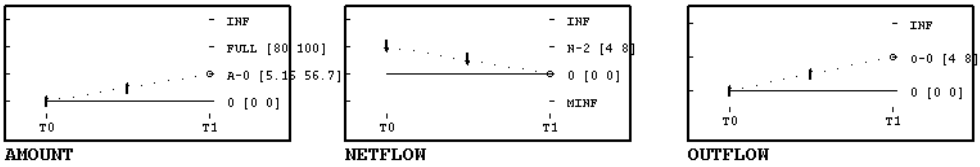


Figure 5.5: Semi-quantitative behaviour of bathtub example

QSIM+Q2).

5.3.2 Outputs

By including semi-quantitative information, some qualitative behaviours, generated by QSIM but inconsistent with quantitative constraints, may be refuted during the interval propagation. Note that the variables' initial values are specified in the initialisation and transaction functions in QSIM, not in QDE (Figure 5.3). Given the initial state with `amount` $= 0$ (an empty bath), Q2 predicts the behaviour tree of bathtub with single branch (Figure 5.4). It indicates the process reaches an equilibrium as its *final state* ($\odot$).

Figure 5.5 shows the variables' trends of this behaviour. Unlike the behaviour trend from qualitative simulation (Figure 4.9), the landmarks in each plot are also associated with a pair of numeric values, which are generated by Q2, to describe the estimated value ranges.

Note that the simulation predicts `amount` will be static at the level of a very coarse range, $[5.15, 56.7]$. It might be caused by two reasons: first, there is a large uncertainty for `inflow`, say $[4, 8]$; second, in this case, Q2 is not so efficient as expected, because no new landmarks are created (at critical time-points) between start and end points of simulation (Figure 5.4), which implies that the intersecting algorithm failed to work. To solve the weakness of Q2, some advanced refinements need to apply.

5.4 Advanced Techniques

One important limitation of Q2 is that the qualitative information inferred remains relatively coarse in some cases, such as when no transition point or intermediate critical time-point is created during simulation (e.g. smooth software evolution process described in Section 8.5.3). The weak quantitative conclusion in such cases is mainly due to the large grain-size of the qualitative behaviour with landmark time-points only for initial and final states of the process.

Some recent advances build on QSIM+Q2 to greatly strengthen the semi-quantitative conclusions that can be drawn from incomplete information.

Q3

Q3 extends Q2 with *step-size refinement*, adaptively inserting landmarks into qualitative intervals to decrease the effective step-size and strengthen semi-quantitative inference. In addition to providing tighter predictions and more refutations, Q3 provides an important new guarantee that the uncertainty in the semi-quantitative behaviour prediction converges to zero as the uncertainty in initial conditions and the maximum step-size converge to zero. Along with Q2 soundness guarantee, this means that semi-quantitative reasoning smoothly spans the gap from purely qualitative simulation on the one hand to numerical simulation on the other.

Dynamic Envelope

Dynamic envelopes are used by the semi-quantitative simulation program NSIM to improve on Q2 by providing much tighter bounds over qualitative intervals. In Q2, behaviour over an interval is only bounded by the rectangle defined by bounds on landmarks at the endpoints of the interval. Dynamic envelops are numerically defined solutions to an *external system* of ODEs that can be derived from the QDE and Q2 bounds, and proved to bound its solutions. They are therefore curvilinear, and take advantage of stronger properties of the QDE.

5.5 Summary

Continuing the concise description of qualitative modelling and simulation in Chapter 4, this chapter introduces its semi-quantitative counterpart as the extension.

Now we are ready to start the exploitation of qualitative/semi-quantitative modelling and simulation in software process research, and to yield the major contribution of this thesis. The next part investigates three distinct but typical software processes, which are different in *purpose*, *domain*, *scope*, and *output* identified in the systematic review (cf. Chapter 3), using the modelling approaches described in this part.

Part III

INNOVATION I: MODELLING

Chapter 6

Modelling Software Staffing Process

The systematic review (cf. Chapter 3) reports the state-of-the-art of SPSM research and extracts the most modelled aspects of software process for the design of this research. This part follows this design and presents the modelling and simulation of the selected software processes, i.e. software staffing process, incremental development process and software evolution process.

This chapter* starts the qualitative and semi-quantitative modelling with a *generic development* process in *project life cycle*. A classic software staffing process is modelled and investigated here at *macro-process* level. As the outcomes, the qualitative and semi-quantitative simulation models can be used to estimate the duration (*time*) and effort (*cost*) of a middle sized software project with workforce change. Table 6.1 highlights the related aspects (in bold) to the software staffing process model in the research design.

The following sections first give a brief review of the related background, and then present the progressive development of qualitative model and semi-

*The work included in this chapter has been partially reported in [ZHKJ06] and [ZK06].

Table 6.1: Staffing process in relation to research design

Purpose level	Problem domain	Model scope	Output variable
cognitive level	**generic development**	single phase	**time**
tactical level	software evolution	multi-phase	**effort**
strategic level	incremental development	**project**	quality
		evolution	size

quantitative constraints for a software staffing process. The simulation results of the qualitative process model is further used to revisit the Brooks' Law. Moreover, an illustrative example of semi-quantitative simulation is presented for the comparison with quantitative alternative models.

6.1 Background

6.1.1 Software Staffing Process & Brooks' Law

Software project staffing includes the activities of managing the human resource in software development [AH89], typically as personnel shortcomings, recruitment, assimilation, and turnover. Since the personnel effort is the most important component and major indicator of software cost, people issues, the focus of software staffing process, has gained recognition as the core of software project management.

Brooks' Law, since its publication, might has been the best-known and widely-endorsed in the literature regarding software staffing process, which states [Bro95]:

> "*Adding manpower to a late software project makes it later.*"

The assertion of Brooks' Law was based on many studies that have demonstrated the negative impacts of communication and training overheads on software development productivity. In spite of the fact, it has not been formally tested so far. Though some previous studies (reviewed in the next subsection) intended to investigate Brooks' Law, most of them employed only a limited set of *quantitative* cases. Later in this chapter, we will investigate this *qualitative* statement through the *qualitative* research vehicle (QSIM) in this thesis, and further test it in the uncertain scenarios (for values and relations) constrained *semi-quantitatively.*

6.1.2 Related Models

A few previous researches have investigated the software staffing process using quantitative models. Three typical models are reviewed in brief as follows:

AHM's Model Abdel-Hamid and Madnick (AHM) modelled the basic process of software human resource management using system dynamics [AHM91]. They assumed two workforce levels, i.e. *new hired workforce* and *experienced workforce*, and then formulated the assimilation process as a first-order exponential delay (refer to the explanation in Chapter 8). They then enhanced manpower assimilation by separating the procedure into four stages with different productivity levels. They restricted the scope of their model to medium-sized software projects.

Madachy's Model Madachy developed a software staffing model using system dynamics as well [Mad08], which examines the classic Brooks' Law. He simplified AHM's model by focusing on the assimilation procedure. His model was built with two connected flow chains representing software development and workforce assimilation. The *requirements* are transformed into *developed software* at *software development rate*. The *new project personnel* are transformed into *experienced personnel* at *personnel allocation rate*.

Both of these staffing process models can be used to verify Brooks' Law [Bro95]. Unlike the study in this chapter, these models used specific numeric values, which were selected from the literature or historical data of company projects, to quantify the factors in the models. Then they simulated the process with the data from particular projects or example as inputs.

Stutzke's Model Stutzke developed a simple model in order to perform a similar investigation, with a similar output [Stu94]. He undertook the analysis of the process and effort of assimilating the new employees, and then tested his model against an actual project in which the manpower was doubled successfully and the original schedule achieved. Stutzke believed that the added burden of communication in a larger project was a second-order effect and did not model it.

6.2 Qualitative Modelling

This section presents the procedure of modelling software staffing process using the qualitative modelling approach described in Chapter 4.

6.2.1 Qualitative Assumptions

The qualitative model of software staffing process was developed by following the qualitative modelling procedure illustrated in Figure 4.1. The '*physical scenario*' is that of adding new staff to a software project. This section conducts Step 2 through Step 4 to implement the modelling approach. The last step '*quantitative behaviours*' will be used in the next section (Section 6.3) within semi-quantitative simulation as the extension of qualitative simulation.

Both AHM's and Madachy's models are quantitative models which consist of a set of ODEs. Here a qualitative model is created on the base of a minimal set of assumptions of the software staffing process in a *generic software project*, and avoid the quantitative relations which came from the particular types of software projects and may not fit other projects or organisations in a general context.

The qualitative model is derived from the following basic assumptions, including the unstated assumptions in the previous models:

Table 6.2: Assumptions of staffing process model

	Assumption	Constraint
1.	No changes of requirements SP	constant S_P
2.	Reworking included in SP during project	constant S_P
3.	SP is transformed to product by RSD	$S_C + S_R = S_P$ $R_{SD} = \frac{d}{dt}(S_C)$
4.	WFT consists of WFEX and WFNW	$WF_T = WF_{EX} + WF_{NW}$
5.	RND (REXD+RNWD) is calculated by multiplying PD by WF	$R_{EXD} = PD_{EX} * WF_{EX}$ $R_{NWD} = PD_{NW} * WF_{NW}$
6.	Increasing WFT increases RCO	$R_{CO} = \boxed{M^+}(WF_T{}^2)$
7.	RSD is difference between RND and RCO	$R_{SD} = R_{ND} - R_{CO}$
8.	PDNW is less than PDEX initially	$PD_{NW} < PD_{EX}$
9.	PDEX is constant on average	constant PD_{EX}
10.	WFNW are assimilated into WFEX by RAS	$WF_{EX} = WF_{ET} + WF_{ED}$ $R_{AS} = \frac{d}{dt}(PD_{NW})$

1. Software requirements *project size* (SP) do not change during the project life-cycle;

2. SP includes the necessary reworking performed in project;

3. SP is transformed from *remaining size* (SR), which initially equals to project size, into the *completed size* (SC) at the *software development rate* (RSD);

4. Project team, i.e. *workforce in total* (WFT), consists of *experienced workforce* (WFEX) and *newly hired workforce* (WFNW);

5. *Nominal development rate* (RND) has linear relationship with workforce (WF) and their productivity (PD);

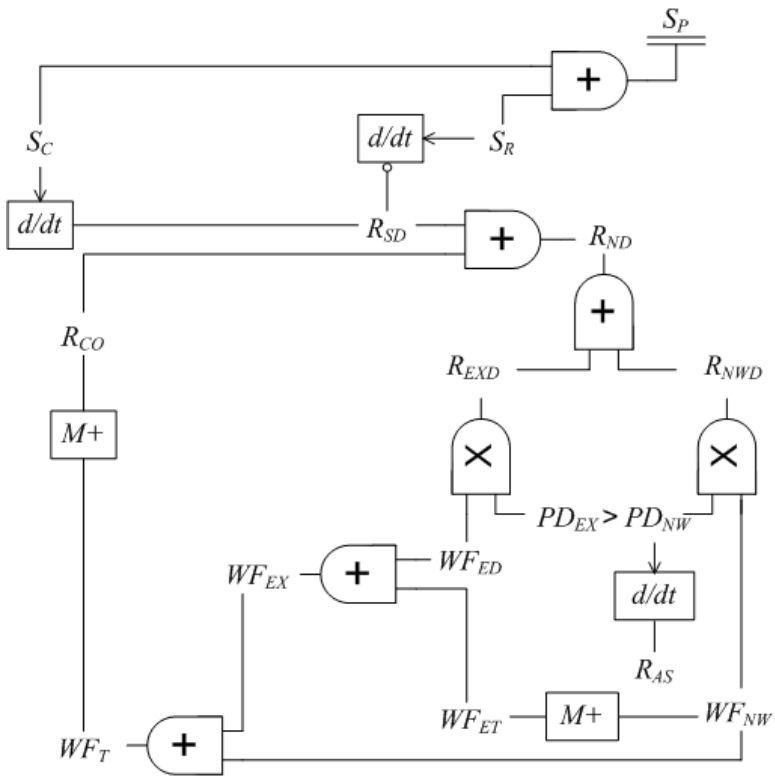
6. Adding more people to a project results in a larger *communication and motivation overhead* (RCO);

7. RCO is the only negative impact on RND;

8. A *new employee's productivity* (PDNW) is initially lower than an *experienced employee's productivity* (PDEX) on average;

9. Average PDEX does not change during project;

10. A new employee must be trained by experienced personnel before reaching their normal productivity level (PDEX).



SP: Project size
SC: Completed size
SR: Remaining size
RSD: Software development rate
RAS: Assimilation rate
WFT: Total Workforce
WFEX: Experienced workforce
WFNW: New hired workforce
RCO: Communication overhead rate
REXD: Experienced employee development rate
RNWD: New employee development rate
RND: Nominal development rate
PDEX: Experienced employee's productivity
PDNW: New hired employee's productivity
WFED: Experienced workforce for development
WFET: Experienced workforce for training

Figure 6.1: Qualitative abstract structure of staffing process

Brooks suggests that communication overhead increases by a factor of $(n(n-1)/2)$, where n is the project team size [Bro95]. It implies that increasing the size of project team increases the effort of communication overhead (Assumption 6).

Further, the qualitative abstract structure was created to graphically illustrate the qualitative constraints of software staffing process.

6.2.2 Qualitative Abstract Structure

Given the proposed assumptions in Section 6.2.1, this step investigates the staffing process of software development.

Initially, *remaining project size* (SR) equals the requirements (SP). As time progresses, SR decreases and will be processed to become *completed project size* (SC), which represents progress made in development. The project will be com-

pleted when Sc equals the original requirements, or Sr drops to zero.

There are two major differential structures included in this model. The first one is *software development rate* (Rsd), which indicates the development speed of software project. It is determined by the two factors: *nominal development rate* (Rnd), and *equivalent communication overhead* (Rco). Rnd can be further decomposed into development rate contributions from the experienced personnel and new personnel.

The *workforce in total* (WFt) equals *experienced workforce* (WFex) plus the *newly hired workforce* (WFnw), each group works at different productivity rates. Because new employees join the project, a portion of experienced staff (WFet) have to leave their development tasks for training the new hirees. The *experienced employee development rate* (Rexd) results only from the *experienced workforce for development* (WFed). A monotonically increasing function M^+ exists between the workforce and corresponding development rate. It means that increasing the manpower in project leads to a higher *nominal development rate* (Rnd).

However, to determine the actual *development rate* (Rsd) in project, we also need to consider the *communication overhead* (Rco), which causes a drop in the actual Rsd below Rnd. It is expressed as a nonlinear function of WFt. AHM quantified the function as $(0.06 * n^2)$ [AHM91]. While we can formulate the relation by the qualitatively accepted Brooks' assumption (Assumption 6) as *"the communication overhead rate monotonically increases with the square of total workforce"*.

The second differential relation is the *assimilation rate* (Ras) of new workforce introduced, which represents how quickly the productivity of new staff increases to the level equivalent to the experienced staff's through training.

6.2.3 Qualitative Differential Equations

Two QDEs were developed to convert the qualitative abstract structure model (Figure 6.1) to formal constraint programs (later input to simulation engine). One QDE is used to describe the normal software development process, and the other to represent the interaction and relations in staff assimilation process. (The complete programmed QDE for staffing process model can be found in Appendix B.)

Normal Development QDE

Figure 6.2 shows the constraint definitions in the first QDE, which represents the normal software development process without new employees being hired. All the constraints are based on the first four assumptions in Section 6.2.1.

```
(constraints
  (add Sc Sr Sp)
  (constant Sp)
  (d/dt Sc Rsd)
  (constant WFnw 0)
  (constant WFex)
  (add WFex WFnw WFt)
  (constant PDex)
  (Mult PDex WFex Rexd)
  (constant Rnwd 0)
  (add Rexd Rnwd Rnd)
  (M+ WFtl Lcm)
  (Mult Rnd Lcm Rcm)
  (add Rsd Rcm Rnd))
```

Figure 6.2: Formal constraints for normal development QDE

Assimilation Process QDE

AHM only formulated the assimilation process as a first-order exponential delay. In terms of the design of the training program applied, the model of this process can be refined in three different ways.

Refinement 1 The newly hired staff will be gradually transferred into the experienced staff pool by the assimilation rate (RAS). In this situation, the amount of new staff will be reduced to zero when the assimilation process finishes. The productivity of new employee stays at its initial low level. As the relationship between *experienced workforce for training* (WFET) and *newly hired workforce* (WFNW) is monotonically positive, the amount of trainer needed gradually decrease to zero in the course.

Refinement 2 There is an *one-off* transfer from the newly hired staff to the experienced staff once the assimilation process finishes. However, the former's average productivity is increasing during the procedure, and will be equal to the latter's productivity at the end (of assimilation). The amount of experienced trainers required is constant until the assimilation terminates, and then all return to normal development process.

Refinement 3 The productivity of the new employee stays at zero during the *orientation period*, since they have to *"learn the project's ground rules, the goals of effort, the plan of work..."* and what has been done in the current project. Their productivity starts to increase after this period until the end of assimilation, or in other words, they start to be assimilated into the experienced workforce pool as the first refinement addressed.

```
(constraints
  (add Sc Sr Sp)
  (constant Sp)
  (d/dt Sc Rsd)
  (constant WFnw)
  (constant WFex)
  (add WFex WFnw WFt)
  (M+ WFnw WFet)
  (add WFed WFet WFex)
  (constant PDex)
  (d/dt PDnw Ras)
  (constant Ras)
  (Mult PDex WFed Rexd)
  (Mult PDnw WFnw Rnwd)
  (add Rexd Rnwd Rnd)
  (M+ WFt Lcm)
  (Mult Rnd Lcm Rcm)
  (add Rsd Rcm Rnd))
```

Figure 6.3: Formal constraints for assimilation process QDE

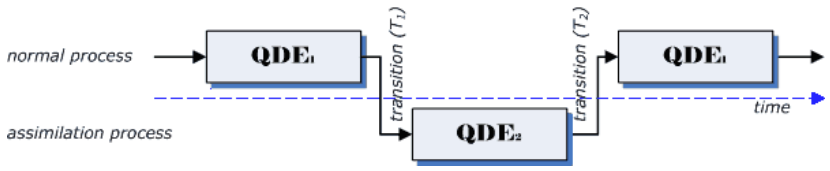


Figure 6.4: QDEs and transitions during simulation

This chapter selects the second refinement, which is different from AHM's model, in development staffing process model. It seems more realistic according to the author's experience and allows a well defined transition point for qualitative modelling. Figure 6.3 shows the second QDE coded.

Figure 6.4 illustrates the relations (transitions) between these two QDEs in model execution. The transition conditions to switching between these two QDEs are the beginning of assimilation when the new employee's productivity start to increase, and the end when their productivity reaches the equivalent level of experienced personnel.

6.3 Semi-Quantitative Constraints

This section extends the qualitative model with semi-quantitative constraints (Q2 equations). Q2 (qualitative+quantitative) equations, including parameter intervals and envelope functions, are the important representations of the incomplete quantitative knowledge and information in semi-quantitative simulation.

6.3.1 Parameter Intervals

The unit and unit conversion are not considered in the qualitative model because the landmark values are symbolic names for unknown real values (like algebraic variables). All quantities are assumed to have appropriate and compatible units in such circumstance. However, since some quantitative interval bounds are involved in semi-quantitative simulation, the units must be explicitly denoted for the interval arithmetic (as specified in Table 6.6).

Ratio of Productivity According to AHM's summary of the literature and interviews, the estimate for the productivity of newly hired workforce relative to that of experienced personnel varies from 0.33 to 0.64. Here, the interval of [0.4, 0.6] is assumed for the quantitative extension.

Assimilation Delay The range of proposed assimilation delay is quite large in the literature. It was set at 80 days in [AHM91], whereas 20 days in [MT00]. Although some project attributes in their studies are different, the team size and project duration are comparable. For EXAMPLE, a prototype project from AHM selected in the case study (see Section 6.4.2), a moderate range (say [60, 80] days) is employed for quantifying this parameter.

The value ranges of SP and WF highly depend on the particular project and organisation. They are specified later in Section 6.4.2.

6.3.2 Envelope Functions

As Figure 6.1 indicates, there are two monotonic functions included in the qualitative staffing process model. We need to correspondingly specify two sets of numerical envelope functions providing bounds on them for Q2 inference.

M^+(WFNW, WFET) When more new employees join in the project, more experienced people have to be assigned to train them. A linear relationship was reported in the literature. AHM summarised the ratio ranges from 15% to 25%, and set it at 20% in their model [AHM91]. The value of 0.25 was used in Madachy's model [MT00]. So the range of [0.15, 0.25] (from AHM) is applied here to bound this envelope function.

```
(envelopes
  ((M+ WFnw WFet)
    (upper (lambda (x) (* x 0.25)))
    (u-inv (lambda (y) (* y 4)))
    (lower (lambda (x) (* x 0.15)))
    (l-inv (lambda (y) (/ y 0.15))))
  (M+ WFtl Lcm)
    (exact (* 0.0006 (square (x))))
    (e-inv (sqrt (/ y 0.0006)))))
```

Figure 6.5: Monotonic envelope functions

M$^+$(WFTL, LCM) Brooks suggests that *communication and motivation overhead* increases by a factor of $n(n-1)/2$, where n is the project team size [Bro95]. It is widely accepted and implies that increasing the size of software project team increases the effort of overheads.

AHM specifically formulate this nonlinear relation as $(0.06n^2)$ between *workforce in total* (WFT) and the percentage of *communication and motivation loss* (LCM). It is consistent with Brooks' assumption. An exact function (i.e. the upper bound equal to the lower) is imported into the Q2 equations. Figure 6.5 illustrates these two specified monotonic envelope functions in `envelopes` section of the QDE.

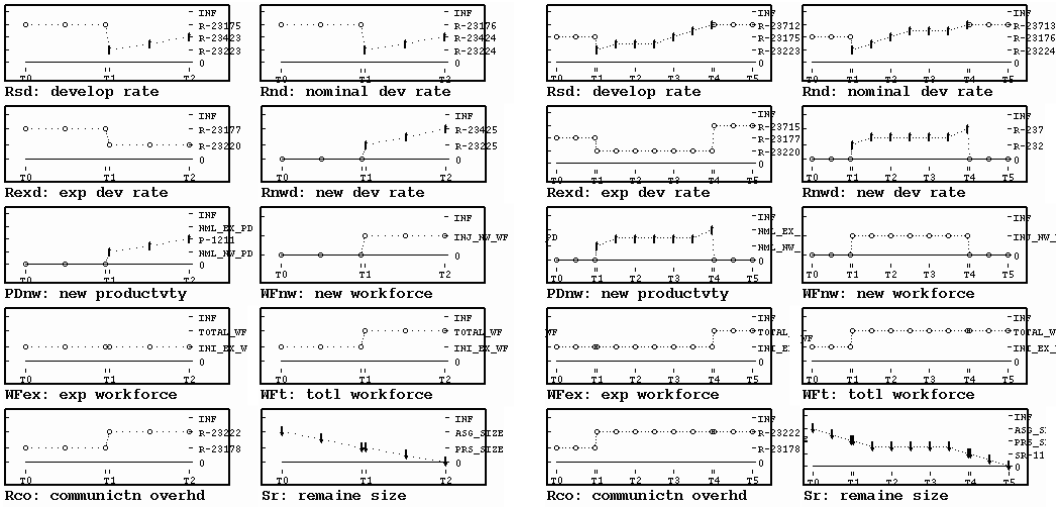
6.4 Case Study: Brooks' Law

With reference to AHM's model, the assimilation procedure is triggered by the workforce gap, which presents the difference between the *workforce in total* and the *workforce sought*. When *workforce sought* is larger than current WFT, new employees will be hired and injected into the software project. However, *workforce sought* usually depends on the project schedule pressure [AHM91], which is decided by the *completed project size* (SC) and other outputs from the detailed model of the software development sector. Hence, to simplify the staffing model and focusing on assimilation impact, the simulation triggers the QDE_2 (cf. Figure 6.4) when *remaining project size* (SR) reaches the size under pressure (PRS_SIZE) in this case.

6.4.1 Qualitative Simulation

Possible behaviours

The qualitative model outputs a number of possible behaviours. They are further filtered by the constraint that the remaining project size becomes zero when sim-



(a) Behaviour 57 of 112 (b) Behaviour 48 of 112

Figure 6.6: Examples of possible behaviours

ulation terminates. Finally, 112 behaviours are generated automatically through simulation (QSIM). The predicted behaviours are divided into two categories:

1. behaviours only passing the first transition point (T_a) when the assimilation procedure is triggered, i.e. project finishes before the assimilation is complete (the second transition point);

2. behaviours also passing the second transition point (T_b) when the assimilation completes, and then being followed by project closure.

Here, T_a is used to represent the transaction time when new staff are added in and T_b identifies the time when all new staffs have been assimilated and transferred to be the experienced workforce. In the Behaviour 48 (shown in Figure 6.6-b), the first time point with the symbol of parallel $^{\parallel}$ (T_1) presents the status at time T_a and the second $^{\parallel}$ T_4 indicates status at time T_b.

Two examples of the above behavior categories are shown in Figure 6.6 *a*) passing T_a only; and *b*) passing both T_a and T_b. Furthermore, four types of variables are identified from the outputs:

1. variables with constant values (SP, RAS, and PDEX);

2. variables change their values only at transition points T_a or T_b (WFT, WFNW, WFEX, WFED, WFET, RCO, and REXD);

3. variables keep varying their values only between T_a and T_b (RSD, RND, RNWD, and PDNW);

Table 6.3: Type 2 variables

	initial value	value @ T_a	T_a to T_b	$*$value @ T_b
WFNW	0	$\uparrow > 0$	no change	$\downarrow 0$
WFEX	> 0	$\downarrow$ [0, initial]	no change	$\uparrow$>initial
WFT	> 0	$\uparrow$>initial	no change	no change
WFED	n/a	$\uparrow > 0$	no change	n/a
WFET	n/a	$\uparrow > 0$	no change	n/a
RCO	> 0	$\uparrow$>initial	no change	no change
REXD	> 0	$\downarrow$ [0, initial]	no change	$\uparrow$>initial

$*$only apply for behaviours passing T_2

Table 6.4: Type 3 variables

	initial value	value @ T_a	T_a to T_b	value @ T_b
RNWD	0	$\uparrow > 0$	$\uparrow$	$\downarrow 0$
RND	> 0	$? > 0$	$\uparrow$	no change
RSD	> 0	$? \geq 0$	$\uparrow$	no change
PDNW	0	$\uparrow > 0$	$\uparrow$	$\downarrow 0$

4. variables changing values throughout the project (SC and SR).

Because the qualitative modelling is used for continuous and dynamic simulation, we are only interested in the behaviours of the latter three types of variables in this subsection (Type 4 variable's behaviours are discussed in the following dedicated subsections). According to the model outputs, it is worthy to examine how these variables change at two transaction points (T_a and T_b), and in the period between them.

Type 2 variables: There are several variables changing within the simple pattern across the predicted behaviours: they jump from initial values to a larger value at T_a, then no further changes occur until T_b or project finishes. Table 6.3 summarises the change patterns of all Type 2 variables through two transitions.

Type 3 variables: The variables in this category present more complicated and nonlinear changes in some behaviours. To identify the difference with Type 2 variables, we focus on the value changes at T_a, and between two transitions. Table 6.4 shows the change patterns of all Type 3 variables through two transitions. The increasing trend is found for this type variables between T_a and T_b.

Software Development Rate

As the project duration and development speed are directly determined by *software development rate* (RSD), it can be regarded as the vital variable of the model. It also exhibits complicated changes among the possible behaviours generated by this model.

RSD declines sharply in most predicted possible behaviours when new personnel are introduced at the transition point (T_a). Then it recovers gradually due to the increasing productivity of the novices. It may exceed the initial value in some scenarios where the project finishes after the assimilation period.

The relations between RSD and SR are examined by the phase view (cf. Chapter 4) that is plotted in Figure 6.7. The dashed line arrows indicate the time progress directions. The changes of RSD are shown on vertical dimension and the corresponding values of SR presented on horizontal axis. Each dot in the plots indicates both values of RSD and SR at the same transition time point. The left most dot in each plot represents the end of project.

There are increasing trends of RSD in all behaviours after first transition (T_a). The pattern of RSD can be identified as "$\sqrt{\ }$" shape in most behaviours as shown in Figure 6.7-a and -b. The immediate drops are found when adding new staff into project team at PRS_SIZE. The final software development rate will achieve the values below (b) or above (a) the initial rate.

Nonetheless, this qualitative model also outputs some other patterns inconsistent with Brooks' Law. Figure 6.7 (c) indicates no significant behaviour change of development rate at PRS_SIZE, and (d) shows an immediate increase at PRS_SIZE.

Project Completion

We may determine whether the project will be later, as stated by Brooks' Law, by comparing the simulated project's (with assimilation process) closure time point with the original scenario without new workforce hired. Figure 6.8 shows the phase views of the comparison: the horizontal axis indicates the value of SC with new staff introduced, where the vertical represents SC in the original project. The right most points in each plot represent the end of project. The right plots (d, e, f) include two transition points during project, whereas the left plots (a, b, c) have single transition. Instead of one specific result generated from the quantitative models, this model outputs three states identified by examining in which scenarios the project achieve the requirement size (ASG_SIZE):

1. projects where assimilation finishes later than the original schedule, as Figure 6.8-a and -d;

2. projects where assimilation finishes earlier than the original schedule, as Figure 6.8-b and -e;

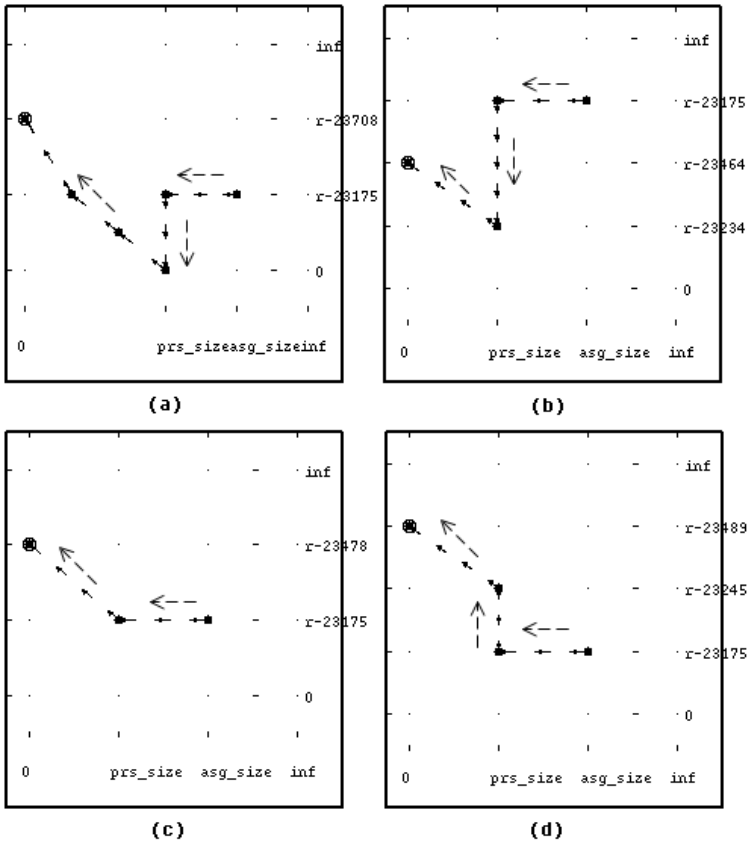


Figure 6.7: Phase view of RSD vs. SR

3. projects where duration with assimilation just equals the original schedule, as Figure 6.8-c and -f.

The first state (a, d) is consistent with Brooks' Law. The results indicate that the above three final states are all possible for software project in which the new workforce are added in. Therefore, only given "*adding manpower to a late project*" will not necessarily "*make it later*".

6.4.2 Semi-Quantitative Simulation

Baseline Project

To illustrate some results of the semi-quantitative simulation (SQSIM) model, AHM's EXAMPLE project is selected as a baseline project. EXAMPLE is a medium-size project with 64 KDSI, and used COCOMO to calculate the workforce level. The main attributes of EXAMPLE are summarised in Table 6.5.

Two scenarios designed by AHM to investigate the aggressive manpower acquisition policy are to add new personnel at day 260, when testing starts, and

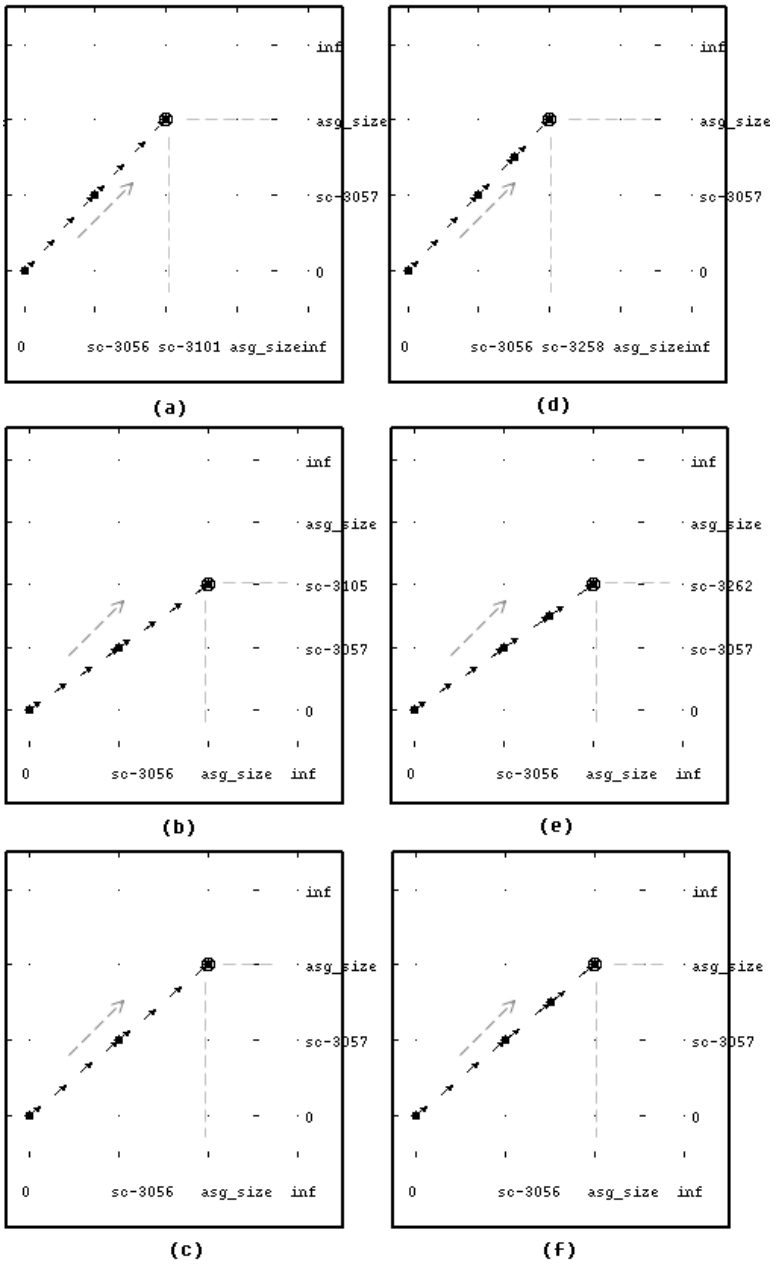


Figure 6.8: Phase view of project schedules

Table 6.5: Major attributes of EXAMPLE project

Attribute	Value
Project size	64 KDSI
Man-days	3,795 man-days
Duration	430 days
Initial team size	4 men
Maximum team size	[8, 18] men
Average ratio of productivity	0.5
Ratio of experienced mentors to novices	0.2
Average productivity of experienced workforce	36 DSI/man-day

Table 6.6: Value ranges for quantitative quantifying

Parameter	Value Range
Project size	64 KDSI
Initial experienced workforce	[4, 5] men
Newly hired workforce	$[3,4]^1/[11,12]^2/[7,8]^3$ men
Nominal productivity of experienced personnel	36 DSI/man-day
Ratio of experienced mentors to novices	[.15, .25]
Ratio of productivity	[.4, .6]
Assimilation delay	[60, 80] days

increase the total workforce to 8 and 18 [AHM91]. This subsection replicates these two scenarios in semi-quantitative simulation, and adds one more medium scenario for the further comparison:

Scenario 1: adding [3, 4] new developers on day 260;

Scenario 2: adding [11, 12] new developers on day 260;

Scenario 3: adding [7, 8] new developers on day 260.

With reference to the semi-quantitative constraints (Section 6.3) and EXAMPLE (Table 6.5) project, the numeric ranges (as shown in Table 6.6) are assigned for parameters of the semi-quantitative model.

Accordingly, the monotonic envelope function $\mathsf{M}^{+}(\mathsf{WF_{ET}}, \mathsf{WF_{NW}})$ has been updated with the value ranges of workforce as illustrated in Figure 6.9. The area of the dashed-line rectangle between two linear equations indicates the possible value range of the envelope function for each scenario.

