Qualitative vs. Quantitative Software Process Simulation Modeling: Conversion and Comparison

He Zhang
Lero Research Centre
Department of Computer Science
and Information Systems
University of Limerick, Ireland
he.zhang@lero.ie

Barbara Kitchenham
School of Computer Science
and Mathematics
Keele University, UK
barbara@cs.keele.ac.uk

Ross Jeffery
National ICT Australia
School of Computer Science
and Engineering, University
of New South Wales
ross.jeffery@nicta.com.au

Abstract

Software Process Simulation Modeling (SPSM) research has increased in the past two decades. However, most of these models are quantitative, which require detailed understanding and accurate measurement. As the continuous work to our previous studies in qualitative modeling of software process, this paper aims to investigate the structure equivalence and model conversion between quantitative and qualitative process modeling, and to compare the characteristics and performance of these two approaches by modeling and simulating a software evolution process. Following the model conversion scheme, the reference quantitative (SD) model and the corresponding qualitative model become comparable. The results present their different capabilities and interesting perspectives, and further the potential use of qualitative modeling in software process research.

1 Introduction

Software Process Simulation Modeling (SPSM) was introduced into the software engineering domain by Abdel-Hamid and others' pioneering work summarized in [1]. In the last two decades, it has been emerging as an effective tool to help evaluate and manage changes made to software projects and organizations. However, most of software process models for simulation are quantitative, and require a detailed understanding and accurate measurement of software processes, which rely on reliable and precise history data. Unfortunately, in many cases these data are not readily available, which limits its adoption in practice.

As the counterpart of quantitative modeling, qualitative approach is able to cope with the lack of precise knowledge by modeling at a more abstract level than quantitative modeling. Our previous work explored qualitative modeling and simulation of software process at different process scales [16, 17], and justified its value in software process research.

This paper first develops a model conversion scheme between quantitative and qualitative process models. It further reports our study of comparing the modeling characteristics and performance between these two approaches by modeling and simulating the same software process.

1.1 The Systematic Review

In 2007, we undertook a systematic literature review of SPSM research in the past decade (1998-2007). This review revisited and updated the state-of-the-art of SPSM, summarized the 10-years progress, discovered trends, and suggested possible directions in this research domain [18, 19].

The systematic review found *phase*, *project*, and *product evolution* as the most modeled software process scales; identified *generic development*, *software evolution*, *software process improvement*, and *incremental development* as the top four in the most interesting topics in SPSM research. As we previously investigated qualitative modeling of *generic development* (software staffing process) and *incremental development* at *project* [16] and *phase* [17] scale, this paper compares these two modeling approaches with focus on *software evolution*. Moreover, the successful modeling and simulating software processes at all the above scale levels provides evidence for justifying the value of qualitative modeling approach in software process research [15].

1.2 Reference Model Selection

In addition to the modeling of software evolution process, more importantly, the goal of this paper is to develop

IEEE
computer
society

and implement an element level mapping for converting a typical quantitative simulation model into a corresponding qualitative model; and moreover, compares the model structures and outputs of quantitative simulation against its counterpart, the qualitative process model. To achieve this goal, a typical quantitative software process simulation model has to be selected as the reference model in this study for model conversion and comparison.

Therefore, the selection of reference quantitative model is the first and important step in this study. Apart from the findings derived from the systematic review (Section 1.1), the selection criteria also need to include two extra aspects:

Paradigm The reference model should be constructed with a typical quantitative simulation modeling paradigm, which has been widely accepted by SPSM research. According to the systematic literature review described in [18], 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 simulation model is one type of continuous simulation with the incomplete feature of discrete transition. From the point of comparability, SD becomes the selected modeling paradigm for comparison due to its wide use in SPSM and its 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 than construction of a complicated model.

In terms of the above criteria of the reference quantitative process 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, some of them available in [14, 2, 4, 12, 13]. Among them, Wernick and Hall's model [13] consists of single module, whose structure is simpler than others'. Moreover, their model is the most recent SD model of software evolution process found in the primary studies of the systematic review.

The following sections of this paper are structured as follows. Section 2 briefly introduces our conversion scheme between quantitative (SD) and qualitative process models. Section 3 describes software evolution process and replicates a simplified SD model of software evolution. In Section 4, we develop a corresponding qualitative model of software evolution by applying the model conversion scheme. A comparison of characteristics and performance between these two models is discussed in Section 5. We present our current conclusions in Section 6.