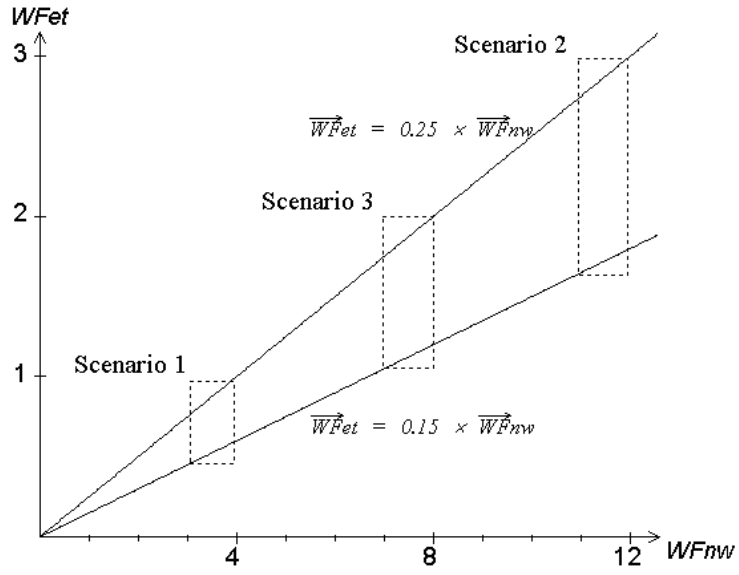


Figure 6.9: Envelope functions of required experienced developers for training

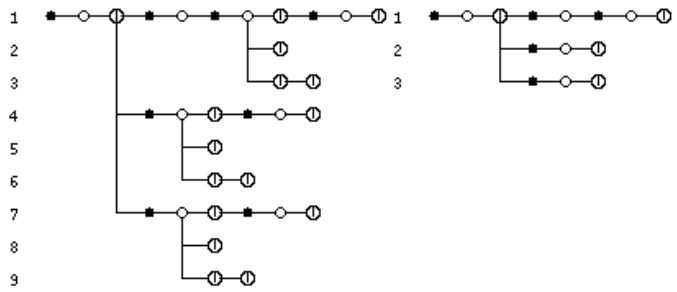


Figure 6.10: Possible behaviour trees for EXAMPLE project

Possible behaviours

In Section 6.4.1 the QSIM staffing process model generates 112 possible behaviours, which can be classified into two types.

After the quantitative constraint propagation, Q2 outputs only 9 behaviours for Scenario 1 and 3 behaviours for Scenario 2 as final solutions (shown in Figure 6.10). Comparing with the behaviours generated by QSIM, all behaviours of Scenario 2 plus 3 behaviours of Scenario 1 are Type 1, i.e. passing only one transition, and other behaviours of Scenario 1 are Type 2, i.e. including two transitions. (The behaviours of Scenario 3 are a mix of 2 of Type 1 plus 3 of Type 2.)

Figure 6.11 shows the changes of main variables in Behaviour 1 of Scenario 1.

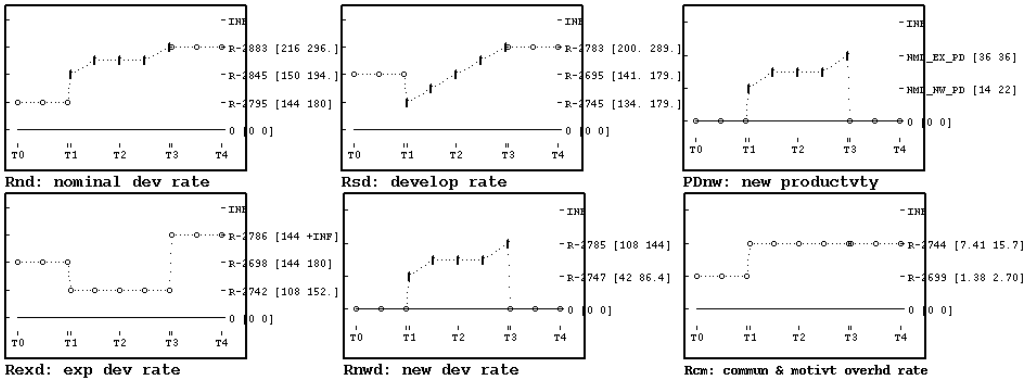


Figure 6.11: Behaviour 1 for Scenario 1 of EXAMPLE project

They are similar to the changes in most behaviours except the different numeric ranges of the landmarks.

Software Development Rate

Because the software project duration and development speed are directly determined by *software development rate* (RSD), it can be considered as the vital variable in this model. It also exhibits complicated changes and major differences among the possible behaviours generated.

Figure 6.12 shows three behaviour patterns of RSD in Scenario 2, which are similar to those in Scenario 1: when the new hirees are introduced into the project, *1*) RND ascends and RSD falls immediately; *2*) RND undergoes an instantaneous increase and RSD starts to increase gradually; *3*) both of them jump straightway.

All behaviours indicate that the training overhead is less than the additional development rate by the new workforce. Pattern 1 implies the *communication and motivation overhead* (RCM) is greater than the increase of *nominal development rate* (RND). It results in the drop-off of RSD. Whereas, RCM is equal to and less than the extra RND in Pattern 2 and 3.

It is noticeable that Pattern 2 and 3 generated through Q2 may be inconsistent with Brooks' Law, i.e. adding manpower brings no negative impacts on the overall software development productivity.

Project Completion

The *remaining size* (SR) and *completed size* (SC) are the variables changing in opposite directions over the project, and being decided by *software development rate* (RSD). By investigating its behaviours in Figure 6.12, the positive impact of new staff's introduction can be easily identified (Pattern 2 and 3), and it must lead to shrinking of project elapsed time.

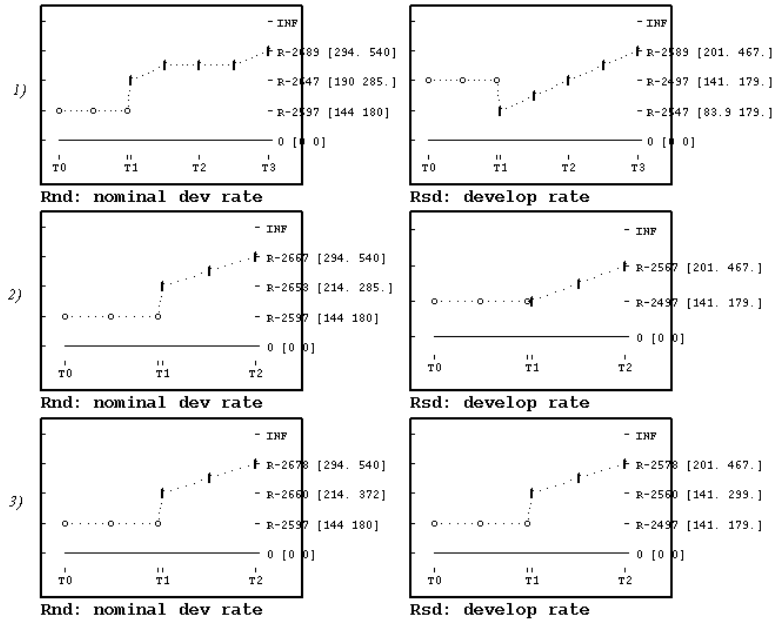


Figure 6.12: Behaviours of RND and RSD in Scenario 2

Table 6.7: Simulated project completion

Scenario	Completion	Remark
S1	**[339, 423]** days	add [3, 4] developers on day 260
S2	**[309, 386]** days	add [11, 12] developers on day 260
S3	**[320, 396]** days	add [7, 8] developers on day 260

Table 6.7 illustrates the simulated project closure time for each scenario. Instead of one single-value result generated from the quantitative model, the semi-quantitative model outputs all possible value ranges by simulating the project under the given scenarios. Figure 6.13 shows the envelopes against the original completion time (430 days). Each rectangle in dashed-line indicates the introduced new workforce and the corresponding possible project duration. Given these three scenarios, it is guaranteed that the EXAMPLE project can finish earlier.

6.5 Model Comparison & Discussion

Because a qualitative model (with its semi-quantitative constraints) simulates all possible behaviours and states of a specific system (with incomplete quantitative knowledge), such a model cannot be fully evaluated by particular sets of

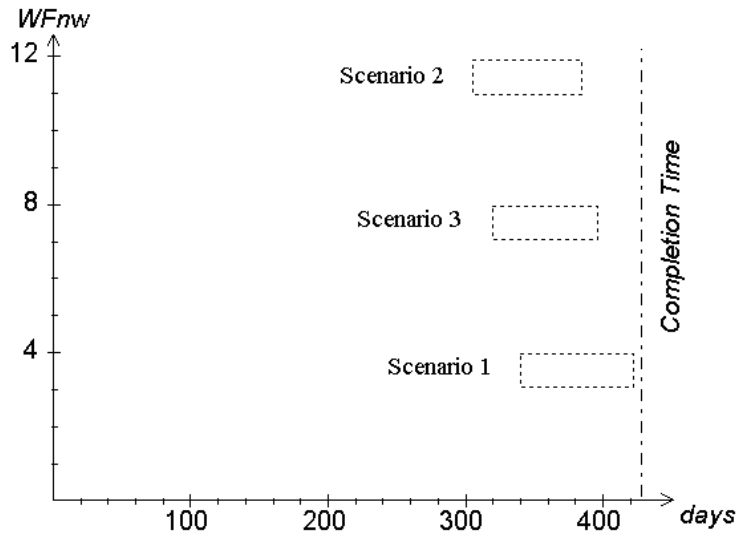


Figure 6.13: Project durations of scenarios

quantitative evidences.

This section conducts the evaluation by comparing the qualitative and semi-quantitative outputs with the existing calibrated quantitative staffing models, and further comparing with the empirical evidence.

6.5.1 Comparison with Outputs of Related Models

Stutzke's model will not be compared with the results of QSIM and SQSIM modelling, because the models developed in this chapter consider the increasing effort on communication, which is not included in his study.

Both AHM's and Madachy's models show the staffing process and examine Brooks' Law under predetermined conditions, since the equations and parameters must take on quantitative values.

AHM's model provides detailed insights into what happens under several assumptions as to how much manpower is added, and when [AHM91]. They identified that the new personnel always have an immediate negative impact on software development rate, and concluded that *"adding more people to a late project always makes it more costly, but it does not always cause it to be completed later"*. In particular, adding extra workforce early in the schedule is much safer than adding them later.

From quantitative aspect, according to AHM's simulation results, the completion times of Scenario 1 and 2 are less than, but close to 400 days [AHM91]. They are covered by the possible value ranges generated by the semi-quantitative model.

By performing the sensitivity study of his model, Madachy clarified Brooks' Law qualitatively as *"adding manpower to a late software project makes it later if too much is added too late"* [MT00].

According to the simulation outputs in this chapter, two major findings can be concluded as fellows:

Findings 1 Adding new personnel might bring the immediate negative or positive impact on software development rate, but it does not always make a project finish later (cf. Figure 6.12).

Findings 2 Adding a large number of new personnel in a late phase (e.g. testing) obviously makes project costly, and might contribute more slightly to the schedule improvement (cf. Figure 6.13).

All the results of previous quantitative models can be obtained by this work. The quantitative models always assume that project schedule merely depends on the amount of new manpower and the introduction time. However, they did not take into consideration the impacts of other factors associated with project and organisation (see Section 6.5.3 for further discussion).

6.5.2 Comparison with Empirical Evidence

Jeffery investigated the relationships between productivity, effort and elapsed time that the time-sensitive cost models claimed for software development. He collected and analysed the data on 47 projects from 4 large organisations in Australia [Jef87]. The plot of the ratio of the actual effort to estimated effort against the ratio of actual elapsed time to estimated elapsed time is shown as Figure 6.14.

The gap between the actual remaining effort and the estimated remaining effort results in the gap of manpower, which directly leads to workforce sought (cf. Section 6.4). Project managers may decide to hire new workforce, as they perceive that more employees are sought to complete the additional workloads.

In Figure 6.14, Quadrant 1 and 2 reflect the scenarios with more actual effort needed than estimated, whereas Quadrants 1 and 4 cover the cases of elapsed time expansion. The points in Quadrant 1, which represent more effort and longer elapsed time, are consistent with the situation that Brooks' Law claims. However, Quadrant 2 includes the projects with more effort but less elapsed time, which is inconsistent to Brooks' Law.

The points falling in both Quadrant 1 and 2 are consistent with the states predicted through the above qualitative simulation. We can also find from the scattergram that when the actual effort exceeded the estimate, most projects could finish within the actual elapsed time close to the estimate.

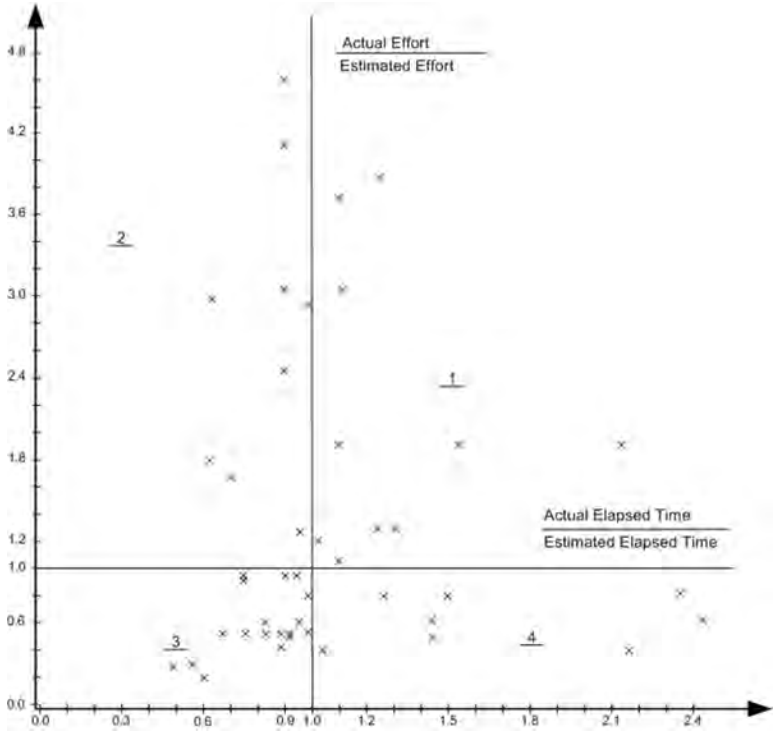


Figure 6.14: Effort ratio vs. elapsed time ratio

6.5.3 Discussion

In Section 6.4.1, it was already found that the project with new employees introduced might be completed earlier, later, or at the estimated schedule through the qualitative simulation. The three final states are all possible with the basic qualitative assumptions (shown in Figure 6.8).

Furthermore, given the specific scenarios, the EXAMPLE project with new employees hired can be completed earlier than the original completion time through SQSIM. It is inconsistent with Brooks' Law.

Even though other researchers found Brooks' Law might be questionable, they emphasised the causes as being the number of new manpower hired and when, which are the inputs with the values they can change in the quantitative models. However, we can identify other factors that may impact the result through qualitative simulation. These factors can be located among the variables associated with the qualitatively constrained monotonic formulae, i.e. M^+ or M^-.

Communication Overheads (Rco) Even though it is widely accepted that communication overheads account for a second-order effect on added effort, the relationship with workforce may vary between projects, even different tasks. It can be expressed by the variance of the constant coefficients of the relationship.

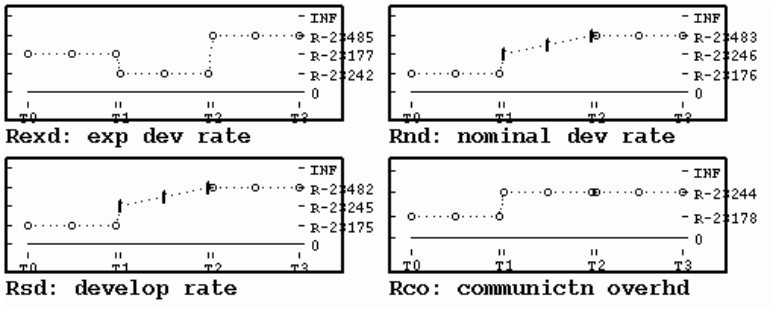


Figure 6.15: Behavior 80 of 112

For instance, when the project hires new staff to conduct the testing tasks, the communications between them are not so significant as during initial phase of the development. Hence, the drop of RSD due to RCO may be small.

In the scenario shown as Figure 6.15, though *experienced employee development rate* (REXD) declines sharply at the first transition, the overall *software development rate* (RSD) increases significantly at T_1. This might be caused by the situation when many new employees were added in project. They lead to the increase of RND, but no significant increase in RCO occurs.

Assimilation Rate (RAS) AHM decomposed the *assimilation rate* (RAS) into *"hiring delay"* plus *"transfer delay"*, which assumed to be 40 and 80 days respectively. Madachy set the delay to be at the average of 20 days. The actual assimilation rate is decided by multiple factors, which may include the average novice's competency, the mentor's teaching skills and programme design, and the tasks being assigned to the new personnel, and so on. In the qualitative model, one qualitative rate without specific numeric value is involved and all consequent possibilities and corresponding impacts can be examined through simulation. One quantitative value range instead of a single value was applied in the semi-quantitative constraint, and then all consequent possibilities and corresponding impacts can be predicted through the simulation. It can avoid the loss of any possibilities.

New Employee's Productivity (PDNW) The low productivity of new manpower was believed to be the vital factor that leads to the immediate decline of the entire development rate. Its value varies in the quantitative models: it is set to 0.8 of nominal productivity in Madachy's model, and AHM separated it into 4 levels from 0.1 to 0.9. In contrast, Assumption 8 in qualitative modelling (Section 6.2.1) only claims a lower productivity of the new employees than the experienced employees. In addition, the semi-quantitative constraint assumes this low productivity of new employees is between [0.4, 0.6] of experienced personnel's. In practice, if the project managers weigh the schedule constraint more than

cost, they can pay more to hire the software professionals with potential higher productivity. Then they will be absorbed quickly, and more importantly, may increase quality (fewer defects) and improve the productivity of the original team.

Ratio of Experienced Mentors to Novices Madachy's model assumed the default value of 0.25 that indicates one experienced person is needed to train four new employees until they are fully assimilated. AHM estimated the range from 15% to 25%, and there are no explicitly proposed formulae in literature. However, the ratio might be lower if the new team members are internal and transferred from the similar projects. The qualitative model and semi-quantitative constraint can allow all these possibilities and simulate their corresponding states without further details given.

Management Adjustment None of previous models takes into account the management adjustment, such as the fact that the work must be repartitioned in practice, a process addressed by Brooks.

The above comparison and discussion reveal a complex structure that is not limited in the relationship between "*when to add new workforce in project*" and "*how many new personnel need to be hired*", which were focused by previous studies.

In terms of the analysis in this chapter, the practical meaning of Brooks' Law may be that it presents a simplified approximation to the truth of software staffing process for project managers, and as a rule of thumb to warn them against blindly making the simplistic response to a late project. However, by applying appropriate recruitment, training and task assignment strategies, it is possible to assimilate new staff without encountering schedule problems.

6.6 Summary

Following the research design, this chapter investigates a *generic development project* with focus on its related staffing process using qualitative and semi-quantitative approaches.

1. It develops the *first* qualitative *explanatory structural* model of software development process for simulation study.
2. This model is further extended to be the *first* semi-quantitative software process model by applying incomplete quantitative constraints.
3. It is the *first* study that comprehensively demonstrates the the entire procedure of qualitative/semi-quantitative modelling and simulation of a software process.

4. This study initially tests the *feasibility* and *applicability* of qualitative/semi-quantitative simulation in modelling a *generic* software process with a minimum set of assumptions and constraints.

5. The model is further used to revisit Brooks' Law in a qualitative form, and provides new insights rather than those through previous quantitative studies.

6. The model is evaluated by comparing its outputs with the results from the related quantitative models and empirical evidence.

The next chapter presents the modelling and simulation of another typical software process, an incremental development process, but at a fine-grained level by looking into the phases in each iteration.

Chapter 7

Modelling Incremental Development Process

Chapter 6 describes the first qualitative structural software process model and its semi-quantitative version, which focus on software staffing process over project life cycle. Following the design of this research (cf. Section 3.6), this chapter* explores the qualitative and semi-quantitative modelling of software process at a fine-grained *scope* by looking into the incremental development at phase level. For this purpose, an incremental development process (one of the most modelled *domain* in SPSM identified in Chapter 3) is chosen and investigated with the *software quality* concern by focusing its test-and-fix phase (sub-process) in an intermediate increment. Table 7.1 shows the related aspects (in bold) of the incremental development model in the research design. The items already investigated in the last chapter (strikethrough) are ignored here to maintain the emphasis of the modelling and simulation in this chapter.

*The work included in this chapter has been partially reported in [ZKJ07d] and [ZKKJ08].

Table 7.1: Incremental development in relation to research design

Purpose level	Problem domain	Model scope	Output variable
~~cognitive level~~	~~generic development~~	**single phase**	~~time~~
tactical level	software evolution	**multi-phase**	~~effort~~
strategic level	**incremental development**	~~project~~	**quality**
		evolution	size

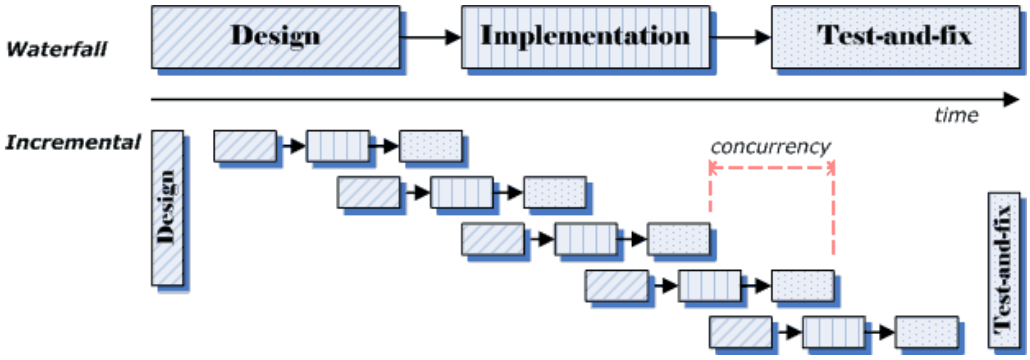


Figure 7.1: Waterfall vs. incremental development

7.1 Background

7.1.1 Incremental Development Processes

Incremental development is a broad terminology of software development process, which may cover iterative development, versioned implementation, spiral model, and so on. The basic idea is to divide the development into several smaller increments, which are gradually accumulated to become the complete system [Kar01].

The essence of incremental development is illustrated in Figure 7.1 by comparison with the traditional waterfall life cycles model. Basically, there are three phases in one increment, i.e. design (analysis), implementation, and test-and-fix.

This chapter only models the implementation and test-and-fix processes of an intermediate increment. After the current release is implemented, the progress enters a test-and-fix process, where this release is thoroughly tested and corrected. During this period, no new functionality is added into the release. The purpose of the process is to make sure that each release provides a robust foundation for its succeeding releases.

7.1.2 Conceptual Software Quality Model

In this study, we assume that any reduction in software defects that remain in a product improves the quality of that product. Accordingly, the defect level is used as the indicator of *software quality* in this chapter.

Boehm described the "*software defect introduction and removal model*" in [Boe81], which is analogous to the "*tank and pipe*" model introduced by Jones [Jon75]. Based on their models, a fundamental conceptual quality model is developed focusing on test-and-fix process of incremental development, as shown in Figure 7.2.

The graphic model shows that defects conceptually flow into a cascade of tanks through various defect source and transferring pipes. Be aware that the quality assurance activities, e.g. walk-through and inspection, are not included

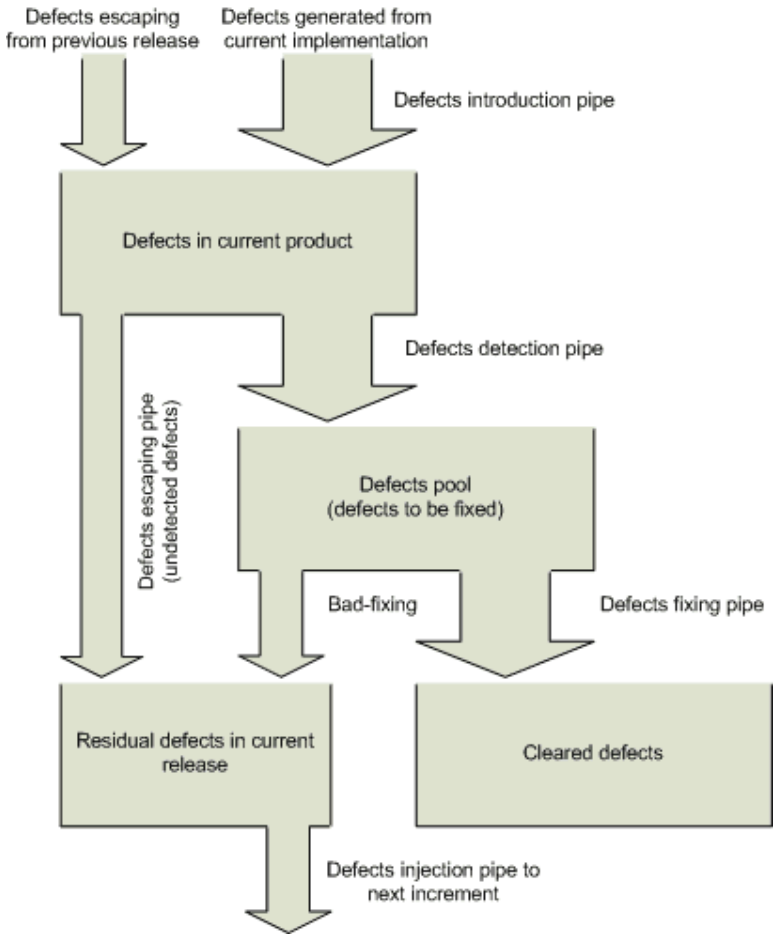


Figure 7.2: "*Tank-pipe*" software quality model of incremental development

in this model at current stage. The figure depicts that defects are drained off through the defect detection and defect fixing pipes. The residual defects of current release will be input to next increment. In the following sections of this chapter, '*error*' will be used as the equivalent term to '*defect*'.

7.1.3 Related Models

The software testing (or test-and-fix) processes have been investigated by researchers using quantitative modelling and simulation. This subsection provides brief descriptions and comments of these models separately.

AHM's Model Abdel-Hamid and Madnick (AHM) modelled the basic software testing process, which is a sector of their integrated software process model using system dynamics [AHM91]. However, their model is based on the waterfall test-

Table 7.2: Summary of related software testing process models

	Process	Model type	Life cycle	Error type	Dynamic
AHM's model	test	SD	waterfall	active, positive	yes
Huff's model	test-fix	analytic	incremental	new, old	no
Tvedt's model	test	SD	incremental	unknown	yes
Cangussu's model	test-fix	DTS (control theory)	n/a	single	yes

ing process, rather than the incremental testing. They did not differentiate the newly generated errors and the escaped errors from the test-and-fix process of the previous increments, which influence the current testing performance. Plus, their model neglects the switchover phenomenon of error fixing productivity.

Huff's Model Huff *et al.* developed the alternative causal model for the test-and-fix process of incremental development [HSS86]. They quantified the relations by quantitative equations. Nevertheless, their models did not identify the reproduction of remaining active errors in the succeeding increments.

Tvedt's Model Tvedt developed a comprehensive process model of concurrent incremental development [Tve96]. He considered the impacts on defect creation from engineer's capability, technical risk, and interdependency among the concurrent increments, which is not the case of our process. However, the active and passive errors were not explicitly handled in his model.

Cangussu's Model Cangussu *et al.* developed a software test process model based on concepts and techniques from control theory [CDM02]. Their model reinforces modelling the continuous feedback during test process, and computes the effort required and schedule slippage to meet the quality objectives under changing process environment. As AHM's model, they did not identify the error categories, and treated them as the same. In addition, they concentrated on the control-feedback during one test process alone, and omitted the influence between implementation and test processes.

The characteristics of the above quantitative models are summarised in Table 7.2.

7.2 Qualitative Modelling

In the context of incremental development, the primary purpose of the test-and-fix process model is to examine whether the increment can be completed within the desired time frame with the required quality. This section presents qualitative models for the implementation of an intermediate increment and its associated test-and-fix process.

The qualitative models consist of two interconnected models, which are developed to model the implementation (error generation) process and test-and-fix (error detection and correction) process for producing an intermediate release. At the qualitative modelling stage, this section investigates the incremental development processes, and abstract error-related features (assumptions) in qualitative form from them.

7.2.1 Modelling Implementation Process

One widely used linear model for software implementation is employed as the basic skeleton of this model, i.e. given workforce (WF), release size (S), and unit productivity (PD), the elapsed calendar days to complete the release can be calculated by $S/(WF * PD)$. However, because we are interested in the processes related to error generation and detection during each increment, more features related to error generation need to be included in this model. In addition, the management overheads (including communications, adaptations, staff's absence, configuration management, and so on) need to be considered as well when calculating the elapsed time.

During the implementation phase, there are two basic types of errors generated: '*active errors*' and '*passive errors*'. '*Active errors*' are errors which unless detected and retired will generate additional errors (active and passive) in subsequent increments. '*Passive errors*' remain in code until they are detected and retired but do not give rise to additional errors in subsequent increments.

Active and *passive* errors are considered in the incremental implementation process because they impact the number of errors introduced into the incremental test-and-fix process. If we suppose the system is developed with an incremental top-down strategy, then in the early releases, most of errors committed are in the core or high level components and become active. If these errors are not detected, they tend to propagate through the succeeding increments that build on one another.

Therefore, the errors generated through each increment arise in two ways. The first is through the development of the construction of new functionality required in each increment (increment size, S_i); the second is the errors generated by existing active errors. However, for many undetected active errors, the reproduction will not continue after producing one or two '*generations*' of errors [AHM91]. In

this case, they effectively become undetected passive errors.

The qualitative assumptions for modelling the implementation process are explicitly given below:

1. All resources are focused on one increment, i.e. increments are sequentially linked with each other, and there are no concurrent increments at any time;
2. The size of current release (S_i) does not change during current increment;
3. Current release size (S_i) includes the necessary work, e.g. design, coding, unit testing and rework except system testing and defect fixing in this release;
4. More old active errors (E_{AO}) from previous releases will reproduce more active and passive errors (E_{ARp} and E_{PRp}) in current implementation;
5. Increasing software development rate (R_{SD}) increases both active and passive error generation rate (R_{EAG} and R_{EPG});
6. Increasing the team size (WF) leads to a larger implementation overheads (R_{Oh});
7. The average development productivity (PD_{Im}) does not change during current implementation;
8. A fraction of remaining active errors (E_{ARt}) are retired to become old passive errors (E_{PO}) in each intermediate release.

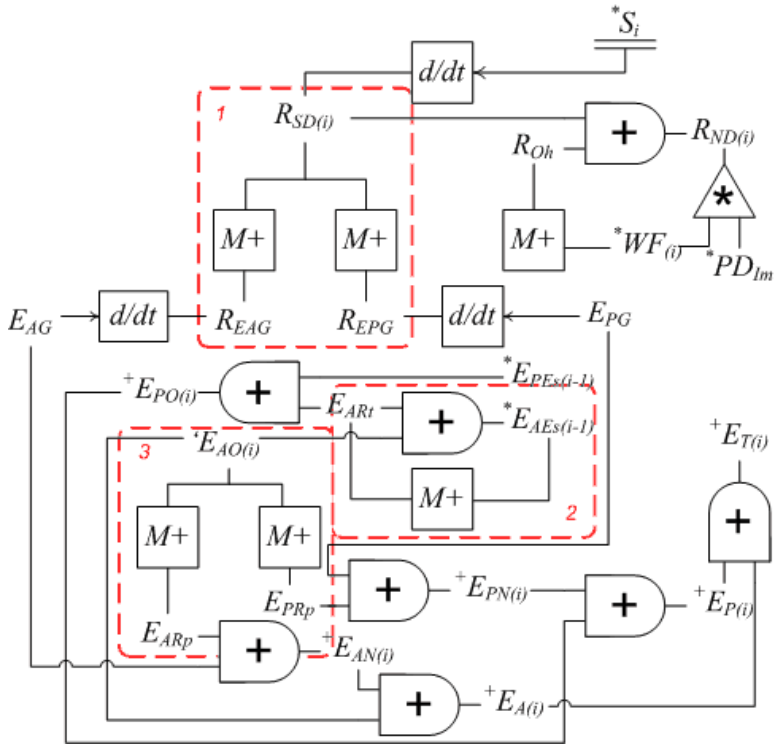
Based on the above qualitative assumptions, the qualitative model of implementation process is given in Figure 7.3, where the asterisk '*' denotes an input parameter, and the plus '$^+$' indicates an output parameter.

7.2.2 Modelling Test-and-Fix Process

The objective of test-and-fix process is to achieve an "*acceptable level of quality*", meaning that a certain percentage of defects will remain unidentified upon release of the software [Gal04]. During the incremental development, a small percentage of defects may escape from the current test-and-fix process, and remain in the implementation of next increment.

In test-and-fix process, a specific test suite is run and analysed, detected errors are reported, assigned, and eventually fixed. In practice, the test cases are usually performed by a stand-alone team to avoid any potential bias. The work of correcting errors mostly falls to the team who implement the design and coding.

The test suite contains multiple test cases, which are prepared in advance, and ready prior to test-and-fix. During the test-and-fix, the black-box testing



S(i): Size of increment i	EAEs(i): Escaping active errors from increment i
RSD: Software implementation rate	EPEs(i): Escaping passive errors from increment i
RND: Nominal development rate	EARt: Retired active errors
ROh: Management overhead	EAO: Old active errors in increment i
WF(i): Workforce for increment i	EPO(i): Old passive errors in increment i
EARp: Reproduced active errors	EAN(i): New active errors in increment i
EPRp: Reproduced passive errors	EPN(i): New passive errors in increment i
REAG: Active error generation rate	EA(i): Active errors in increment i
REPG: Passive error generation rate	EP(i): Passive errors in increment i
EAG: Generated active errors	PDIm: Implementation productivity
EPG: Generated passive errors	ET(i): Total errors in increment i

Figure 7.3: Qualitative model of implementation process in one increment

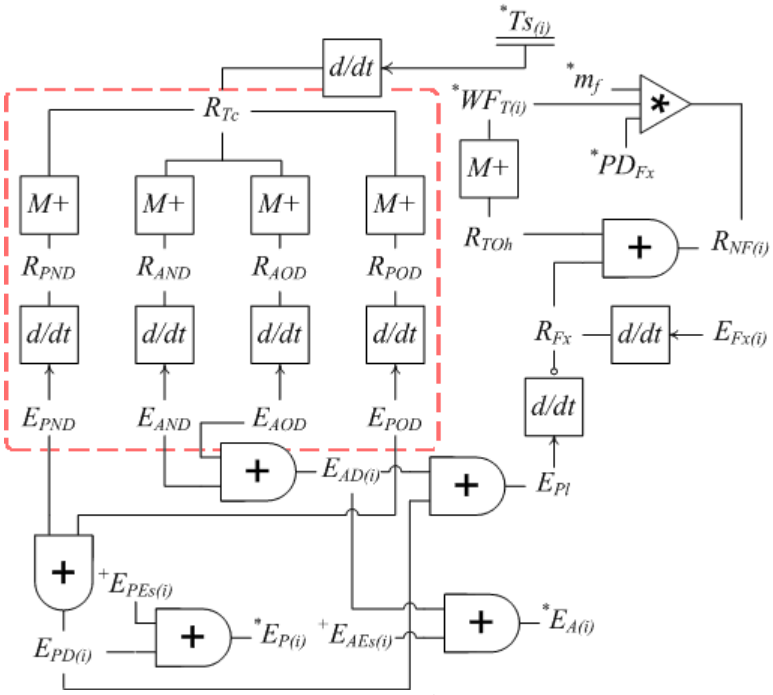
strategy is applied here, since we assume that unit testing (based on white box testing) takes place during the implementation process. The time spent detecting errors depends on the size of test suite (or the number of test cases in queue) of the current release, instead of the release size (lines of code or function points). However, the time spent on correcting errors relates to the number of the detected defects.

Huff *et al.* argued that the defect fixing rate (productivity) slows as average length of defect queue drops below a certain number [HSS86]. This switchover phenomenon normally happens near the end of defect fixing work, when a small number of defects in the queue for developers. A single multiplier is used here to reflect the productivity drop.

The qualitative assumptions for modelling the test-and-fix process are summarised as:

1. Test suite (TS) is prepared before test-and-fix starts, and does not change during the process;
2. Errors (active and passive, old and new) are uniformly distributed across the test suite for the current release. In the other words, given the implementation and test suite, completing more test cases produces more detected errors;
3. Errors generated in current release (EAN and EPN) have a higher probability of being detected with the same test case than the old errors (EAO and EPO) remaining from previous releases;
4. The average effort required to correct errors (PDFx) is constant throughout the process;
5. The team clearing errors is the same team developing current release;
6. Increasing the team size leads to larger test-and-fix overheads (RTOh);
7. All detected errors in current release (EFx(i)) must be cleared before the next release;
8. There is no new error generated (badfix) during defect correction;
9. The defect fixing rate (RNF) is a multiple of the productivity rate (PDFx), the work force size (WF) and a multiplier (m_f) which decreases after the switchover.

Here, we focus on the '*new*' and '*old*' errors in test-and-fix process model (Figure 7.4), as they are associated with different probabilities (defect hitting rate) and efforts to detect them, which ultimately influence the performance of the testing process.



TS(i): Test suite for increment i
RTc: Testing rate by test case
m_f: Multiplier for fixing switchover
PDFx: Productivity for error fixing
RTOh: Error fixing overheads
RNF: Normal error fixing rate
RFx: Error fixing rate
EAND: Detected active new errors
EAOD: Detected active old errors
RPND: Passive new error detection rate
RAND: Active new error detection rate
EPI: Detected errors in pool to be fixed
RAOD: Active old error detection rate
RPOD: Passive old error detection rate
EPND: Detected passive new errors
EPOD: Detected passive old errors
EAD(i): Detected active errors in increment i
EPD(i): Detected passive old errors
EPEs(i): Escaping passive errors from increment i
EAEs(i): Escaping active errors from increment i
EP(i): Passive errors available in increment i
EA(i): Active errors available in increment i
EFx(i): Errors fixed in increment i

Figure 7.4: Qualitative model of test-and-fix process in one increment

The model can have multiple exits when performing simulation. This process is not completed until all detected defects are fixed, or a required percentage of test cases are passed within a desired period, and so on.

Model Execution

These two models connect each other iteratively to model the entire incremental development. Four QDEs were coded to represent the above qualitative models: QDEA for implementation process model, and QDEB, QDEC, and QDED for test-and-fix process model. Figure 7.5 shows the connections among these QDEs. This formal representation is required for the further execution of the qualitative model.

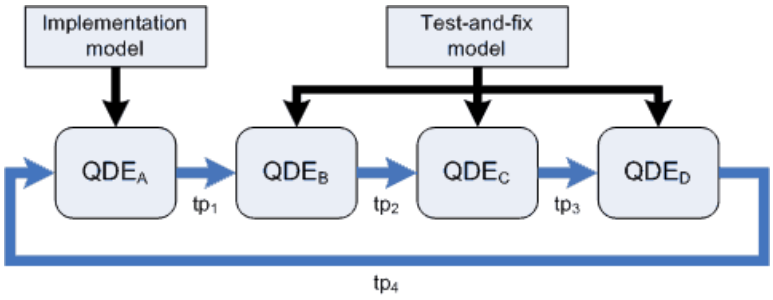


Figure 7.5: Model transitions and iterations

These QDEs are connected with four transitions: *Transition 1* (tp_1), when the implementation phase finishes; *Transition 2* (tp_2), when the accumulated detected defects reach the threshold that triggers bug fixing; *Transition 3* (tp_3), when the average length of defect queue for developers drops to the switchover point; *Transition 4* (tp_4), when detected defects are fixed, and the current release is done, it further ignites the next increment.

7.3 Semi-Quantitative Constraints

This section shows how quantitative constraints can be added to the qualitative models to obtain the corresponding semi-quantitative models.

7.3.1 Quantifying Implementation Process

As indicated in Figure 7.3, there are six monotonic functions in the qualitative implementation model. Thus, we need to specify the envelope functions for them separately.

To simplify the discussion based on the above-mentioned assumptions, linear relationships are assumed between software development rate and error generation rate, between residual active errors and retired errors, and between old active errors and propagated errors (indicated by dashed line in Figure 7.3). Therefore, some auxiliary parameters need to be introduced with the quantitative constraints: error densities (EDAt and EDPt) in Portion 1, error retirement rate (ERRt) in Portion 2, and error reproducing rates (REAR and REPR) in Portion 3 (Figure 7.6). The discussion on the relation between team size (WF) and overheads (ROh) can be found in Chapter 6.

Based on the above assumptions, the envelope functions for these monotonic constraints can be converted to the assignment of value ranges. For a ten-increment project, with reference to [AHM91], active error retirement ratio can be quantified by the value ranges in Table 7.3.

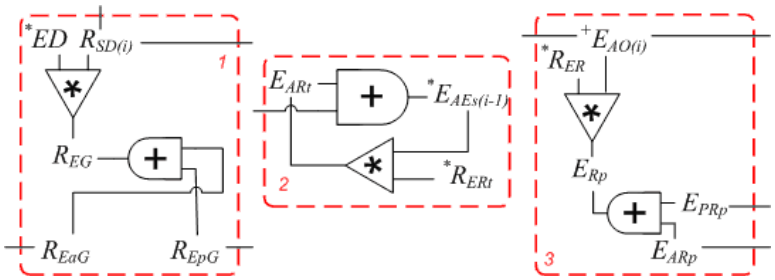


Figure 7.6: Refinement of implementation process model

Table 7.3: Value ranges for active error retirement ratio

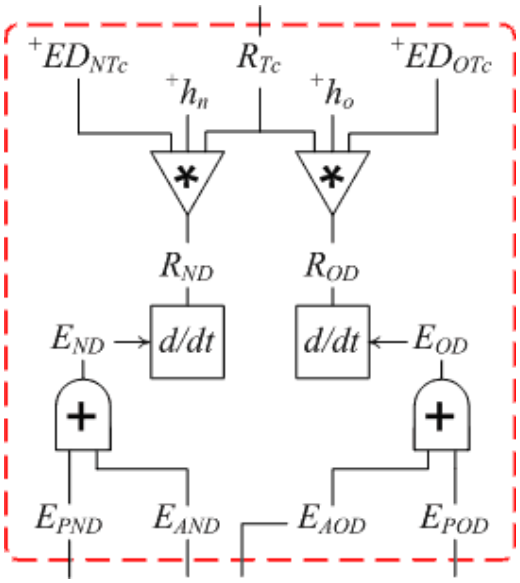
Increment	1,2,3	4,5,6	7,8	9	10
RERt	[0, 0]	[.05, .15]	[.2, .4]	[.5, .9]	[1, 1]

7.3.2 Quantifying Test-and-Fix Process

In the qualitative test-and-fix model (Figure 7.4), four types of errors, the combinations of new-old and active-passive error types, are modeled separately. The new errors and old errors are associated with different hitting rates (Assumption 3, Section 7.2.2) in testing. Nevertheless, the qualitative model provides no indication of this difference, which needs to be quantified here.

The error detection rate may change across the test cases. Its value highly depends on the applied testing strategy, the design of the test case, and the portfolio of test suite. In the context of quantitative constraint, we postulate that the hitting rates do not change across the testing cases. Then two hitting rates, new error hitting rate h_n and old error hitting rate h_o ($h_n > h_o$), and corresponding error detection rates can be used to specify the relations (Figure 7.7). Their value ranges can be calculated based on history records, such as [.85, 1.0] (h_n) and [0, .1] (h_o) of nominal detecting rate.

The estimate of multiplier (m_f) is based upon the time period when the average length of a developer's error queue falls below one particular value. Huff *et al.* argued that typically a developer spends one quarter time waiting for a defect to arrive, and three quarters handling a single defect in queue [HSS86]. However, this value is difficult to be directly observed and measured. It is therefore more sensible to define the range of value, like [.75, .85], for a general context.



EDNTc: Detected new errors per test case
EDOTc: Detected old errors per test case
h_n: Hitting rate of new errors
END: Detected new errors
EOD: Detected old errors
h_o: Hitting rate of old errors

Figure 7.7: Refinement of test-and-fix process model

7.4 Case Study: Incremental Development

7.4.1 Qualitative Simulation

Given the qualitative assumptions and QDE models, no specific numeric values are needed to perform qualitative simulation. We can simulate one intermediate increment by assuming the volume of residual active and passive errors (EAEs, EPEs) from previous releases qualitatively. The simulation generates 72 possible qualitative behaviours for this intermediate increment. Figure 7.8 shows part of the behaviour tree containing Behaviour 1 to 36. One example behaviour (Behaviour 26 of 72) depicts the trends and states of four key variables in Figure 7.9. The difference between the first half of behaviours (Behaviour 1 to 36) and the second half (Behaviour 37 to 72) is whether the residual active errors from previous release(s) are partially retired or entirely retired. In terms of the difference of this qualitative initial condition, the behaviour tree is branched at its root. The four transition points ('⊕') through each course indicate the landmarks of one increment process (see Figure 7.5).

Most of the qualitative behaviours observed by simulation represent the complicated relationships in the variable space, including EAEs vs. EAN, EPEs vs. EPN, EEs (EAEs + EPEs) vs. TS, RFx vs. RTc, etc. For example, the value of EAEs can be increased due to a low active error retirement rate RERt and/or a high active error generation rate (REAG) and/or a high active error propagation rate (REAR),

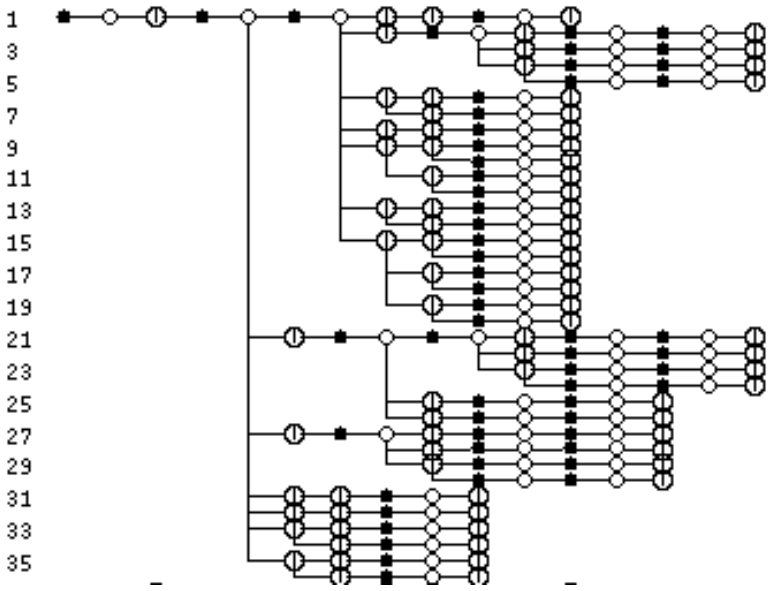


Figure 7.8: Part of behavior tree from qualitative simulation

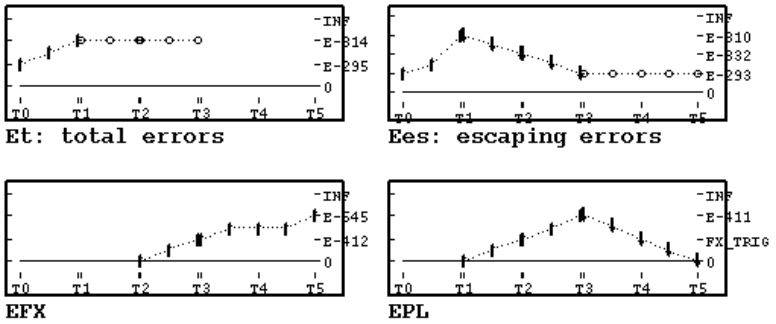


Figure 7.9: Behavior 26 of 72

and so on.

7.4.2 Semi-Quantitative Simulation

Baseline Project

A baseline project is needed in order to assess the performance of the semi-quantitative model by comparison. When using the qualitative simulation approach, a set of qualitative behaviours is generated, which are difficult to compare with one specific project. The baseline project for quantitative comparison is derived from data reported by Tvedt [Tve96]. The characteristics of this project are given in Table 7.4. The requirements size of the baseline project was 80,000

Table 7.4: Characteristics of the baseline project

Attributes	Values
Project size	90,667Loc
Number of increments	3
–Increment 1	26,667Loc
–Increment 2	32,000Loc
–Increment 3	32,000Loc
Project schedule	250days (1year)
Project team size	15 experienced engineers
Estimated budget	3750 man-days

lines of code written in COBOL. As the incremental development is applied in this project, it causes 10,667 lines of code as the estimated development overheads, which results in the total project size of 90,667 lines of code. The project was scheduled to last one calendar year, with a workforce level of 15 full-time engineers from start to finish.

Simulation Results

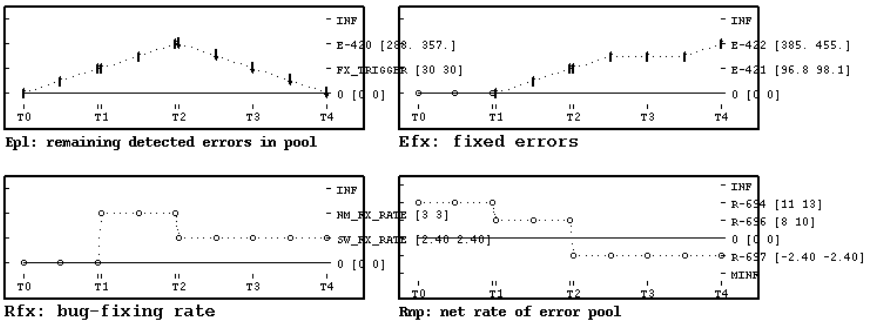
By applying the semi-quantitative incremental development process model (qualitative model and quantitative constraints) on the baseline project, the simulation is able to generate one semi-quantitative behaviour for each increment. The behaviours of some key variables are plotted in Figure 7.10.

Comparison with Baseline Project

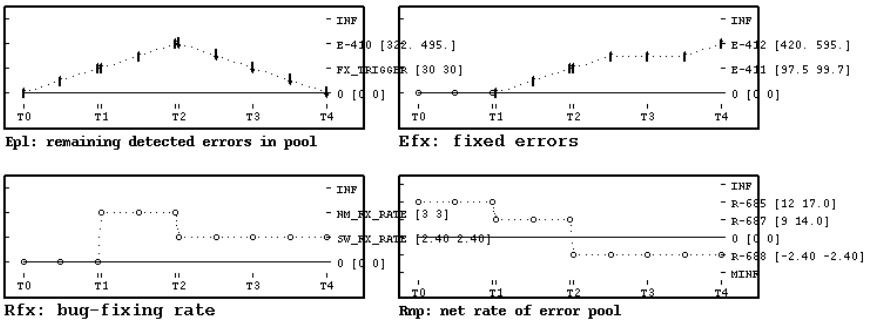
Tvedt's model used the project shown in Table 7.4. The variables in test-fix process are also affected by auxiliary factors from other related sectors, such as human resource. The semi-quantitative model treats different types of defects separately, gains more insights into error propagation across increments and allows some ignorance of the impacts from other sectors.

In contrast to this model, Tvedt's model simulated the incremental development with concurrency that allows overlapping between increments. Thus, it is not possible to compare the elapsed times with it. Here, only the simulated quality features (shown in Table 7.5) are presented for the brief comparison . The semi-quantitative model estimated that the project could produce the software product with no more than 414 defects, which is consistent with the simulation result generated from Tvedt's model.

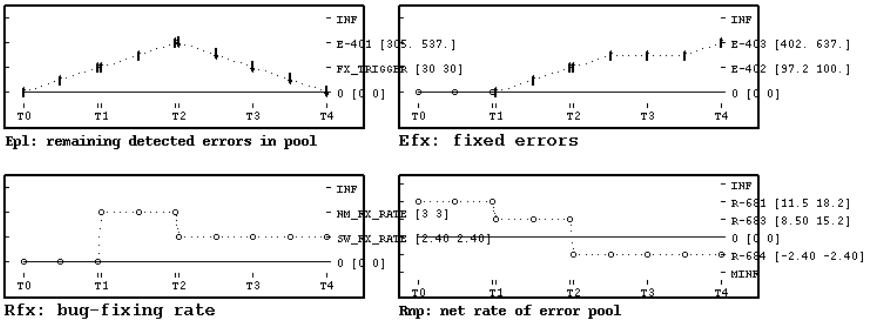
Note that the error value ranges extend significantly during Increment 2. It is mainly caused by the increased uncertainty of active error generation ratio



(a) behaviour of increment 1



(b) behaviour of increment 2



(c) behaviour of increment 3

Figure 7.10: Semi-quantitative behaviours of baseline project's test-and-fix process

Table 7.5: Simulated defect levels for the baseline project

	E_N	E_{Fx}	E_{Es}	Density
Increment 1	[480, 533]	[385, 455]	[25, 148]	≤5.5/KLoc
Increment 2	[598, 781]	[420, 595]	[3, 361]	≤6.1/KLoc
Increment 3	[576, 816]	[402, 637]	**[0, 414]**	**≤4.2/KLoc**
Tvedt's model			**279**	**3.5/KLoc**

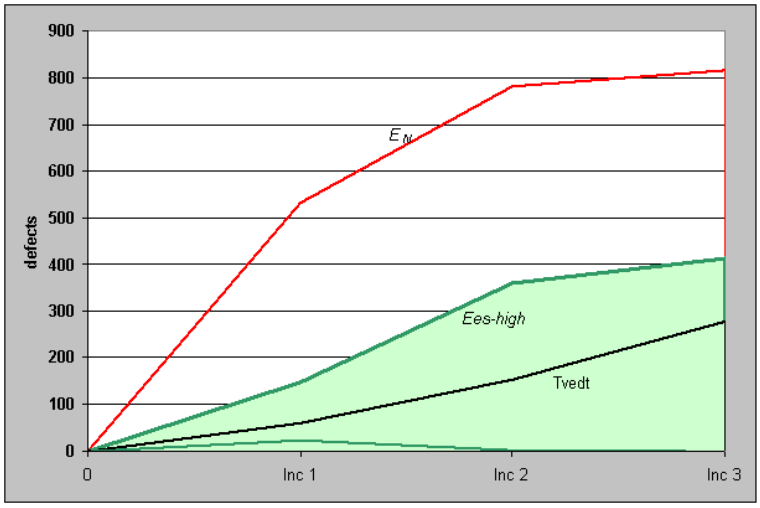


Figure 7.11: Simulated defect levels for the baseline project

Table 7.6: Value ranges of RAEG for 3 increments

Increment	1	**2**	3
rRAEG	[.95, 1.0]	**[.2, .9]**	[0, .25]

(rRAEG[†], Table 7.6). As there are only three increments in the baseline project, it leads to the difficulty in estimating the ratio between RAEG and RPEG, which might be simply assumed in purely quantitative modelling.

The quantitative constraints that were applied in this semi-quantitative model are mostly defined based on literature and experience data, which have yet been completely calibrated with the baseline project. In this example, only the baseline project information is used to specify the initial state (initial value ranges of input variables) and control the simulation (transitions and iterations). The simulation results demonstrate that this approach can produce reasonable prediction ranges (shadow area in Figure 7.11) when the specific process information is incomplete or uncertain.

7.5 Summary

This chapter further investigates the qualitative and semi-quantitative approaches for modelling and simulating software processes, but at a fine-grained level, and

[†]rRAEG = RAEG / (RAEG + RPEG)

reports a study of modelling incremental development focusing its test-and-fix process.

The qualitative and semi-quantitative models of incremental development processes discussed in this chapter provides a solid demonstration and possibility in modelling at a finer-grained (phase) level than higher project life cycle level (in Chapter 6).

The model structure described in this chapter can be further transformed into quantitative forms of simulation modelling. For example, given more details of specific process definition and policy (cf. [ZJZ08]), a hybrid simulation model of incremental development, which integrates SD and DES paradigms horizontally, was developed for investigating the possible process change in test-and-fix process on *micro-process* level.

The next chapter describes a comparison and conversion from a continuous quantitative modelling approach (system dynamics) to qualitative and semi-quantitative modelling, and vice versa. It also tests the capability of qualitative and semi-quantitative approaches in modelling a long-term software evolution process.

Chapter 8

Quantitative vs. Qualitative/Semi-Quantitative Process Simulation

Chapter 6 and 7 explore qualitative/semi-quantitative modelling of software processes at *project* and *phase* level. This chapter continues to follow the design of research (cf. Section 3.6), but develops a long-term software *evolution* process model for simulation using the approaches. Table 8.1 highlights the outstanding characteristics (in bold), identified in the research design, for specifying the modelling and simulation study in this chapter.

8.1 Reference Model Selection

In addition to the modelling of software evolution process, this chapter also develops and implements an element level mapping for converting a typical quantitative simulation model into an equivalent qualitative model, and moreover, compares the model structures and outputs of quantitative simulation against its

Table 8.1: Evolution process in relation to research design

Purpose level	Problem domain	Model scope	Output variable
~~cognitive level~~	~~generic development~~	~~single phase~~	~~time~~
~~tactical level~~	**software evolution**	~~multi-phase~~	~~effort~~
strategic level	~~incremental development~~	~~project~~	~~quality~~
		evolution	**size**

counterparts, the qualitative and semi-quantitative models. To achieve this goal, a typical quantitative software process simulation model has been selected as the reference model in this study for model conversion and comparison.

The selection of reference model is the first and important step in this study. Besides the characteristics of this model defined in Table 8.1, the selection criteria have to include two extra aspects:

Paradigm The reference model should be constructed with a typical quantitative simulation modelling paradigm, which has been widely accepted by SPSM research. According to the systematic literature review described in Chapter 3, system dynamics (SD) and discrete-event simulation (DES) are the two most widely used techniques out of ten simulation paradigms in SPSM. Both of them are quantitative. On the other hand, qualitative and semi-quantitative simulation (QSIM/SQSIM) are one type of continuous simulation with the incomplete feature of discrete transition. From the point of comparability, SD becomes the selected modelling paradigm due to its wide use in SPSM and continuous characteristic.

Complexity The reference model needs to contain most common elements and relations of the chosen software evolution process. However, the structure complexity of the reference model should be clear and simple enough to ensure the emphasis of this research on model conversion and comparison, rather construction of a complicated model.

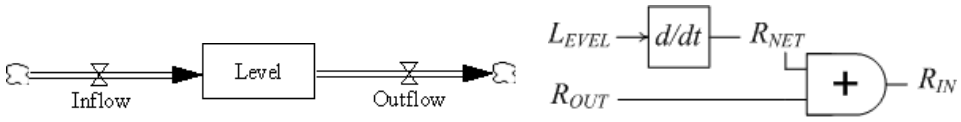
According to the research design and the above criteria of the reference model, an SD model of software evolution process is the first choice for this study. There are several candidate models published in the last decade available in [WL99, CLRW00, KLRW01, WH02, WH04]. Among them, Wernick and Hall's model [WH04] consists of a single module, whose structure is simpler than others'. Moreover, their model is the most recent SD model of evolution process in the primary studies of the systematic review.

8.2 Model Conversion

In Chapter 3, system dynamics, as a typical continuous quantitative simulation technique, has been briefly introduced. This section presents the elements of SD modelling, and describes how to convert them for qualitative and semi-quantitative modelling.

8.2.1 Causal Loop Diagram

In SD several modelling components and tools are used to capture the structure of systems, including causal loop diagram (CLD), level and rate, and delay. Among



(a) System Dynamics (b) Qualitative Modelling

Figure 8.1: General structures of level and rate

them, CLD is well suited to represent interdependencies and feedback processes. A CLD consists of variables connected by arrows denoting the causal influences among the variables. Each causal link (arrow) might be assigned a polarity, either positive (+) or negative (-) to indicate how dependent variable changes when the independent variable changes. The causal links are quantified in SD, but CLD does not reflect such quantification.

In qualitative diagramming, i.e. *abstract structure diagram* (ASD) in QSIM (cf. Chapter 4), the notations and links are more explicit and clear. Sum and product relations are explicitly represented by `add` (or `sum`) and `mult` identifiers. Other basic arithmetic relations, e.g. subtraction and division, can also be derived from them. The complicated or unknown positive (+) and negative (-) dependencies can be denoted as the monotonic increasing `M+` and decreasing `M-` functions in qualitative modelling (see Chapter 4). Therefore, a CLD can be converted into the corresponding qualitative diagram, but needs more explicit and specific qualitative assumptions.

8.2.2 Level & Rate

Level (or stock) and rate (or flow) are the central concepts of system dynamics. Levels are absorbing inflow rate, and accumulating the difference between the inflow to a process and its outflow. SD uses a particular diagramming notation for stocks and flows (Figure 8.1-a). Valves control the flow rates. Clouds represent the sources and sinks for the flows, which are both outside of the model boundary.

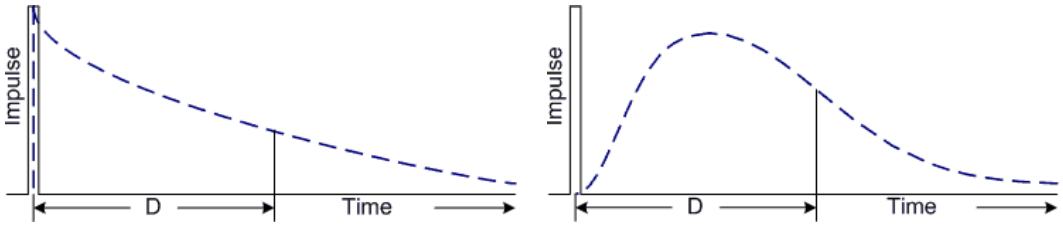
The structure represented in Figure 8.1-a corresponds exactly to the following integral equation:

$$Level\,(t) = \int_{t_0}^{t} [inflow\,(s) - outflow\,(s)]ds + Level\,(t_0) \tag{8.1}$$

Equivalently, the net rate of level change, its derivative, is the inflow less the outflow, defining the differential equation:

$$\frac{\partial}{\partial t}\,(Level) = inflow\,(t) - outflow\,(t) \tag{8.2}$$

Hence, this dynamics of systems can be modelled by *differential* relation (described in Chapter 4) in ASD. Unlike SD diagramming, the rate difference (net



(a) first-order delay (b) third-order delay

Figure 8.2: Exponential delay curves

flow) has to be explicitly presented in qualitative diagram. Figure 8.1-b shows the converted level and rate in qualitative modelling diagram.

8.2.3 Delay

Delay in System Dynamics

Forrester identified two characteristics of a delay [For69]. One is the length of time expressing the average delay D, which fully determines the "*steady-state*" effect of the delay. In steady state the flow rate multiplied by the average delay gives the quantity in transit in the delay. The other describes its "*transient response*", which tells how the time shape of the outflow is related to the time shape of the inflow when the inflow rate is changing over time. The delays with the same average delay time (D) can have quite different transient responses to changes in input rate (like plot a and b in Figure 8.2).

Exponential delay is the most frequently used delay class in SD. Figure 8.2 shows two common types of exponential delay: first-order delay (a), and third-order delay (b). Mathematically speaking, an n^{th}-order delay is equivalent to n cascaded single-order delays, with each single-order delay having a delay time of $\frac{D}{n}$ [For69].

Since time is treated qualitatively in qualitative simulation, there is no meaning to handle a delay with "*qualitative*" length. Therefore, QSIM algorithm does not explicitly consider delay phenomenon, and neither include built-in function of delay. As semi-quantitative simulation offers the capability of dealing with numeric values in reasoning, the delay function is necessary to maintain the integrity during model conversion. Unfortunately, in terms of the author's knowledge so far, there is no such function built in the available simulation tools or packages. The objective of this section is to implement the (exponential) '*delay*' functions using QSIM algorithm framework.

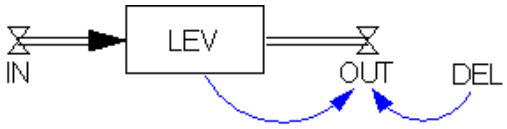


Figure 8.3: First-order exponential delay in SD

Table 8.2: First-order exponential delay

	$OUT_{[i, i+1]} = DELAY1\ (IN_{[i, i+1]}, DEL)$
	$OUT_{[i, i+1]} = LEV_{[i]} / DEL$
	$LEV_{[i]} = LEV_{[i-1]} + (DT)(IN_{[i-1, i]} - OUT_{[i-1, i]})$
$OUT_{[i, i+1]}$	the outflow rate between time i and i+1
$LEV_{[i]}$	the level stored for delay at time i
DEL	the average delay time
DT	the time step between successive evaluations of equation
IN	the inflow rate between time i-1 and i

First-Order Delay

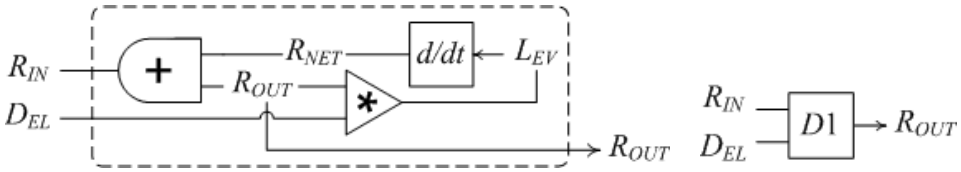
First-order and third-order exponential delays are two of the most common delays used in the SD models of software process. Figure 8.3 is a first-order delay presented in SD. Given an exogenous inflow rate (IN from other part of system), a first-order delay consists of a simple level (LEV) and a rate of outflow (OUT) that depends on the level and on the delay time (DEL).

Table 8.2 shows the mathematical equations of fist-order delay. The outflow rate (OUT) is equal to the level (LEV) divided by the average delay (DEL).

A first-order delay is composed of four model elements (Figure 8.3): one level, two rates, and one auxiliary variable (*delay*). Following the model element conversion discussed in previous sections, the structure of first-order delay is converted and represented with ASD notations (Figure 8.4-a). A new ASD notation (Figure 8.4-b), with two inputs (*inflow* and *delay*) and one output (*outflow*), is created to abstract this structure and avoid the redundant complexity of qualitative model. Figure 8.5 shows the corresponding qualitative constraints in QDE (LISP).

Third-Order Delay

A third-order delay is the equivalent of three first-order delays cascaded on one after another, so that the output of the first is the input to the second, and the output of the second is the input to the third. Figure 8.6 illustrates the structure of a third-order delay in SD format.



(a) implementation (b) notation

Figure 8.4: Implementation of first-order delay in ASD

```
(constraints
  (add      Rnet Rout Rin)
  (constant Del)
  (d/dt     Lev Rnet)
  (mult     Rout Del Lev))
```

Figure 8.5: Implemented constraints of first-order delay in QSIM

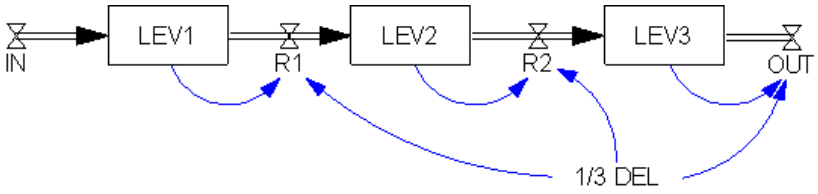


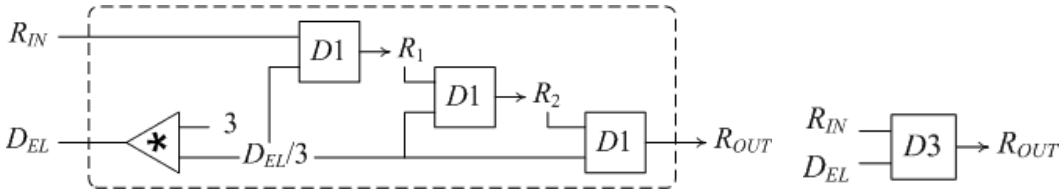
Figure 8.6: Third-order exponential delay in SD

Table 8.3 shows the equations for calculating a third-order delay. The outflow rate (OUT) is equal to the level (LEV) divided by the average delay (DEL).

By using the new ASD notation created for first-order delay (Figure 8.4-b), the structure of a third-order delay can be represented in Figure 8.7-a. Again, another new ASD notation (Figure 8.7-b), with two inputs and one output, is

Table 8.3: Third-order exponential delay

$OUT_{[i,\ i+1]} = DELAY3\ (IN_{[i,\ i+1]},\ DEL)$
$R1_{[i,\ i+1]} = LEV1_{[i]} \ /\ (\frac{1}{3}DEL)$
$LEV1_{[i]} = LEV1_{[i-1]} + (DT)(IN_{[i-1,\ i]} - R1_{[i-1,\ i]})$
$R2_{[i,\ i+1]} = LEV2_{[i]} \ /\ (\frac{1}{3}DEL)$
$LEV2_{[i]} = LEV2_{[i-1]} + (DT)(R1_{[i-1,\ i]} - R2_{[i-1,\ i]})$
$OUT_{[i,\ i+1]} = LEV3_{[i]} \ /\ (\frac{1}{3}DEL)$
$LEV3_{[i]} = LEV3_{[i-1]} + (DT)(R2_{[i-1,\ i]} - OUT_{[i-1,\ i]})$
$LEV_{[i]} = LEV1_{[i]} + LEV2_{[i]} + LEV3_{[i]}$



(a) implementation (b) notation

Figure 8.7: Implementation of third-order delay in ASD

created to abstract this more complicated structure.

Here, the implemented third-order delay also demonstrates how to construct an n^{th}-order delay by using the basic first-order delay in QDE (based on QSIM algorithm framework).

8.3 Reference System Dynamics Model

According to the selection criteria identified and discussed in Section 8.1, Wernick and Hall's SD model [WH04] of software evolution process (published in ProSim 2004) is chosen as the reference model in this study. This section introduces the model background, software evolution processes, and replicates this model for further comparison.

8.3.1 Software Evolution Process

With respect to the results of the systematic review (Chapter 3), software evolution process is one of the most investigated software processes in SPSM. The insights obtained from previous studies indicated that software evolution could be systematically studied and exploited using SPSM approaches. They also suggested that to some extent software evolution is a disciplined phenomenon as illustrated, for example, by the regularity of functional growth patterns [LR03]. Models of such patterns permitted the forecasting of future overall system growth and growth rates. Moreover, the observed patterns of behaviour appearing yielded common phenomenological interpretations.

Feedback Hypotheses in Software Evolution

Basically, four important feedback structures, identified by previous studies, are used in model construction of software evolution processes.

Inertia-like (*anti-regressive*) effect due to system growth The first hypothesis is that increasing the size of a software system and changes in unanticipated directions will over time reduce the enhancement and modification of that system [WH02]. The increasing size and complexity of the software system as its

structure is degraded by efforts to make changes unanticipated when the system architecture was designed. These changes may result in a decay in software architecture. Meanwhile, new changes also have to be fitted into an existing system structure, and as the software grows, there are more existing components into which each new change needs to be fitted. Thereby, software developers are occupied on tasks specifically intended to maintain the system structure, and to compensate for the software ageing effects, which are referred to as '*anti-regressive*' activities [KLRW01].

Effects of decreasing knowledge coverage The increasing complexity of a software system also reduces the developer's ability to change the system because of a decrease in coverage of developer's knowledge of the system components, their composition and interactions [WH02]. As the software grows, the amount of knowledge needed to support future changes grows as well, but at a faster rate, as the implementation of each new component of the software needs to be seen in the context of *all* of the existing system [Tur96, Tur02]. If the developer's knowledge does not grow at this rate, it may fall behind the knowledge needed to support further changes.

Generation (*progressive* effect) of new requirements The release of upgraded software with new functionalities enables users to exploit opportunities for novel or extended system use, which in turn result in demand for further functionalities [CLRW00]. This positive feedback is recognised as '*progressive*' type of work, which represents the evolution activities that enhance software functionality by modification of or addition to the code and/or the documentation [KLRW01].

Correction of fault implementation After the release and adoption of software, a small proportion of units (requirements) is gradually found not to be implemented as originally or correctly specified. They are eliminated from the specification, but may be replaced in the requirements by new or changed equivalents [WL99]. The rate of completion of successful implementation can be reflected by a *success* or *failure* percentage.

Each of these hypothetical drivers of specification change is reflected in (positive or negative) feedback loops in SD modelling, again calculated as portions of the successful implementation flow.

8.3.2 A Simplified Model of Software Evolution

Model Description

The reference model (shown in Figure 8.8) was developed using Vensim simulation environment (from Ventana Systems, Inc.). Although it is a simplified model, it

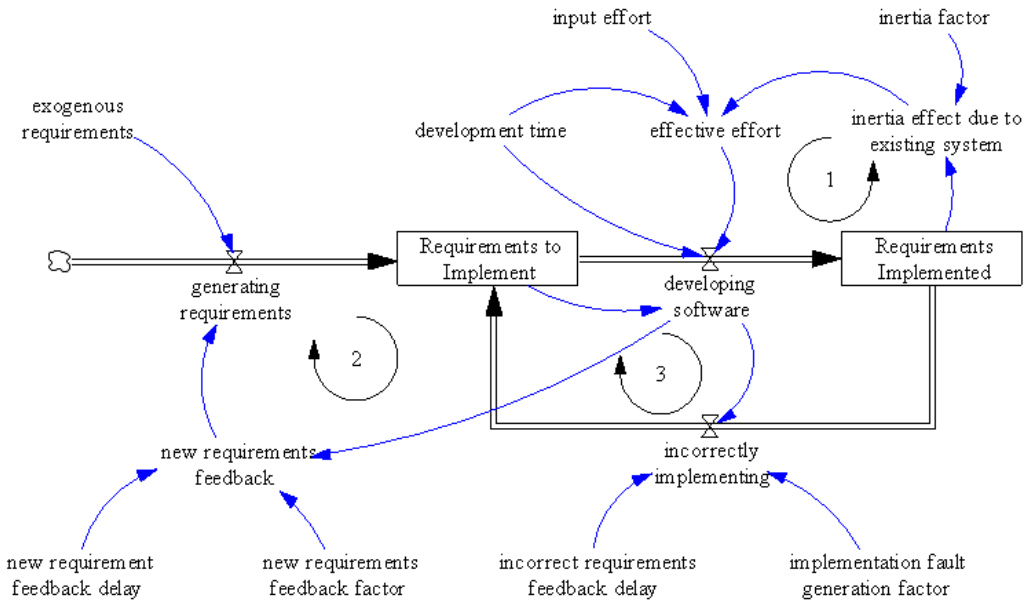


Figure 8.8: The reference SD model of software evolution

incorporates and reflects three of the typical feedback loops described in the last subsection that are indicated by the loop numbers in the model. They are feedback structures representing the system inertia effect (anti-regressive activities, Loop 1), the generation of new requirements (progressive activities, Loop 2), and the correction of faults in previous implementation (Loop 3).

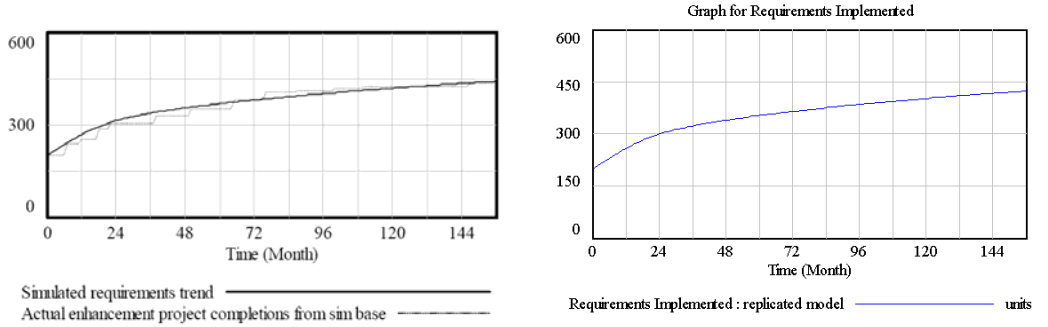
The '*size*' of software system has been abstracted into a number of arbitrary-sized '*units*' (or '*modules*') of requirements, since it is a more informative reflection of software evolution, which is more likely driven by changes in functionality than by low-level thinking with '*code*' [Tur96]. Plus, it avoids issues related to specific metric of code size.

The delay used in the reference model is a *third-order* delay, which fits the technical software process [WL99]. It represents the time delays caused by some entity passing through the phases of a process made up of a sequence of sub-processes, each of which depends for its input on the output of the previous sub-process.

Model Calibration

The calibration inputs to the reference model are based on actual data for the evolution of the ICL VME mainframe operating system as described in [CLRW00]. To guarantee the replication of the reference model, most of these variable inputs are kept in this chapter.