2 Structure Equivalence and Model Conversion

This section presents the elements of quantitative (SD) modeling, and describes how to convert them for qualitative and semi-quantitative modeling.

2.1 Causal Loop Diagram

In SD several modeling components and tools are used to capture the structure of systems, including *causal loop diagram* (CLD), *level* and *rate*, and *delay*. Among 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 [5], 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 modeling [6]. Therefore, a CLD can be converted into its corresponding qualitative diagram, but needs more explicit and specific qualitative assumptions.

2.2 Level & Rate

Level (or stock) and rate (or flow) are the central concepts of SD modeling. 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 1-a). Valves control the flow rates. Clouds represent the sources and sinks for the flows, which are both outside of the model boundary.
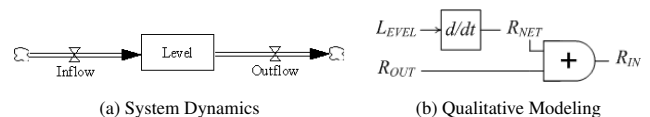

(a) System Dynamics   (b) Qualitative Modeling

Figure 1. General structures of level and rate

The structure represented in Figure 1-a corresponds exactly to the following integral equation:

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

$$(1)$$

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

$$\frac{\partial}{\partial t}\left(Level\right) = inflow\left(t\right) - outflow\left(t\right) \qquad (2)$$

Hence, this dynamics of systems can be modeled by *differential* relation [5] in ASD. Unlike SD diagramming, the rate difference (net flow) has to be explicitly presented in qualitative diagram. Figure 1-b shows the converted level and rate in qualitative modeling diagram.

2.3 Delay

Forrester identified two characteristics of a delay [3]. 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 2).

Exponential delay is the most frequently used delay class in SD. Figure 2 shows two common types of exponential delay: first-order delay (a), and third-order delay (b). Mathematically speaking, an nth-order delay is equivalent to n cascaded single-order delays, with each single-order delay having a delay time of $\frac{D}{n}$ [3].

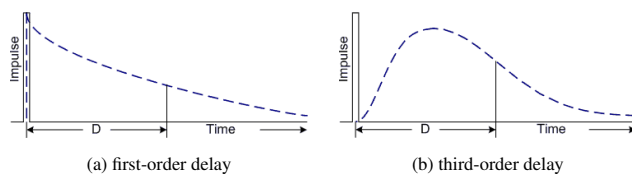(a) first-order delay   (b) third-order delay

Figure 2. Exponential delay curves

Since time is treated qualitatively in qualitative modeling and simulation, there is no meaning to handle a delay with '*qualitative*' length. Therefore, QSIM algorithm [8] does not explicitly consider delay phenomenon, and neither include built-in mechanism of delay.

More details of structure equivalence and model conversion between qualitative and quantitative process modeling can be found in [15].

3 Reference Quantitative Model

This section introduces the background of the software process modeled, software evolution processes, and replicates this model for further comparison.

3.1 Software Evolution Process

With respect to the results of the systematic review [18], 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 [7]. Models of such patterns permitted the forecasting of future overall system growth and growth rates. Moreover, the observed patterns of behavior appearing yielded common phenomenological interpretations.

3.1.1 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 [12]. 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 aging effects, which are referred to as '*anti-regressive*' activities [4].

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 [12]. 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 [10, 11]. 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 [2]. This positive feedback is recognized 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 [4].

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 [14]. 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 modeling, again calculated as portions of the successful implementation flow.

3.2 A Simplified Quantitative Model

3.2.1   Model Description

The reference quantitative model (shown in Figure 3) was developed using Vensim simulation modeling environment (from Ventana Systems, Inc.). Although it is a simplified model, it 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*' [10]. 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 [14]. 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.

3.2.2   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 [2]. To guarantee the replication of the reference model, most of these variable inputs are kept in this study.