Neither Wernick and Hall's SD model nor Turski's reference model [Tur02]



(a) system size simulated by the reference model

(b) system size simulated by the replicated model

Figure 8.9: Simulation of implemented requirements over time

explicitly quantified the linear coefficients for *inertia effect due to existing system*. Therefore, during the reconstruction of the reference SD model, it is assumed that the inertia effect starts at 1 when simulation starts, then decreases as the inverse cube of the system size (*Requirements Implemented*).

Note that the replicated reference model in this chapter is based on the SD flow graphs, inputs, outputs, and relation equations published in [CLRW00, KLRW01, WH02, WH04] (no full version models published). As result of this divergence, the output of the replicated model is slightly different from the reference model. Figure 8.9 shows the overall evolution trends are similar to each other, and the only difference on simulated system sizes, which can be regarded as the scaling effect of *inertia factor*.

To maintain the completeness, the new requirements generated from exogenous events (*exogenous requirements*) are included in the model. But they were set to 0 in Wernick and Hall's study.

Sensitivity to Policy Change

The reference model has been subjected to a further sensitivity analysis to investigate the effects of changes in policy inputs. Wernick and Hall introduced five policy factors to the reference model, which are underlined in Figure 8.10. Sensitivity analysis of these factors have therefore been undertaken. Vensim allows Monte Carlo simulations by varying the values of one or more input variables in terms of probability distributions. For this study, Wernick and Hall varied each of the policy input from its default value of 1, using a normal distribution with a standard deviation of 0.25. For instance, the values greater than 1 of '*inertia scaling policy*' indicates the higher maintainability and evolvability of the system.

A similar sensitivity analysis design was applied to the replicated model. To simplify the discussion, three of their five policy factors are selected to investigate

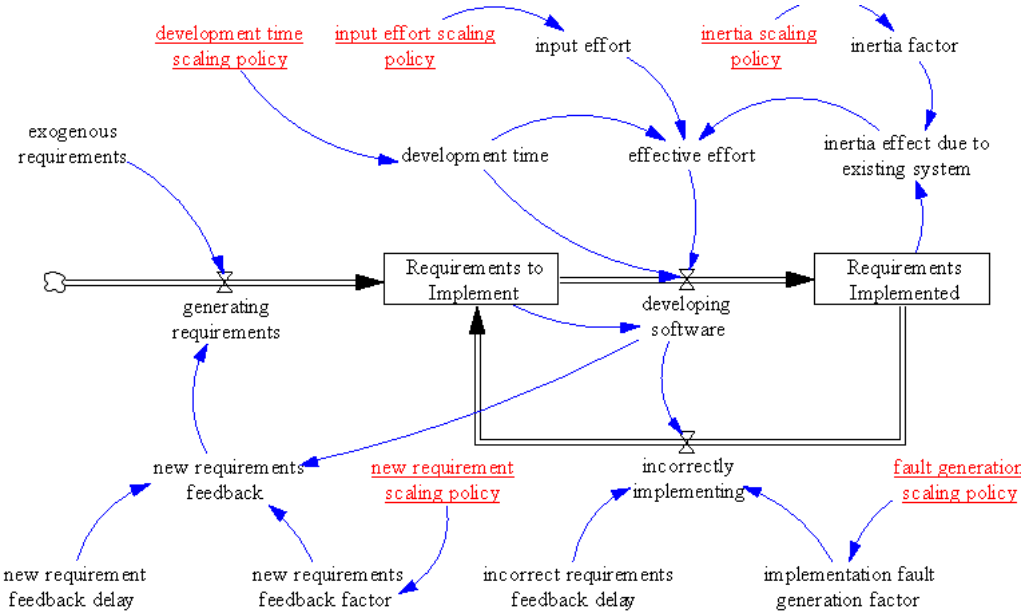


Figure 8.10: The SD model of software evolution for policy investigation

the policy sensitivity of three feedback loops respectively. Figure 8.11 shows the distributions of simulated system size growth and volume of requirements over time for 1000 runs for each parameter varied separately. The solid line in each plot indicates the mean result, and the regions either side of it contain 50%, 75%, 95%, and 100% of the simulated results respectively.

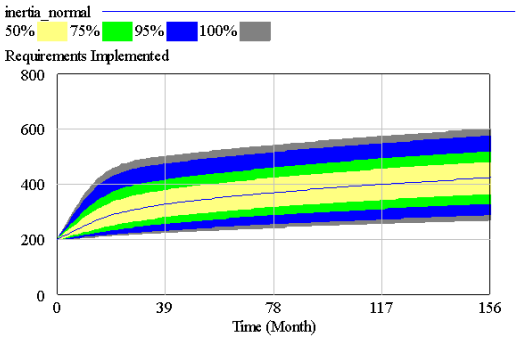
8.4 Corresponding Qualitative & SemiQ Models

8.4.1 Qualitative Model

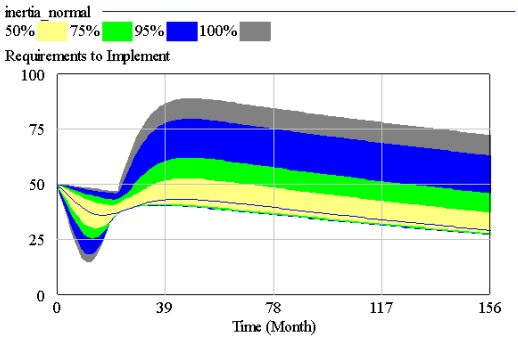
As the first step of qualitative modelling, the qualitative assumptions need to be abstracted from the real world system (Chapter 4). However, due to the quantitative (SD) reference model available, an inverse procedure has to be followed here. The SD model should be converted into qualitative model based on the discussion in Section 8.2. After that, the qualitative assumptions can be extracted from the corresponding qualitative model.

Figure 8.12 shows the corresponding qualitative model. As mentioned early, it is not necessary to model and simulate a delay with '*qualitative*' length. Thus, the qualitative model does not explicitly include the three delay relations in the reference model.

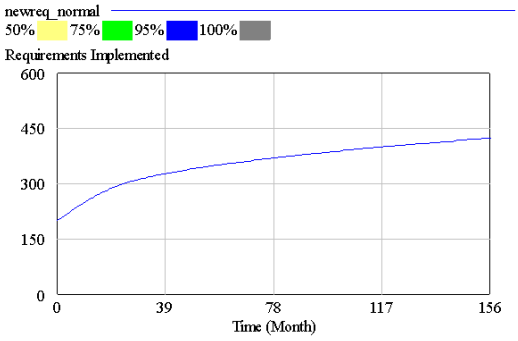
Based on the converted qualitative model, the inherent qualitative assumptions in the SD reference model can be revealed:



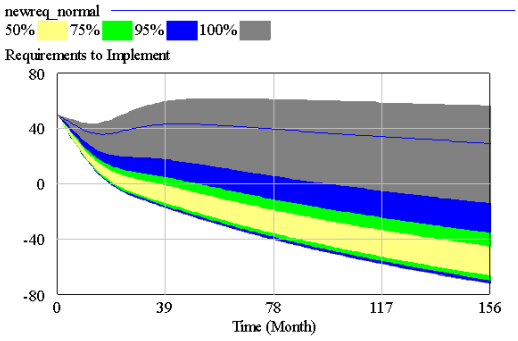
(a) Simp by *inertia*



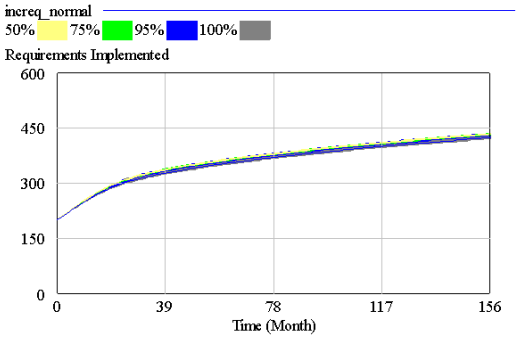
(b) Sreq by *inertia*



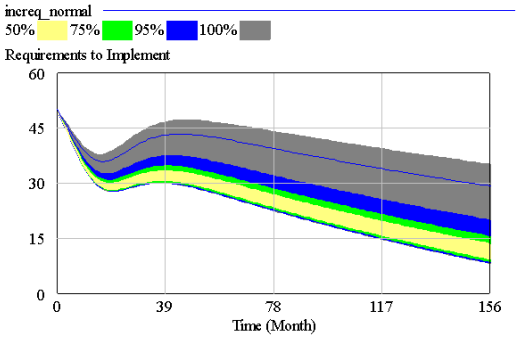
(c) Simp by *new requirement*



(d) Sreq by *new requirement*

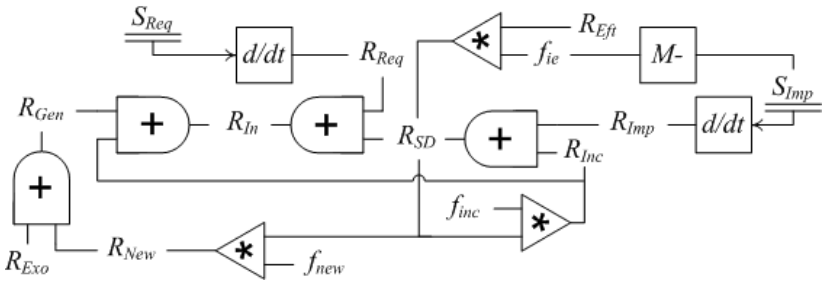


(e) Simp by *fault generation*



(f) Sreq by *fault generation*

Figure 8.11: Sensitivity of policy change for reference model



SReq: Requirements to implement
RReq: Requirement generation rate
RNew: New requirement feedback rate
RSD: Software development rate
RIn: Requirement input rate
REft: Effective effort rate
f_{new}: new requirement feedback factor

SImp: Requirements implemented
RImp: Requirement implementation rate
RInc: Incorrect implementation rate
RGen: Requirement generation rate
RExo: Exogenous requirement rate
f_{ie}: inertia factor
f_{inc}: fault generation factor

Figure 8.12: Qualitative model of software evolution

1. *Requirements to implement* (SReq) come from *exogenous requirements*, *new requirements feedback* and *incorrect requirements feedback*;

2. SReq is transferred to *Requirements implemented* (SImp) at *software developing* rate (RSD);

3. The *incorrectly implemented requirements*, as a small portion of SImp is returned to SReq for rework;

4. Increasing existing system size (SImp) incurs more effort needed for '*anti-regressive activities*', and decreases RSD;

5. The *input effort* (REft) has linear relationship with RSD;

6. The *new requirement feedback* (RNew) has linear relationship with RSD;

7. The *incorrectly implementation* (RInc) has linear relationship with RSD;

8. The development team size does not change (neither recruitment nor turnover) during the evolution process;

9. There is no *exogenous requirements* (RExo $= 0$) during the evolution process.

8.4.2 Semi-Quantitative Model

Figure 8.13 is the graph of qualitative model with more constraints introduced. To be noticed, the newly introduced notations representing the exponential delays (Section 8.2.3) are included in semi-quantitative constraints.

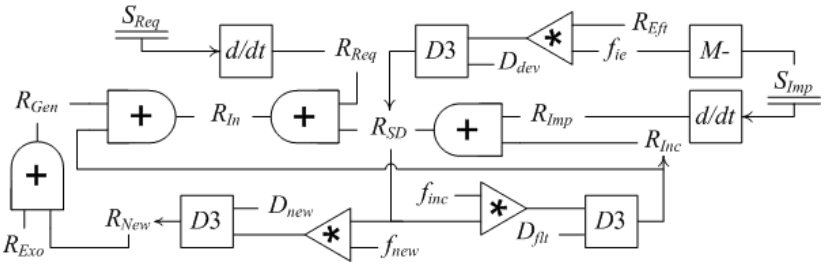


Figure 8.13: Semi-quantitative model of software evolution

One monotonic function (M−, between f_{ie} and SImp) is included in the qualitative model. This nonlinear relation needs to be quantified at this stage. Nevertheless, the inertia factor was quantified as a multiple of the inverse square [WH02] or inverse cube [WH04] of existing system size respectively. Therefore, an envelop function (Equation 8.3) is suitable for this case.

$$f_{ie} = \left[\left(\frac{\lambda}{S_{imp}^3} \right), \left(\frac{\lambda}{S_{imp}^2} \right) \right] \tag{8.3}$$

where λ is suitable constant, and determined from historic data.

8.5 Simulation Results Comparison

8.5.1 Qualitative Simulation

The reference model set the simulation terminated at predefined time point (the 156^{th} month). However, as time is treated qualitatively in the corresponding model, this termination condition does not work in QSIM. Moreover, the oscillation phenomena are observed in some simulated behaviour patterns. So the qualitative simulation cannot stop itself on these behaviours until runs out of memory. In experiment, the simulation was set to halt after generating a '*large*' number of behaviours, which is sufficient to observe the behaviour patterns.

The qualitative simulation generates a diversity of behaviours of the evolution process, most of which are the combinations of varying patterns of important variables. Figure 8.14 shows the typical behaviour patterns of some important variables in the model.

Requirements Implemented The simulated behaviour of SImp presents growing trend at all time (Figure 8.14-a), which is also quantitatively reflected in Figure 8.9 and Figure 8.11-a, c, e. It is because *developing software* is always greater than *incorrectly implementing*.

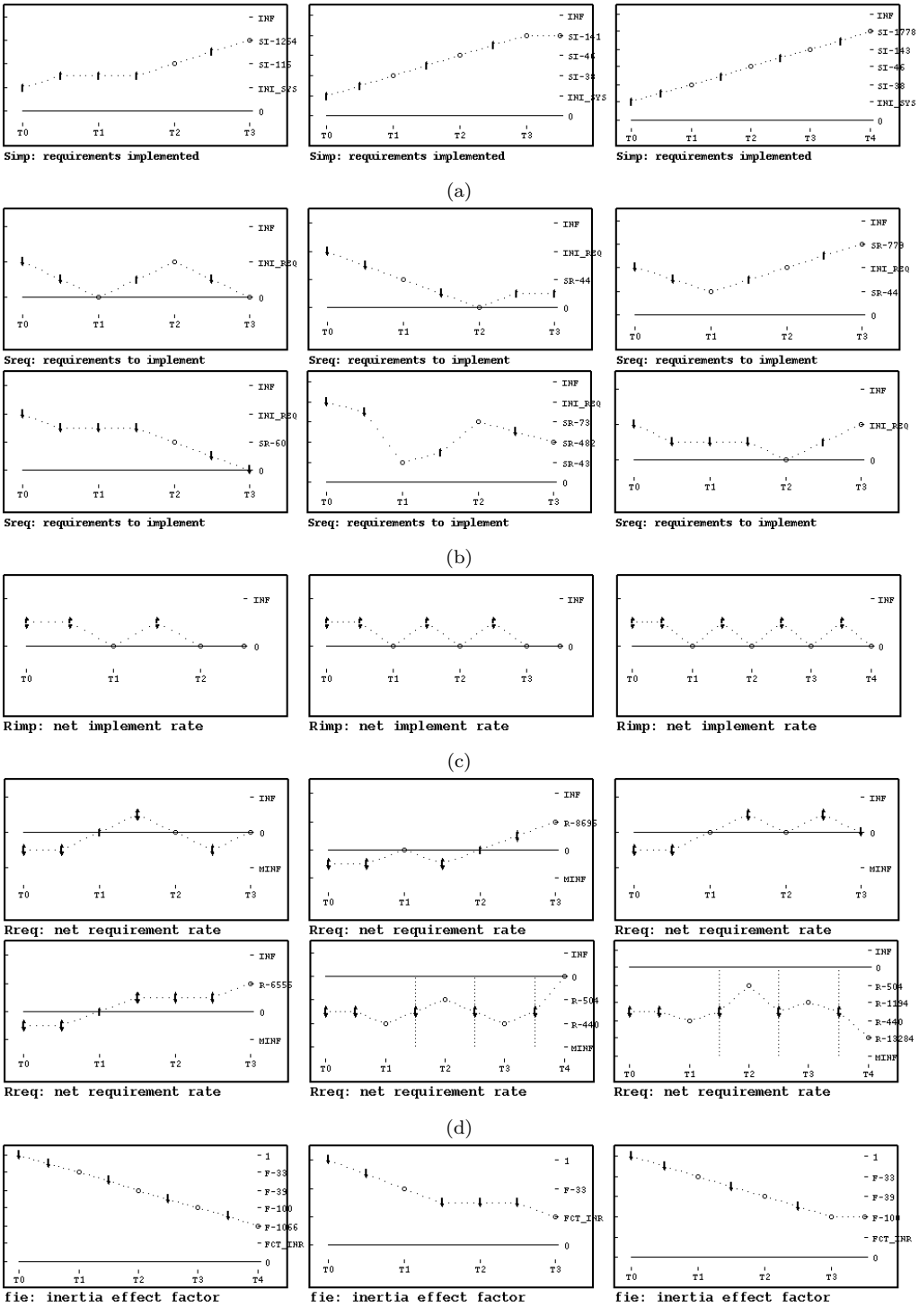


Figure 8.14: Behaviors of qualitative simulation

Requirements to Implement The possible changes of S_{Req} are more complicated than S_{Imp}. Overall, it may gradually drop (down to zero), then rebound to a certain level (Figure 8.14-b). The qualitative simulation reflects the result from reference model (Figure 8.11-b, d, f).

Requirement implementation rate It is easy to identify the oscillation of R_{Imp} from the simulated possible behaviour (Figure 8.14-c). The evolution process may reach the equilibrium state after one, two, three oscillations or more, even keep oscillating for ever, which is one of the main reasons for why the simulation cannot stop itself. The oscillation phenomenon is to a large degree consistent with the qualitative behaviours observed by Ramil and Smith's study in [RS02], which constructed qualitative simulation model based on analytic model, instead of continuous casual model in this chapter.

Requirement generation rate Figure 8.14-d reflects the '*unpredictable*' behaviour of R_{Req}. It may oscillate cross, over or below zero. Therefore, the quantitative constraints have to be applied for R_{Req} to generate more specific and stable behaviours.

Other variables, such as R_{SD}, R_{inc}, R_{new}, and f_{ie}, keep decreasing from the start of the simulation. In some cases, they can finally reach an equilibrium state (Figure 8.14-e).

8.5.2 Single-Point Value Simulation

The comparison between quantitative simulation (system dynamics) and semi-quantitative simulation can be conducted from two aspects: simulation with single-point value and value range. Traditionally, purely quantitative simulation always assign a single-point (numeric) value for each input variable during each run of simulation. In contrast, semi-quantitative simulation treats the value ranges as the single-point values in quantitative approach. To realise the single-point value simulation within semi-quantitative approach, the upper and lower bounds should be set as the same. In the other words, the interval must be zero, which presents the single-point value within value range format. Therefore, the input variables of semi-quantitative model are set with the same value as in the reference model (Section 8.3.2) for comparison.

On the other hand, the envelop function should be also replaced with the '*exact*' function to eliminate the uncertainty. Note that the '*exact*' function (Equation 8.4) is used to replace the '*envelop*' function (Equation 8.3) for comparison.

$$f_{ie} = \left[\left(\frac{\lambda}{S_{imp}^3} \right), \left(\frac{\lambda}{S_{imp}^3} \right) \right] \tag{8.4}$$

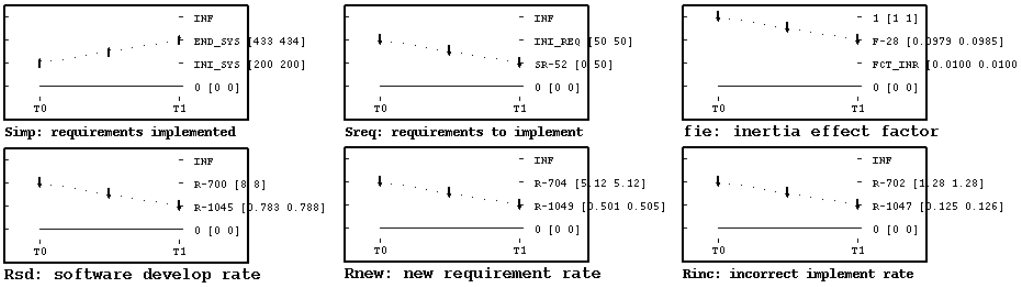


Figure 8.15: Behavior of semi-quantitative simulation with single-point values

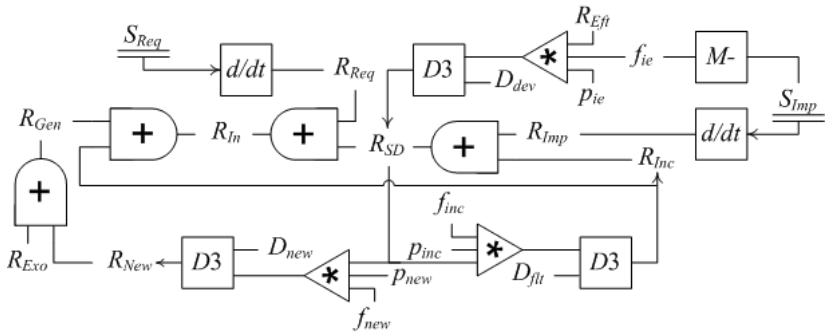
Different from purely quantitative simulation, the semi-quantitative simulation generates nine behaviours, even with single-point settings. Although the simulation is preset to terminate at the 156^{th} month (as the SD model), Q2 algorithm also includes some behaviours that may terminate in a range covering this value, such as [29.6, 156] months. It is because SQSIM is based on value range, rather than single-point value for purely quantitative simulation. Any behaviours covering this condition are generated by algorithm. For this comparison, we are only interested in the behaviours exactly terminated at the preset time point. By removing the '*invalid*' behaviours, there are two behaviours consistent to the reference model that terminates exactly at Month 156. Figure 8.15 depicts the varying trends of some important variables.

The only difference between these two behaviours is that S_{Req} may finish at zero or in the rang [0, 50] units when the simulation terminates. The simulation in Section 8.3.2 predicts that the system size may grows up to 433.75 units at the 156^{th} month. Both the valid behaviours from SQSIM produce the close value range, [433, 434] units, for S_{Imp}. Moreover, it is interesting that SQSIM presents other variables (e.g. f_{ie}, R_{SD}, R_{New} and R_{Inc}) in its inherent format (value range) as well. It is possibly caused by the slim deviation between the function and its inverse version, which are required for describing the envelop functions.

Compared with the simulation result of system dynamics (presented in Figure 8.9), the graphic result of semi-quantitative simulation only depicts the monotonic trend of S_{Imp}, but lacks detailed shape, which depends on the number of landmark created in the course of simulation. It can be enriched by inserting more landmarks (like Q3 algorithm).

8.5.3 Value-Range Simulation

In Section 8.3.2, several policy factors are introduced by Wernick and Hall for sensitivity analysis (Figure 8.10). This section emphasises on the value range comparison, between the Monte Carlo simulations of these inputs in terms of probability distributions and semi-quantitative simulations with the correspond-



p_{ie}: inertia scaling policy factor
p_{inc}: implementation fault feedback policy factor
p_{new}: new requirement feedback policy factor

Figure 8.16: Semi-quantitative evolution model with policy factors

Table 8.4: Value range comparison

	SImp by f_{ie}	SImp by RNew	SImp by RInc
SD	[270, 601]	[434, 434]	[424, 443]
SQSIM (Q2)	[220, 627]	[423, 525]	[410, 527]
SQSIM (Q3)	[263, 610]	[430, 455]	[420, 471]

ing value ranges. Therefore, the policy factors needs to be added in the semi-quantitative model as well. However, their probability distributions are not necessary in this form of simulation. Figure 8.16 shows the updated semi-quantitative model with the policy factors (p_{ie}, p_{new}, and p_{inc}).

The results of SQSIM are summarised in Table 8.4, compared with SD (the first row). The value ranges on the second row are the simulated results generated by normal QSIM+Q2 algorithm. Although they are consistent with SD results (covering the intervals of SD), it is easy to observe that they remain very coarse. It is mainly because the evolution behaviours (SImp) are monotonic and smooth, which, unlike the models described in previous chapter, include no transition points and few *critical* time points either. As a result, the landmarks inserted into qualitative intervals are not sufficient to generate finer ranges to reduce the uncertainty.

To improve the performance of SQSIM, Q3 algorithm, which extends Q2 with '*step-size refinement*' (cf. Chapter 5), is further applied to obtain finer-grained value ranges. The simulation results (corresponding to the step-size of 10) are listed on the bottom row in Table 8.4. It demonstrates that the accuracy of SQSIM can be improved significantly by adaptively introducing additional landmarks. The remaining difference between these two approaches is probably caused by *1*) the different reasoning mechanism of SQSIM that is based on the be-

haviour chattering technique instead of single-point calculation used in SD; *2*) the sampling and assumed probability distribution (in quantitative simulation) cause some missing points; *3*) step-size of Q3 has not been completely optimised. Along with the Q3 and other advanced refinement techniques (e.g. dynamic envelopes), semi-quantitative simulation can smoothly span the gap from qualitative simulation on the one hand to purely quantitative simulation on the other [Kui94].

Note that one unrealistic phenomenon is observed in the sensitivity analysis of SD model (Figure 8.11-d). Given some scenarios, the simulated *requirements to implement* (S_{Req}) might be less than zero. However, it is difficult to understand the requirements with a negative value, which may further lead to other variables' deviation from real world cases. In semi-quantitative modelling, it can be prevented in advance with one required step of modelling, an explicit quantity space declaration (`Sreq (0 inf)`). In contrast, it is difficult for SD modelers to involve and check such constraints in their modelling stage.

Overall, both qualitative/semi-quantitative simulation and purely quantitative simulation (system dynamics) compared here have their strength and weakness. In modelling, compared with causal loop diagramming, the qualitative approach starts at explicitly stated qualitative assumptions, and then converts them into more specific and clearer constraints. Thus, it provides a more rigorous approach. Moreover, a CLD model does not offer simulation capability, but a QSIM model does. Both CLD and QSIM model can be quantified to become their quantitative/semi-quantitative counterpart.

In simulation, both of SD and SQSIM can produce *similar* results with single-point values. SD can present the variable's varying trend with more details during the course. Whereas, the SQSIM approach reflects trends more qualitatively, because it shares the same plotting mechanism with QSIM. When dealing with uncertainty, the value range (or discrete values) and its associated probability distribution is required for any stochastic (quantitative) simulation. In contrast, semi-quantitative approach handles uncertainty with value range and envelop function, and can omit their probability. As a result, the quantitative approach can produce relatively *precise* numeric result with probability as well. In some cases, results of semi-quantitative simulation are coarse, and need to be refined.

As SQSIM is extended from QSIM, it provides a more rigorous and robust mechanism in modelling, and then is able to avoid unrealistic results sometimes generated in simulation.

8.6 Summary

This chapter first introduced a model conversion scheme between system dynamics and qualitative/semi-quantitative modelling based on an element level

mapping. By developing the corresponding model of the reference SD model using this scheme, the characteristics of modelling and simulation become comparable between these two approaches. The major contributions of this chapter are highlighted as the follows:

1. An element level mapping from CLD and equations of system dynamics to ASD of qualitative/semi-quantitative modelling is established, and vice versa.

2. A model conversion scheme from a quantitative (SD) model to qualitative/semi-quantitative model is implemented by using the element level mapping. From a given qualitative model and quantification, it is possible to covert and construct its corresponding SD model as well.

3. The n^{th}-order delay is introduced into semi-quantitative modelling and simulation, and implemented in QSIM algorithm framework.

4. The software evolution processes are revisited by using qualitative and semi-quantitative modelling and simulation.

5. The modelling procedure and characteristics are compared between system dynamics and qualitative modelling; and further the comparison of simulation capability and results between system dynamics and qualitative/semi-quantitative approaches are presented.

Following the predefined research design, this part has tested some of the research questions stated in Chapter 1, i.e. *feasibility*, *adaptability*, and part of *uniqueness* of qualitative and semi-quantitative modelling for simulation. The next part will further investigate their *uniqueness* and *practicability* by developing the pragmatic approach and integrated framework for facilitating their adoption in real world software practice.

Part IV

INNOVATION II: ADOPTION

Chapter 9

SQSIM-Based Software Project Management

Part III has explored and tested the feasibility and adaptability of qualitative and semi-quantitative modelling approaches by developing a range of software process models. This part provides some initial thinking in the use and adoption of these approaches in software engineering practice.

Semi-quantitative modelling and simulation has been introduced in Part II and applied in Part III as a new paradigm for constructing software process models. In this chapter*, a novel approach to software project management is proposed to fit the evolved definition of project success, and applies SQSIM as the core technique to achieve this goal. An example is also provided to illustrate the use of this approach in practice.

This chapter first introduces the evolved definition of project success motivating this approach, and then briefly discusses the aspects of software project management that are able to benefit from the semi-quantitative modelling. The proposed project planning and control approach is next described in detail. It is followed by an example application with an SQSIM software process model. Some considerations related to this approach are discussed as well.

9.1 Motivation: Software Project Success

Historically, the success of software project has meant getting the job done within the constraints of success metrics, such as time, cost, and quality. By using this conventional definition, success could be visualised as a single point on a combination of success factor grid [Ker05]. However, few projects, especially those

*The work included in this chapter has been partially reported in [ZKJ07c] and [ZKJ07a].

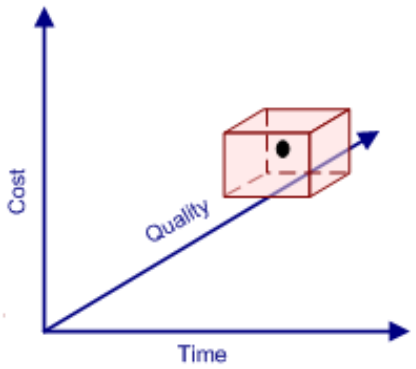


Figure 9.1: Project success: point or cube?

requiring innovation (like software projects), can achieve such an exact target. Due to the ever increasing complexity and diversity of innovative projects, the definition of project success needs to evolve as well.

In software practice, few projects are ever completed without tradeoffs or changes to time, cost, and quality. Therefore, Kerzner argues in his definitive text on project management that "*project success might still occur without exactly hitting this single point target*" [Ker06]. In this regard, the success of contemporary projects might be better defined as a box or a hyper-cube of project success metrics, rather than a single point in multi-dimensions (Figure 9.1), and the project is evaluated as successful if it finishes at any point inside the box or cube. The most common metrics for software project success can be extracted from the results of the systematic review (cf. Section 3.3.3). They may include schedule, cost, quality (defects), size (functionality), market share, return of investment (ROI), and so on. Furthermore, there may exist different priorities among these success factors.

Accordingly, such a contemporary success definition requires project management methods capable of dealing with uncertainty and contingency in multi-dimensional space based on a range of values. Most traditional quantitative techniques, however, usually support single point predictions of project outcomes, rather than prediction of a range of possible outcomes. In contrast, semi-quantitative simulation has the inherent capability of reasoning with value ranges in multiple dimensions, and can facilitate decision-making based on project success factors represented as a range of acceptable values in many dimensions. The following sections explain how semi-quantitative modelling and simulation can support flexible software project management.

9.2 Managing Software Project Semi-Quantitatively

DeMarco identified two broad categories of solutions for managing software projects: '*political solutions for political problems*', and '*technical solutions for technical problems*' [DeM82]. The thinking and solution for managing software projects developed in this chapter concerns the latter only.

For the *technical problems*, most *technical solutions* are purely quantitative. This section discusses the possibilities of semi-quantitative modelling and simulation from several aspects in support of the '*technical solutions*'.

Estimation

A process simulation model is a system with inputs and outputs. As an example, the input and output variables of the incremental development model are denoted in Figure 7.3 and Figure 7.4 (in Chapter 7). The inputs can be further classified into *direct* and *indirect* types. The values of *direct* inputs can be quantitatively and explicitly specified or measured, which enables a manager to allocate, observe and change their values, such as TS, WFT, EFx, h_o, and h_n (in Figure 7.4 and Figure 7.7).

On the other hand, some *indirect* inputs cannot be directly observed, measured, or quantified, such as RERt and m_f (in Figure 7.4 and Figure 7.6). However, these are required to produce model outputs. Although some statistical techniques, for example control charts [FC99], may to a certain degree provide the estimation based on analysis of historical process data, blindly applying single-point value assignment (especially for the complicated model) may result in the dramatic uncertainty propagation, and unrealistic outputs.

In contrast, semi-quantitative modelling is capable of handling the estimation of uncertainty and imprecision intuitively on *indirect* input parameters.

Planning

Planning is an iterative estimating and refining process. Using a process model, the planning process consists of assigning values to the input parameters of the model, and then computing or reasoning about the output parameters as predicted results for planning.

Most current project planning methods are purely quantitative based and use a single point target in multi-dimensional space (i.e. cost, quality, delivery date, etc.) to define project success, which may results in much time consumed in the iterative process. They handle uncertainty by using statistical techniques, such as Monte Carlo Simulation. The corresponding distributions, however, have to be assumed over the range sometimes.

Semi-quantitative modelling can speed up this iterative process by using interval arithmetic and constraint propagation techniques, rather than single-point

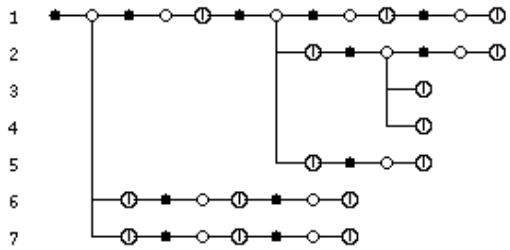


Figure 9.2: Behaviour tree for one scenario of test-fix process

value calculation. The specific tools or artifacts can be developed to support this process, such as element-effect matrix (described later in Section 9.3).

By considering multiple uncertain factors, semi-quantitative modelling does not force you to accept some '*accurate*' single-point estimates during planning (actually you can turn value range to single-point by minimising the interval to zero, cf Chapter 8). Instead, given the intervals of inputs, it produces the guaranteed value range you can expect.

Tracking and Control

As presented in Part III, the behaviour tree is an important output generated by semi-quantitative simulation. In contrast with the traditional methods, it depicts the road map with all possible alternative routes of the process. For example, Figure 9.2 shows a behaviour tree for one scenario of the test-and-fix process (discussed in Chapter 7). Each dot point indicates one critical time point during the process progress. The transition points are denoted by landmarks. All critical time points and transition points need to be identified and recorded as the check-points for process tracking, and its artifact, control metric table (described later in Section 9.3), is then created for observing whether the process is under control.

Decision Making

No decision making is needed if the process lives up to its prediction. However, progress often diverges from expectations. By using behaviour tree and its artifact for project control, decisions can be made at check-points when the process is not on track.

Decision making answers to a series of '*what if*' questions, such as "what if we increase the staffing level, or extend the time period?" Semi-quantitative modelling can maintain the integrity of the prediction when a manager struggles to cope with multiple uncertain factors and has to make a tradeoff.

In addition, though the process performance indicators are represented with value ranges in semi-quantitative modelling, it also helps to extract qualitative

assessment from them for decision making and management, which are useful and important in many cases.

The next section further details a novel and pragmatic SQSIM-based approach for the planning and control activities that correspond to the contemporary success definition.

9.3 A SQSIM-Based Approach for Planning & Control

9.3.1 Planning & Control

Comprehensive planning and control are two of the most important aspects of any project [MH86]. Research into the success of information systems (IS) projects has identified project planning and control as two of the top three important factors affecting project success [Reh96].

Project Planning In general terms, PMBoK defines project planning as "*to define and mature the project scope, develop management plan, and identify and schedule the project activities and resources that occur within the project*" [PMI04].

For complex problems such as a software project, planning is essential to facilitate understanding of the problem and implementation of the solution. It is also estimated that the planning process of project management should require approximately 35% of the project manager's effort over the life of project [Cla02]. Project planning requires the project manager to think through the project and remain focused on the final goal, i.e. project success, which is delivered at the end.

Project Control As defined by PMBoK, project control is "*to compare actual performance with planned performance, analyse variances, assess trends, and evaluate possible alternatives*" [PMI04].

The approach developed and proposed in this section focuses on applying SQSIM-based methods in support of the *technical solutions* for the quantitative aspect of software project management in terms of the contemporary definition of project success. It consists of two connected phases: *project planning* and *project control*.

9.3.2 Phase 1: Project Planning

The approach proposed here includes an iterative refinement method for software project planning. However, the desired results may converge rapidly between the adjoining iterations using SQSIM, when a realistic solution exists. As illustrated in Figure 9.3, the planning approach consists of five distinct steps: *1*) defining

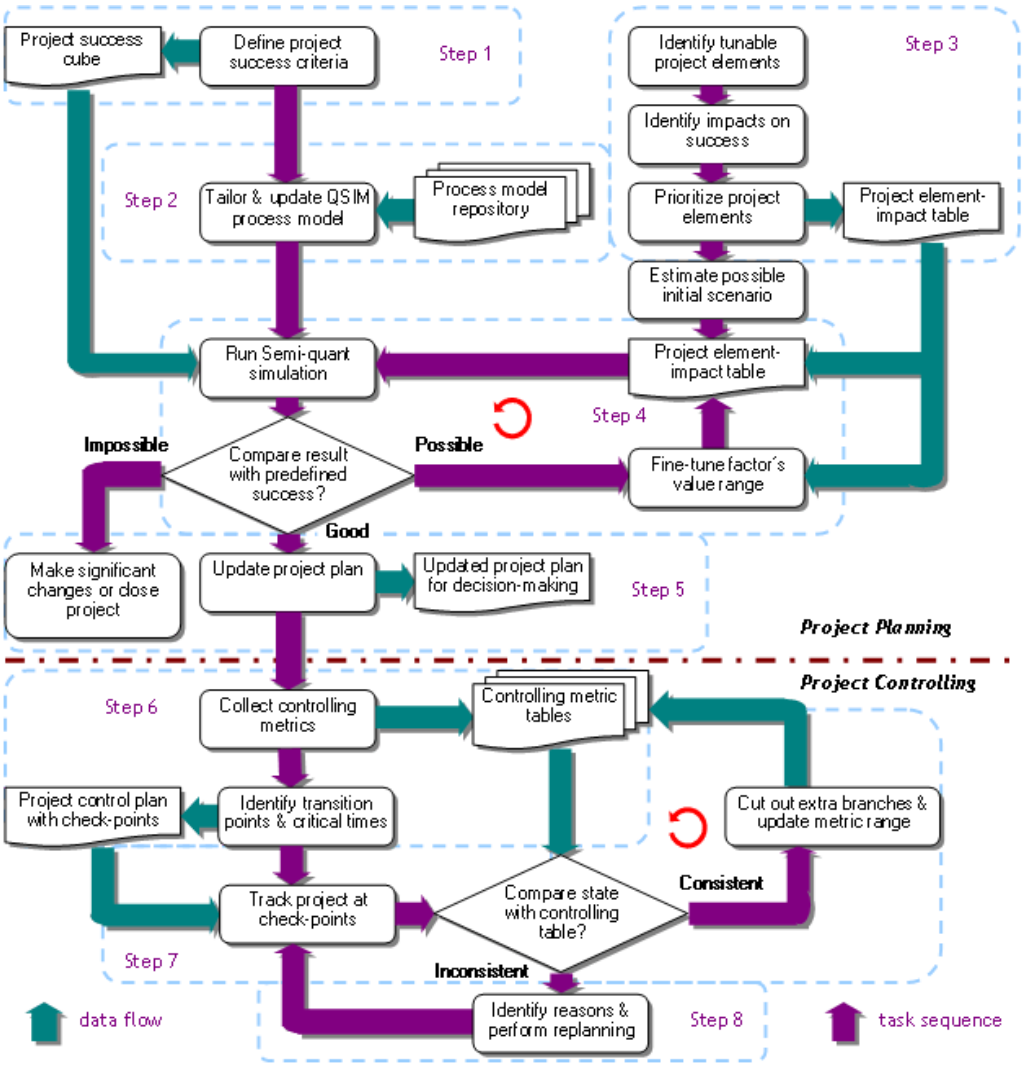


Figure 9.3: Project planning and control with Semi-Quantitative Simulation

project success criteria; *2*) tailoring and updating process model; *3*) creating project element-effect matrix; *4*) simulating and fine tuning; and *5*) updating project plan.