Neither Wernick and Hall's SD model nor Turski's reference model [11] 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 section is based on the SD flow graphs, inputs, outputs, and relation equations published in [2, 4, 12, 13] (no full version models published). As result of this divergence, the output of the replicated model is slightly different from the reference model. Figure 4 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.
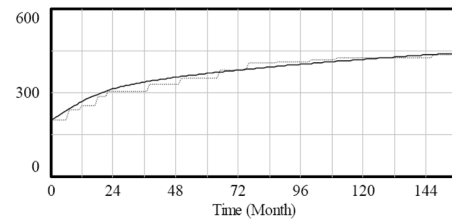
(a) system size simulated by the reference model
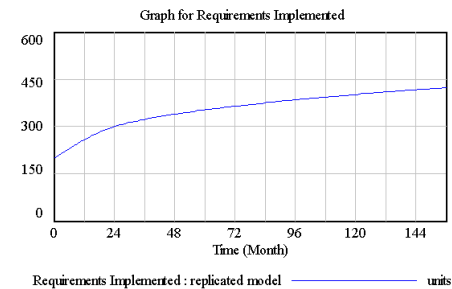
(b) system size simulated by the replicated model

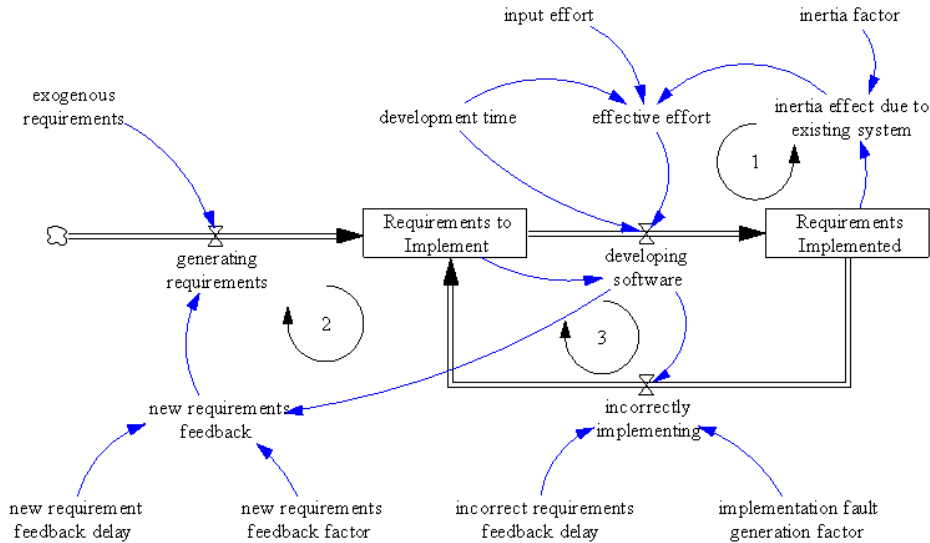**Figure 4. Simulation of implemented requirements over time**

Figure 3. Reference quantitative model of software evolution

3.2.3 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 5. 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 modeled 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 the policy sensitivity of three feedback loops respectively. Figure 6 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.

4 Corresponding Qualitative Model

As the first step of qualitative modeling, the qualitative assumptions need to be abstracted from the real world system [5]. 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 scheme discussed in Section 2. After that, the qualitative assumptions can be extracted from the corresponding qualitative model.

Figure 7 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.

S_{Req}: Requirements to implement
R_{Req}: Requirement generation rate
R_{New}: New requirement feedback rate
R_{SD}: Software development rate
R_{In}: Requirement input rate
R_{Eft}: Effective effort rate
f_{new}: new requirement feedback factor

S_{Imp}: Requirements implemented
R_{Imp}: Requirement implementation rate
R_{Inc}: Incorrect implementation rate
R_{Gen}: Requirement generation rate
R_{Exo}: Exogenous requirement rate
f_{ie}: inertia factor
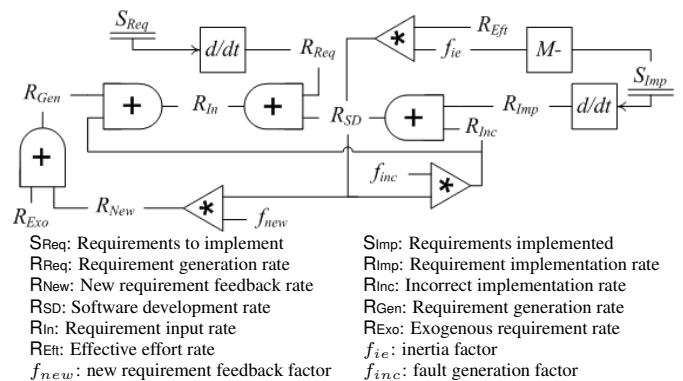f_{inc}: fault generation factor

Figure 7. Corresponding qualitative model of software evolution

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

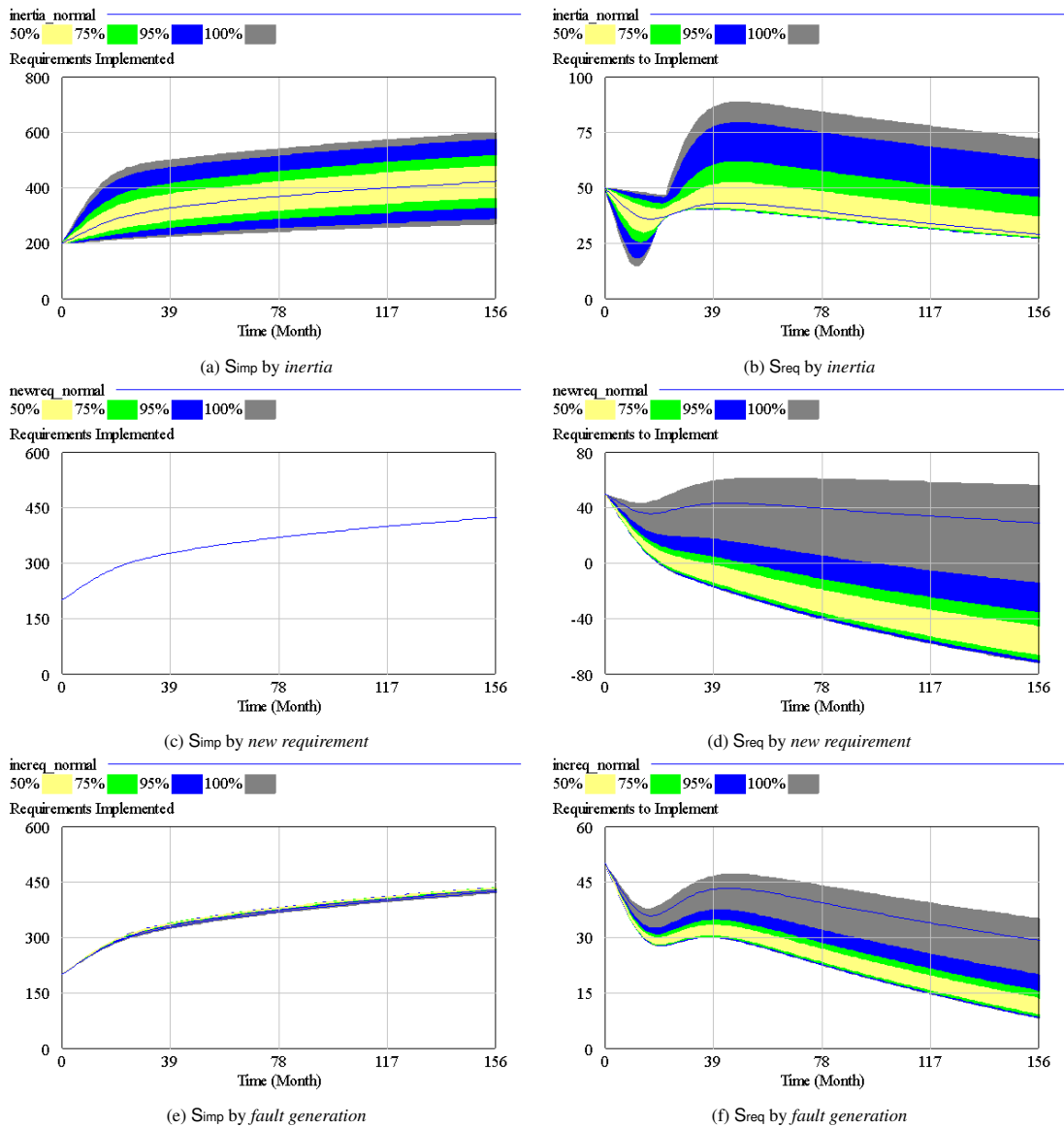1. *Requirements to implement* (S_{Req}) come from *exoge-*