Step 1: defining project success criteria. Unlike the following steps, the first step of this planning approach emphasises the business aspect instead of the technical aspect of project planning. In the real life project planning, a variety of stakeholders may be involved in specifying the project success criteria, i.e. defining the success cube. The output of this step is the project success metric list, in other words, the criteria that are required to define the success for this project, their importance and preference, plus the value

ranges accepted, which are further visualised as project success cube.

Step 2: tailoring and updating process model(s). According to the nature of planned project and the organisational context, one or more reference process models are selected from the literature or organisation's model repository. This topic is too complicated to be included in this chapter. The reference models have to be tailored to fit the project's characteristics, and to be updated with its specific information.

Step 3: creating element-effect matrix. Not all elements can be changed in a process model. Only a few tunable elements are identified in this step, and any value change to these elements and their combination may induce the changes in the project plan and consequent project performance. These elements are further prioritised in order of their importance and contribution to the success factors in element-effect matrix, which also includes their qualitative impact on success metrics. Table 9.2 gives an example of the element-effect matrix.

Step 4: simulating and fine tuning. All outputs from the above three steps, including success criteria, process model(s), and element-effect matrix, are used as inputs to the semi-quantitative simulation (see below).

Step 5: updating project plan. When the simulation produces an acceptable prediction of the project outcomes, the project plan will be updated according to this result. In contrast, if there is a large deviation from the success criteria (such as Impossible state illustrated in Figure 9.4), the management should consider cancelling the project.

Simulation Iteration.

The iteration procedure with semi-quantitative simulation can be regarded as a planning optimising phase to find a fitted solution (a reasonable project plan) progressively for the predefined project success criteria. The generated results converge rapidly if the success criteria are realistic.

The process model selected from Step 2 is coded with the specific element values (ranges) and initial states of the project, and then is executed by QSIM and Q2 (or Q3), which generates all possible behaviours and predicts the project completion states. These states are compared with the success criteria (defined in Step 1). Either Impossible or Good results cause an exit from the iteration procedure. Otherwise, if result is Possible, the values of tunable elements need to be further refined in terms of the element-effect matrix created in Step 3, and the next iteration is triggered with their updates.

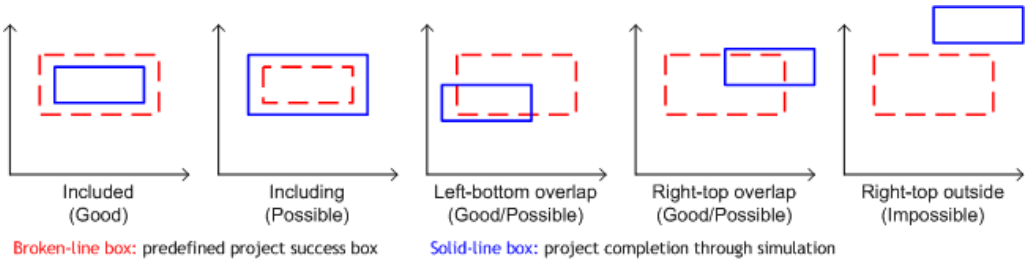


Figure 9.4: Comparing simulation result vs. predefined success criteria

Refinement Strategy.

Here, five types of project completion state (shown in Figure 9.4) are presented for contrasting with the predefined project success criteria. They are used as guidance to indicate if the simulation iteration needs to continue with further refinement or stop. The strategies for other states, e.g. Right-Bottom, can be deduced similarly.

The Included state indicates the predicted project completion falls into the success cube. As the project success defined in Section 9.1, we can easily identify that it is a 'Good' plan for project success.

The counterpart of Included state is Including state, which covers the success area with extra space. This state means Possible success, i.e. the project can finish in success cube or outside. It needs to shrink with further refinement.

Another state is that the project completion area locates at the Left-Bottom of the success cube, but with overlap. It may be translated to Good for some metrics, such as cost and schedule. But for some others, such as earned value and scope (functionality), the overlapping only implies the Possible success, and then the iteration procedure has to continue. Similar discussion applies to the Right-Top overlapping state. When the project completion area locates Outside the success cube (no overlapping existing), the proposed project will be mostly evaluated as Impossible. If no significant changes are available, the project is recommended to be cancelled. The refinement strategy can be further extended and applied to multi-dimension or hyper-cube of project success criteria.

9.3.3 Phase 2: Project Control

You control a project to the extent that you manage to ensure the minimum of surprises along the way [DeM82]. The best-controlled project is the one that best lives up to its predictions. The semi-quantitative controlling approach can provide a flexible way to track and control project progress, and help the project manager observe whether the project is under control. Following project planning, the project control approach contains three major steps: *6*) creating control metric

tables; *7*) tracking project at check-points; *8*) identifying problems and replanning (see Figure 9.3).

Step 6: creating control metric tables. SQSIM generates behaviour tree and all possible behaviours for each simulation scenario. The final project state cube is calculated as the union of the value ranges predicted by the branches. The behaviour tree serves as the road map for the project (like Figure 9.2). The distinction from the traditional methods is that it depicts the alternative routes, given the existing uncertainty. Figure 9.6 is another simple behaviour tree with five branches. The transition points are indicated as landmarks (ⓛ) in behaviour tree.

Once reaching a Good solution for project planning, the control metric table should be created for each measurable variable based on its predictions of all behaviours. Table 9.3 is an example control table. The transition points and critical time points need to be identified from the behaviour tree, and added to project control plan as the check-points.

Step 7: tracking project at check-points. According to the control plan, the performance indicators are measured and compared at the check-points. The project can shift between the branches (possible behaviours), and its progress state can be identified with the corresponding value ranges in control table. If the progress is consistent with the estimated value ranges, it indicates the project under control, then the extra branches (inconsistent with actual project state) should be cut out, and the control table can be updated with the refined value ranges. Correspondingly, the project completion state is refined and updated with the remaining branches.

Step 8: identifying problems and replanning. When inconsistency is found against any branch at check-points, it alerts that the project might be out of control. Under this situation, problems have to be identified, assigned, and corrected, and replanning needs to be performed.

9.4 Illustrative Example

This section presents a simple application of the SQSIM-based planning and control approach, and shows how the project management benefits from this novel approach. To illustrate its value in software practice, this section employs the software process model focusing on the staffing process, which is described in Chapter 6. This model is chosen here for two main reasons: first, it models the software staffing process at a project level, rather than one particular phase of development (like the test-and-fix process described in Chapter 7); second, it is a simplified model that ignores some impacts from other project sectors,

Table 9.1: Project success criteria

success metric	value range
project completion	[245, 390] days
total expenditure	[2150, 3000] man-days

which avoids the excessive project detail and maintains our focus. For clarity and ease of understanding, the project success constraints are defined in only two dimensions in this example.

9.4.1 Baseline Project

Abdel-Hamid and Madnick's EXAMPLE project [AHM91] is selected as the baseline project again for the purpose of illustration. This project was first used in Chapter 6. The main attributes of EXAMPLE project are summarised in Table 6.5. As originally planned, the project can be delivered on day 430. In this example, we set this project in a realistic project scenario.

Considering any contingency issues, such as leave or sickness, the initial project team size is defined with (the value range of) [4, 5] developers. The EXAMPLE project proceeded as planned until a request for change (RFC) is raised by marketing department on day 240. It reports that a major competitor plans to release a similar software product in the near future, and argues that their own product must be released at least two months earlier than the original schedule to remain competitive. After one-week's analysis and discussion across the organisation, management approves the RFC with the condition that the new total expenditure must be no more than 3000 man-days. The project manager is responsible for making the corresponding changes to the project plan.

9.4.2 Planning Phase

This is a typical project replanning scenario on the fly. First, the project manager tries to figure out a Good solution using the proposed SQSIM-based planning approach.

Step 1. The project manager updates the project plan on day 245 (one week after RFC). The changed project has to be completed two months (40 working days) earlier than the original schedule. In other words, the current project closure targets at day 390. The original estimated budget of the project is 2150 man-days. The project success criteria are updated correspondingly (Table 9.1).

Table 9.2: Project element-effect matrix

element	*time*	*cost*	*constraint*
new workforce	[+/-]	[+]	[0, 12] developers
productivity ratio	[-]	[-]	[.4, .6]
assimilation delay	[+]	[+]	[60, 80] days
remaining size	[+]	[+]	

Step 2. The qualitative process model (shown in Figure 6.1) is chosen in this case and used for the simulation.

Step 3. By examining the elements of the process model for simulation, four tunable elements can be identified: *new workforce*, *productivity ratio*, *assimilation delay*, and *remaining project size* (Table 9.2). The value of *new workforce* indicates how many new developers are introduced into the project. The available manpower resource is up to 12 developers for this project. Chapter 6 explains the quantitative constraints in detail.

It is noticeable that the first three elements are related to introducing more developers into the project. The fourth element, i.e. reducing *remaining project size* (functionality), is not desired by the clients, so it is ranked at the bottom. Considering the time for recruiting, the new staff can join the project team in three weeks, i.e. recruitment delay for 15 days. The project completion is correspondingly refined as [260, 390] days. Because the values of the second and third elements are highly dependent on the quality of the new staff, it is hard to refine the value ranges before the assimilation. Therefore, the project manager plans to start the simulation by adding extra workforce into the project without altering the uncertainty on the last two elements.

Iteration 1. We initially introduce [3, 4] developers into the project to initiate the simulation process. The simulation predicts that this project can finish on day [339, 423], and the completion cube is depicted in Figure 9.5. Comparing the original completion time and project success cube (in dashed-line), this decision slightly improves the product release schedule (but only guaranteed for 7 days) within the acceptable cost performance. However, the delivery date is still behind the required release date (day 390).

Iteration 2. One positive finding through Iteration 1 is that adding extra workforce may shorten the project duration. To amplify this positive effect, the project manager introduces [11, 12] developers in this iteration. It generates 3 possible behaviours this time, which predict the project may finish on day [309, 386], a

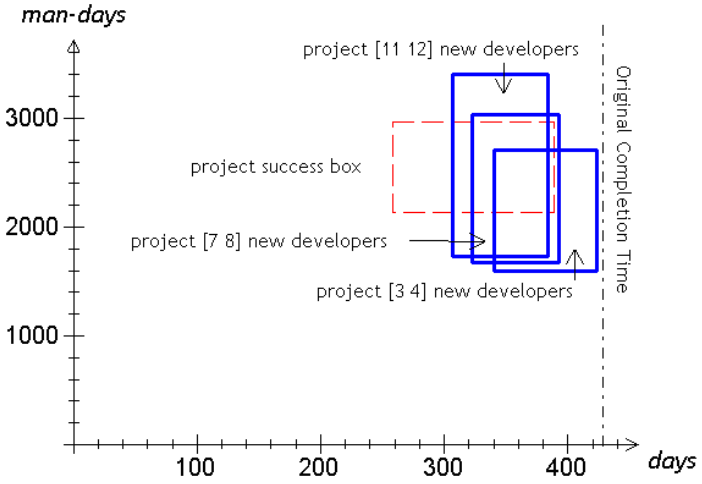


Figure 9.5: Project completion cube through iterations

bit earlier than requested release date. However, the completion cube indicates that the project cost may increase significantly and reach much higher than the acceptable budget (see Figure 9.5). Although the financial performance looks terrible this time, it further verifies the positive contribution of extra workforce to schedule.

Iteration 3. With respect to the impacts identified in Table 9.2, fewer developers need to be added in this iteration to reduce the possible high expenditure caused in Iteration 2. The project manager chooses a modest number of developers, say [7, 8] developers, for this iteration. Five possible behaviours are generated (shown in Figure 9.6), and indicate the project may finish at day [320, 396], before or after the new members of staff are fully assimilated.

Both schedule and cost are slightly over the requested success cube. This means that the solution corresponds to the Right-Top overlapping state in Figure 9.4. Analysing the impacts of increasing the workforce across iterations, it is not difficult to find that reducing the number of extra staff (for Iteration 3) will incur a further delay of the project; and conversely, introducing more developers will result in a higher cost.

Step 5. After negotiating with the senior management and marketing department, they reach the agreement to update the project plan with new time frame of [320, 396] days and budget of [1700, 3068] man-days. Meanwhile, the management gives up the last option (in Table 9.2) of sacrificing software functionality or quality. Given this update of project success criteria, the project manager plans

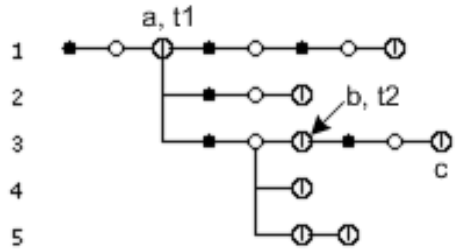


Figure 9.6: Behaviour tree of Iteration 3

to recruit [7, 8] developers into the team with confidence that the projects will be successfully completed on time.

9.4.3 Control Phase

Figure 9.6 is the project behaviour tree generated for this simplified case by Iteration 3. It depicts five possible behaviours: three of them, i.e. Behaviour 1/2/4, have one transition only (t_1, when new workforce is introduced) before the project completion; Behaviour 5 ends exactly at the second transition point (t_2, when assimilation ends at); only Behaviour 3 passes two transitions (t_1, t_2). The behaviours are distinguished by the variables' trends (e.g. value going up or down).

Step 6. Based on the behaviour tree, the control metric table is developed for each measurable variable to track its changes at check-points. The most important check-point is transition point t_2 that indicates the end of assimilation. Table 9.3 is an example control metric table for Rsd (software development rate) and Sc (completed software size).

Step 7. When the project progresses to t_2, the project state is compared to the controlling tables. Because only Behaviour 3 goes through the second transition point, if we are aware of the end of assimilation and Sc falls into the range of [44, 64] KDSI, we can predict the schedule might reach 396 days. Correspondingly, the behaviour branch 1, 2, 4, and 5 can be cut out. On the other hand, if the project closes during the assimilation, it may happen in the time period of [323, 340] days.

Step 8. One unexpected situation might be that the assimilation finishes, but the project progresses to the outside of (lower than) the range [44, 64] KDSI. It means the project is out of control. The project manager has to identify and

Table 9.3: Control metric table example

No.	RSD@t_1	RSD@t_2	Est. duration
1.	[123, 179]	[203, 387]	[325, 340]
2.	[141, 179]	[203, 387]	[325, 340]
3.	[141, 179]	[277, 403]	[320, 396]
4.	[141, 179]	[201, 403]	[323, 340]
5.	[141, 179]	[277, 403]	[323, 340]

correct the assignable problems immediately. Replanning should be carried out to update the project end state.

9.5 Related Considerations

Multi-Dimension Simulation

In the EXAMPLE project (in Section 9.4), only two dimensions (project schedule and cost) are used to define the project success cube. The simplified scenario helps to explain our approach. However, in the real world, management may consider more factors simultaneously, and needs to determine the tradeoffs between all of them for decision making. SQSIM provides this capability of reasoning among alternative process behaviours in multiple dimensions.

Success Factors

In real software projects, many success factors can be defined at the planning stage, and can be reasoned about using the multi-dimension capability of SQSIM. The example uses a simplified 2-D view of project success. However, normally a variety of stakeholders are involved in the project planning process. Most of them, including the project manager, development team, project clients, and senior management, may possess very different perspectives and expectations of the project outcomes.

For instance, the marketing department (i.e. the internal project client) hopes to release a new software product earlier than the competitors. On the other hand, the development team estimates the required project duration in terms of their own experience. Therefore, both groups have to compromise with each other on the value ranges in dimension(s) of project success cube, accepting that delivery is impossible for developers before the lower value, and release is useless for clients after the higher value.

Table 9.4: Element-effect matrix of test-and-fix process model

element	EPEs	EAEs	EAD	EPD	EFx	t
TS	[-]	[-]	[+]	[+]	[+]	[+]
WFT	[N]	[N]	[N]	[N]	[N]	[-]
PDFx	[N]	[N]	[N]	[N]	[N]	[-]
m_f	[N]	[N]	[N]	[N]	[N]	[-]
EP	[+]	[+]	[+]	[+]	[+]	[+]
EA	[+]	[+]	[+]	[+]	[+]	[+]
EDANTc	[N]	[-]	[+]	[N]	[+]	[+]
...	...	...	...	...	...	...
h_o	[-]	[-]	[+]	[+]	[+]	[+]
h_n	[-]	[-]	[+]	[+]	[+]	[+]

[+]: positive relation; [-]: negative relation; [N]: no explicit relation

Element-Effect Matrix

An element-effect matrix should be developed with the qualitative process model. With respect to project's characteristics and organisation's context, different projects may contain quite different tunable element set at Step 3, even if they apply the same reference process model. A manager or estimator will learn from the matrix and the iterations how a combination of input elements influences the project performance (outputs), and how sensitive the (software process) model is to the changes in certain parameters. Table 9.4 gives another example of element-effect matrix for the test-and-fix process model developed in Chapter 7.

Good Solutions

Our approach is to find one '*good*' solution that guarantees the project falls into the success cube. However, the outcome is not a unique one fitting the predefined project success, but one of the possible 'Good' solutions. Different iterations may produce slightly different solutions. Project managers have to identify the tradeoff (solution) among the success factors required by clients and executives, allowing for the resources available to the project. With regard to the definition of project success in Section 9.1, there is no difference (better or worse) among the all possible Good results. Thus, the solution obtained through this approach can assist in the planning and controlling process by offering both flexibility and contingency tolerance.

9.6 Summary

Based on the explorative modelling of software processes using qualitative and semi-quantitative approaches in Part III, this part aims to present the considerations on the practical use and adoption of the promising approaches in software engineering practice.

This chapter focuses the potential but pragmatic use of SQSIM in assisting of software project management.

1. It first introduces the contemporary definition of project success to software process research.

2. It then outlines the initial thinking in the unique uses of SQSIM in software project management different from traditional quantitative approaches.

3. A novel approach for project planning and control using SQSIM is developed as a solution, which matches a contemporary definition of project success.

4. A simplified illustrative project example is given for demonstrating the application of this approach and its unique values.

The semi-quantitative approach is presented in this chapter as a powerful technique for planning and controlling software project with uncertainty. It is able to depict all possible routes of project progress offering a project manager the flexibility and confidence to cope with uncertainty and contingency during the software development, and guaranteeing the integrity of final project states predicted. In contrast, many traditional approaches deliver only one-point sample of the set of possible solutions in the success cube. Therefore, the SQSIM-based approach is able to be an indispensable supplement to contemporary software project management.

The next chapter discusses the interaction between process simulation and process improvement, and develops an initial version framework for supporting the adoption of SPSM in practice. It further justifies the unique value of QSIM/SQSIM for the software organisations at low maturity levels.

Chapter 10

Adopting Process Simulation in Software Organisations

Chapter 9 presents the initial considerations about SQSIM-based software project management and develops a pragmatic approach for project planning and control. This chapter, from a different perspective, discusses the relationships between SPSM and SPI, and focuses on the practical adoption of process simulation in software organisations. It proposes a primary framework as a general guideline for the adoption of SPSM, which integrates QSIM/SQSIM and other typical simulation paradigms.

10.1 Motivation

Process simulation methods were initially introduced to software engineering comprehensively by Abdel-Hamid's [AHM91] and others' efforts in the late 1980s (cf. Chapter 2). However, in the author's experience, especially from the Australian software industry, these methods are seldom adopted in practice. In terms of the findings from the systematic review, one possible reason might be the lack of guidance for supporting the adoption of appropriate modelling and simulation paradigms within a specific organisation's context.

As addressed by Kellner *et al.*, "*no single modelling approach or tool is the most natural and convenient one to use in all software process situations*" [KMR99]. However, they only took into account the factors of purposes, questions, and desired result variables, and discussed their relations to the capability of simulation paradigms. Unfortunately, the organisation's context (precisely process maturity discussed in this chapter), which is crucial to influence the success of adoption of process simulation in practice, has been often omitted in most of previous

research. This chapter* aims to stimulate research and trigger debate in this research direction.

CMM(I)-based process improvement has been discussed for many years in SPSM community. For instance, Christie argued that CMM-based process improvement can benefit from process simulation, and that simulation can help to tackle different questions on each CMM level [Chr99]. However, he did not distinguish the different simulation techniques in his discussion. Raffo *et al.* further suggested that process simulation serves as an organisation's strategy for achieving a higher process capability and moving to higher CMM levels [RVM99]. Nevertheless, it can be asserted that there is no '*one-size-fits-all*' simulation solution for all organisational contexts, in particular organisations at different CMM(I) maturity levels. The selection of a suitable process simulation paradigm is the first necessary step to realise the value of simulation for software organisations. Unfortunately, there is a lack of the generic guidance on how to select the appropriate simulation paradigm(s). With reference to the capability descriptions of CMM/CMMI maturity levels and the requirements for quantitative simulation modelling, there exists a significant gap between them at lower maturity levels.

In contrast to most previous discussions focusing on the simulation's positive contributions to CMM-based process improvement, the following sections in this chapter argue that adverse effects can also occur. Primarily, a framework (version 1.0) is proposed to support the selection of appropriate simulation paradigms by mapping the selected process simulation techniques to their related CMMI levels. This provides general guidelines for the adoption of process simulation in software organisations. In addition, our discussion intends to justify the following propositions:

1. Software organisations that have achieved high levels of process maturity will find it relatively easy to adopt process simulation as part of their standard software project and process management processes.

2. Once having achieved high process maturity levels, an organisation can refine process simulation models used at lower levels by incorporating into the models more detailed process measurement data.

3. Compared with commonly used quantitative simulation methods, qualitative and semi-quantitative process models can be more appropriate for adoption in software organisations at low process maturity levels.

This chapter first briefly reviews the related concepts of process simulation and CMMI for defining the framework scope, and describes the interaction between them. It further explains the primary framework and justifies the mapping

*The work included in this chapter has been partially reported in [ZKJ07b].

across the maturity levels . Finally, some associated issues to the current version framework are discussed.

10.2 Scope: SPSM & CMMI

This chapter presents a framework that links four different process modelling paradigms to CMMI maturity levels and transition process between maturity levels. This section justifies the choice of process modelling paradigms selected for inclusion in the initial version of the framework and provides a brief description of their scope. It also gives a brief overview of CMMI by explaining the relationships between *process area groups* and *maturity levels*. The interaction between SPSM and SPI (CMMI) is also discussed.

10.2.1 Software Process Simulation Modelling

Scope of Purposes

With reference to the findings from the systematic review (cf. Chapter 3), the purposes for undertaking simulations of software process models are grouped into three levels, which updated and enhanced the reasons given by Kellner, Madachy, and Raffo [KMR99]. They are repeated here for quick reference: the ***cognitive*** level contains the purposes of *1*) understanding, *2*) communication, *3*) process investigation, *4*) training and learning; the ***tactical*** and ***strategic*** levels include the purposes of *5*) prediction and planning, *6*) control and operational management, *7*) risk management, *8*) process improvement, *9*) technology adoption, *10*) tradeoff analysis and optimising.

Note that the objectives on the *cognitive* level can benefit from all simulation paradigms no matter what maturity level the organisation is on. For example, even a Level 1 organisation can apply a role-playing simulation game (RPG), and gain insight from it. However, such application is not the case of simulation in support of the real process execution in a practical situation. Thus, the framework described in this chapter focuses only on the purposes on the *tactical* and *strategic* levels of purposes.

Simulation Paradigms

The systematic review of ProSim series (1998-2007) publications (cf. Chapter 3) also identified *system dynamics* (SD, 49%) and *discrete-event simulation* (DES, 31%) as the most frequently used process simulation modelling techniques. For this reason, they are considered for inclusion in the framework (v1.0). Another reason for selecting these approaches is that they represent two typical simulation approaches. The former is the widely applied *continuous* simulation paradigm which captures higher level project or product considerations and shows how

feedback loops connect a variety of business characteristics. In contrast, *discrete* simulation is the modelling of systems in which the state variable changes only at a discrete set of points (events) in time. It is excellent at capturing well-defined process tasks, incorporating, queueing and scheduling considerations.

However, both techniques are purely quantitative approaches for modelling and simulating systems. As software organisations at lower ends of CMMI lack the ability in quantitative management, they are difficult to obtain major benefits from these approaches (in-depth discussion in Section 10.3). Accordingly, the framework must include other simulation paradigms, e.g. *qualitative simulation* (QSIM) and *semi-quantitative simulation* (SQSIM), for the requirements of low CMMI maturity levels.

Qualitative simulation models reflect the systems in the real world at an abstract level. Fewer assumptions are required than for purely quantitative approaches. As introduced and demonstrated in Part II and III, the outputs generated by QSIM are all the possible behaviours of the system, whose states are described by qualitative landmarks, instead of numeric values.

As an extension of QSIM, SQSIM focuses on the use of bounding intervals to represent partial quantitative knowledge. This paradigm provides a seamless transition between purely qualitative and quantitative approaches (cf. Chapter 8). In the previous chapters, these qualitative and semi-quantitative approaches have been demonstrated by developing qualitative software process models and bounding with quantitative constraints.

Table 10.1 gives a brief summary of the selected simulation paradigms in terms of the type of processes they are used to model and information needed to apply the models.

10.2.2 Process Maturity Model: CMMI

As the successor of CMM, CMMI describes the practices for software process change, and framework for measuring the compliance of organisations. CMMI selects only the most important topics for process improvement, and then groups those topics into '*areas*'. It represents ten years of lessons learnt from many external and internal consultants, based on applying continuous improvement to CMM itself [Kas04].

Representation

Unlike its predecessor, CMMI offers two representations, i.e. *staged models* for assessing organisational maturity and *continuous models* for measuring process capability. The main difference between *maturity levels* (MLs) and *capability levels* (CLs) is the representation they belong to and how they are applied. Table 10.2 shows the maturity levels of staged representation [SEI02a, SEI02b].

Table 10.1: Summary of selected paradigms for Framework v1.0

Paradigm	Simulation type	Process type	Data requirements	Model driver
QSIM	Continuous	Macro-process	Qualitative values of parameters and relationships among parameters	Qualitative time, i.e. landmarks for describing critical events
SQSIM	Continuous	Macro-process	Numeric ranges for parameters, and envelop functions for relationships	Qualitative time or quantitative interval for describing critical events
SD	Continuous	Macro-process	Numerical values for each parameter	Quantitative & continuously evolved time intervals
DES	Discrete	Micro-process	Numerical values (plus probability) for each parameter	Quantitative queueing attributes in discrete time set
Hybrid	as per modules	Macro- & micro-process	as per modules	as per modules

Here, the ***staged representation*** is chosen for the framework for the following reasons:

1. It provides a recommended path of improvement evolution (as shown in Figure 10.1, refer to Section 10.3.3) for the entire organisation based on the last decade's best practices.

2. It allows comparisons across software organisations by using appraised process maturity levels.

3. The single rating can be used as the indicator of the organisation's overall maturity level, and provides an easy mapping to process simulation paradigms.

4. It provides a smooth migration from CMM to CMMI.

Table 10.2: CMMI Staged Representation

Maturity Level	Staged Representation
1	Initial
2	Managed
3	Defined
4	Quantitative Managed
5	Optimising

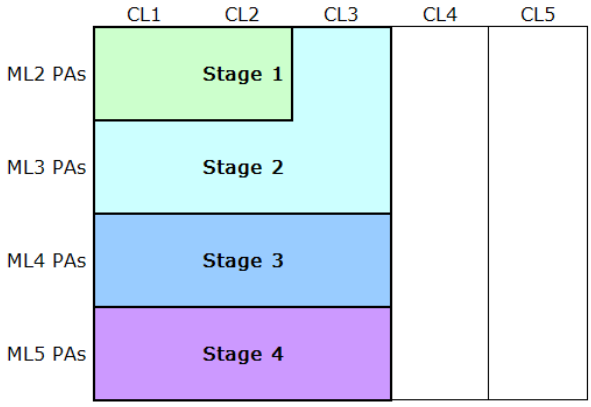
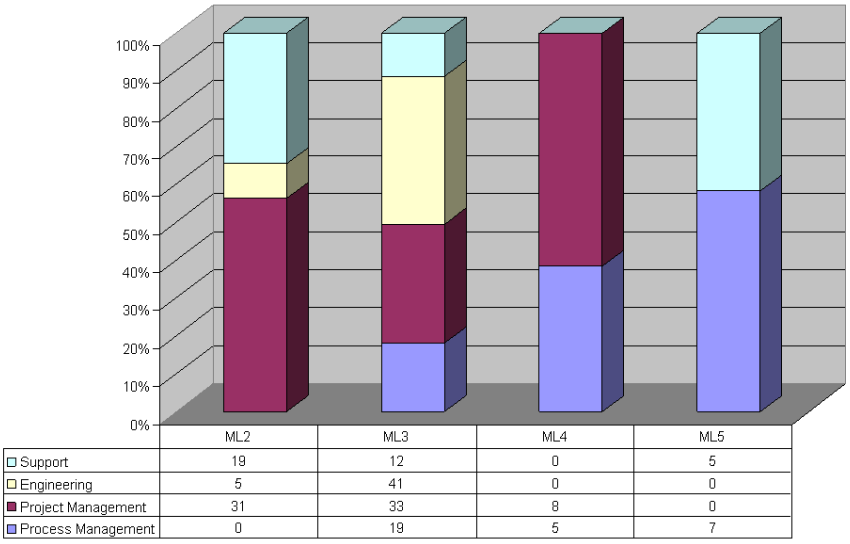


Figure 10.1: Capability profile for maturity levels

Process Areas

CMMI contains 25 *process areas* (PAs) and 185 *specific practices* (SPs) grouped into four categories in terms of their scopes (Figure 10.2). *Project Management* process areas consist of project management activities related to planning, monitoring, and controlling project. *Process Management* process areas provide the organisation with capability of performing cross-project activities related to defining, deploying, implementing, monitoring, appraising, measuring, and improving processes. *Engineering* process areas cover product development and maintenance activities shared across engineering disciplines. They define the product development processes rather than discipline-specific processes (e.g. software engineering). Since the *Support* process areas address processes that are used in the context of performing other process areas [SEI02a, SEI02b], the first three process area groups are the main aspects considered in the initial version of the framework.



	ML2	ML3	ML4	ML5
Support	19	12	0	5
Engineering	5	41	0	0
Project Management	31	33	8	0
Process Management	0	19	5	7

Figure 10.2: Distribution of Specific Practices across maturity levels

Practices

The required component of the CMMI models is the '*goal*' that represents a desirable end state, and indicates that a certain degree of project and process control has been achieved. A *specific goal* (SG) is unique to a single process area; in contrast, a *generic goal* (GG) may apply across all of the process areas. Therefore, the proposed framework mainly focuses on *specific goals* and *specific practices*, which represent the '*expected*' means of achieving the goal, and their different emphases at maturity levels. The number of *specific practices* applied to each maturity level is calculated and categorised into process area groups (Figure 10.2). Though the allocated effort varies across the practices, and even for the same practice performed among different software organisations, in general speaking, this comparison to a large extent illustrates the emphasis of improved process areas on each maturity level.

10.2.3 Interaction between SPSM & CMMI

In terms of the findings from the systematic review, software process improvement (SPI) has been recognised as one of important motivations of SPSM research (cf. Section 10.2.1). Accordingly, most of previous research solely argued and justified the function of SPSM in support of SPI [Chr99, RVM99]. On the contrary, the adverse effects were seldom discussed.

Figure 10.3 depicts the interaction between SPSM and CMMI. At the top of the diagram, the arrow indicates the support from SPSM to SPI by producing

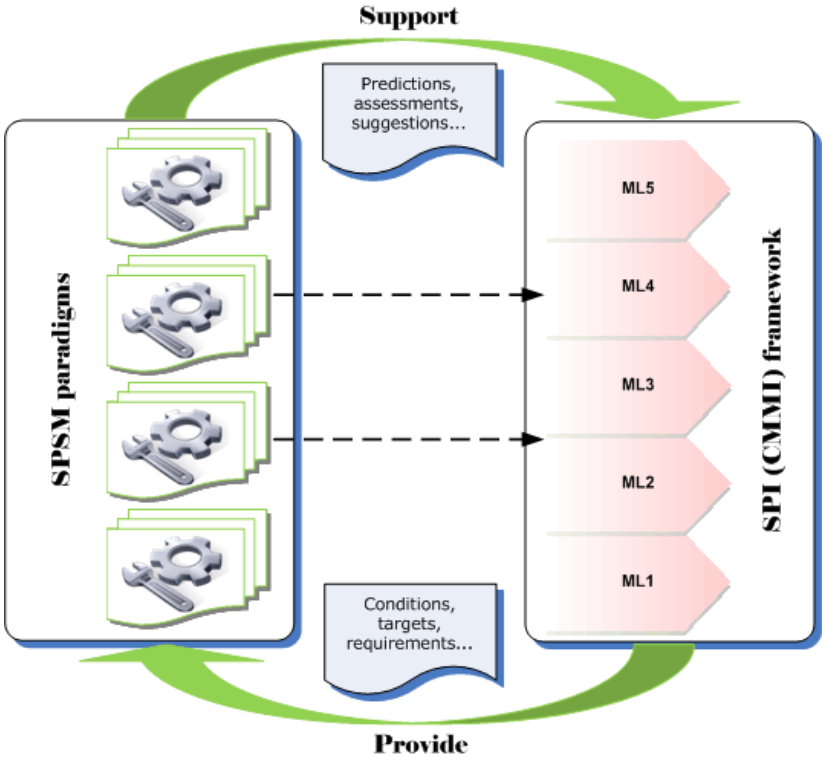


Figure 10.3: Interaction between SPSM & CMMI

process predictions, assessments, and suggestions to process changes through simulation studies. The arrow at bottom shows that the effects of SPI on SPSM exist as well. In a CMMI-based SPI framework, the identification of maturity level of a software organisation can provide the conditions for adopting particular simulation paradigm(s), and targets for performing simulation studies. By appropriately and progressively introducing and adopting process simulation paradigms in software organisations, a positive feedback loop, e.g. from SPSM to SPI and further to SPSM, can be established in match of the continuous improvement targeted by SPI programs.

When a software organisation achieves a particular CMMI maturity level, it can be assessed as capable to adopt particular simulation paradigm(s); on the other hand, maturity levels are static points during the course of continuous introduction of new simulations and the improvement of process to the higher levels supported by their adoption. The purpose of this framework is to help organisations secure the success of SPSM adoption, maximise the benefits gained from process simulation, and use the appropriate paradigm(s) needed to achieve higher maturity levels.

10.3 Adoption Framework (version 1.0)

According to the introduction of SPSM in Chapter 3, simulation models may be discrete or continuous, quantitative or qualitative, for macro- or micro-process research, or mixed. The choice of whether to use a discrete or continuous (or mixed) simulation model is a function of the characteristics of the system and the objective of study [BCNN05]. For a software organisation, CMMI provides an assessment framework for the organisation's capability (i.e. characteristics of the system) and the target of process improvements (which is one of important objectives of study).

As CMMI depicts a progressive path to achieve continuous process improvement, likewise the adoption framework (proposed in this chapter) depends on the continuing improvement in the organisation's process capability maturity. The mapping is implemented by analysing the characteristics of software organisation and the practices introduced at each maturity level, and comparing with the inherent requirements and functions of each simulation paradigm. This section also provides simulation model of one well-defined software process as an example for each transition.

10.3.1 Framework Overview

Figure 10.4 shows the adoption framework version 1.0. It depicts the introduction and adoption of the process simulation paradigm in an organisation is also a process, rather than a single point action of CMMI appraisal. Hence, one particular simulation paradigm is normally introduced during the transition between two adjoining CMMI maturity levels.

As an organisation increases its process capability, it can provide more precise and reliable process information with richer details both of the process itself and the metrics used to describe process attributes. Thus, it is able to adopt low level (*micro-process*) or more sophisticated process modelling paradigms that require detailed, quantitative process information. This framework, therefore, visualises an evolution path from high level (*macro-process*) abstract to low level (*micro-process*) more complex simulation paradigms that mirrors the CMMI improvement process, i.e. from macro-process to micro-process, from qualitative to quantitative, from continuous to discrete and then hybrid (see Section 10.3.6).

At the maturity levels beyond ML1, the framework offers more than one choice of simulation paradigm to software organisation. To use the framework, imagine that you stand at one particular position (regarding your maturity level), say ML2, and look right. Then you can make the relatively mature decision from the simulation paradigms you can see, i.e. SD, QSIM, and SQSIM for ML2 (see the following sections for detailed explanation). These simulation paradigms are recommended to fit your organisation by this framework.

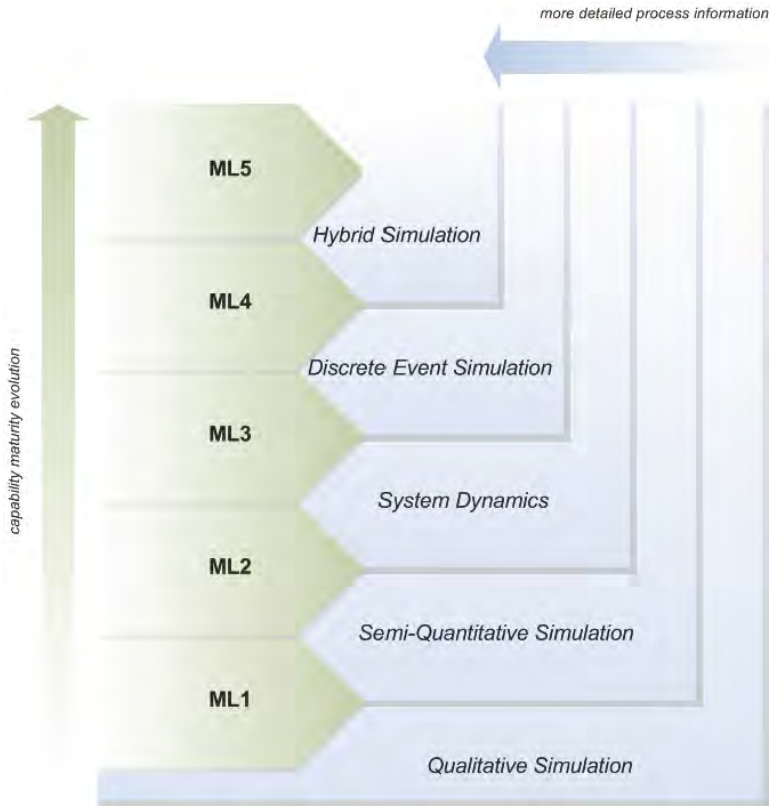


Figure 10.4: Framework (v. 1.0) of adopting SPSM in CMMI organisations

10.3.2 Starting at ML1

At the entry level, software processes are performed in a chaotic and unstable organisational environment. "*Maturity level 1 organisations are characterised by a tendency to over commit, abandon processes in the time crisis.*" [SEI02b] Given a poorly-defined process, significant uncertainty and high risk associated within such situation, project success and process performance can rarely be predicted quantitatively. Because of the large variance and contingency of the development behaviours, in most cases, the organisation is unable to repeat their past success.

Although it is an ad-hoc level and no stable processes are followed in organisation, some qualitative knowledge and models still apply (e.g. Brooks' Law and defect amplification across development phases). Qualitative assumptions can be abstracted from these models, corresponding to general knowledge about the software development process. A ***qualitative simulation*** model can be then developed based on these qualitative assumptions.

With reference to the demonstration and discussion in Part III, QSIM is able to simulate the qualitative behaviours of software processes, as well as reason possible trends of project progress given the incomplete and uncertain process

information. The simulation results can facilitate the qualitative management and prediction of development process, which is more realistic in software organisations at ML1.

Example. The software process focusing on staffing level is modelled and simulated as an example for each transition in this chapter. Brooks' Law might be the most well-known statement regarding the software staffing issue. It argues "*adding manpower to a late software project makes it later*" [Bro95], and has a negative impact on software development productivity. The qualitative simulation model for examining Brooks' Law was built on ten basic qualitative assumptions of the software staffing process (cf. Chapter 6), such as "*adding more people to a project results in a larger communication overheads*", and "*new employees' productivity is initially lower than experienced developer's productivity*". This model generates all possible behaviours to describe the software staffing process. Even without quantitative information, the simulation results can justify that under some scenarios adding more people may help the project complete earlier than the original schedule.

10.3.3 Transitioning from ML1 to ML2

In progressing to ML2, organisations start to apply the generic and specific practices. As illustrated in Figure 10.2, over 55% *specific practices* implemented in this transition concentrate on adopting '***basic project management***' methodologies (*process areas*). On the other hand, no *specific practice* of *process management* is introduced until achieving ML2. Therefore, the main improvements are expected on the project or management level, not the process level. For instance, the estimates of attributes of the work products are established and maintained (PP-SP1.3[†]); based on estimation rationale, project effort and the cost for work products are established (PP-SP1.4); project risks are identified and analysed (PP-SP2.2); the actual values of the project parameters are monitored against the project plan (PMC-SP1.1); the project's progress, performance and issues are periodically reviewed (PMC-SP1.6), and so on.

However, quantitative project management processes must be adopted progressively. It takes time not only to accumulate sufficient project history data, but to specify how measurement data will be obtained, stored, analysed, and reported (MA-SP1.3/1.4). This implies there might be significant variance, even inconsistency, in the data collected during this transition. In addition, it can be noted in Figure 10.1 that all adopted process areas at current maturity level (for ML2) are targeted at CL2 (capability level 2), other than CL3 required for all the higher maturity levels. This implies only primary quantitative project management capability is expected at maturity level 2.

[†]The code of CMMI specific practice, refer to [SEI02a] and [SEI02b] for detailed description.

Obviously, the discrete paradigm is not appropriate for the simulation during this transition, because of the absence of the *specific practices* of process management. In contrast, the continuous paradigms are more suitable for capturing the project or product characteristics at high (macro-process) level. However, in light of the lack of complete and reliable history data from ML1, the estimates of project metrics are highly based on project manager's personal experience and incomplete history information. Though quantitative simulation can cope with the uncertainty with stochastic methods, e.g. Monte Carlo Simulation, unfortunately, the number of uncertain factors may be too many to handle in this way, and the statistical distributions may be unknown or unstable at this stage. Thereby, blindly adopting a purely quantitative simulation within such context may result in *over-optimistic* or *-pessimistic* predictions, and may finally discourage the implementation of process simulation due to the unreasonable expectations.

As the extension of *qualitative simulation*, ***semi-quantitative simulation*** provides a seamless transition between qualitative and purely quantitative approaches. It can be introduced as a lens with a smooth zoom to match the organisation's immature but continuously improving quantitative capability during this transition. The discipline of semi-quantitative modelling will encourage project managers to estimate in terms of ranges of values. This effectively avoids the mistake of being more precise in process simulations than an organisation's actual process capability warrants.

Example. The software staffing process model is extended with the quantitative constraints in Chapter 6. Since the uncertainty is still high at level 2, and only incomplete historical project data is available, SQSIM assigns envelope functions to the relations in the model, and value ranges to the inputs and its initial state. When the stricter quantitative constraints are applied during process progressing, the simulation may produce fewer but more precise behaviours for the specific staffing process (cf. the example in Chapter 9). Given the primary and limited quantitative management capability (between ML1 and ML2), SQSIM is able to provide the possible behaviours of development process for decision-making while maintaining the integrity of the final solution.

10.3.4 Transitioning from ML2 to ML3

Once the organisation achieves ML2, projects can be managed and a few successful project management practices can be repeated. As over 55% of all CMMI practices must be implemented successfully here in order to reach ML3 (as shown in Figure 10.2), the transition to ML3 will produce the most distinct improvements across the maturity levels. The main process areas in this transition are *Engineering* (39%) and *Project Management* (31.5%). The '***advanced project management***' practices (except '*quantitative project management*' processes)

are introduced, such as to establish and maintain the project's defined process (IPPD-SP1.1); to define the parameters used to analyse and categorise risks and control risk management effort (RM-SP1.2), and so on.

System dynamics is suggested to be introduced during this transition for more precise management based on experience and knowledge. SD is a dynamic feedback system, sometimes refined as a goal-seeking system. It is possible to study the interaction of control policies, exogenous events and feedback structures producing dynamic behaviour, such as rise, drop or oscillation (an example SD process model included in Chapter 8). SD simulates the software process as a set of performance indicators. Most of them are active during the whole project or project phases.

Although as one alternative solution beyond ML2, the SQSIM offers the capability of purely quantitative continuous simulation (cf. Chapter 8), SD is preferred here for its wide application (the most popular simulation paradigm applied in SPSM during the past decade, cf. Chapter 3). However, the SQSIM models developed in the last transition can be refined and converted into an SD model by following the model converting scheme discussed in Chapter 8.

Meanwhile, the organisation starts to adopt the '***basic process management***' practices (for ML3), such as to establish and maintain the description of the process needs and objectives for the organisation (OPF-SP1.1); to establish and maintain the organisation's set of standard processes (OPD-SP1.1); to deploy organisational process assets across the organisation (OPF-SP1.1), etc. Thus, the organisation can only partially and locally benefit from discrete-event modelling prior to ML3 when these practices are well defined and implemented across the parts or the whole organisation.

Example. Madachy developed an SD model of the software staffing process to examine the Brooks' Law [Mad08]. He simplified Abdel-Hamid and Madnick's model [AHM91] by focusing on the assimilation procedure. The model was built using a set of specific numeric values, which were selected from the literature or historical data of company projects, to represent the relations in the model. Further, the process was simulated with the data from specific projects as inputs. His model generates single deterministic behaviour through one simulation, and the impact of different staffing policies were analysed by comparing the numeric values describing the project states through multiple runs.

10.3.5 Transitioning from ML3 to ML4

When some software processes become well-defined at ML3, it is appropriate time to completely introduce ***discrete-event simulation*** (Figure 10.4). A '*defined process*' clearly states: *purpose*, *inputs*, *entry criteria*, *activities*, *roles*, *measures*, *verification steps*, *outputs*, and *exit criteria* [SEI02b]. The discrete models, which

are often represented as queueing systems, are capable of capturing a well specified process, which is composed of the above process elements. DES is suitable to model a queueing system, which is observed by arrival rate, service time, queue capability and discipline [BCNN05]. The entities will be moved from one queue to another during simulation.

Discrete-event simulation is a typical method employed in stochastic queueing models. All pre-defined rules, such as arrival rates and service times, will be sampled from the appropriate distributions. At ML3, the organisation's measurement repository has been established and maintained (OPD-SP1.4), and the process asset library has been established and maintained (OPD-SP1.5), and so on. These practices provide the precondition for applying statistical methods and tailoring at the process level.

ML4 aims to achieve a '***quantitatively managed process***'. A critical distinction between a '*well defined process*' and '*quantitatively managed process*' is the predictability of process performance. The latter implies using appropriate statistical techniques to manage process performance so that the future performance can be predicted [SEI02a]. Discrete event modelling tries to answer '*what if*' questions in process changes. The model is run many times with different input variables, entity allocations and statistical distributions. The results are collected and examined to support the '***quantitative project management***' and improve the '***organisational process performance***', which contain all (13) *specific practices* implemented at ML4.

Example. Antoniol *et al.* developed three different queueing models, composed of nodes assessment, technical analysis, enactment and unit testing, to model a software maintenance process [ADD04]. Stochastic discrete simulation was then used to compute the required team size (for the different nodes of each model) under the constraint to complete maintenance activities. In terms of the clearly defined process (which is required for ML3 and above), several simulations were carried out by changing team size (*servers*) for each node until all expected work packets were processed by the deadline. Project staffing levels were then refined to reach a compromise between personnel cost and waiting time.

10.3.6 Transitioning from ML4 to ML5

In the initial version framework, four typical simulation paradigms are introduced at ML1 through ML3 separately. ML4, where this transition starts, is characterised as a ***quantitatively managed*** project and process. After previous transitions along with maturity increase, an ML4 software organisation possesses the competency to employ and institutionalise the above-mentioned four simulation paradigms progressively. At ML4, ***hybrid simulation*** is proposed

by combining multiple simulations and modelling techniques to help organisation achieve continuous optimisation.

Hybrid modelling means not only employing the different modelling paradigms concurrently (which may be implemented prior to ML4), but developing an integrated model with modules created by different paradigms. For example, when pilot process and new technology are considered to implement process improvement (OID-SP1.3), qualitative or semi-quantitative modelling may be employed for the specific module due to the limited knowledge about a specific process; a combination of SD and DES can facilitate the causal analysis of selected defects and other problems (CAR-SP1.2) on project and process levels, and further help the evaluation of changes on process performance (CAR-SP2.2).

Hybrid simulation is encouraged during this transition to select *quantitative* and/or *qualitative* modelling, *continuous* and/or *discrete* simulation, using *vertical* or *horizontal* integration (cf. Chapter 3), to effectively and flexibly capture and simulate the realistic and complex software processes, and to maximise the benefits gained from SPSM.

Example. As an example, Raffo and Setamanit develop a hybrid model that simulates the global software development (GSD) process with the component of staffing process [RS05, SWR07]. The discrete event and system dynamic paradigms compliment each other and together enable the construction of models that capture both the dynamic nature of project variables and the complex sequences of discrete activities that take place. Their hybrid model was implemented using vertical integration. At a high level, their model has three major components: DES sub-model, SD sub-model, and Interaction Effect sub-model. The SD sub-model consists of a global SD sub-model and a site-specific sub-model, which include human resources (HR) modules. The modules deal with HR management, which involves hiring, training, assimilation, and transferring workforce. Whereas the DES sub-models simulate how tasks are allocated and specific activities are performed on site and global scales.

10.4 Related Considerations

Maturity Assessment

As CMMI is a widely-accepted and easily-accessed SPI program, it is chosen here as the premier process maturity model in the current version framework. Nevertheless, the framework is not limited to the organisations with official CMMI appraisal, but also applies to most software organisations, as any of them operates at one particular CMMI maturity level. The framework provides a general and approximate guideline for selecting and adopting process simulation. An organisation can perform self-appraisal, compare the capability characteristics

defined at CMMI maturity levels, and then select the most suitable simulation(s) recommended by the framework at the appropriate maturity level or transition.

Paradigm across Levels

As shown in Figure 10.4, each process simulation paradigm involved in this framework can also be applied at the maturity levels above its introduction level/transition. For instance, when a Level 5 organisation plans to adopt a new technology or a new software process, qualitative simulation may help to gain insight in the possible implication of the change. As another example, the success of a contemporary project might be better defined as a *cube* of metrics than a single point [Ker06]. Since semi-quantitative approach has the inherent capability of coping with uncertainty in multi-dimensions, it can facilitate the decision-making under this condition even for the organisations at higher maturity levels (cf. Chapter 9).

SQSIM and CMMI

Semi-quantitative simulation identifies all possible behaviour consistent with the process constraints and predicts process performance for each behaviour with value ranges. Thus, it is able to provide an outcome that is consistent with an organisation's current process knowledge. Furthermore, as an organisation's process knowledge increases along with the implementation of CMMI (or other SPI) program, the SQSIM models can be refined by introducing narrower value ranges and updated envelope functions to produce more accurate prediction without significant changes in model structure.

Selection Factors

Besides the consideration of an organisation's maturity level, selection and adoption of process simulation paradigms are also related to other constraints, such as expertise with simulation and modelling tools, previous adoption experience. Process simulation should focus on the needs of an organisation in the context of its business environment and the needs of an organisation's current projects. Transition between paradigms is not mutually exclusive to each other, prior introduced and adopted techniques can be retained, optimised, and further integrated with the newly introduced paradigm.

Adoption Process

Along with the evolution of process capability, different process simulation paradigms are introduced into an organisation incrementally and separately. In the staged representation of CMMI model, each maturity level forms a necessary foundation

on which to build the next level, so trying to skip maturity levels is usually counterproductive [SEI02b]. Though organisations can introduce specific simulation paradigms at any time they choose (even before they are ready for advance to the recommended maturity level by the framework), similarly, skipping the adoption of simulation paradigm(s) at lower maturity level(s) is not recommended in this framework. For example, some organisations may try to collect the detailed process data for discrete event simulation, but they are likely to suffer from the inconsistency in processes and measurement definitions.

At present, the initial version framework includes only four typical process simulation paradigms. However, its structure is open and can adapt other simulation paradigms, especially for the paradigms identified in the systematic review, and locate them at appropriate positions in the future.

10.5 Summary

The previous chapters (in Part III and IV) focus on the simulation modelling and application of QSIM/SQSIM in software engineering research and practice. The paradigm scope is extended in this chapter for investigating the adoption of SPSM in a specific organisation, which was seldom discussed in SPSM community before.

1. This chapter argues the interactions between SPSM and (CMMI-based) SPI.
2. It emphasises the important impact of organisation's context on the success of SPSM adoption, and first reveals the positive relation from process maturity model to SPSM for supporting adoption in real world.
3. The initial version of a framework has be proposed by analysing the organisational characteristics on CMMI maturity levels and requirements of the typical process simulation paradigms, and establishing an appropriate mapping between them.
4. The inclusion of qualitative and semi-quantitative simulations in the framework presents the unique value of these approaches in SPSM practice, especially for the organisations at lower maturity levels.
5. The framework provides value especially to software organisations as a primary guideline for selecting and adopting process simulation by assessing their own CMMI maturity levels.

Chapter 11

Discussion & Conclusion

The focus of this thesis is on qualitative/semi-quantitative modelling and simulation of software development processes, which are able to support both software process/project management and process improvement.

As the last chapter of this thesis, the following sections summarise the research achievements of this work, discuss more considerations regarding these approaches, as well as identify the limitations and propose future research.

11.1 Research Achievements

The major achievement of this research lies in the exploration, development, and validation of the use and usefulness of qualitative/semi-quantitative modelling and simulation in software process research and practice. This research produces value on both theoretical and practical aspects. The specific achievements can be summarised as follows:

1. Systematic literature review of the work in software process simulation modelling.

    a) The first systematic literature review reflecting the progress and state-of-the-art of SPSM research over ten years (1998-2007);

    b) Theoretical updates and enhancements for answering essential '*why*', '*what*' and '*how*' questions in SPSM research;

    c) Underlying trends and future directions of SPSM research discovered from the review results;

    d) Guideline and reference derived from the review used as evidence for the design of research in this thesis.

2. Development of a range of qualitative and semi-quantitative software process models for simulation.
    a) Development of the first use of structural qualitative software process models for simulation;
    b) Introduction of the semi-quantitative modelling concept and paradigm in software engineering research by developing software process models for simulation;
    c) Demonstrations of qualitative/semi-quantitative modelling and simulation of software processes in specific domains, at different time scales, and for three levels of purposes;
    d) Development and experiment of qualitative and semi-quantitative simulation models with transitions and iterations at discrete phase level;
    e) Acquisition of new insights in software staffing process by revisiting Brooks' Law with qualitative assumptions and simulation.
3. Comparison between the typical quantitative continuous modelling approach and qualitative/semi-quantitative approaches.
    a) Comparison between casual-loop diagramming and qualitative modelling, and development of model conversion scheme from CLD to ASD;
    b) Introduction and implementation of *first-order* and *higher-order* delays (originated in system dynamics) in QSIM/SQSIM mechanism;
    c) Comparison between system dynamics and qualitative/semi-quantitative simulation from a variety of aspects;
    d) Illustration and discussion of the theoretical and structural equivalence between SD process model and corresponding QSIM/SQSIM process models.
4. Framework for software project and process management based on semi-quantitative modelling and simulation.
    a) Development of SQSIM based software project planning approach for securing project success;
    b) Development of SQSIM based software project control approach in support of process management on the fly;
    c) Development of methods and tools for supporting software project management, such as element-effect table, and control metric table.
5. Identification of the usefulness and unique uses of qualitative and semi-quantitative approaches in SPSM for supporting software process improvement.

a) Identification of the bidirectional interaction and effects between SPSM and SPI;

b) Development of the primary framework for successfully adopting software process simulation in CMMI organisations;

c) Justification of the unique values of QSIM/SQSIM for low maturity software organisations.

11.2 Discussion

11.2.1 Potentials

Apart from the applications of qualitative/semi-quantitative modelling proposed in Part IV, more potential uses of these approaches in software engineering can be further developed. Some are briefly discussed below.

Qualitative Modelling The qualitative modelling (from qualitative assumptions to formal QDEs) presents a basic structure of the process and abstracts its pertinent characteristics. It serves as a vehicle to enhance the qualitative understanding of a specific process, and further the model is able to accommodate the qualitative agreement among the stakeholders involved in development. The qualitative models can also be shared and reused across software organisations, no matter at what maturity level they are (cf. Chapter 10) . Therefore, qualitative modelling can be used to extract and manage common knowledge and insights of software processes at a high level. It may also contribute to model reuse of *continuous* based process simulation, which was identified as one possible direction in Chapter 3.

Furthermore, qualitative modelling and simulation may provide the possibility for dynamical evaluation of semantic aspects of process simulation models, especially for *continuous* models, in the future.

Quantitative Refinement Because a qualitative model reflects the skeleton of the problem under investigation, in most cases, there is no need to amend the qualitative model significantly during refinement. Instead, managers can focus more on the quantitative constraints (with their gradually enriched experience), which improves process prediction and measurement.

Additionally, semi-quantitative constraints provide a means to continuously refine the qualitative model with the improved knowledge or with the emerging certainty in the course of process. Different organisations, even different projects, can develop their own specific quantitative constraints and extensions based on the same qualitative model.

11.2.2 Alternatives

There are two broad categories of methods dealing with uncertainty: *probabilistic* and *non-probabilistic* [GB91]. Semi-quantitative modelling falls into the latter. This subsection compares semi-quantitative modelling and simulation with other typical methods used in handling uncertainty in software engineering.

Monte Carlo Method Monte Carlo simulation is one popular quantitative method for analysing software process. Although it takes many samples of the value range, unfortunately, they are still a finite set. Thus, this method cannot guarantee all possibilities fall into their solution. This problem turns to be more subtle when more factors change simultaneously, which may result in missing some important behaviours. Furthermore, the computational cost of using Monte Carlo method dramatically increases when dealing with a large variable space. In contrast, the cost of semi-quantitative simulation does not depend on the size of the variable space. It is a function of the number of distinct qualitative behaviours predicted [Kui94]. Moreover, semi-quantitative modelling allows for the existence of many uncertain elements without needing to assume their statistical distribution over the range.

Statistical Process Control This control technique (SPC), as described by Florac and Carleton [FC99], provides useful method and tool to improve the controllability of development process. Graphic tools, e.g. control charts and capability histograms, are used to analyse the process. The applicability of SPC is independent of the life cycle model and development methodology. However, it does require that a rigorous measurement process be instituted with the development process. In addition, when the number of parameters that affect the process is relatively high, the combinations of possible values are even higher and analysis of all alternatives becomes very difficult, if not impractical.

Fuzzy Logic As another non-probabilistic method, Fuzzy logic provides a typical set-valued quantitative approach. It describes the real world system with fuzzy set, which is a fuzzy subset of the universe of discourse [GB91]. It applies a rough boundary to handle the uncertainty, and the mapping to fuzzy set is in an arbitrary way, linear or nonlinear. In contrast, semi-quantitative modelling describes the system boundary with real numeric values, which maintains the precision while coping with uncertainty.

Each approach possesses its advantages and limitations. The selection depends on the user's capability and requirements. Semi-quantitative modelling performs modelling and simulation by refinement: define a set of possible solution, and shrink it by cutting out the illegal or illogical behaviours. This approach guar-

antees integrity of the solution. In addition, it produces not only the final states of project, but all possible process behaviours (routes) consistent with the quantitative constraints, i.e. value ranges and envelope functions, which can be used for in-progress process tracking.

Semi-quantitative modelling still requires the process to be observed and measured quantitatively. Plus, it does not mean assigning a value rang to every parameter, nor an envelope to every equation. It provides a manager with an alternative modelling approach and flexibility in managing process. The capability to apply finer intervals reflects the organisation's maturity.

11.3 Limitations & Future Work

11.3.1 Limitations

This subsection discusses the limitations from two aspects: the approach and the supporting tools, as well as the research reported in this thesis.

Approach

Despite of the power of *qualitative* and *semi-quantitative* approaches in dealing with complexity and uncertainty of software processes, currently, some weakness of theses proposed approaches still remain and limit their use in the following aspects:

- Ease of use: A QDE has to be defined by Common Lisp formally, which assumes the modeler possessing the basic knowledge of AI and LISP programming language. Thus, it is often time-consuming to program and to debug a QSIM/SQIM model.
- Interoperability: As the simulation engines, QSIM and Q2 do not allow the interoperation during the run of simulation, which implies the modeler cannot make change in the light of the live progress of simulation.
- Development environment: There is no visual integrated development environment (IDE) available for semi-quantitative simulation, which hinders the rapid modelling for SQSIM.
- Granularity: Though QSIM/SQSIM offer some features for discrete modelling, such as model transition and iteration (cf. Chapter 7), currently, their use focuses on continuous process modelling and macro-process research.
- Integratability: Recently, QSIM/SQSIM strongly rely on reasoning techniques and AI programming language (e.g. LISP), which leads to the difficulty in integration with conventional simulation paradigms.

Thesis

Though the research questions addressed in Chapter 1 have been tested through this thesis, some limitations of this research still remain.

Apart from the above limitations of the proposed approaches, as few qualitative/semi-quantitative software process models were developed prior to this research, the diversity of process models using them need to be further enriched. On the other hand, the QSIM/SQSIM process models are desired to validated directly with empirical data.

The distinct and unique characteristics of qualitative/semi-quantitative modelling presented in this thesis determines the difference in model evaluation from conventional quantitative approaches. A systematic and theoretical model evaluation method is as yet unavailable.

In addition, considering the novelty of these approaches in software engineering domain, they currently still lack the empirical evidence from practice for supporting and improving the methodology and framework developed in this thesis.

11.3.2 Future Work

In spite of the existence of the limitations, qualitative and semi-quantitative approaches proposed in this thesis offer a number of interesting perspectives and promising features for modelling and simulating software processes. Possible future work in this direction can be identified but not limited to:

1. Modelling and investigation of more software processes: This thesis models three typical but distinct software development processes at different scopes. In the future, more software processes need to be modelled for identifying the fitness and appropriateness of qualitative and semi-quantitative approaches.

2. Experiment using alternative implementation schemes: QSIM and its extension Q2 are the implementation scheme applied in this research, which has its advantages and weaknesses. It is worthwhile to investigate the alternative implementation schemes fitting the specific needs in software process research, such as the Garp3 workbench for qualitative modelling and simulation.

3. Integration with conventional quantitative approaches: Developing specific interfaces for information conversion and execution transition between QSIM/SQSIM and conventional quantitative approaches for constructing hybrid/integrated process simulation models.

4. Implementation of discrete process modelling: Qualitative modelling and simulation offers some basic discrete characteristics. Experimenting and

implementing (partial) discrete or hybrid modelling of software process for enhancing its adaptability.

5. Development of innovative uses: Developing new uses based on the capabilities offered by qualitative and semi-quantitative approaches, such as the abstraction and management of software process knowledge for model reuse, and semantic evaluation of process simulation models.

6. Development of integrated tool kits for management: In order to implement semi-quantitative simulation based software project/process management and encourage its use in practice, an integrated tool kits need to be developed for executing automatic simulation over iterations and generating the artifacts.

Bibliography

[ADD04] Giuliano Antoniol, Giuseppe A. DiLucca, and Massimiliano DiPenta. Assessing staffing needs for a software maintenance project through queuing simulation. *IEEE Transactions on Software Engineering*, 30(1), 2004. [cited at p. 178]

[AEPR08] Ahmed Al-Emran, Dietmar Pfahl, and Gunther Ruhe. A method for replanning of software releases using discrete-event simulation. *Software Process: Improvement and Practice*, 13(1):19–33, 2008. [cited at p. 43]

[AH89] Tarek K Abdel-Hamid. The dynamics of software project staffing: A system dynamics based simulation approach. *IEEE Transactions on Software Engineering*, 15(2), 1989. [cited at p. 84]

[AHM91] Tarek K Abdel-Hamid and Stuart E Madnick. *Software Project Dynamics: An Integrated Approach*. Prentice Hall, Englewood Cliffs, N.J., 1991. [cited at p. 5, 20, 24, 27, 42, 84, 88, 91, 92, 98, 102, 111, 113, 118, 158, 165, 177]

[BAB+00] Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer, and Bert Steece. *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000. [cited at p. 20]

[BCNN05] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, Englewood Cliffs, NJ, 4 edition, 2005. [cited at p. 23, 43, 173, 178]

[BDVW06] Marcio de O. Barros, Alexandre R. Dantas, Gustavo O. Veronese, and Claudia M. L. Werner. Model-driven game development: Experience and model enhancements in software project management education. *Software Process: Improvement and Practice*, 11(4):411–421, 2006. [cited at p. 46]

[Bec00] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2000. [cited at p. 19]

[BF04] Bert Bredeweg and Kenneth D. Forbus. Qualitative modeling in education. *AI Magazine*, 24(4):35–46, 2004. [cited at p. 71]

[BKHV97] Ulrike Becker-Kornstaedt, Dirk Hamann, and Martin Verlage. Descriptive modeling of software processes. In *3rd Conference on Software Process Improvement*, Barcelona, Spain, 1997. [cited at p. 18]

[Boe81] Barry Boehm. *Software Engineering Economics*. Prentice Hall, Englewood, Cliffs, NJ, 1981. [cited at p. 20, 38, 110]

[Boe86] Barry Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4), 1986. [cited at p. 19]

[Bro95] Frederick P. Brooks. *The Mythical Manmonth: Essays on Software Engineering*. Addison-Wesley Longman, anniversary edition, 1995. [cited at p. 3, 84, 85, 87, 92, 175]

[BS03] Bert Bredeweg and Peter Struss. Current topics in qualitative reasoning. *AI Magazine*, 24(4), 2003. [cited at p. 59]

[BSB01] Gautam Biswas, Daniel Schwartz, and John Bransford. Technology support for complex problem solving: From sad environments to ai. In Kenneth D. Forbus and Paul J. Feltovich, editors, *Smart Machines in Education: The Coming Revolution in Educational Technology*. AAAI Press, 2001. [cited at p. 71]

[BSBL05] Bert Bredeweg, Paulo Salles, Anders Bouwer, and Jochem Liem. Framework for conceptual qr description of case studies. Technical report, Human Computer Studies (HCS) laboratory, University of Amsterdam, 2005. [cited at p. 71]

[Car03] John S. II Carson. Introduction to modeling and simulation. In *Winter Simulation Conference (WSC)*, pages 7–13, New Orleans, LA, 2003. IEEE Press. [cited at p. 21, 23]

[CBK06] KeungSik Choi, Doo-Hwan Bae, and TagGon Kim. An approach to a hybrid software process simulation using the devs formalism. *Software Process: Improvement and Practice*, 11(4):373–383, 2006. [cited at p. 52]

[CDM02] Joao W. Cangussu, Raymond A. DeCarlo, and Aditya P. Mathur. A formal model of the software test process. *IEEE Transactions on Software Engineering*, 28(8), 2002. [cited at p. 112]

[CGC06] Yu Chen, Gerald C. Gannod, and James S. Collofello. A software product line process simulator. *Software Process: Improvement and Practice*, 11(4):385–409, 2006. [cited at p. 43]

[Chr99] Alan M. Christie. Simulation in support of cmm-based process improvement. *Journal of Systems and Software*, 46(2/3), 1999. [cited at p. 166, 171]

[CKO92] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9), 1992. [cited at p. 11, 17, 20]

[Cla02] Terry A. Clark. *Project Management for Planners: A Practical Guide*. Planners Press, American Planning Association, 2002. [cited at p. 153]

[CLRW00] B.W. Chatters, Manny Lehman, Juan F. Ramil, and Paul Wernick. Modelling a software evolution process: A long-term case study. *Software Process: Improvement and Practice*, 5(2-3), 2000. [cited at p. 128, 134, 135, 136]

[Coc02] Alistair Cockburn. *Agile Software Development.* Addison-Wesley, 2002. [cited at p. 19]

[Cox90] Brad J. Cox. Planning the software industrial revolution. *IEEE Software*, 7(6):25–33, 1990. [cited at p. 1]

[Coy96] R.G. Coyle. *System Dynamics Modelling: A Practical Approach.* Chapman & Hall/CRC, 1996. [cited at p. 42]

[DeM82] Tom DeMarco. *Controlling Software Projects: Management, Measurement and Estimation.* Yourdon Press, New York, 1982. [cited at p. 151, 156]

[DHK+07] Paul Davidsson, Johan Holmgren, Hans Kyhlback, Dawit Mengistu, and Marie Persson. Applications of agent based simulation. In *International-Workshop on Multi-Agent-Based Simulation (MABS'06)*, pages 15–27, Hakodate, Japan, 2007. Springer-Verlag. [cited at p. 46]

[DKJ05] T. Dyba, Barbara Kitchenham, and M. Jorgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):158–165, 2005. [cited at p. 28]

[DL99] Tom DeMarco and Timothy Lister. *Peopleware.* Dorset House, 2nd edition, 1999. [cited at p. 3]

[ES05] Jacky Estublier and Sonia Sanlaville. Business processes and workflow coordination of web services. In *IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE'05)*, pages 85–88, Hong Kong, China, 2005. IEEE Computer Society. [cited at p. 14]

[EVL+03] Jacky Estublier, Jorge Villalobos, Anh-Tuyet Le, Sonia Sanlaville, and Germn Vega. An approach and framework for extensible process support system. In *9th European Workshop on Software Process Technology (EWSPT'03)*, Helsinki, Finland, 2003. Springer. [cited at p. 21]

[FC99] William A. Florac and Anita D. Careton. *Measuring the Software Process: Statistical Process Control for Software Process Improvement.* Addison-Wesley, 1999. [cited at p. 151, 186]

[FKRT94] Adam Farquhar, Benjamin Kuipers, Jeff Rickel, and David Throop. Qsim: The program and its use. Technical report, Department of Computer Sciences, University of Texas, 1994. [cited at p. 71, 78]

[FMN+04] Norman E. Fenton, William Marsh, Martin Neil, Patrick Cates, Simon Forey, and Manesh Tailor. Making resource decisions for software projects. In *26th International Conference on Software Engineering (ICSE'04)*, pages 397–406, Edinburgh, UK, 2004. IEEE Computer Society. [cited at p. 20]

[FNM+07] Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Paul Krause, and Rajat Mishra. Predicting software defects in varying development lifecycles using bayesian nets. *Information & Software Technology*, 49:32–43, 2007. [cited at p. 20]

[For69] Jay W. Forrester. *Industrial Dynamics*. System Dynamics Series. Pegasus Communications, 1969. [cited at p. 42, 130]

[FP97] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS, 2nd edition, 1997. [cited at p. 11, 17]

[Gal04] Daniel Galin. *Software Quality Assurance: from Theory to Implementation*. Pearson, 2004. [cited at p. 114]

[GB91] Jerzy W. Grzymala-Busse. *Managing Uncertainty in Expert Systems*. Kluwer Academic Publishers, 1991. [cited at p. 186]

[HMK+94] Volkmar Haase, Richard Messnarz, Gunter Koch, Hans Jrgen Kugler, and Paul Decrinis. Bootstrap: Fine-tuning process assessment. *IEEE Software*, 11(4):25 – 35, 1994. [cited at p. 16]

[Hou00] Daniel Xavier Houston. *A Software Project Simulation Model for Risk Management*. PhD thesis, Arizona State University, 2000. [cited at p. 20, 24]

[HRD+01] Martin Host, Bjorn Regnell, Johan Natt och Dag, Josef Nedstam, and Christian Nyberg. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *Journal of Systems and Software*, 59(3):323–332, 2001. [cited at p. 43]

[HS00] Mikel Harry and Richard Schroeder. *Six Sigma*. Random House, 2000. [cited at p. 16]

[HSS86] Karen E. Huff, Joan V. Sroka, and Dennis D. Struble. Quantitative models for managing software development processes. *Software Engineering Journal*, 1(1):17–23, 1986. [cited at p. 112, 116, 119]

[HZJ06] Ming Huo, He Zhang, and Ross Jeffery. A systematic approach to process enactment analysis as input to software process improvement or tailoring. In *13th Asia-Pacific Software Engineering Conference (APSEC'06)*, pages 401–408, Bangalore, 2006. IEEE Computer Society. [cited at p. 21]

[HZJ08] Ming Huo, He Zhang, and Ross Jeffery. Detection of consistent patterns from process enactment data. In *International Conference on Software Process (ICSP'08)*, pages 174–185, Leipzig, Germany, 2008. Springer. [cited at p. 21]

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Object Technology Series. Addison-Wesley, 1999. [cited at p. 19]

[Jef87] Ross D. Jeffery. Time-sensitive cost models in the commercial mis environment. *IEEE Transactions on Software Engineering*, SE-13(7), 1987. [cited at p. 103]

[Jef06] Ross Jeffery. Exploring the business process-software process relationship. In *Software Process Workshop/International Workshop on Software Process Simulation and Modeling (SPW/ProSim)*, pages 11–14, Shanghai, China, 2006. Spinger. [cited at p. 14]

[Jon75] Capers Jones. Programming defect removal. In *GUIDE 40*, 1975. [cited at p. 110]

[Kar01] Even-Andre Karlsson. Incremental development - terminology and guidelines. In Shi-Kuo Chang, editor, *Handbook of Software Engineering and Knowledge Engineering*, volume 1. World Scientific, 2001. [cited at p. 110]

[Kas04] Tim Kasse. *Practical Insight into CMMI.* Artech House, 2004. [cited at p. 168]

[KDJ04] Barbara Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. In *26th International Conference on Software Engineering, (ICSE'04)*. IEEE Computer Society, 2004. [cited at p. 28]

[Kel88] Marc I. Kellner. Modeling the software maintenance process: Analytic summary models. In *Conference on Software Maintenance*, pages 279–283, Phenix, AZ, 1988. IEEE Computer Society. [cited at p. 19]

[Ker05] Harold Kerzner. *Using the Project Management Maturity Model: Strategic Planning for Project Management.* John Wiley & Sons, 2nd edition, 2005. [cited at p. 149]

[Ker06] Harold Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling.* John Wiley & Sons, 9th edition, 2006. [cited at p. 150, 180]