349

(a) S_{imp} by *inertia*

(b) S_{req} by *inertia*

(c) S_{imp} by *new requirement*

(d) S_{req} by *new requirement*

(e) S_{imp} by *fault generation*

(f) S_{req} by *fault generation*

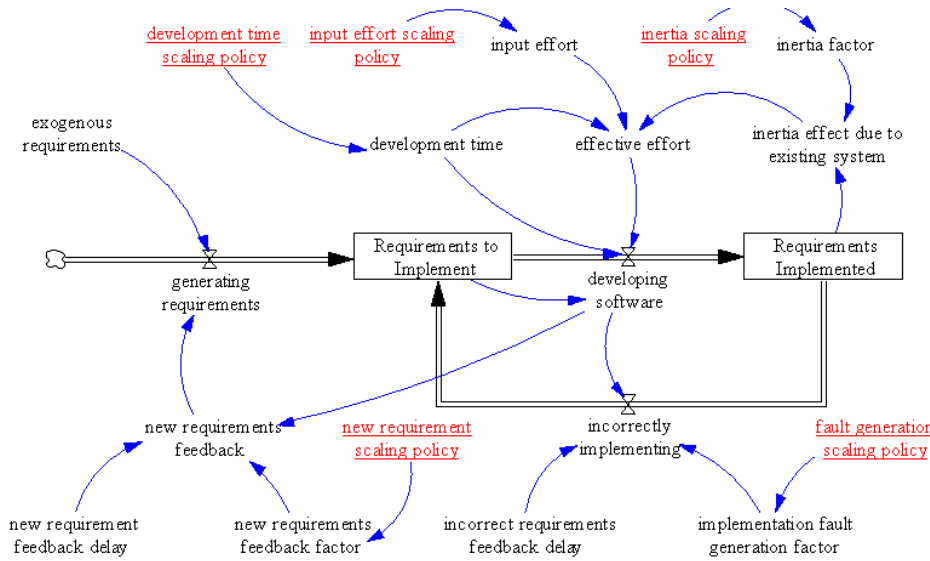**Figure 6. Sensitivity of policy change for reference model**

Figure 5. Quantitative model of software evolution for policy investigation

nous requirements, *new requirements feedback* and *incorrect requirements feedback*;

2. S_{Req} is transferred to *Requirements implemented* (S_{Imp}) at *software developing* rate (R_{SD});

3. The *incorrectly implemented requirements*, as a small portion of S_{Imp} is returned to S_{Req} for rework;

4. Increasing existing system size (S_{Imp}) incurs more effort needed for '*anti-regressive activities*', and decreases R_{SD};

5. The *input effort* (R_{Eft}) has linear relationship with R_{SD};

6. The *new requirement feedback* (R_{New}) has linear relationship with R_{SD};

7. The *incorrectly implementation* (R_{Inc}) has linear relationship with R_{SD};

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

9. There is no *exogenous requirements* ($R_{Exo} = 0$) during the evolution process.

5   Simulation & Comparison

The reference model set the simulation terminated at predefined time point (the 156th 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 behavior patterns. So the qualitative simulation cannot stop itself on these behaviors until runs out of memory. In experiment, the simulation was set to halt after generating a '*large*' number of behaviors, which is sufficient to observe the behavior patterns.

The qualitative simulation generates a diversity of behaviors of the evolution process, most of which are the combinations of varying patterns of important variables. Figure 8 shows the typical behavior patterns of some important variables in the model.

Requirements Implemented   The simulated behavior of S_{Imp} presents growing trend at all time (Figure 8-a), which is also quantitatively reflected in Figure 4 and Figure 6-a, c, e. It is because *developing software* is always greater than *incorrectly implementing*.

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-b). The qualitative simulation reflects the result from reference model (Figure 6-b, d, f).

Requirement implementation rate   It is easy to identify the oscillation of R_{Imp} from the simulated possible behavior (Figure 8-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
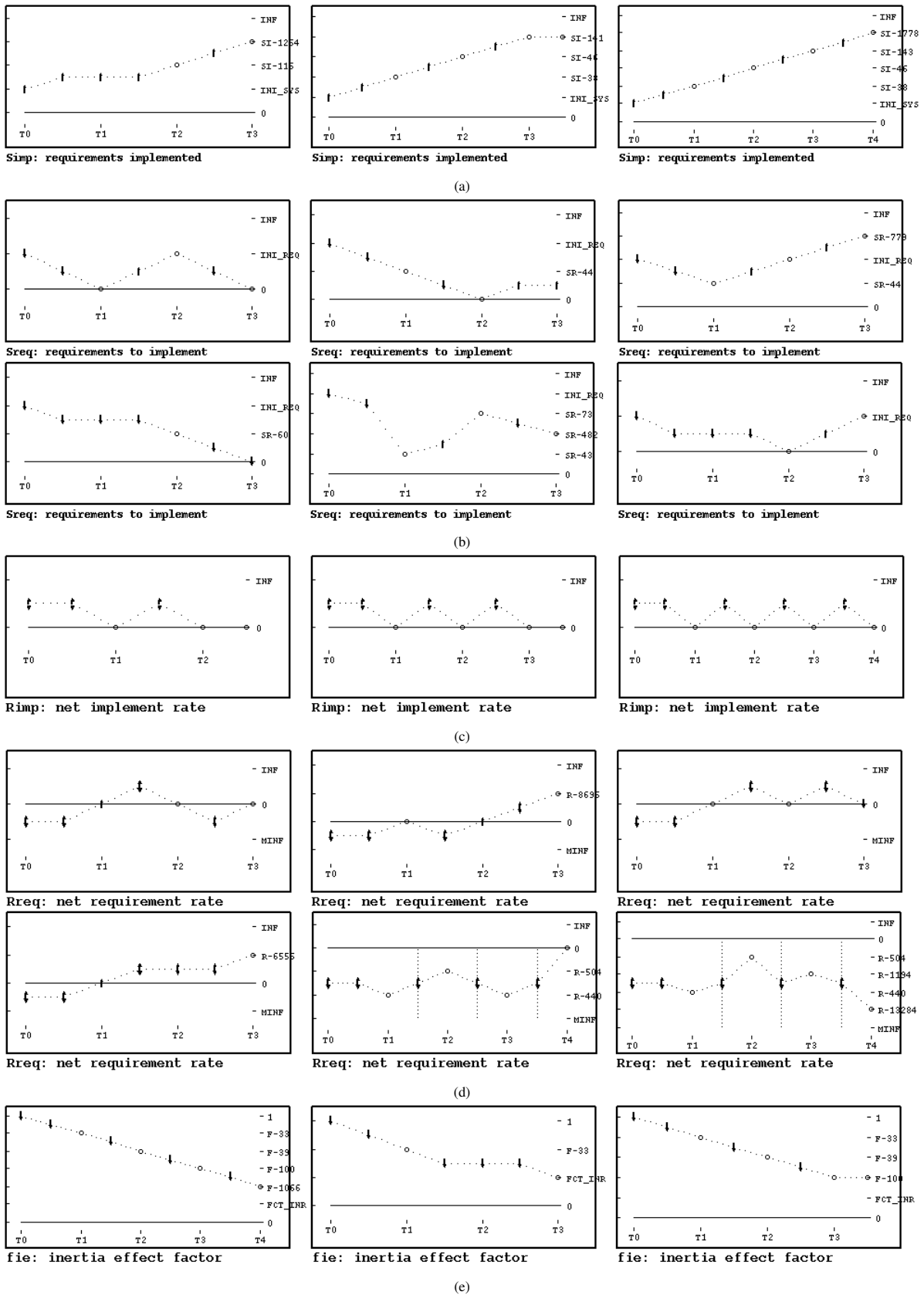
Figure 8. Behaviors of qualitative simulation

phenomenon is to a large degree consistent with the qualitative behaviors observed by Ramil and Smith's study in [9], which constructed qualitative simulation model based on analytic model, instead of continuous structural model in this paper.

Requirement generation rate Figure 8-d reflects the '*unpredictable*' behavior 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 behaviors.

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-e).

Note that one unrealistic phenomenon is observed in the sensitivity analysis of SD model (Figure 6-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 qualitative modeling, it can be avoided in advance with one required step of modeling, an explicit quantity space declaration (`Sreq (0 inf)`). In contrast, it is difficult for SD modelers to involve and check such constraints in their modeling stage.

Overall, both qualitative and quantitative (system dynamics) modeling compared here have their strength and weakness. In modeling, 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 or semi-quantitative counterpart.

In simulation, the quantitative (SD) model can present variable's varying trend with more details during the course. Whereas, the qualitative (QSIM) model reflects trends at a more abstract level. However, when dealing with uncertainty, the value range (or discrete values) and associated probability distribution are required for any (stochastic) quantitative simulation. In contrast, qualitative process model can allow the lack of such information in simulation. According to the modeling process and simulation results, the qualitative modeling provides a more rigorous and robust mechanism in constructing process models, and then is able to avoid unrealistic results sometimes generated in simulation.

6 Conclusions

Our previous studies investigated the use of qualitative modeling in software engineering, and constructed qualitative process models at *phase* and *project* scale levels. As the continuous and enhanced work, this paper first discussed structure equivalence between qualitative and quantitative (system dynamics) process models, and developed a model conversion scheme based on an element level mapping. By developing the corresponding qualitative model of the reference quantitative (SD) model using this scheme, the characteristics and performance of modeling and simulation become comparable between these two approaches. The major contributions of this paper are highlighted as the follows:

1. An element level mapping from CLD and equations of system dynamics to ASD of qualitative modeling is established, and vice versa.

2. A model conversion scheme from a quantitative (SD) model to qualitative model is implemented by using the element level mapping. Given a qualitative model and its quantification, it is possible to covert and construct its corresponding SD model as well.

3. The software evolution processes are revisited by using qualitative modeling and simulation.

4. The modeling process and characteristics are compared between system dynamics and qualitative modeling; and further the comparison of simulation capability and results between quantitative (SD) and qualitative approaches are presented.

The quantitative and qualitative modeling approaches compared in this paper offer a number of different capabilities and interesting perspectives in software process research. The future work in this direction can be continued as follows but not limited to:

1. Investigating model conversion between quantitative and qualitative approaches in constructing discrete process model.

2. Integrating qualitative and quantitative modeling in development of hybrid software process model.

7. Acknowledgments

References

[1] T. K. Abdel-Hamid and S. E. Madnick. *Software Project Dynamics: An Integrated Approach*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[2] B. Chatters, M. Lehman, J. F. Ramil, and P. Wernick. Modelling a software evolution process: A long-term case study. *Software Process: Improvement and Practice*, 5(2-3), 2000.

[3] J. W. Forrester. *Industrial Dynamics*. System Dynamics Series. Pegasus Communications, 1969.

[4] G. Kahen, M. Lehman, J. F. Ramil, and P. Wernick. System dynamics modeling of software evolution processes for policy investigation: Approach and example. *Journal of Systems and Software*, 59(3), 2001.

[5] B. Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, 1994.

[6] B. Kuipers. Qualitative simulation. In R. Meyers, editor, *Encyclopedia of Physical Science and Technology*, pages 287–300. Academic Press, NY, 2001.

[7] M. Lehman and J. F. Ramil. Software evolution - background, theory, practice. *Information Processing Letters*, 88, 2003.

[8] U. Qualitative Reasoning Group. Qsim. http://www.cs.utexas.edu/users/qr/QR-software.html. University of Texas.

[9] J. F. Ramil and N. Smith. Qualitative simulation of models of software evolution. *Software Process: Improvement and Practice*, 7(3-4), 2002.

[10] W. M. Turski. The reference model for smooth growth of software systems. *IEEE Transactions on Software Engineering*, 22(8), 1996.

[11] W. M. Turski. The reference model for smooth growth of software systems revisitied. *IEEE Transactions on Software Engineering*, 28(8), 2002.

[12] P. Wernick and T. Hall. Simulating global software evolution processes by combining simple models: An initial study. *Software Process: Improvement and Practice*, 7(3-4), 2002.

[13] P. Wernick and T. 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.

[14] P. Wernick and M. Lehman. Software process white box modelling for feast/1. *Journal of Systems and Software*, 46(2-3), 1999.

[15] H. Zhang. *Qualitative and Semi-quantitative Modelling and Simulation of Software Engineering Processes*. PhD thesis, University of New South Wales, 2008.

[16] H. Zhang, M. Huo, B. Kitchenham, and R. 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.

[17] H. Zhang, J. Keung, B. Kitchenham, and R. 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.

[18] H. Zhang, B. Kitchenham, and D. Pfahl. Reflections on 10 years of software process simulation modelling: A systematic review. In *International Conference on Software Process (ICSP'08)*, Leipzig, Germany, 2008. Springer.

[19] H. Zhang, B. Kitchenham, and D. Pfahl. Software process simulation modeling: Facts, trends, and directions. In *15th Asia-Pacific Software Engineering Conference (APSEC'08)*, Beijing, China, 2008. IEEE Computer Society.