[KFF+91] Marc I. Kellner, Peter H. Feiler, Anthony Finkelstein, Takuya Katayama, Leon J. Osterweil, Maria H. Penedo, and H. Dieter Rombach. Ispw-6 software process example. In *6th International Software Process Workshop (ISPW)*, pages 176–186, Hakodate, Hokkaido, Japan, 1991. IEEE Computer Society. [cited at p. 24]

[Kit04] Barbara Kitchenham. Procedures for undertaking systematic reviews. Technical report, Computer Science Department, Keele University and National ICT Australia, 2004. [cited at p. 28]

[Kit07] Barbara Kitchenham. Guidelines for performing systematic literature reviews in software engineering. Technical report, Software Engineering Group, School of Computer Science and Mathematics, Keele University, and Department of Computer Science, University of Durham, April 2007. [cited at p. 28, 29]

[Kle08] Jack P.C. Kleijnen. *Design and Analysis of Simulation Experiments.* International Series in Operations Research and management Science. Springer Science+Business Media, 2008. [cited at p. 18]

[KLRW01] Goel Kahen, Manny Lehman, Juan F. Ramil, and Paul Wernick. System dynamics modeling of software evolution processes for policy investigation: Approach and example. *Journal of Systems and Software*, 59(3), 2001. [cited at p. 128, 134, 136]

[KMR99] Marc I. Kellner, Raymond J. Madachy, and David M. Raffo. Software process simulation modeling: Why? what? how? *Journal of Systems and Software*, 46(2/3), 1999. [cited at p. 28, 36, 41, 47, 165, 167]

[Kui94] Benjamin Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, 1994. [cited at p. 59, 62, 69, 71, 78, 145, 186]

[Kui01] Benjamin Kuipers. Qualitative simulation. In R.A Meyers, editor, *Encyclopedia of Physical Science and Technology*, pages 287–300. Academic Press, NY, 2001. [cited at p. 59]

[Li05] Mingshu Li. Expanding the horizons of software development processes: A 3-d integrated methodology. In *Software Process Workshop (SPW'05)*, pages 54–67, Beijing, China, 2005. Springer. [cited at p. 12]

[Li07] Mingshu Li. Triso-model: A new approach to integrated software process assessment and improvement. *Software Process: Improvement and Practice*, 12(5):387–398, 2007. [cited at p. 12]

[LK00] Averill M. Law and W.David Kelton. *Simulation Modelling and Analysis*. McGraw Hill, 3rd edition, 2000. [cited at p. 44]

[LR03] Manny Lehman and Juan F. Ramil. Software evolution - background, theory, practice. *Information Processing Letters*, 88, 2003. [cited at p. 133]

[Mad94] Raymond Joseph Madachy. *A Software Project Dynamics Model for Process Cost, Schedule, and Risk Assessment*. PhD thesis, University of Southern California, 1994. [cited at p. 20, 24]

[Mad08] Raymond J. Madachy. *Software Process Dynamics*. Wiley-IEEE Press, 2008. [cited at p. 85, 177]

[Mak92] Hing-Yin Mak. *System Dynamics and Discrete Event Simulation Modelling*. PhD thesis, University of London, 1992. [cited at p. 44]

[Mar02] Robert Hayne Martin. *A Hybrid Model for the Software Development Process*. PhD thesis, Portland State University, 2002. [cited at p. 21, 25]

[MH86] H.J. McNeil and K.O. Hartley. Project planning and performance. *Project Management Journal*, March:36–44, 1986. [cited at p. 153]

[MT00] Ray Madachy and Denton Tarbet. Case studies in software process modeling with system dynamics. *Software Process: Improvement and Practice*, 5(2-3):133–146, 2000. [cited at p. 91, 103]

[NH05] Emily Oh Navarro and Andre van der Hoek. Software process modeling for an educational software engineering simulation game. *Software Process: Improvement and Practice*, 10(3):311–325, 2005. [cited at p. 46]

[Ost05] Leon J. Osterweil. Unifying microprocess and macroprocess research. In *Software Process Workshop (SPW)*, Beijing, 2005. Springer. [cited at p. 13, 14, 49]

[Ost07] Leon J. Osterweil. What we learn from the study of ubiquitous processes. *Software Process: Improvement and Practice*, 12(5):399–414, 2007. [cited at p. 50]

[PA06] Shari Lawrence Pfleeger and Joanne M. Atlee. *Software Engineering: Theory and Practice*. Prentice Hall, 3rd edition, 2006. [cited at p. 11]

[Pad02] Frank Padberg. A discrete simulation model for assessing software project scheduling policies. *Software Process: Improvement and Practice*, 7(3-4), 2002. [cited at p. 43]

[PCCW93] M. Paulk, B. Curtis, M. Chrissis, and C. Weber. Capability maturity model for software (version 1.1). Technical report, Software Engineering Institute (SEI), 1993. [cited at p. 16]

[Pfa01] Dietmar Pfahl. *An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations*. PhD thesis, University of Kaiserslautern, 2001. [cited at p. 12, 20, 25]

[Pid04] Michael Pidd. *Computer Simulation in Management Science*. Wiley, 5th edition, 2004. [cited at p. 43]

[PMI04] PMI. *A Guide to the Project Management Body of Knowledge*. Project Management Institute, 3rd edition, 2004. [cited at p. 36, 153]

[Pre05] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 6th edition, 2005. [cited at p. 11, 12, 15]

[Put80] Lawrence H. Putnam. *Software Cost Estimating and Life-Cycle Control: getting the Software Numbers*. Computer Society Press, 1980. [cited at p. 20]

[QRG] UT Qualitative Reasoning Group. Qsim. http://www.cs.utexas.edu/users/qr/QR-software.html. University of Texas. [cited at p. 71]

[Raf96] David Mitchell Raffo. *Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes on Process Performance*. PhD thesis, Carnegie Mellon University, 1996. [cited at p. 21, 24, 43]

[Reh96] H. Rehessar. Project management success factors. Technical Report CAESAR Technical Report 96/01, University of New South Wales, 1996. [cited at p. 153]

[Roy70] Winston W. Royce. Managing the development of large software systems: Concepts and techniques. In *IEEE WESCON*, pages 1–9, 1970. [cited at p. 18]

[RS02] Juan F. Ramil and Neil Smith. Qualitative simulation of models of software evolution. *Software Process: Improvement and Practice*, 7(3-4), 2002. [cited at p. 45, 142]

[RS05] David M. Raffo and Siri-on Setamanit. A simulation model for global software development project. In *International Workshop on Software Process Simulation and Modeling (ProSim)*, St. Louis, MO, 2005. [cited at p. 179]

[Rus98] Ioana Rus. *Modeling the Impact on Cost and Schedule on Software Quality Engineering Practices*. PhD thesis, Arizona State University, 1998. [cited at p. 20, 24]

[RV95] H. Dieter Rombach and Martin Verlage. Directions in software process research. *Advances in Computers*, 41:1–63, 1995. [cited at p. 14]

[RVM99] David M. Raffo, Joseph V. Vandeville, and Robert H. Martin. Software process simulation to achieve higher cmm levels. *Journal of Systems and Software*, 46(2/3), 1999. [cited at p. 166, 171]

[SAGO02] Antonio J. Suarez, Pedro J. Abad, Rafael M. Gasca, and Juan A. Ortega. Qualitative simulation of human resources subsystem in software development projects. In *16th International Workshop on Qualitative Reasoning*, Sitges, Spain, 2002. [cited at p. 45]

[SB01] Ken Schwaber and Mike Beedle. *Agile Software Development with SCRUM*. Prentice Hall, 2001. [cited at p. 19, 50]

[SC06] Benjamin Stopford and Steve Counsell. Simulating the structural evolution of software. In *Software Process Workshop/International Workshop on Software Process Simulation and Modeling (SPW/ProSim'06)*, pages 294–301, Shanghai, China, 2006. Springer-Verlag. [cited at p. 46]

[SCR06] Neil Smith, Andrea Capiluppi, and Juan Fernandez Ramil. Agent-based simulation of open source evolution. *Software Process: Improvement and Practice*, 11(4):423–434, 2006. [cited at p. 46]

[SEI02a] CMMI Product Team SEI. Capability maturity model integration (cmmise/sw/ippd, v1.1), continuous representation. Technical report, Software Engineering Institute, Carnegie Mellon University, 2002. [cited at p. 16, 168, 170, 175, 178]

[SEI02b] CMMI Product Team SEI. Capability maturity model integration (cmmise/sw/ippd, v1.1), staged representation. Technical report, Software Engineering Institute, Carnegie Mellon University, 2002. [cited at p. 16, 168, 170, 174, 175, 177, 181]

[SGI98] The chaos report 1998. Technical report, Standish Group International, 1998. [cited at p. 1]

[SGI06] The chaos report 2006. Technical report, Standish Group International, 2006. [cited at p. 1]

[Som07] Ian Sommerville. *Software Engineering*. Addison-Wesley, 8th edition, 2007. [cited at p. 12, 15, 18]

[SS97] Martin Shepperd and Chris Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(11):736–743, 1997. [cited at p. 20]

[Sta97] Jennifer Stapleton. *DSDM Dynamic Systems Development Method: The Method in Practice*. Addison Wesley, 1997. [cited at p. 19]

[Stu94] R. D. Stutzke. A mathematical expression of brooks's law. In *9th International Forum on COCOMO and Cost Modeling*, Los Angeles, 1994. [cited at p. 85]

[SWR07] Siri-on Setamanit, Wayne Wakeland, and David Raffo. Using simulation to evaluate global software development task allocation strategies. *Software Process: Improvement and Practice*, 12(5):491–503, 2007. [cited at p. 52, 179]

[Tur96] Wladyslaw M. Turski. The reference model for smooth growth of software systems. *IEEE Transactions on Software Engineering*, 22(8), 1996. [cited at p. 134, 135]

[Tur02] Wladyslaw M. Turski. The reference model for smooth growth of software systems revisitied. *IEEE Transactions on Software Engineering*, 28(8), 2002. [cited at p. 134, 135]

[Tve96] John Douglas Tvedt. *An Extensive Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time*. PhD thesis, Arizona State University, 1996. [cited at p. 20, 24, 112, 121]

[Wal94] Ernest Wallmuller. *Software Quality Assurance: A Practical Approach*. Prentice Hall, 1994. [cited at p. 15]

[WCL+00] Alexander Wise, Aaron G. Cass, Barbara Staudt Lerner, Eric K. McCall, Leon J. Osterweil, and Stanley M. Sutton. Using little-jil to coordinate agents in software engineering. In *Automated Software Engineering Conference (ASE)*, Grenoble, France, 2000. [cited at p. 21]

[WH02] Paul Wernick and Tracy Hall. Simulating global software evolution processes by combining simple models: An initial study. *Software Process: Improvement and Practice*, 7(3-4), 2002. [cited at p. 128, 133, 134, 136, 140]

[WH04] Paul Wernick and Tracy Hall. A policy investigation model for long-term software evolution processes. In *5th International Workshop on Software Process Simulation Modeling (ProSim'04)*, pages 206–214, Edinburgh, Scotland, 2004. [cited at p. 128, 133, 136, 140]

[WL99] Paul Wernick and Manny Lehman. Software process white box modelling for feast/1. *Journal of Systems and Software*, 46(2-3), 1999. [cited at p. 128, 134, 135]

[Woo02] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002. [cited at p. 45, 50]

[YP06] Levent Yilmaz and Jared Phillips. Organization-theoretic perspective for simulation modeling of agile software processes. In *Software Process Workshop/International Workshop on Software Process Simulation and Modeling (SPW/ProSim'06)*, pages 234–241, Shanghai, China, 2006. Springer-Verlag. [cited at p. 46]

[ZHKJ06] He Zhang, Ming Huo, Barbara Kitchenham, and Ross Jeffery. Qualitative simulation model for software engineering process. In *17th Australian Software Engineering Conference (ASWEC'06)*, pages 391–400, Sydney, Australia, 2006. IEEE Computer Society. [cited at p. 83]

[ZJZ08] He Zhang, Ross Jeffery, and Liming Zhu. Hybrid modeling of test-and-fix processes in incremental development. In *International Conference on Software Process (ICSP'08)*, pages 333–344, Leipzig, Germany, 2008. Springer. [cited at p. 52, 125]

[ZK06] He Zhang and Barbara. Kitchenham. Semi-quantitative simulation modeling of software engineering process. In *Software Process Workshop/International Workshop on Software Process Simulation and Modeling (SPW/ProSim'06)*, pages 242–253, Shanghai, China, 2006. Springer. [cited at p. 83]

[ZKJ07a] He Zhang, Barbara Kitchenham, and Ross Jeffery. Achieving software project success: A semi-quantitative approach. In *International Conference on Software Process (ICSP'07)*, pages 332–343, Minneapolis, MN, 2007. Springer. [cited at p. 149]

[ZKJ07b] He Zhang, Barbara Kitchenham, and Ross Jeffery. A framework for adopting software process simulation in cmmi organizations. In *International Conference on Software Process (ICSP'07)*, pages 320–331, Minneapolis, MN, 2007. Springer. [cited at p. 166]

[ZKJ07c] He Zhang, Barbara Kitchenham, and Ross Jeffery. Planning software project success with semi-quantitative reasoning. In *18th Australian Software Engineering Conference (ASWEC'07)*, pages 369–378, Melbourne, Australia, 2007. IEEE Computer Society. [cited at p. 149]

[ZKJ07d] He Zhang, Barbara Kitchenham, and Ross Jeffery. A semiq model of test-and-fix process of incremental development. In *1st International Workshop on Software Productivity Analysis and Cost Estimation (SPACE'07)*, pages 23–29, Nagoya, Japan, 2007. Information Processing Society (IPS) of Japan. [cited at p. 109]

[ZKKJ08] He Zhang, Jacky Keung, Barbara Kitchenham, and Ross Jeffery. Semi-quantitative modeling for managing software development processes. In *19th Australian Software Engineering Conference (ASWEC'08)*, pages 66–75, Perth, Australia, 2008. IEEE Computer Society. [cited at p. 109]

[ZKP08a] He Zhang, Barbara Kitchenham, and Dietmar Pfahl. Reflections on 10 years of software process simulation modeling: A systematic review. In *International Conference on Software Process (ICSP'08)*, pages 345–356, Leipzig, Germany, 2008. Springer. [cited at p. 24, 27]

[ZKP08b] He Zhang, Barbara Kitchenham, and Dietmar Pfahl. Software process simulation modeling: Facts, trends, and directions. In *15th Asia-Pacific Software Engineering Conference (APSEC'08)*, Beijing, China, 2008. IEEE Computer Society. [cited at p. 27]

[ZKP08c] He Zhang, Barbara Kitchenham, and Dietmar Pfahl. Software process simulation over decade: Trends discovery from a systematic review. In *2nd International Symposium on Empirical Software Engineering and Measurement (ESEM'08)*, Kaiserslautern, Germany, 2008. ACM. [cited at p. 27]

Appendices

Appendix A

Supplements for Systematic Literature Review

A.1 List of Primary Studies (Stage 1)

ProSim 1998

[1] Marc I. Kellner, Raymond J. Madachy, David M. Raffo. Software Process Simulation Modeling: Why? What? How? *Journal of Systems and Software*, 46(2-3), 1999. - [B*]

[2] Alan M. Christie. Simulation in Support of CMM-Based Process Improvement. *Journal of Systems and Software*, 46(2-3), 1999. - [C]

[3] A. Drappa, J. Ludewig. Quantitative modeling for the interactive simulation of software projects. *Journal of Systems and Software*, 46(2-3), 1999. - [A/B]

[4] M. M. Lehman, J. F. Ramil. The impact of feedback in the global software process. *Journal of Systems and Software*, 46(2-3), 1999. - [D]

[5] Dietmar Pfahl, Karl Lebsanft. Integration of system dynamics modeling with descriptive process modeling and goal-oriented measurement. *Journal of Systems and Software*, 46(2-3), 1999. - [B/C/D]

[6] Antony Powell, Keith Mander, Duncan Brown. Strategies for lifecycle concurrency and iteration CA system dynamics approach. *Journal of Systems and Software*, 46(2-3), 1999. - [A]

[7] David M. Raffo, Joseph V. Vandeville, Robert H. Martin. Software process simulation to achieve higher CMM levels. *Journal of Systems and Software*, 46(2-3), 1999. - [D]

[8] Ioana Rus, James Collofello, Peter Lakey. Software process simulation for reliability management. *Journal of Systems and Software*, 46(2-3), 1999. - [A]

*Indicator of study category, the definition cf. Chapter 3

[9] Walt Scacchi. Experience with software process simulation and modeling. *Journal of Systems and Software*, 46(2-3), 1999. - [D]

[10] Pual Wernick, M. M. Lehman. Software process white box modelling for FEAST/1. *Journal of Systems and Software*, 46(2-3), 1999. - [A]

[11] Judson Williford, Andrew Chang. Modeling the FedEx IT division: a system dynamics approach to strategic IT planning. *Journal of Systems and Software*, 46(2-3), 1999. - [A/D]

[12] Dundar Kocaoglu, Robert Martin, David Raffo. Moving Toward a Unified Model for Software Development. *1st International Workshop on Software Process Simulation and Modeling (ProSim)*, 1998. - [B]

[13] Ana Martinez-Garcia, Brian Warboys. From RADs to DESs: A Mapping from Process Models to Discrete Event Simulation. *1st International Workshop on Software Process Simulation and Modeling (ProSim)*, 1998. - [B]

ProSim 1999

[14] B. W. Chatters, M. M. Lehman, J. F. Ramil, P. Wernick. Modelling a software evolution process: a long-term case study. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [D]

[15] Alan M. Christie, Mary Jo Staley. Organizational and social simulation of a software requirements development process. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [A]

[16] Peter Henderson, Yvonne Howard. Simulating a process strategy for large scale software development using systems dynamics. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [A]

[17] Ray Madachy, Denton Tarbet. Case studies in software process modeling with system dynamics. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [A/D]

[18] Robert H. Martin, David Raffo. A model of the software development process using both continuous and discrete models. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [A/B]

[19] David Raffo, Warren Harrison, Joseph Vandeville. Coordinating models and metrics to manage software projects. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [C/D]

[20] Stephen T. Roehling, James S. Collofello, Brian G. Hermann, Dwight E. Smith-Daniels. System dynamics modeling applied to software outsourcing decision support. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [A]

[21] Walt Scacchi. Understanding software process redesign using modeling, analysis and simulation. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [C]

[22] Friedrich Stallinger. Software process simulation to support ISO/IEC 15504 based software process improvement. *Software Process: Improvement and Practice*, 5(2-3), 2000. - [A]

ProSim 2000

[23] Paolo Donzelli, Giuseppe Iazeolla. Hybrid simulation modelling of the software process. *Journal of Systems and Software*, 59(3), 2001. - [A]

[24] Robert Martin, David Raffo. Application of a hybrid process simulation model to a software development project. *Journal of Systems and Software*, 59(3), 2001. - [A]

[25] Dan X. Houston, Gerald T. Mackulak, James S. Collofello. Stochastic simulation of risk factor potential effects for software development risk management. *Journal of Systems and Software*, 59(3), 2001. - [A]

[26] Dan X. Houston, Susan Ferreira, James S. Collofello, Douglas C. Montgomery, Gerald T. Mackulak. Behavioral characterization: finding and using the influential factors in software process simulation models. *Journal of Systems and Software*, 59(3), 2001. - [D]

[27] G. Kahen, M.M. Lehman, J.F. Ramil, P. Wernick. System dynamics modelling of software evolution processes for policy investigation: approach and example. *Journal of Systems and Software*, 59(3), 2001. - [A]

[28] Dietmar Pfahl, Marco Klemm, Gunther Ruhe. A CBT module with integrated simulation component for software project management education and training. *Journal of Systems and Software*, 59(3), 2001. - [A/D]

[29] Mercedes Ruiz, Isabel Ramos, Miguel Toro. A simplified model of software project dynamics. *Journal of Systems and Software*, 59(3), 2001. - [A]

[30] Friedrich Stallinger, Paul Grunbacher. System dynamics modelling and simulation of collaborative requirements engineering. *Journal of Systems and Software*, 59(3), 2001. *Journal of Systems and Software*, 59(3), 2001. - [A]

[31] Martin Host, Bjorn Regnell, Johan Natt och Dag, Josef Nedstam, Christian Nyberg. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *Journal of Systems and Software*, 59(3), 2001. - [A]

[32] S. James Choi, Walt Scacchi. Modeling and simulation software acquisition process architectures. *Journal of Systems and Software*, 59(3), 2001. - [A/B]

[33] Volker Gruhn, Ursula Wellen. Analysing a process landscape by simulation. *Journal of Systems and Software*, 59(3), 2001. - [B]

[34] Peter Henderson, Yvonne Margaret Howard, Robert John Walters. A tool for evaluation of the software development process. *Journal of Systems and Software*, 59(3), 2001. - [B]

[35] Alan M. Christie, David A. Fisher. Simulating the Emergent Behavior of Complex Software-Intensive Organizations. *3rd International Workshop on Software Process Simulation and Modeling (ProSim)*, 2000. - [B]

[36] David Raffo, Marc Kellner. Analyzing Process Improvements Using the Process Tradeoff Analysis Method. *3rd International Workshop on Software Process Simulation and Modeling (ProSim)*, 2000. - [D]

SPIP 2002†

[37] Juan F. Ramil, Neil Smith. Qualitative simulation of models of software evolution. *Software Process: Improvement and Practice*, 7(3-4), 2002. - [A]

[38] Paul Wernick, Tracy Hall. Simulating global software evolution processes by combining simple models: an initial study. *Software Process: Improvement and Practice*, 7(3-4), 2002. - [A]

[39] Frank Padberg. A discrete simulation model for assessing software project scheduling policies. *Software Process: Improvement and Practice*, 7(3-4), 2002. - [A]

[40] Eliza Chiang, Tim Menzies. Simulations for very early lifecycle quality evaluations. *Software Process: Improvement and Practice*, 7(3-4), 2002. - [A/B/D]

[41] Marcio De Oliveira Barros, Claudia Maria Lima Werner, Guilherme Horta Travassos. A system dynamics metamodel for software process modeling. *Software Process: Improvement and Practice*, 7(3-4), 2002. - [B]

[42] I. P. Antoniades, I. Stamelos, L. Angelis, G. L. Bleris. A novel simulation model for the development process of open source software projects. *Software Process: Improvement and Practice*, 7(3-4), 2002. - [A]

[43] Dietmar Pfahl, Gunther Ruhe. IMMoS: a methodology for integrated measurement, modelling and simulation. *Software Process: Improvement and Practice*, 7(3-4), 2002. - [B/C]

ProSim 2003

[44] Joao W. Cangussu. A software test process stochastic control model based on CMM characterization *Software Process: Improvement and Practice*, 9(2-3), 2004. - [A]

[45] Tobias Haberlein. Common structures in system dynamics models of software acquisition projects. *Software Process: Improvement and Practice*, 9(2-3), 2004. - [A]

†The ProSim 2002 Workshop was cancelled due to a variety of circumstances including the apprehension over air travel following the tragedy of September 11, 2001, and the relocation of the ICSE 2002. However, the selected papers from the submission had been published in the special issue of SPIP 2002.

[46] Mercedes Ruiz, Isabel Ramos, Miguel Toro. An integrated framework for simulation-based software process improvement. *Software Process: Improvement and Practice*, 9(2-3), 2004. - [C]

[47] Alejandro Fernandez, Badie Garzaldeen, Ines Grutzner, Jurgen Munch. Guided support for collaborative modeling, enactment and simulation of software development processes. *Software Process: Improvement and Practice*, 9(2-3), 2004. - [B]

[48] Wayne W. Wakeland, Robert H. Martin, David Raffo. Using design of experiments, sensitivity analysis, and hybrid simulation to evaluate changes to a software development process: a case study. *Software Process: Improvement and Practice*, 9(2-3), 2004. - [B]

[49] Ioana Rus, Holger Neu, Jurgen Munch. A Systematic Methodology for Developing Discrete Event Simulation Models of Software Development Processes. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [B]

[50] David Raffo, Greg Spehar, Umanath Nayak. Generalized Simulation Models: What, Why and How? *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [C]

[51] Rizwan Ahmed, Tracy Hall, Paul Wernick. A Proposed Framework for Evaluating Software Process Simulation Models. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [B]

[52] Tomas Berling, Carina Andersson, Martin Host, Christian Nyberg. Adaptation of a Simulation Model Template for Testing to an Industrial Project. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [A/C]

[53] Susan Ferreira, James Collofello, Dan Shunk, Gerald Mackulak, Philip Wolfe. Utilization of Process Modeling and Simulation in Understanding the Effects of Requirements Volatility in Software Development. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [A]

[54] Dan Houston. A Case Study in Software Enhancements as Six Sigma Process Improvements: Simulating Productivity Savings. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [A/D]

[55] Peter B. Lakey. A Hybrid Software Process Simulation Model for Project Management. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [A/C]

[56] Jurgen Munch, Dieter Rombach, Ioana Rus. Creating an Advanced Software Engineering Laboratory by Combining Empirical Studies with Process Simulation. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [C]

[57] Holger Neu, Thomas Hanne, Jrgen Mnch, Stefan Nickel, Andreas Wirsen. Creating a Code Inspection Model for Simulation-based Decision Support. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [A]

[58] Holger Neu, Ioana Rus. Reuse in Software Process Simulation Modeling. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [B]

[59] Dietmar Pfahl, Gunther Ruhe. Goal-Oriented Measurement plus System Dynamics - A Hybrid and Evolutionary Approach. *4th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2003. - [B/C]

ProSim 2004

[60] Neil Smith, Andrea Capiluppi and Juan F. Ramil. A Study of Open Source Software Evolution Data using Qualitative Simulation. *Software Process: Improvement and Practice*, 10(3), 2005. - [D]

[61] Wayne Wakeland, Stephen Shervais, David Raffo. Heuristic Optimization as a V&V Tool for Software Process Simulation Models. *Software Process: Improvement and Practice*, 10(3), 2005. - [B]

[62] Emily Oh Navarro, Andre van der Hoek. Software Process Modeling for an Educational Software Engineering Simulation Game. *Software Process: Improvement and Practice*, 10(3), 2005. - [A/B]

[63] Thomas Birkholzer, Christoph Dickmann, Research Section Jurgen Vaupel, Laura Dantas. An Interactive Software Management Simulator based on the CMMI Framework. *Software Process: Improvement and Practice*, 10(3), 2005. - [A]

[64] David Raffo, Umanath Nayak, Siri-on Setamanit, Patrick Sullivan, Wayne Wakeland. Using Software Process Simulation to Assess the Impact of IV&V Activities. *5th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2004. - [D]

[65] Dietmar Pfahl, Michael Stupperrich, Tatyana Krivobokova. PL-SIM: A Generic Simulation Model for Studying Strategic SPI in the Automotive Industry. *5th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2004. - [A/D]

[66] Bjorn Regnell, Bengt Ljungquist, Lena Karlsson. Investigation of Requirements Selection Quality in Market-Driven Software Process using an Open Source Discrete Event Simulation Framework. *5th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2004. - [A]

[67] Mercedes Ruiz, Isabel Ramos, Miguel Toro. Building Software Process Models with a Multitier Architecture. *5th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2004. - [B]

[68] Paul Wernick, Tracy Hall. A Policy Investigation Model for Long-term Software Evolution Processes. *5th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2004. - [A]

[69] Robert H. Martin. What Makes Software Management Hard? *5th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2004. - [A]

ProSim 2005

[70] Marco Melis, Ivana Turnu, Alessandra Cau, Giulio Concas. Evaluating the Impact of Test-First Programming and Pair Programming through Software Process Simulation. *Software Process: Improvement and Practice*, 11(4), 2006. - [A]

[71] Dan Houston. An experience in facilitating process improvement with an integration problem reporting process simulation. *Software Process: Improvement and Practice*, 11(4), 2006. - [A/D]

[72] KeungSik Choi, Doo-Hwan Bae, TagGon Kim. An approach to a hybrid software process simulation using the DEVS formalism. *Software Process: Improvement and Practice*, 11(4), 2006. - [A/B]

[73] Yu Chen, Gerald C. Gannod, James S. Collofello. A software product line process simulator. *Software Process: Improvement and Practice*, 11(4), 2006. - [A]

[74] Marcio de O. Barros, Alexandre R. Dantas, Gustavo O. Veronese, Claudia M. L.Werner. Model-driven game development: experience and model enhancements in software project management education. *Software Process: Improvement and Practice*, 11(4), 2006. - [A/B/D]

[75] Neil Smith, Andrea Capiluppi, Juan Fernandez-Ramil. Agent-based simulation of open source evolution. *Software Process: Improvement and Practice*, 11(4), 2006. - [A]

[76] Gregorio Robles, Juan Julian Merelo, Jesus M. Gonzalez-Barahona. Self-organized development in libre software projects: a model based on the stigmergy concept. *6th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2005. - [A/B]

[77] Charbel Noujeim, Jorg Sandrock, Christof Weinhardt. Economic Analysis of Integrated Software Development and Consulting Companies. *6th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2005. - [A]

[78] Ray Madachy. Software Process and Business Value Modeling. *6th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2005. - [C]

[79] David M. Raffo, Tim Menzies. Evaluating the Impact of a New Technology Using Simulation: The Case for Mining Software Repositories. *6th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2005. - [D]

[80] Nuria Hurtado, Mercedes Ruiz, Jesus Torres. Towards Interactive Systems Usability Improvement through Simulation Modeling. *6th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2005. - [A]

[81] David Raffo, Umanatha Nayak, Siri-on Setamanit, Wayne Wakeland. Implementing Generalized Simulation Models. *6th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2005. - [C]

[82] Niniek Angkasaputra, Dietmar Pfahl. Towards an Agile Development Process of Software Process Simulation. *6th International Workshop on Software Process Simulation and Modeling (ProSim)*, 2005. - [B]

SPW/ProSim 2006

[83] Dietmar Pfahl, Ahmed Al-Emran, Gunther Ruhe. A System Dynamics Simulation Model for Analyzing the Stability of Software Release Plans. *Software Process: Improvement and Practice*, 12(5), 2007. - [A]

[84] Siri-on Setamanit, Wayne Wakeland, David Raffo. Using Simulation to Evaluate Global Software Development Task Allocation Strategies. *Software Process: Improvement and Practice*, 12(5), 2007. - [A]

[85] Benjamin Stopford, Steve Counsell. Simulating the Structural Evolution of Software. *Software Process Workshop/International Workshop on Software Process Simulation and Modeling (SPW/ProSim)*, 2006. - [A]

[86] Levent Yilmaz, Jared Phillips. Organization-Theoretic Perspective for Simulation Modeling of Agile Software Processes. *Software Process Workshop/International Workshop on Software Process Simulation and Modeling (SPW/ProSim)*, 2006. - [A]

[87] He Zhang, Barbara Kitchenham. Semi-quantitative Simulation Modeling of Software Engineering Process. *Software Process Workshop/International Workshop on Software Process Simulation and Modeling (SPW/ProSim)*, 2006. - [A/B]

[88] Raymond Madachy. Reusable Model Structures and Behaviors for Software Processes. *Software Process Workshop/International Workshop on Software Process Simulation and Modeling (SPW/ProSim)*, 2006. - [B/C]

ICSP 2007[‡]

[89] Ahmed Al-Emran, Dietmar Pfahl, Gunther Ruhe. A Method for Re-planning of Software Releases Using Discrete-event Simulation. *Software Process: Improvement and Practice*, 13(1), 2008. - [A]

[90] David Raffo, Robert Ferguson, Siri-on Setamanit, Bhuricha Sethanandha. Evaluating the Impact of Requirements Analysis Tools Using Simulation. *Software Process: Improvement and Practice*, 13(1), 2008. - [D]

[91] Florian Deissenboeck, Markus Pizka. The Economic Impact of Software Process Variations. *International Conference on Software Process*, 2007. - [A/D]

[92] Christoph Dickmann, Harald Klein, Thomas Birkholzer, Wolfgang Fietz, Jurgen Vaupel, Ludger Meyer. Deriving a Valid Process Simulation from Real World Experiences. *International Conference on Software Process*, 2007. - [C/D]

[93] Makoto Nonaka, Liming Zhu, Muhammad Ali Babar, Mark Staples. Project Delay Variability Simulation in Software Product Line Development. *International Conference on Software Process*, 2007. - [A]

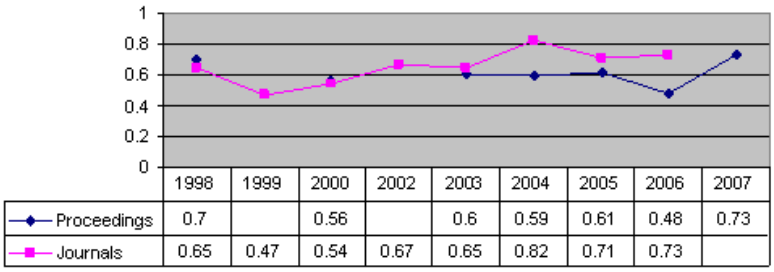
[94] Antony Powell, John Murdoch, Nick Tudor. Modeling Risk-Benefit Assumptions in Technology Substitution. *International Conference on Software Process*, 2007. - [A/C]

[‡]The ProSim Workshop continued as a special track of ICSP since 2007.

[95] He Zhang, Barbara Kitchenham, Ross Jeffery. A Framework for Adopting Software Process Simulation in CMMI Organizations. *International Conference on Software Process*, 2007. - [C]

[96] He Zhang, Barbara Kitchenham, and Ross Jeffery. Achieving Software Project Success: A Semi-quantitative Approach. *International Conference on Software Process*, 2007. - [C]

A.2 Study Quality Assessment



	1998	1999	2000	2002	2003	2004	2005	2006	2007
Proceedings	0.7		0.56		0.6	0.59	0.61	0.48	0.73
Journals	0.65	0.47	0.54	0.67	0.65	0.82	0.71	0.73	

Figure A.1: Average study quality per source and year

A.2.1 Assessment Criteria

The quality of a primary study is assessed with the help of a checklist (Table A.1), which specifies the questions to each study category separately. For each question, the study's quality is evaluated as '*yes*' (y), '*partial*' (p), or '*no*' (n), which are scored with the value 1, 0.5, and 0 respectively. The studies were evaluated by the principal researcher (author), and a selection of approximately 30% was checked by the secondary researcher. Disagreements were resolved by the principal researcher.

A.2.2 Assessment Results (Stage 1)

The primary studies were assessed for quality using the criteria in Table A.1. The normalised average quality score per source type (proceedings and journals) and year is presented in Figure A.1. The study quality of workshop proceedings was stable from 2000 to 2005. In most cases, the quality of journal articles was equal to or better than the proceedings papers of the same year. Overall, many Category A studies failed to explicitly address the conditions of model/simulator adoption. For Category B and C studies, the limitations associated with the paradigm/method/solution were rarely discussed.

Table A.1: Study quality assessment checklist

No.	Question	Score
Common questions (for all categories)		
1	Did the study clearly state the aims/research questions?	'y/p/n'
2	Did the study review the related work for the problem?	'y/p/n'
3	Did the study discuss related issues, and compare with the alternatives?	'y/p/n'
4	Did the study recommend the further continuous research?	'y/p/n'
Questions for Category **A**		
5	Are the model's assumptions explained explicitly?	'y/p/n'
6	Is the model construction fully described?	'y/p/n'
7	Did the study explain why choosing the applied simulation paradigm(s)?	'y/p/n'
8	Are the conditions when the model adoption explained?	'y/p/n'
9	Did the study avoid any selection bias exist during experiment design?	'y/p/n'
10	Has the model been trialled on an industry scale problem?	'y/p/n'
11	Did the study carry out a sensitivity or residual analysis?	'y/p/n'
12	Are any model evaluation methods applied on the model?	'y/p/n'
13	Does the study interpret the findings?	'y/p/n'
Questions for Category **B** & **C**		
14	Are the scopes of the method/paradigm/solution clearly defined?	'y/p/n'
15	Are the modelling approach/method/environment clearly defined?	'y/p/n'
16	Are the problems that the study addresses defined with appropriate SE examples?	'y/p/n'
17	Did the study specify the limitations of the argued paradigm/method/solution?	'y/p/n'
18	Did the empirical evidence include support the arguments of the study?	'y/p/n'
Questions for Category **D**		
19	Can the experience be used for validating and calibrating simulation model/modelling?	'y/p/n'
20	Are the best practices or lessons learnt extracted from experience?	'y/p/n'

Appendix B

Model Implementations

B.1 Software Staffing Process Models

Software staffing process model for qualitative simulation

```
(define-QDE normal-software-develop
  (text "Software project development with all experienced workforce")
  (quantity-spaces
    (Sp (0 ASG_SIZE inf)                "Sp: project size")
    (Sc (0 ASG_SIZE inf)                "Sc: complete size")
    (Sr (0 PRS_SIZE ASG_SIZE inf)       "Sr: remaining size")
    (Rsd (0 inf)                        "Rsd: develop rate")
    (mRsd (minf 0)                      "mRsd: minus dev rate")
    (Rnd (0 inf)                        "Rnd: nominal dev rate")
    (WFt (0 INI_EX_WF TOTAL_WF inf)     "WFt: totl workforce")
    (WFex (0 INI_EX_WF TOTAL_WF inf)    "WFex: exp workforce")
    (WFnw (0 inf)                       "WFnw: new workforce")
    (PDex (0 NML_EX_PD inf)             "PDex: exp productvty")
    (PDnw (0 NML_NW_PD NML_EX_PD inf)   "PDnw: new productvty")
    (Rexd (0 inf)                       "Rexd: exp dev rate")
    (Rnwd (0 inf)                       "Rnwd: new dev rate")
    (Rco (0 inf)                        "Rco: communictn ovhd"))

  (constraints
    ((add Sc Sr Sp) (0 ASG_SIZE ASG_SIZE) (ASG_SIZE 0 ASG_SIZE))
    ((constant Sp ASG_SIZE))
    ((constant WFt))
    ((constant WFnw))
    ((constant WFex))
    ((add WFex WFnw WFt) (0 0 0) (INI_EX_WF 0 INI_EX_WF))
    ((d/dt Sc Rsd))
```

```
    ((d/dt Sr mRsd))
    ((minus Rsd mRsd))
    ((add Rsd Rco Rnd))
    ((add Rexd Rnwd Rnd))
    ((constant PDex NML_EX_PD))
    ((Mult PDex WFex Rexd))
    ((Mult PDnw WFnw Rnwd))
    ((M+ WFt Rco) (0 0) (inf inf)))

  (transitions
    ((Sr (PRS_SIZE dec)) -> transition-to-assimilating)
    ((Sr (0 dec)) -> t)))

(define-QDE software-develop-assimilation
  (text "Software project development during assimilation")
  (quantity-spaces
    (Sp (0 ASG_SIZE inf)                "Sp: project size")
    (Sc (0 ASG_SIZE inf)                "Sc: complete size")
    (Sr (0 PRS_SIZE ASG_SIZE inf)       "Sr: remaine size")
    (Rsd (0 inf)                        "Rsd: develop rate")
    (mRsd (minf 0)                      "mRsd: minus dev rate")
    (Rnd (0 inf)                        "Rnd: nominal dev rate")
    (WFt (0 INI_EX_WF TOTAL_WF inf)     "WFt: totl workforce")
    (WFex (0 INI_EX_WF TOTAL_WF inf)    "WFex: exp workforce")
    (WFed (0 INI_EX_WF TOTAL_WF inf)    "WFed: exp dev wkfr")
    (WFet (0 INI_EX_WF inf)             "WFet: exp train wkfr")
    (WFnw (0 INJ_NW_WF inf)             "WFnw: new workforce")
    (PDex (0 NML_EX_PD inf)             "PDex: exp productvty")
    (PDnw (0 NML_NW_PD NML_EX_PD inf)   "PDnw: new productvty")
    (Rexd (0 inf)                       "Rexd: exp dev rate")
    (Rnwd (0 inf)                       "Rnwd: new dev rate")
    (Ras (0 inf)                        "Ras: assimilate rate")
    (Rco (0 inf)                        "Rco: communictn ovhd"))

  (constraints
    ((add Sc Sr Sp) (0 ASG_SIZE ASG_SIZE) (ASG_SIZE 0 ASG_SIZE))
    ((constant Sp ASG_SIZE))
    ((constant WFt))
    ((constant WFnw))
    ((constant WFex))
    ((add WFex WFnw WFt) (0 0 0) (INI_EX_WF 0 INI_EX_WF) (INI_EX_WF INJ_NW_WF TOTAL_WF))
    ((add WFed WFet WFex))
    ((M+ WFnw WFet) (0 0) (inf inf))
    ((d/dt Sc Rsd))
    ((d/dt Sr mRsd))
    ((minus Rsd mRsd))
    ((add Rsd Rco Rnd))
```

```
    ((add Rexd Rnwd Rnd))
    ((constant PDex NML_EX_PD))
    ((Mult PDex WFed Rexd))
    ((Mult PDnw WFnw Rnwd))
    ((d/dt PDnw Ras))
    ((constant Ras))
    ((M+ WFt Rco) (0 0) (inf inf)))

  (transitions
    ((PDnw (NML_EX_PD inc)) -> transition-to-experienced)
    ((Sr (0 dec)) -> t)))

(defun transition-to-assimilating (project-state)
  (create-transition-state
    :from-state project-state
    :to-qde software-develop-assimilation
    :assert '((WFnw (INJ_NW_WF std))
              (WFet ((0 INI_EX_WF) std))
              (WFed ((0 INI_EX_WF) std))
              (PDnw (NML_NW_PD inc))
              (WFt (TOTAL_WF std)))
    :inherit-qmag '(Sc Sr WFex)))

(defun transition-to-normal (workforce-state)
  (create-transition-state
    :from-state workforce-state
    :to-qde normal-software-develop
    :assert '((WFnw (0 std))
              (WFex (TOTAL_WF std))
              (PDnw (0 std)))
    :inherit-qmag '(Sc Sr WFt Rsd Rnd Rco)))

(defun start-dev-project ()
  (let* ((sim (make-sim :state-limit *))
  (initial-state
    (make-new-state
      :from-qde normal-software-develop
      :sim sim
      :assert-values '((Sp (ASG_SIZE std))
                       (Sc (0 inc))
                       (Sr (ASG_SIZE dec))
                       (WFt (INI_EX_WF std))
                       (WFex (INI_EX_WF std))
                       (WFnw (0 std))
                       (Rco ((0 inf) std))
                       (PDnw (0 std))
                       (PDex (NML_EX_PD std)))
```

```
      :text "Start software development project with experienced staff")))
    (qsim initial-state)))
```

Software staffing process model for semi-quantitative simulation

```
(define-QDE normal-software-develop
  (text "Software project development with all experienced workforce")
  (quantity-spaces
    (Sp (0 ASG_SIZE inf)                "Sp: project size")
    (Sc (0 ASG_SIZE inf)                "Sc: complete size")
    (Sr (0 PRS_SIZE ASG_SIZE inf)       "Sr: remaine size")
    (Rsd (0 inf)                        "Rsd: develop rate")
    (mRsd (minf 0)                      "mRsd: minus dev rate")
    (Rnd (0 inf)                        "Rnd: nominal dev rate")
    (Rcl (0 inf))
    (Rexd (0 inf)                       "Rexd: exp dev rate")
    (Rnwd (0 inf)                       "Rnwd: new dev rate")
    (WFtl (0 inf)                       "WFtl: totl workforce")
    (WFex (0 INI_EX_WF inf)             "WFex: exp workforce")
    (WFnw (0 inf)                       "WFnw: new workforce")
    (PDex (0 NML_EX_PD inf)             "PDex: exp productvty")
    (PDnw (0 NML_NW_PD NML_EX_PD inf)   "PDnw: new productvty")
    (Lcm (0 1 inf)                      "Lcm: commun & motivt overhd loss")
    (Rcm (0 inf)                        "Rcm: commun & motivt overhd rate"))

  (constraints
    ((add Sc Sr Sp) (0 ASG_SIZE ASG_SIZE) (ASG_SIZE 0 ASG_SIZE))
    ((constant Sp ASG_SIZE))
    ((constant WFtl))
    ((constant WFnw))
    ((constant WFex))
    ((add WFex WFnw WFtl) (0 0 0))
    ((d/dt Sc Rsd))
    ((d/dt Sr mRsd))
    ((minus Rsd mRsd))
    ((Mult PDex WFtl Rcl) (0 0 0))
    ((Mult Rcl Lcm Rcm))
    ((add Rsd Rcm Rnd))
    ((add Rexd Rnwd Rnd))
    ((constant PDex))
    ((Mult PDex WFex Rexd) (0 0 0))
    ((Mult PDnw WFnw Rnwd) (0 0 0))
    ((M+ WFtl Lcm) (0 0) (inf 1)))

  (transitions
    ((Sr (PRS_SIZE dec)) -> transition-to-assimilating)
    ((Sr (0 dec)) -> t))
```

```
  (initial-ranges
    ((Sp ASG_SIZE) (64000 64000))       ; DSI
    ((Sc ASG_SIZE) (64000 64000))       ; DSI
    ((Sr ASG_SIZE) (64000 64000))       ; DSI
    ((Sr PRS_SIZE) (25300 25300))       ; DSI (when to inject new workforce)
    ((WFex INI_EX_WF) (4 5))
    ((PDex NML_EX_PD) (36 36))          ; DSI/man-day
    ((PDnw NML_NW_PD) (15 22))          ; DSI/man-day
    ((PDnw NML_EX_PD) (36 36)))         ; DSI/man-day

(envelopes
  ((M+ WFtl Lcm) (exact (lambda (x) (* 0.0006 (* x x))))
                 (e-inv (lambda (y) (sqrt (/ y 0.0006)))))))

(defun transition-to-assimilating (project-state)
  (create-transition-state
    :from-state project-state
    :to-qde software-develop-assimilation
    :assert '((WFnw (INJ_NW_WF    std))
              (PDnw (NML_NW_PD    inc))
              (PDdf (ORG_DF_PD    std))
              (Das  (ASS_DLY      std)))
    :inherit-qmag '(Sp Sc Sr WFex PDex Time)
    :inherit-qdir '(Sp Sc Sr WFex PDex Time)
    :text "start assimilation"))

(define-QDE software-develop-assimilation
  (text "Software project development during assimilation")
  (quantity-spaces
    (Sp (0 ASG_SIZE inf)                "Sp: project size")
    (Sc (0 ASG_SIZE inf)                "Sc: complete size")
    (Sr (0 PRS_SIZE ASG_SIZE inf)       "Sr: remaine size")
    (Rsd (0 inf)                        "Rsd: develop rate")
    (mRsd (minf 0)                      "mRsd: minus dev rate")
    (Rnd (0 inf)                        "Rnd: nominal dev rate")
    (Rcl (0 inf))
    (Rexd (0 inf)                       "Rexd: exp dev rate")
    (Rnwd (0 inf)                       "Rnwd: new dev rate")
    (WFtl (0 inf)                       "WFtl: totl workforce")
    (WFex (0 INI_EX_WF inf)             "WFex: exp workforce")
    (WFed (0 inf)                       "WFed: exp dev wkfr")
    (WFet (0 inf)                       "WFet: exp train wkfr")
    (WFnw (0 INJ_NW_WF inf)             "WFnw: new workforce")
    (PDex (0 NML_EX_PD inf)             "PDex: exp productvty")
    (PDnw (0 NML_NW_PD NML_EX_PD inf)   "PDnw: new productvty")
    (PDdf (0 ORG_DF_PD inf))
```

```
  (Das (0 ASS_DLY inf))
  (Ras (0 inf)                          "Ras: assimilate rate")
  (Lcm (0 1 inf)                        "Lcm: commun & motivt overhd loss")
  (Rcm (0 inf)                          "Rcm: commun & motivt overhd rate"))

(constraints
  ((add Sc Sr Sp) (0 ASG_SIZE ASG_SIZE) (ASG_SIZE 0 ASG_SIZE))
  ((constant Sp ASG_SIZE))
  ((constant WFtl))
  ((constant WFnw))
  ((constant WFex))
  ((add WFex WFnw WFtl) (0 0 0))
  ((add WFed WFet WFex))
  ((M+ WFnw WFet) (0 0) (inf inf))
  ((d/dt Sc Rsd))
  ((d/dt Sr mRsd))
  ((minus Rsd mRsd))
  ((Mult PDex WFtl Rcl) (0 0 0))
  ((Mult Rcl Lcm Rcm))
  ((add Rsd Rcm Rnd))
  ((add Rexd Rnwd Rnd))
  ((constant PDex))
  ((constant PDdf))
  ((constant Das))
  ((Mult Ras Das PDdf) (0 0 0))
  ((Mult PDex WFed Rexd) (0 0 0))
  ((Mult PDnw WFnw Rnwd) (0 0 0))
  ((d/dt PDnw Ras))
  ((constant Ras))
  ((M+ WFtl Lcm) (0 0) (inf 1)))

(transitions
  ((PDnw (NML_EX_PD inc)) -> transition-to-normal)
  ((Sr (0 dec)) -> t))

(initial-ranges
  ((Sp ASG_SIZE) (64000 64000))       ; DSI
  ((Sc ASG_SIZE) (64000 64000))       ; DSI
  ((Sr ASG_SIZE) (64000 64000))       ; DSI
  ((Sr PRS_SIZE) (25300 25300))       ; DSI (when to inject new workforce)
  ((WFex INI_EX_WF) (4 5))
  ((WFnw INJ_NW_WF) (7 8))
  ((PDex NML_EX_PD) (36 36))          ; DSI/man-day
  ((PDnw NML_NW_PD) (14 22))          ; DSI/man-day
  ((PDnw NML_EX_PD) (36 36))          ; DSI/man-day
  ((PDdf ORG_DF_PD) (14 22))
  ((Das ASS_DLY) (60 80)))            ; day
```

```
  (envelopes
    ((M+ WFnw WFet) (upper (lambda (x) (truncate (+ (* x 0.25) 0.5))))
                    (u-inv (lambda (y) (* y 4)))
                    (lower (lambda (x) (truncate (+ (* x 0.15) 0.5))))
                    (l-inv (lambda (y) (truncate (/ y 0.15)))))
    ((M+ WFtl Lcm) (exact (lambda (x) (* 0.0006 (* x x))))
                   (e-inv (lambda (y) (sqrt (/ y 0.0006)))))))

(defun transition-to-normal (workforce-state)
  (create-transition-state
    :from-state workforce-state
    :to-qde normal-software-develop
    :assert '((WFnw (0        std))
              (PDnw (0        std)))
    :inherit-qmag '(Sc Sr WFtl PDex Rsd Rnd Lcm Time)))

(defun start-dev-project ()
  (let* ((sim (make-sim :state-limit *
                        :Q2-constraints t))
         (initial-state (make-new-state
                          :from-qde normal-software-develop
                          :sim sim
                          :assert-values
                            '((Sp   (ASG_SIZE   std))
                              (Sc   (0          inc))
                              (Sr   (ASG_SIZE   dec))
                              (WFex (INI_EX_WF  std))
                              (WFnw (0          std))
                              (PDex (NML_EX_PD  std))
                              (PDnw (0          std)))
                          :text "start software project with experienced staff")))
         (qsim initial-state)))
```

B.2 Incremental Development Process Models

Incremental development process model for semi-quantitative simulation

```
(define-QDE implementation
  (text "Implementation process of one increment")
  (quantity-spaces
    ; normal development ----------------------------------------------
    (S    (0 RLS_SIZE inf)      "S: release size")
    (Rsd  (0 inf)               "Rsd: development rate")
    (Rnd  (0 inf)               "Rnd: nominal development rate")
    (Rioh (0 inf)               "Rioh: overheads rate")
    (WF   (0 ASG_WF inf)        "WF: development team size")
```

```
  (PDim (0 IMP_PD inf)        "PDim: avg implement productivity")
  ; error generation ------------------------------------------------
  (ErDn (0 NML_ED inf)        "ErDn: nominal error density")
  (Reg  (0 ER_GEN_RATE inf)   "Reg: error generation rate")
  (rERt (0 ACER_RT_RATIO 1)   "rERt: active error retired ratio")
  (Eg   (0 inf)               "Eg: error generation rate")
  (rEag (0 ACER_GEN_RATIO 1)  "rEag: active error generation ratio")
  (Eag  (0 inf)               "Eag: generated active errors")
  (Epg  (0 inf)               "Epg: generated passive errors")
  (Eaes (0 ESC_AC_ER inf)     "Eaes: escaping active errors")
  (Epes (0 ESC_PS_ER inf)     "Epes: escaping passive errors")
  (Eart (0 inf)               "Eart: retired active errors")
  (Epo  (0 inf)               "Epo: old passive errors")
  (Eao  (0 inf)               "Eao: old active errors")
  (Eprp (0 inf)               "Eprp: reproduced passive errors")
  (Earp (0 inf)               "Earp: reproduced active errors")
  (Epn  (0 inf)               "Epn: new passive errors")
  (Ean  (0 inf)               "Ean: new active errors")
  (Eo   (0 inf)               "Eo: old errors")
  (En   (0 inf)               "En: new errors")
  (Ea   (0 inf)               "Ea: active errors")
  (Ep   (0 inf)               "Ep: passive errors")
  (Et   (0 inf)               "Et: total errors"))

(constraints
  ; normal development ----------------------------------------------
  ((d/dt S Rsd))
  ((constant PDim IMP_PD))
  ((constant WF ASG_WF))
  ((add Rioh Rsd Rnd) (0 0 0))
  ((M+ WF Rioh) (0 0))
  ((Mult PDim WF Rnd) (0 0 0))
  ; error generation ------------------------------------------------
  ((constant ErDn NML_ED))
  ((mult ErDn Rsd Reg))
  ((d/dt Eg Reg))
  ((constant Eaes)) ; input variable from previous release
  ((constant Epes)) ; input variable from previous release
  ((constant rERt)) ; value changes between increments
  ((constant rEag)) ; value changes between increments
  ((mult Eaes rERt Eart) (0 0 0))
  ((add Eart Eao Eaes) (0 0 0))
  ((add Epes Eart Epo) (0 0 0))
  ((M+ Eao Earp) (0 0))
  ((M+ Eao Eprp) (0 0))
  ((add Eag Epg Eg))
  ((mult Eg rEag Eag))
```

```
    ((add Epg Eprp Epn) (0 0 0))
    ((add Eag Earp Ean) (0 0 0))
    ((add Eao Ean Ea) (0 0 0))
    ((add Epo Epn Ep) (0 0 0))
    ((add Ean Epn En) (0 0 0))
    ((add Eao Epo Eo) (0 0 0))
    ((add En Eo Et) (0 0 0)))

  (transitions
    ((S (RLS_SIZE inc)) -> transition-to-test-and-fix)))

  (initial-ranges
    ((S    RSL_SIZE)       (26667 26667)) ; inc1:26667, inc2:32000, inc3:32000 (LOC)
    ((W    ASG_WF)         (15  15))
    ((ErDn NML_ERDN)       (0.018 0.020)) ; (error/LOC)
    ((rEag ACER_GEN_RATIO) (0.90 0.95)) ; inc1:[.9 .95], inc2:[.2 .85], inc3:[0 .15]
    ((rERt ACER_RT_RATIO)  (0 0))) ; inc1:[0 0], inc2:[.05 .15], inc3:[.2 1]

(defun transition-to-test-and-fix (implementation-state)
  (create-transition-state
    :from-state implementation-state
    :to-qde test-and-fix
    :assert '((Ts    (0         inc))
              (PDnfx (NML_FX_PD std))
              (Rtc   (TEST_RATE std))
              (Eand  (0         inc))
              (Epnd  (0         inc))
              (Eaod  (0         inc))
              (Epod  (0         inc)))
    :inherit-qmag '(WF Eao Epo Ean Epn Et)
    :inherit-qdir '(WF)
    :text "start test-and-fix of increment"))

(define-QDE test-and-fix
  (text "Test-and-fix process of one increment")
  (quantity-spaces
    (Ts    (0 TS_SIZE inf)        "Ts: test suite size")
    (Rtc   (0 TEST_RATE inf)      "Rtc: testing rate by test case")
    (EDtc  (0 ER_TC_ED inf)       "EDtc: error density in test case")
    (Red   (0 inf)                "Red: error detecting rate (per day)")
    (ho    (0 OL_ER_HT_RATIO 1)   "ho: old error hitting ratio")
    (hn    (0 NW_ER_HT_RATIO 1)   "hn: new error hitting ratio")
    (Rnt   (0 inf)                "Rnt: net rate of error pool")
    (Rpd   (0 inf)                "Rpd: detected passive errors")
    (Rad   (0 inf)                "Rad: detected active errors")
    (Rpnd  (0 inf)                "Rpnd: passive new error detecting rate")
    (Rand  (0 inf)                "Rand: active new error detecting rate")
```

```
  (Rpod  (0 inf)               "Rpod: passive old error detecting rate")
  (Raod  (0 inf)               "Raod: active old error detecting rate")
  (Epnd  (0 inf)               "Epnd: detected passive new errors")
  (Eand  (0 inf)               "Eand: detected active new errors")
  (Epod  (0 inf)               "Epod: detected passive old errors")
  (Eaod  (0 inf)               "Eaod: detected active old errors")
  (Ead   (0 inf)               "Ead: detected active errors")
  (Epd   (0 inf)               "Epd: detected passive errors")
  (Epnes (0 inf)               "Epnes: escaping passive new errors")
  (Eanes (0 inf)               "Eanes: escaping active new errors")
  (Epoes (0 inf)               "Epoes: escaping passive old errors")
  (Eaoes (0 inf)               "Eaoes: escaping active old errors")
  (Epes  (0 inf)               "Epes: escaping passive errors")
  (Eaes  (0 inf)               "Eaes: escaping active errors")
  (Epn   (0 inf)               "Epn: new passive errors")
  (Ean   (0 inf)               "Ean: new active errors")
  (Epo   (0 inf)               "Epo: old passive errors")
  (Eao   (0 inf)               "Eao: old active errors")
  (Epl   (0 inf)               "Epl: detected errors in pool")
  (Etd   (0 inf)               "Etd: detected errors") ;
  (Ees   (0 inf)               "Ees: escaped error from testing")
  (Rnf   (0 inf)               "Rnf: nominal bug-fixing rate")
  (WF    (0 ASG_WF inf)        "WF: development team size")
  (PDfx  (0 inf)               "PDfx: productivity of bug-fixing")
  (PDnfx (0 NML_FX_PD inf)     "PDfx: nominal productivity of bug-fixing")
  (Rfoh  (0 inf)               "Rfoh: bug-fixing overheads rate")
  (mf    (0 PD_SW_MLT 1)       "mf: multiplier of bug-fixing switchover")
  (Rfx   (0 inf)               "Ref: bug-fixing rate")
  (Efx   (0 inf)               "Efx: fixed errors")
  (Et    (0 inf)               "Et: total errors in increment"))

(constraints
  ((d/dt Ts Rtc))
  ((constant Rtc TEST_RATE))
  ((constant EDtc ER_TC_ED))
  ((Mult Rtc EDtc Rned))
  ((Mult Rned h
  ((d/dt Etd Red))
  ((constant PDnfx  NML_FX_PD))
  ((constant WF))
  ((constant Et))
  ((constant Ean))
  ((constant Epn))
  ((constant Eao))
  ((constant Epo))
  ((M+ Rtc Rpnd))
  ((M+ Rtc Rand))
```

```
    ((M+ Rtc Rpod))
    ((M+ Rtc Raod))
    ((add Rpnd Rpod Rpd))
    ((add Rand Raod Rad))
    ((add Rpd Rad Red))
    ((d/dt Epnd Rpnd))
    ((d/dt Eand Rand))
    ((d/dt Epod Rpod))
    ((d/dt Eaod Raod))
    ((add Eand Eanes Ean) (0 0 0))
    ((add Epnd Epnes Epn) (0 0 0))
    ((add Eaod Eaoes Eao) (0 0 0))
    ((add Epod Epoes Epo) (0 0 0))
    ((add Epnes Epoes Epes) (0 0 0))
    ((add Eanes Eaoes Eaes) (0 0 0))
    ((add Eaes Epes Ees) (0 0 0))
    ((add Eand Eaod Ead))
    ((add Epnd Epod Epd))
    ((add Ead Epd Etd) (0 0 0))
    ((add Ees Etd Et) (0 0 0))
    ((add Efx Epl Etd))
    ((constant mf PD_SW_MLT))
    ((constant Rfoh))
    ((add Rfx Rnt Red)) ;
    ((d/dt Epl Rnt))
    ((d/dt Efx Rfx))
    ((add Rfx Rfoh Rnf))
    ((M+ WF Rfoh))
    ((Mult PDnfx mf PDfx))
    ((Mult PDfx WF Rnf)))

  (transitions
    ((Ees (0 dec)) -> transition-to-fix-only)
    ((Ts (TS_SIZE inc)) -> transition-to-fix-only)))

  (initial-ranges
    ((Ts    RSL_SIZE)        (14 14)) ; test_cases
    ((W     ASG_WF)          (15 15))
    ((hn    NW_ER_HT_RATIO)  (0.80 0.95))
    ((ho    OL_ER_HT_RATIO)  (0.05 0.15))
    ((mf    PD_SW_MLT)       (1 1)))

(defun transition-to-fix-only (test-fix-state)
  (create-transition-state
    :from-state test-fix-state
    :to-qde fix-only
    :assert '()
```

```
    :inherit-qdir '(WF Rnf PDfx PDnfx Rfoh mf Et Rfx Efx)
    :inherit-qmag '(WF Epl Etd Ees Rnf PDnfx Rfoh mf Et Rfx Efx)))

(define-QDE fix-only
  (text "Fix-only process of one increment")
  (quantity-spaces
    (Epl   (0 inf)              "Epl: detected errors remaining in pool")
    (Etd   (0 inf)              "Etd: detected errors")
    (Ees   (0 inf)              "Ees: escaped error from testing")
    (Rnf   (0 inf)              "Rnf: nominal bug-fixing rate")
    (WF    (0 ASG_WF inf)       "WF: development team size")
    (PDfx  (0 inf)              "PDfx: productivity of bug-fixing")
    (PDnfx (0 NML_FX_PD inf))
    (Rfoh  (0 inf)              "Rfoh: bug-fixing overheads rate")
    (mf    (0 PD_SW_MLT 1)      "mf: multiplier of productivity switchover")
    (mRfx  (minf 0)             "mRfx: minus bug-fixing rate")
    (Rfx   (0 inf)              "Ref: bug-fixing rate")
    (Efx   (0 inf)              "Efx: fixed errors")
    (Et    (0 inf)              "Et: total errors in increment"))

  (constraints
    ((constant PDnfx NML_FX_PD))
    ((constant WF))
    ((constant Et))
    ((constant Etd))
    ((constant Ees))
    ((constant mf PD_SW_MLT))
    ((constant Rfoh))
    ((d/dt Epl mRfx))
    ((d/dt Efx Rfx))
    ((minus Rfx mRfx))
    ((add Rfx Rfoh Rnf))
    ((M+ WF Rfoh))
    ((Mult PDnfx mf PDfx))
    ((Mult PDfx WF Rnf)))

  (transitions
    ((Epl (0 dec)) -> transition-to-implementation)))

(defun start-simulation ()
  (let* ((sim (make-sim :state-limit *))
         (initial-state (make-new-state
                         :from-qde implementation
                         :sim sim
                         :assert-values
                           '((S      (0          inc))
                             (WF     (ASG_WF     std))
```

```
                              (Rtc    (TEST_RATE   std))
                              (rERt   (AC_RT_RATE  std))
                              (PDnfx  (NML_FX_PD   std))
                              (Eag    (0           inc))
                              (Epg    (0           inc))
                              (Eaes   (ESC_AC_ER   std))
                              (Epes   (ESC_PS_ER   std))
                              (Rfoh   ((0 inf)     std)))
                         :text "start the increment development"))
        (qsim initial-state)))
```

B.3 Software Evolution Process Models

Software evolution process model for qualitative simulation

```
(define-QDE evolution
  (text "Software evolution process")
  (quantity-spaces
    (Sreq    (0 INI_REQ inf)    "Sreq: requirements to implement")
    (Simp    (0 INI_SYS inf)    "Simp: requirements implemented")
    (Rreq    (minf 0 inf)       "Rreq: net requirement rate")
    (Rimp    (0 inf)            "Rimp: net implement rate")
    (Rsd     (0 inf)            "Rsd: software develop rate")
    (Rin     (0 inf)            "Rin: requirement input rate")
    (Reft    (0 RAT_EFT inf)    "Reft: effective effort rate")
    (Rinc    (0 inf)            "Rinc: incorrect implement rate")
    (Rgen    (0 inf)            "Rgen: requirement generate rate")
    (Rexo    (0 inf)            "Rexo: exogenous requirement rate")
    (Rnew    (0 inf)            "Rnew: new requirement rate")
    (fie     (0 FCT_INR 1)      "fie: inertia effect factor")
    (finc    (0 FCT_INC 1)      "finc: fault generate factor")
    (fnew    (0 FCT_NEW 1)      "fnew: new feedback factor")
    (timer   (0 sim_end inf))
    (pace    (0 time_step inf)))

  (constraints
    ((d/dt Sreq Rreq))
    ((d/dt Simp Rimp))
    ((add Rreq Rsd Rin))
    ((add Rimp Rinc Rsd))
    ((add Rinc Rgen Rin))
    ((add Rnew Rexo Rgen))
    ((constant Rexo 0))
    ((M- Simp fie))
    ((mult Reft fie Rsd))
    ((mult Rsd finc Rinc))
    ((mult Rsd fnew Rnew))
```

```
    ((constant Reft RAT_EFT))
    ((constant finc FCT_INC))
    ((constant fnew FCT_NEW))
    ((d/dt timer pace))
    ((constant pace time_step)))

  (transitions
    ((timer (sim_end inc)) -> t))

(defun start-simulation ()
  (let* ((sim (make-sim :state-limit *))
         (initial-state (make-new-state
                         :from-qde evolution
                         :sim sim
                         :assert-values
                           '((Sreq    (INI_REQ    dec))
                             (Simp    (INI_SYS    inc))
                             (Reft    (RAT_EFT    std))
                             (fie     (1          dec))
                             (timer   (0          inc))
                             (pace    (time_step  std)))
                         :text "start the evolution process")))
         (qsim initial-state)))
```

Software evolution process model for semi-quantitative simulation

```
(define-QDE evolution
  (text "Software evolution process")
  (quantity-spaces
    (Sreq    (0 INI_REQ inf)    "Sreq: requirements to implement")
    (Simp    (0 INI_SYS inf)    "Simp: requirements implemented")
    (Rreq    (minf 0 inf)       "Rreq: net requirement rate")
    (Rimp    (0 inf)            "Rimp: net implement rate")
    (Rsd     (0 inf)            "Rsd: software develop rate")
    (Rin     (0 inf)            "Rin: requirement input rate")
    (Reft    (0 RAT_EFT inf)    "Reft: effective effort rate")
    (Rinc    (0 inf)            "Rinc: incorrect implement rate")
    (Ddev    (0 DEL_DEV inf))
    (Dflt    (0 DEL_FLT inf))
    (Dnew    (0 DEL_NEW inf))
    (Rgen    (0 inf)            "Rgen: requirement generate rate")
    (Rexo    (0 inf)            "Rexo: exogenous requirement rate")
    (Rnew    (0 inf)            "Rnew: new requirement rate")
    (fie     (0 FCT_INR 1)      "fie: inertia effect factor")
    (finc    (0 FCT_INC 1)      "finc: fault generate factor")
    (fnew    (0 FCT_NEW 1)      "fnew: new feedback factor")
    (timer   (0 sim_end inf)))
```

```
  (constraints
    ((d/dt Sreq Rreq))
    ((d/dt Simp Rimp))
    ((add Rreq Rsd Rin))
    ((add Rimp Rinc Rsd))
    ((add Rinc Rgen Rin))
    ((add Rnew Rexo Rgen))
    ((constant Rexo 0))
    ((M- Simp fie))
    ((mult Reft fie Rsd1))
    ((d3 Rsd1 Ddev Rsd))
    ((mult Rsd finc Rinc1))
    ((d3 Rinc1 Dflt Rinc))
    ((mult Rsd fnew Rnew1))
    ((d3 Rnew1 Dnew Rnew))
    ((constant Reft RAT_EFT))
    ((constant finc FCT_INC))
    ((constant fnew FCT_NEW)))

  (transitions
    ((timer (sim_end inc)) -> t))

  (initial-ranges
    ((Sreq  INI_REQ)    (50 50))
    ((Simp  INI_SYS)    (200 200))
    ((Reft  RAT_EFT)    (8 8))
    ((fie   FCT_INR)    (0.01 0.01))
    ((finc  FCT_INC)    (0.12 0.20))
    ((fnew  FCT_NEW)    (0.64 0.64))
    ((timer sim_end)    (156 156)))

  (envelopes
    ((M- Simp fie) (exact (lambda (x) (expt (/ 200 x) 3)))
                   (e-inv (lambda (y) (/ 200 (expt y 1/3)))))))

(defun start-simulation ()
  (let* ((sim (make-sim :state-limit *
                        :Q2-constraints t))
         (initial-state (make-new-state
                          :from-qde evolution
                          :sim sim
                          :assert-values
                            '((Sreq    (INI_REQ    dec))
                              (Simp    (INI_SYS    inc))
                              (Reft    (RAT_EFT    std))
                              (fie     (1          dec))
```

```
                        (timer   (0          inc)))
                    :text "start the evolution process")))
    (qsim initial-state)))
```

Appendix C

Data Extraction Form

Reflections of 10-Years' Progress on ProSim: A Systematic Review

Data Extraction Form

Reviewer: Study title:

First author: Source: ◯ ProSim workshops ◯ ICSP ◯ JSS ◯ SPIP Pub year: 2003

Study category: (one study can be classified into one or more categories, e.g. study introduces a new simulation paradigm and developes a simulator)

- ☐ A: software process simulation models or simulators (go to evaluate the questions in section A)
- ☐ B: process simulation modeling paradigms, methodology, or environment (go to section B, C)
- ☐ C: application, methodology, or guidelines for adopting process simulation in SE (go to section B, D)
- ☐ D: experience reports of software process simulation in practice (go to section B, E)

Section A:

How many models or simulators are described in this study? (append extra pages if more than two models included in a single study)

☐ Model #1:

Simulation purpose(s): ☐ strategic mgmt ☐ planning ☐ control & operational mgmt ☐ understanding ☐ training & learning ☐ process imprvmt & technology adoption ☐ N/A or

Problem domain in SE: e.g. open-source developmnt, software product-line, or global development

Model complexity: ◯ single module or ◯ integrated model, module num:

Modeling paradigm(s): ☐ system dynamics ☐ discrete-event ☐ state-based ☐ knowledge(rule)-based ☐ game theory ☐ agent-based ☐ hybrid ☐ qualitative ☐ N/A or

Simulation tool: ◯ Vensim ◯ Extend ◯ Stella/iThink ◯ N/A, or

(tool type) ◯ COTS ◯ open-source ◯ academic ◯ one-off developmt ◯ plug-ins ◯ other ◯ N/A

Model scope: ◯ portion of life-cycle if so, phase: ◯ requirement ◯ design ◯ code ◯ test or

◯ developmt project time span: ◯ <6months ◯ 6~12mon ◯ 12~24mon ◯ >24mon ◯ N/A

◯ multiple projects organization: ◯ <1 project team ◯ 1 project/product team ◯ multiple teams

◯ long-term product evolution geography: ◯ single site ◯ multi-site ◯ N/A country:

Output variables: (answer key questions) ☐ effort/cost ☐ time ☐ quality/defect ☐ resource ☐ productivity or

☐ Model #2:

Simulation purpose(s): ☐ strategic mgmt ☐ planning ☐ control & operational mgmt ☐ understanding ☐ training & learning ☐ process imprvmt & technology adoption ☐ N/A or

Problem domain in SE: e.g. open-source developmnt, software product-line, or global development

Model complexity: ◯ single module or ◯ integrated model, module num:

Modeling paradigm(s): ☐ system dynamics ☐ discrete-event ☐ state-based ☐ knowledge(rule)-based ☐ game theory ☐ agent-based ☐ hybrid ☐ qualitative ☐ N/A or

Simulation tool: ◯ Vensim ◯ Extend ◯ Stella/iThink ◯ N/A, or

(tool type) ◯ COTS ◯ open-source ◯ academic ◯ one-off developmt ◯ plug-ins ◯ other ◯ N/A

Model scope: ◯ portion of life-cycle if so, phase: ◯ requirement ◯ design ◯ code ◯ test or

◯ developmt project time span: ◯ <6months ◯ 6~12mon ◯ 12~24mon ◯ >24mon ◯ N/A

◯ multiple projects organization: ◯ <1 project team ◯ 1 project/product team ◯ multiple teams

◯ long-term product evolution geography: ◯ single site ◯ multi-site ◯ N/A country

Output variables: (answer key questions) ☐ effort/cost ☐ time ☐ quality/defect ☐ resource ☐ productivity or

Reflections of 10-Years' Progress on ProSim: A Systematic Review

Data Extraction Form

Section B:

Simulation purpose(s): ☐ strategic mgmt ☐ control & operational mgmt ☐ training & learning ☐ understanding ☐ N/A
☐ planning ☐ process imprvmt & technology adoption or

Modeling paradigm(s): ☐ system dynamics (continuous) ☐ discrete-event ☐ state-based ☐ rule-based ☐ agent-based
☐ game thry ☐ knowledge-based ☐ hybrid ☐ N/A or

Section C:

Study for: ☐ strategy/perspective ☐ simulation paradigm ☐ modeling method ☐ simulation environmt/tool ☐ others

Section D:

Question focused:

Solution proposed:

Effect (expected):

Section E:

Exp. source: ☐ industry ☐ government ☐ education ☐ other Outcome: ◯ success ◯ fail ◯ mixed

Arguments: (supported by exp.)

Study quality assessment: (assess quality questions related to identified study category. yes=1.0, partial=0.5, no=0)

Did the study clearly state the aims or research questions? ◯ yes, explicit & pertinent ◯ partial ◯ no

Did the study review the related work for the problem? ◯ yes, >=3 ◯ partial, 1~2 ◯ no

Did the study discuss related issues, and compare with the alternatives? ◯ yes, >=3 ◯ partial, 1~2 ◯ no

Did the study recommend the future continous research? ◯ yes, highly related ◯ partial ◯ no

A

Are model's assumptions explained explicitly (or supplementary meterial given, e.g. webpage)? ◯ yes, most ◯ partial ◯ no

Is model construction fully described (or supplementary meterial given, e.g. webpage)? ◯ yes, detailed ◯ partial, brief ◯ no

Did the study explain why choosing the simulation paradigm(s)? ◯ yes ◯ no

Are the conditions of model adoption explained? ◯ yes, explicit ◯ partial, brief ◯ no

Did the study attempt avoid the selection bias during experiment design? ◯ yes, effective ◯ partial ◯ no

Has the model been trialed on an industry scale problem? ◯ yes, fully ◯ partial ◯ no

Did the study carry out a sensitivity or residual analysis? ◯ yes ◯ no

Are any model evaluation methods applied on the model? ◯ yes, >=2 ◯ partial, 1method ◯ no

Did the study interpret the findings? ◯ yes, detailed ◯ partial, brief ◯ no

B

Are the scopes of the method/paradigm/solution clearly defined? ◯ yes, explicit ◯ partial, implicit ◯ no

C

Are the modeling approach/method/environment clearly defined? ◯ yes, detailed/clear ◯ partial, brief ◯ no

Are the problems that the study addresses explained with appropriate SE examples? ◯ yes, appropriate ◯ partial ◯ no

Did the study epecify the limitations of the argued paradigm/method/solution? ◯ yes, explicit ◯ partial, brief ◯ no

Did the empirical evidence support the arguemtns of the study? ◯ yes, fully ◯ partial ◯ no

D

Is the experience used for validating and calibrating simulation model/modeling? ◯ yes, fully ◯ partial ◯ no

Are best practices or lessons learnt extracted from experienced? ◯ yes, both ◯ partial, single ◯ no

Final comments: