Software Project Simulator for Effective Software Process Improvement

Osamu Mizuno

June 2001

Software Project Simulator for Effective Software Process Improvement

Osamu Mizuno

June 2001

Dissertation submitted to the Graduate School of Engineering Science of
Osaka University in partial fulfillment of the requirements
for the degree of Doctor of Engineering

Abstract

Assuring software quality and productivity is a major problem facing software engineering. In general, software quality affects productivity, since software products with higher quality require less rework and maintenance. To develop and maintain the software products, developers may use various activities, methods, practices and transformations. A software process is defined as a set of such activities, methods, practices and transformations that people use to develop and maintain software and the associated products, e.g., project plans, design documents, code, test cases, and user manuals. Software process improvement is a deliberate methodology for improving software processes, and it includes understanding, definition, measurement, and optimization of the processes. Process improvement is indispensable for improving the quality of software products and the productivity of the development process.

In a certain company (hereafter, called Company *A*) that develops software embedded in social infrastructure systems, low quality and low productivity became major problems about 10 years ago. Since such problems were primarily caused by disorganized projects, that is, the projects were executed under loose management, the company recognized the necessity of improving the software process for such disorganized projects. To promote process improvement, Company *A* then established a group called the Software Engineering Process Group (SEPG). The SEPG collected data on software metrics from software development projects, and faced the challenge of identifying disorganized projects based on the statistical analysis of the collected data. By means of such activities, the SEPG

succeeded in clearly identifying disorganized projects. As a result, the SEPG's objective was shifted from the identification of disorganized projects to the improvement of their processes. However, even if process improvement activities were devised, they were seldom applied to actual projects, since those activities require a large amount of resources, and software developers were skeptical about the effectiveness of improvement activities.

This kind of problem is common to other software organizations, and a cost-effective approach for software process improvement is strongly required. Simulation techniques have been introduced to answer such a requirement, and software process simulation is known to be one of the most cost-effective solutions for facilitating process improvement activities. Our goal has been to establish a new software simulation technique with a high simulation accuracy. We have also aimed at investigating its applicability to actual software process improvement in Company *A*.

In this paper, we propose a new model for describing software processes and a simulation method for software projects that aims at effective software process improvement. A formal model is developed based on experience and data from software development processes at Company *A*. The model consists of a Project model and a Process model. The Project model focuses on three key components: activities, products, and developers of a project. The Process model includes a set of Activity models, each specifying design, coding, review, test, and debug activities using a Generalized Stochastic Petri Net (GSPN). The new model handles the effect of human factors by introducing the concept of "workload". Next, we develop a simulator that supports the description of a software process, and executes the software process according to the description. As a result of its execution, we obtain estimated values for the quality, cost and delivery date of the target process. Experimental results show the applicability of the proposed simulator.

Based on the developed simulator, we simulate 3 process improvements: (1)

improvement of parallel execution of activities, (2) improvement of test phase planning, and (3) application to risk prediction.

First, we investigate experimentally the influence of "parallel execution" of activities in a development process. Parallel execution is a situation where an activity, such as coding, begins before the previous activity, such as module design, finishes. In a standard waterfall model, the situation should be avoided because it tends to cause confusion in a project. However, in actual development projects in Company *A*, such parallel executions frequently occur in order to reduce the development schedule. We thus try to show the effect of the parallel execution, and to improve parallel execution in Company *A*. To do so, we simulate two cases: a case without any parallel execution and a case with parallel execution between module design and coding activities. The results of two cases are compared and evaluated.

Next, we try to apply the proposed simulator to update a project plan. Generally, a development plan should be updated according to the actual progress of a project. However, some of the solutions for updating a development plan are less than optimal. Hence, project plans are updated based on the experience and intuition of developers. We thus apply the proposed simulator to support updating a development plan. To do so, we consider a simple development process consisting of design and debug phases, which is derived from the actual development process in Company *A*. We then propose a two-phase project control method that examines the initial development plan at the end of the design phase, updates it to current status of the development process, and executes a debug phase under a new (updated) plan. In order to show its usefulness, we define 3 imaginary projects: a project that executes the debug phase under the initial plan, a project that applies the proposed approach, and a project that follows a uniform (that is, trivial) plan.

Finally, we present an extension of the proposed simulator in order to apply risk prediction with a cost estimating capability. We have developed a risk

predicting system to find "risky" projects by statistical analyses on risk questionnaires from project managers in Company *A*. In this approach, only a probability of riskiness was calculated for a project. The managers, however, wanted to be given concrete proof why a software project becomes risky. To present proof that a software project is becoming risky, we try to extend the simulator so that it can deal with risk factors. To do so, fluctuation in skill level and deadline pressure are represented by modifying parameters in the simulator. A case study confirms that the extended simulator can estimate the development cost under some typical risks.

This dissertation is organized as follows: The first and second chapters are introductory. In Chapter 1, we briefly summarize related progress and topics in software process simulation and describe the outline of the dissertation. In Chapter 2, we describe the background for software process improvement and we also show process improvement activities performed in the targeted organization.

In Chapter 3, we define a model to describe a software process using Generalized Stochastic Petri Nets (GSPN). Examples describing typical activities in a development process are also shown. we then implement a simulator based on the proposed model, and evaluate the simulator experimentally based on actual project data.

In Chapter 4, we describe the first application of the simulator to process improvement. In this chapter, we show that the parallel execution of development activities (such as executing module design and coding activities simultaneously) increase project costs and residual faults.

In Chapter 5, we show the second process improvement example. In this application, we use several simulations to investigate the effect of dynamic updating of a project plan.

In Chapter 6, we extend the application of the simulator to risk prediction. We adopt some extensions to the simulator to manage risk factors in software projects.

Finally, I conclude this dissertation with a summary and directions for future work in Chapter 7.

List of Major Publications

(1) Yuji Hirayama, Osamu Mizuno, Shinji Kusumoto, and Tohru Kikuno, "Hierarchical project management model for quantitative evaluation of software process," *Proc. of International Symposium on Software Engineering for the Next Generation*, pp.40–49 (February 1996).

(2) Osamu Mizuno, Yuji Hirayama, Shinji Kusumoto, and Tohru Kikuno, "Application of generalized stochastic Petri net to quantitative evaluation of software process," *Proc. of 1996 IEEE International Conference on Systems, Man and Cybernetics*, pp.3192–3197 (October 1996).

(3) Shinji Kusumoto, Osamu Mizuno, Yuji Hirayama, Tohru Kikuno, Yasunari Takagi, and Keishi Sakamoto, "A new project simulator based on generalized stochastic Petri net," *Proc. of 19th International Conference on Software Engineering (ICSE'97)*, pp.293–303 (May 1997).

(4) Katsumi Inagaki, Yasunari Takagi, Keishi Sakamoto, and Osamu Mizuno, "Analyzing the cost estimation accuracy in software project respect to productivity and quality," *Proc. of International Symposium on Future Software Technology '97*, pp.372–377 (October 1997).

(5) Osamu Mizuno, Shinji Kusumoto, Tohru Kikuno, Yasunari Takagi, and Keishi Sakamoto, "Estimating the number of faults using simulator based on generalized stochastic Petri net," *Proc. of 6th Asian Test Symposium*, pp.269–274 (November 1997).

(6) Osamu Mizuno, Shinji Kusumoto, Tohru Kikuno, Yasunari Takagi, and Keishi Sakamoto, "Experimental evaluation of two-phase project control for software development process," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E81-A, no.4, pp.605–614 (April 1998).

(7) Osamu Mizuno, Tohru Kikuno, Katsumi Inagaki, Yasunari Takagi, and Keishi Sakamoto, "Analyzing effects of cost estimation accuracy on quality and productivity," *Proc. of 20th International Conference on Software Engineering (ICSE'98)*, pp.410–419 (April 1998).

(8) Osamu Mizuno, Shinji Kusumoto, and Tohru Kikuno, "Customization of software project simulator for improving estimation accuracy," *Proc. of 9th International Symposium on Software Reliability Engineering, Fast Abstracts & Industrial Practices*, pp.47–48 (November 1998).

(9) Satoru Uehara, Osamu Mizuno, Yumi Itou, and Tohru Kikuno, "An MVC-based analysis of object-oriented system prototyping for banking related GUI applications — Correlationship between OO metrics and efforts for requirement change —," *Proc. of 4th International Workshop on Object-Oriented Real-time Dependable Systems*, pp.91–104 (January 1999).

(10) Osamu Mizuno and Tohru Kikuno, "Empirical evaluation of review process improvement activities with respect to post-release failure," *Proc. of Empirical Studies of Software Development and Evolution, ICSE'99 Workshop*, pp.50–53 (May 1999).

(11) Satoru Uehara, Osamu Mizuno, and Tohru Kikuno, "A straightforward approach to effort estimation for updating programs in object-oriented prototyping development," *Proc. of 6th Asia-Pacific Software Engineering Conference*, pp.144–151 (December 1999).

(12) Osamu Mizuno, Tohru Kikuno, Katsumi Inagaki, Yasunari Takagi, and

Keishi Sakamoto, "Statistical analysis of deviation of actual cost from estimated cost using actual project data," *Information and Software Technology*, vol.42, pp.465–473 (May 2000).

(13) Osamu Mizuno, Tohru Kikuno, Yasunari Takagi, and Keishi Sakamoto, "Characterization of risky projects based on project managers' evaluation," *Proc. of 22nd International Conference on Software Engineering (ICSE2000)*, pp.387–395 (June 2000).

(14) Nahomi Kikuchi, Osamu Mizuno, and Tohru Kikuno, "Identifying key attributes of projects that affect the field quality of communication software," *Proc. of 24th Annual International Computer Software & Applications Conference (COMPSAC2000)*, pp.176–178 (October 2000).

(15) Masayuki Hirayama, Tetsuya Yamamoto, Takuya Kishimoto, Osamu Mizuno, and Tohru Kikuno, "Systematic generation of software test items based on system behavior from user's viewpoint," *Proc. of 5th International Conference on Probabilistic Safety Assessment and Management (PSAM5)*, pp.2377–2382 (November 2000).

(16) Masayuki Hirayama, Tetsuya Yamamoto, Takuya Kishimoto, Osamu Mizuno, and Tohru Kikuno, "Generating test items for checking illegal behavior in software testing," *Proc. of 9th Asian Test Symposium (ATS2000)*, pp.235–240, (December 2000).

(17) Satoru Uehara, Osamu Mizuno, and Tohru Kikuno, "An implementation of electronic shopping cart on the Web system using component-object technology," *Proc. of 6th Workshop on Object-oriented Real-Time Dependable Systems (WORDS2001)*, pp.85–92, (January 2001).

(18) Shinji Kusumoto, Osamu Mizuno, Tohru Kikuno, Yuji Hirayama, Yasunari Takagi, and Keishi Sakamoto, "Software project simulator for effective process improvement," *Trans. of Information Processing Society of Japan*, vol.42,

no.3, pp.396–408, (March 2001).

(19) Satoru Uehara, Osamu Mizuno, and Tohru Kikuno, "Design of New Mechanism for Context Data Storing on Web System and Its Implementation using Component-Object Technology," *Trans. of IEICE* (in Japanese), vol.J84-D-I, no.6, pp.713–722 (June 2001).

(20) Osamu Mizuno, Daisuke Shimoda, Tohru Kikuno, and Yasunari Takagi, "Enhancing software project simulator toward risk prediction with cost estimation capability," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E84-A, no.11 (To appear in November 2001).

(21) Satoru Uehara, Osamu Mizuno, and Tohru Kikuno, "A new approach to estimate effort to update object-oriented programs in prototyping development," *IEICE Trans. on Information and Systems* (Conditionally accepted).

List of Figures

List of Tables

Acknowledgments

During the course of this work, I have been fortunate to have received assistance from many individuals. I would especially like to thank my supervisor Professor Tohru Kikuno for his continuous support, encouragement, and guidance for this work.

I am also very grateful to the members of my thesis review committee: Professor Katsuro Inoue and Professor Toshinobu Kashiwabara for their invaluable comments and helpful criticisms of this thesis.

Many of the courses that I have taken during my graduate career have been helpful in preparing this thesis. I would like to acknowledge the guidance of Professors Toru Fujiwara, Ken-ichi Hagihara, Akihiro Hashimoto, Teruo Higashino, Masaharu Imai, Tadahiro Kitahashi, Toshimitsu Masuzawa, Hideo Miyahara, Masayuki Murata, Shinji Shimojo, Shin-ichi Tamura, Haruo Takemura, and Ken-ichi Taniguchi.

I would like to express my special thanks to Associate Professor Shinji Kusumoto for his assistance and invaluable advice.

I also would like to thank Mr. Yuji Hirayama and Mr. Kouichi Furusawa, who were engaged in previous studies in our laboratory. Their works help my dissertation so much.

I wish to thank Dr. Keishi Sakamoto, an SPI consultant, for his kind cooperation and insightful advice for my study. I also wish to thank Mr. Yasunari Takagi of OMRON Corporation for his helpful advice and support in this research. I also would like to express my thanks to Mr. Katsumi Inagaki, Mr. Naoki Niihara,

Mr. Toshiki Niki, Mr. Yoshifumi Sakakiya, and Mr. Toshifumi Tanaka of OMRON corporation for their cooperation with data collection from actual software development field.

I would like to acknowledge Dr. Dee Worman of EditScience, Inc. I also would like to thank Mr. John Mackin of Fujitsu Corporation for editing English of a draft of this dissertation. His editing greatly helps me to complete my dissertation.

Thanks are also due to many friends in the Department of Informatics and Mathematical Science at Osaka University who gave me many useful comments.

Contents

Chapter 1

Introduction

1.1 Background

Assuring software quality and productivity is a major problem facing software engineering. In general, software quality affects productivity, since software products with higher quality require less rework and maintenance. To develop and maintain the software products, developers may use various activities, methods, practices and transformations. A software process is defined as a set of such activities, methods, practices and transformations that people use to develop and maintain software and the associated products, e.g., project plans, design documents, code, test cases, and user manuals [17]. Software process improvement is a systematic methodology for improving software processes, and it includes understanding, definition, measurement, and optimization of processes [46]. Process improvement is indispensable for improving the quality of software products and the productivity of the development process.

For the improvement of software development processes, a number of studies and reports have been proposed [3, 6, 7, 13, 20, 28], and they all stress the importance of the following two key activities: (1) to understand and analyze the current status of the software development process, and (2) to construct and execute an improvement plan for the process, based on the results of the analyses. In order

to promote process improvement activities, Humphrey proposed the process maturity model [24]. Based on the process maturity model, Paulk *et al.* proposed the Capability Maturity Model (CMM) [57]. The CMM has been widely accepted and applied to actual software development organizations.

A certain company (hereafter, called Company *A*) was influenced by the process maturity model and established a Software Engineering Process Group (SEPG) in 1992 to promote process improvement. Since its establishment, the SEPG has collected data on software metrics from software projects in the company. We have been participating in the process improvement activities of the SEPG.

An actual example of software process improvement activity in Company *A* is described in [60]. In order to improve the software process, a process improvement procedure, which describes a current process using Petri nets and estimates the benefits gained by the improvement, was proposed. In the procedure, the current software process is described accurately and in detail, and then a feasible action plan is presented to developers. Also, benefits were estimated to evaluate the impacts of the action plan before the action plan is actually implemented. In [60], the proposed procedure was applied to an actual project. By applying the action plan to other projects, it was estimated that 10% of the total effort/KLOC would be reduced in the test phases. The estimated effort reduction was actually attained.

Their experience with the above study encouraged the SEPG, and then our research objective shifted to applying process improvement systematically in Company *A* focusing on quality, cost and delivery date. However, even if activities for process improvement were devised by the SEPG, they were seldom applied to actual projects, since those activities required much resources.

This kind of problem is common to other software organizations, and a cost-effective approach to software process improvement is strongly required. Simulation techniques have been introduced to answer such a requirement, and software process simulation is known to be one of the most cost-effective solutions to facil-

itate such process improvement activities.

Numerous studies on the simulation of software processes have been proposed [4,5,10,12,14,15,21,29,30,34,35,48–53,56,62–64]. However, those methods were not powerful enough to apply to the software process in Company *A*, because we intended to estimate the cost, quality, and duration of software development as well as showing the effectiveness of the process improvement. Our goal was to establish a new software simulation technique that can estimate cost, quality, and duration using highly accurate simulation. We also aimed at investigating its applicability to actual software process improvement in Company *A*.

1.2 Software Project Simulation

As the importance of software process improvement increases, the improvement activity costs become a more critical problem. The process improvement activities (usually including changes to the software process) require a large amount of resources to implement. However, its impact is difficult to predict. Thus, the following question arises: "How can we predict the future impact of theoretically useful methods for process improvement?"

One area of research to answer that question is the software process simulation technique. Using software process simulation, quite feasible answers for some complex situations can be obtained at a reasonable cost. Therefore the software process simulation technique is drawing interest from academic researchers and practitioners alike as an approach for analyzing complex business and policy questions.

Several methods have already been proposed to model and evaluate the software development process. Kellner has proposed the method of evaluating software processes that are described by the modeling tool STATEMATE [29]. This method has demonstrated how process models could be applied to software project management. Next, Lee and Murata have proposed a β-distributed

stochastic Petri net model for software project management [34]. This model is an integrated model using the program evaluation and review technique (PERT) and Petri nets, and is suitable to deal with uncertainty and concurrency problems in large software project management. FUNSOFT Nets [12] and SPADE [4,5] are also models based on Petri nets and are more oriented to process enactment than to process analysis and simulation. They are mainly focused on the evaluation of the time constraints of the process.

Tvedt and Collofello have also evaluated the effectiveness of process improvement on software inspections using the system dynamics model [62]. This method makes it possible to predict the impact of process improvement through the cause-effect relationships of software development. These methods evaluate only the cost and delivery date. They thus cannot evaluate all the important factors of software project: quality, cost and delivery date. Raffo has extended Kellner's method in order to evaluate the quality of software, and applied it to Kellner's Software Process example [51]. This method, however, has not been applied to real software development processes.

Recently, software process simulation is increasingly used to address a variety of issues from the strategic management of software development, to supporting process improvement [30]. For instance, in order to achieve higher levels of CMM, Raffo *et al.* introduced a framework for using process simulation to make decisions about process changes [52]. Their approach is intended to be a key to successful achievement of CMM levels 4 and 5. Christie also showed how simulation can be of significant benefit at all levels of the CMM [10]. Drappa *et al.* proposed the *SESAM* model, which is expressed in a rule-based modeling language, and applied the SESAM model to the training of software project management [14,15]. They also reported that the SESAM model helped the students in their department to easily understand the problems in the software development process. Pfahl *et al.* used a system dynamics-based simulator to refine requirements in a software development [48,49].

1.3 Main Results

The situation in Company *A* was almost the same, that is, the impact of process improvement activity had to be analyzed in advance. Since then, we have started research to simulate the software process. Although many simulation techniques had been proposed, they were not applicable to Company *A*. Thus the main objective is to develop a software project simulator that is applicable to software process improvement activities at Company *A*. The main results to be described in this dissertation are as follows:

1. Development of a software project simulator
2. Improvement of parallel execution of activities
3. Improvement of test process planning
4. Extension of simulator to risk prediction

1.3.1 Development of a software project simulator

First, we propose a new model based on Generalized Stochastic Petri Nets(GSPN) [36] for software projects. The model consists of a Project model and a Process model. The Project model focuses on three key components: activity, product and developer of the project. The Process model includes a set of Activity models, each of which specifies design, coding, review, test, and debug activities respectively using GSPNs. The model can handle the effect of human factors by introducing the concept of "workload". The workload of an activity is defined as the total time needed for a developer with the average capability to complete the activity. The workload can reflect the necessity of communication and the performance of CASE tools, and thus the simulator can evaluate the dynamic aspects of the software project.

Next, in order to support description and execution of the processes described by the Activity model, we develop a software project simulator. The kernel part

of the simulator is developed using the C++ language and the display and the editing functions are developed using the Tcl/Tk. As a result of the simulation, we get estimated values for the quality, cost and completion date of the target process. We then conduct an empirical evaluation. In the experiment, we apply the simulator to real software projects at Company *A* and compare the estimated values with actual data. The experimental results show that the estimated values are quite close to the actual value. As the result, we can show the applicability of the proposed simulator to improve real software projects in the future.

1.3.2 Improvement of parallel execution of activities

The parallel execution of activities is a common problem in software projects, especially in the waterfall-model-based development [58]. An example of parallel execution would be when a coding activity starts before previous module design activity finishes. This technique is usually introduced to reduce development duration when a deadline of a project approaches. However, this technique is not recommended in waterfall-model-based development, since it may increase the development effort. It is important to make the developers recognize the potential problems.

We describe a real software development process with parallel execution of both design and coding activities, and evaluate the process from the viewpoints of quality, cost, and completion date. Furthermore, by using the proposed method, we can analyze how parallel execution of design and coding activities affect the overall development process.

As a result, we observe that parallel execution increases both development effort and duration in experimental simulations. The results were convincing to actual developers in Company *A*.

1.3.3 Improvement of test process planning

The test phase in a development process encounters the following problems or difficulties with respect to controlling the progress of the test phase [22,44].

- Determining the amount of effort needed for the test phase
- Estimating the quality of the delivered code

These problems are tightly coupled. If the effort is too small, the quality will decrease. Even if the effort is more than necessary, the quality may not be improved very much.

Based on these observations, we propose a new method for controlling the progress of software development projects. The key idea is to update or modify the initial plan at intermediate stages in the development project, and to apply the updated plan to the succeeding stages. We then propose to control the progress of the design and coding phases using the initial plan, and then control the progress of the test and debug phases using the updated plan. We call this method two-phase project control.

The updating of the plan should reflect the results of the development project using the initial plan. For instance, data to be considered should include the number of faults introduced into the product and the number of residual faults from the design and coding phases.

We confirm the usefulness of two-phase project control using the proposed simulator. Additionally, in our evaluation, we apply real data to the simulator, collected from actual software development projects in Company *A*.

1.3.4 Extension to risk prediction

Thirdly, we try to estimate the development costs for risky projects [65], which is an important measure to determine risky projects. To do so, the project simulator must have capability to deal with risk factors in a risk questionnaire [41] and to estimate development costs under risk factors.

The initial simulator cannot simulate a project under risks, because it cannot represent risk factors. In an extended simulator, we implement a mechanism that adjusts parameters to deal with the influence of risk factors. As a result, the following typical causes of disorder in risky projects can be represented in the simulator: disorder caused by fluctuation in developer's skill levels and disorder caused by deadline pressure [19].

Finally, we perform a case study to confirm whether a risky project can be simulated. The results show that the enhanced simulator can estimate the development costs for both ideal case and risky case. As a result, we confirm that the simulator shows how much the development cost of a risky project exceeds an estimate.

1.4 Overview of the Dissertation

The rest of this dissertation is organized as follows: Chapter 2 addresses the process improvement activities and shows past activities in Company *A*.

Chapter 3 proposes the new model for project simulation based on the GSPN model, and shows examples to describe typical activities in a development process. We then implement a project simulator based on the proposed model, and the implemented simulator is applied to several actual project data.

Chapter 4 describes the first application to the process improvement. In this chapter, we show that the parallel execution of development activities (such as executing module design and coding activities simultaneously) increases project costs and residual faults.

Chapter 5 shows a second process improvement example. In this application, we investigate the effect of dynamic updating of a project plan by several simulations.

Chapter 6 presents an application of the simulator to risk prediction. We adopt some extensions to the simulator to manage risk factors in software projects.

Finally, Chapter 7 concludes this dissertation with a summary and directions for future work.

Chapter 2

Software Process Improvement Activity

2.1 Background on Software Process Improvement

A software process can be defined as a set of activities, methods, practices and transformations that developers use to develop and maintain software and the associated products [17]. Viewing software development as a process has significantly helped identify the problems that need to be addressed in order to establish effective practices. Because software processes are complex entities, researchers have created many languages and modeling formalisms (such as SPADE [5], FUNSOFT [12], and so on [29,34].).

As in other human-related activities, software processes can exhibit unexpected or undesired behaviors and performance. So practitioners have realized that those processes cannot be defined and frozen definitively. Software processes need to be continuously changed and refined to enhance their ability to handle various requirements from customers.

Quality models and methods for software process improvement thus have been proposed. The Capability Maturity Model (CMM) [57] and ISO 9001 standard [26] have been proposed as quality models. In the Software Engineering

Institute(SEI) at Carnegie Melon University, Humphrey has proposed a framework of software process maturity [24]. After several years of experience with the software process maturity framework, the SEI evolved Humphrey's framework into the CMM. Both the CMM and ISO 9001 standard have been adopted by many organizations to assess their software capability and to provide a pathway for their efforts to improve their processes. As for process improvement methods, SPICE [27], IDEAL [38], Personal Software Process (PSP) [25], and so on have been proposed. They suggest the steps to be accomplished to improve the quality of the software development process.

In the recent years, research in empirical software engineering has also increased. There have been numerous empirical studies and reports [3,6,7,13,20,28] regarding the improvement of software processes, and they have provided many useful insights for practitioners and academic people.

2.2 Process Improvement Activities in Company *A*

2.2.1 Brief introduction of Company *A*

Company *A* is a software development company, whose main product is software embedded in social infrastructure systems (such as ticket vending machines, automated teller machines(ATMs), and point of sales (POS) terminals). Just as such embedded software has grown in size, the development projects have also become large. Since the embedded software is tightly coupled with the functionality of the hardware systems and must meet their release schedules, the quality and timeliness of software availability is critical. Therefore, the need for efficient software project management has increased.

The SEPG (Software Engineering Process Group) is one of groups within the Development Division in Company *A*. The SEPG was established in 1992 to improve the software development process of the development departments, each of which belongs to a different division. The SEPG has been cooperating

with several universities to facilitate this process improvement. They defined their development process, and they have performed exhaustive collection of software metrics [16,43].

The SEPG has applied the following actual process improvement activities to the processes in Company *A*.

2.2.2 Identification of software process and process improvement

Actual experiences of software process improvement at Company *A* were reported in [18,55,59,60]. For effective technology transfer, the SEPG had set three principal goals: (1) motivate developers to improve on their process, (2) describe and define the current software development process correctly and in detail, (3) present a feasible action plan for developers to follow. To attain these goals, the SEPG proposed a process improvement procedure by describing the current process and estimating the benefits gained by an improvement.

The project to be improved was one of a series of embedded software development projects. "Series" implies that there exist multiple projects where similar products are developed in succession. Conventionally, these projects continue for at least three years. When the process improvement started, two projects had already been finished. The next project would start in four months.

First, the SEPG held a series of interviews with developers and drew a flow map in the form of a Petri net [47] describing the current software development process. Next, the group constructed an action plan based on an in-depth analysis of the current flow map, and estimated the benefits to be obtained if this plan were to be rigorously followed. As a result, both the action plan and the benefit estimation were accepted by the developers as a feasible action plan. Applying the action plan to a practical project confirmed that, compared to similar projects, approximately 10% of the total effort/KLOC was reduced in the test phases. We then could suggest that the principal goals and the proposed procedure are effective in reducing the development effort at Company *A*.

2.2.3 Improvement of the planning process

In [39,42], we analyzed the association between a deviation in the actual cost (measured by person-month) from the estimated cost and the quality and productivity of software development projects. Although the obtained results themselves may not be new from the academic point of view, they can motivate developers to join process improvement activities in a software company and thus become a driving force for promoting process improvement.

To be precise, we showed that if a project is performed faithfully under a well organized project plan (that is, the plan is first constructed according to the standards of good writing and then a project is managed and controlled to comply with the plan), then the deviation in actual cost from estimated cost is small. Next, we show statistically that projects with a small deviation in the cost estimate tend to achieve high quality in the final products and high productivity for the development teams. This analysis makes extensive use of actual project data for 37 projects at Company *A*.

2.2.4 Risk investigation and prediction

We have tried to predict the final status of software development projects by statistical methods [41]. The proposed method was based on a questionnaire for the risk factors in software development.

First, we defined a "risky project" from the viewpoint of development cost and duration. We then designed a questionnaire that includes the risk factors that occur in software development. We designed the questionnaire shown in Figure 2.1 to be distributed to software project managers. The questionnaire includes 23 risk factors, which are classified into 5 major categories: Requirements, Estimation, Team organization, Project plan, an Project management. Project managers then returned evaluations on each risk factor as shown in Figure 2.1. The evaluations are of an ordinal evaluation (For example, for evaluations 'High', 'Relatively high', 'Relatively low', and 'Low', the values 3, 2, 1, and 0 are assigned,

	Items	Evaluation
1.	**Requirements**	
1.1	Unreasonable customers.	
1.2	Developers could not elucidate sufficient requirements.	
1.3	Developers misunderstood the requirements of the customer.	
1.4	Lack of interactive agreement regarding requirement specifications between the customer and the developer.	
2.	**Estimations**	
2.1	There were missing items to be estimated; these items were included in the implicit requirements.	
2.2	The importance of estimations was not well recognized.	
2.3	Non-technical pressure rendered estimates of costs and/or schedules unrealistic.	
2.4	Over-optimism in estimating technical issues.	
2.5	Insufficient estimations were carried out using the results of successful projects in the past.	
3.	**Team Organization**	
3.1	Wrong people available (lack of skills, lack of training, lack of expertise).	
3.2	Incorrect staffing (too few people for current task).	
4.	**Planning Capability**	
4.1	Unclear responsibilities and authorities.	
4.2	Inadequate specifications regarding the work product.	
4.3	Inadequate or excessive planning or scheduling of the review process.	
4.4	Lack of commitment on the part of all of the developers with regard to the project plan.	
4.5	Lack of review for the project plan by senior managers.	
4.6	Inadequate control of the development process.	
5.	**Project Management Activities**	
5.1	Lack of risk management on technical matters.	
5.2	Low morale on the part of the developers.	
5.3	Lack of perception on the part of the managers to ensure a concerned effort.	
5.4	Requirement or specification changes were not managed sufficiently.	
5.5	Lack of progress reporting.	
5.6	Lack of data needed to keep track of a project.	

Figure 2.1: Risk investigation questionnaire

respectively).

Based on the responses to the questionnaire, we collected risk assessment data and applied the following logistic model to them:

$$P(Y|x_1, \cdots, x_n) = \frac{e^{b_0+b_1x_1+\cdots+b_nx_n}}{1+e^{b_0+b_1x_1+\cdots+b_nx_n}}$$

where $x_1, \cdots, x_n$ are explanatory variables in the model, and Y is a binary dependent variable that represents whether a project is risky or not. P is the conditional probability that $Y = 1$(i.e. a project is risky) when the values of $x_1, \cdots, x_n$ are determined. We selected the risk factors in the questionnaire as potential explanatory variables, and estimated the coefficients b_i's using the risk assessment data obtained from the responses to the questionnaire. The following is a logistic model constructed for the risk data at Company A [41]:

$$P(Y|x_1, x_2) = \frac{e^{-5.251+2.727x_1+3.984x_2}}{1+e^{-5.251+2.727x_1+3.984x_2}}$$

where x_1 and x_2 are the risk factors "Estimation" and "Planning Capability" in Table 2.1, respectively*.

*In [41], the risk factors were chosen from the categories in Figure 2.1.

We carried out an effectiveness analysis of the model. The results showed that the model can nicely predict risky projects for new data sets with the probability $P(Y|x_1, x_2)$.

Chapter 3

Development of Software Project Simulator

3.1 Proposed Model

3.1.1 Overview

It is necessary to evaluate software processes from the viewpoints of quality, cost, and duration. At first, following the policy in [60] (A brief introduction of [60] was explained in subsection 2.2.2.), we decided to develop the model based on Petri nets. Among many kinds of Petri net models, we selected the Generalized Stochastic Petri Net(GSPN) [36].

Fundamental activities in software development processes in [60] can be described by introducing the probability of the injection and removal of a fault as the firing rate of transitions, in order to estimate the number of residual faults in the products.

The concept of "workload" was then introduced in order to describe the fluctuations in development duration and product size. The interpretation of workload will be given in the next subsection.

There also exist several dynamic factors as follows, which affect the behavior

of the developers in software development:

1. Communication overheads [9].
2. Difference in experience [37,54].
3. Confusion caused by incompleteness in documentation [11].
4. Stress due to tight schedules [19].

Since these factors change the development project duration dynamically in an actual project, it is very difficult to estimate the duration precisely. So three attributes — developers' experience level, product completion rate, and deadline for activities — are incorporated to handle the dynamic effects of human factors. In particular, the "completion rate" of products makes it possible for developers to start concurrently a successive (and thus the next) activity based on incomplete documents developed by the current activity. The degree of incompleteness is controlled by the completion rate. All these attributes will be taken into the Project model later.

3.1.2 Key concept "Workload"

Generally speaking, effort is used to measure the amount of an activity. But the effort does not become clear until the activity is over, and thus the effort is not suitable to determine the amount of not-finished activity. Also, the effort includes not only the amount of work needed purely for execution of the activity, but also the amount of communication among the developers. For example, let us consider an activity of 10 person-days. Even if a single developer could perform this activity in 10 days, 10 developers could not perform it in a single day. One of the main reasons is that the time needed to communicate among the developers increases as the number of the developers increases.

Here, we define the term "workload" for an activity as the total time needed to complete the activity for a developer who has an average capability. A con-

cept similar to workload has been presented in [1]. Our "workload" could be considered as an actual instance of the concept in [1]. Furthermore, the efficiency of the activity under such a condition is quantified as 1. The value of efficiency depends on the environment, such as the number of developers, the necessity of communication and the performance of CASE tools. The development time is calculated as the result of dividing the workload by the efficiency of the activity.

[Example 1] Consider the following two cases of an activity whose workload is 20 hours.

Case 1: Two developers with standard capability execute the activity and ten percent of the total development time is spent for communication.

For this case, if each developer takes part in the activity for 10 hours, then the resulting workload becomes 18 (=10×2×0.9).

Case 2: Four developers with standard capability execute the activity and twenty percent of the total development time is spent for communication.

For this case, if each developer takes part in the activity for 5 hours, then the resulting workload becomes 16 (=5×4×0.8).

If we get the workload of an activity, then we can estimate a development time appropriately for several specific activity conditions dependent on a given environment.

In the proposed model, the workload is assigned to each activity depending on the input products for the activity. That is, for example, the workload for the design activity (W_{design}) is defined in the following formula:

$$W_{design} = s_{design} \times K_{design}.$$

Here, s_{design} denotes the size of the input product for the design activity and K_{design} denotes the workload parameter for the design activity. Before simulation, a workload parameter must be assigned to each activity in the target project.

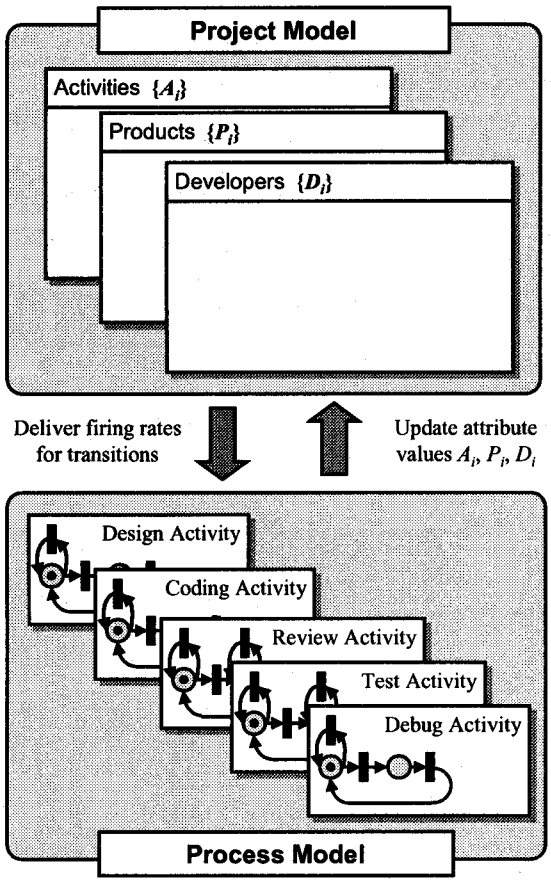


Figure 3.1: Structure of the proposed model

Consumption of the workload assigned to an activity is related to the progress of the activity in the development. Growth in the product can be modeled by changing the size values or the number of faults in the output product.

3.1.3 Structure of the new model

The proposed model consists of a Project model and a Process model. Figure 3.1 shows the structure of the proposed model.

The Project model includes three key components: activities, products and developers. Some attributes are attached to each of them, as shown in Figures 3.2.

The Process model includes a set of Activity models, which include activities

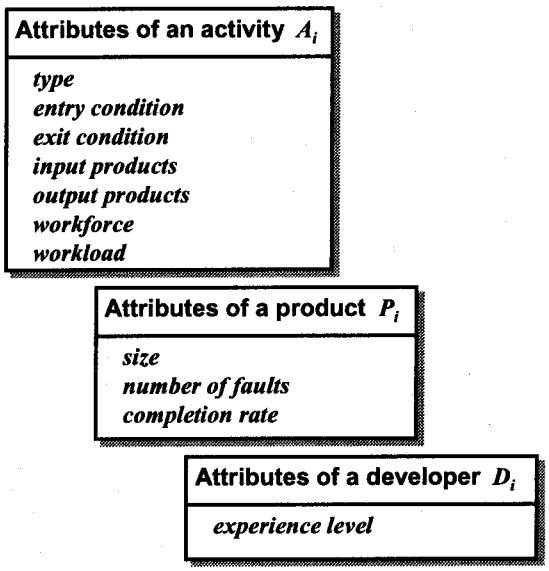


Figure 3.2: Project template

like designing specifications, coding, reviewing, testing, debugging activities.

3.1.4 Project model

The Project model focuses on three key components: activities, products and developers, and attaches several attributes to each of them (See Figure 3.2).

An activity has eight kinds of attributes: *type*, *entry/exit conditions*, *input/output products*, *workforce*, *deadline*, and *workload*. (1) The *type* shows which of the activities it corresponds to and currently refers to either of design, coding, review, test, or debug. (2) The *entry condition* and (3) the *exit condition* specify conditions for beginning and ending the activity, respectively. (4) The *input products* describe the products given to the activity as the input products and the degree of contribution of each input products to workload of the activity. (5) The *output products* describe the output products that are developed in the activity and the weight assigned to each product. Variation in the product size and the number of faults are distributed to the products according to weights. The sum of each of those weights must be one. (6) The *workforce* specifies tuples of the developers who

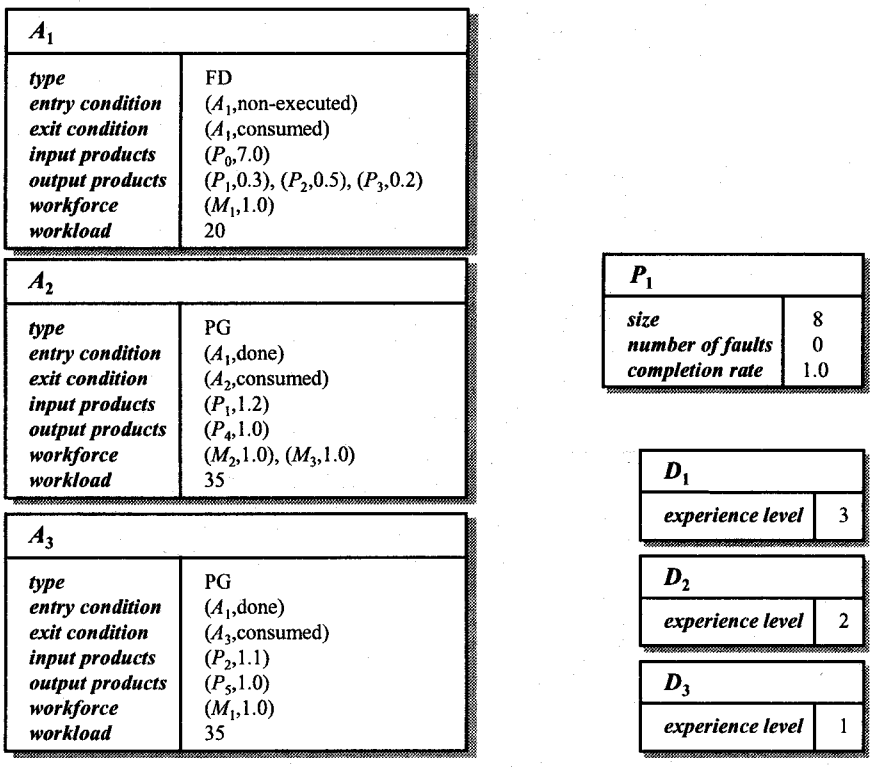
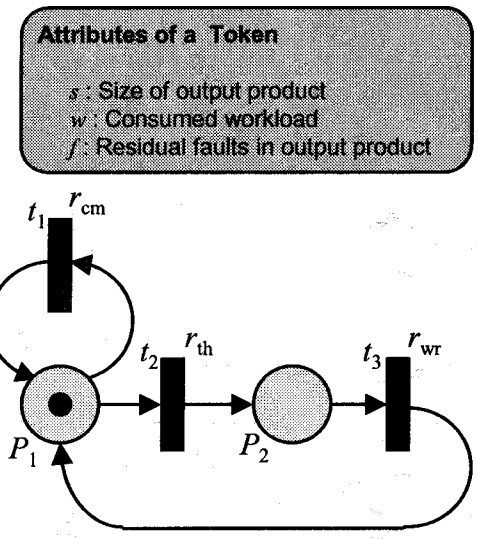


Figure 3.3: Example of project description

engage in the activity and the ratio of time in which each developer can engage in the activity with respect to his or her business hours. (7) The *deadline* represents the date appointed for completion of the activity, which is set in the development plan. (8) The *workload* represents a tuple of the workload assigned to the activity and the amount completed.

A product has three kinds of attributes, which are size, the number of faults and completion rate. (1) *Size* represents the product size in document pages or the lines of source code. (2) *Number of faults* counts faults in the product. (3) *Completion rate* represents the ratio of the completed workload to the assigned workload.

A developer has an attributes *experience level* that is determined according to his/her length of service. We classify developers' experience levels into the following three levels: novice, standard and expert levels. They are quantified as



Transition	t_1	t_2	t_3
Execution function	-	$w=w+1$	$s=s+1$ $f=f+1$ (p_{in})

Figure 3.4: Activity model

discrete values 1, 2 and 3, respectively.

[Example 2] Figure 3.3 shows an example of the Project model description. This project is composed of three activities (A_1, A_2, and A_3), six products and three developers. Now, let us explain the description of activity A_1. *Type* shows that the activity A_1 is a design activity. *Entry condition* (the condition for starting the activity) shows that A_1 can be started at any time, provided that it has not already been started. *Exit condition* (the condition for ending the activity) shows that if all of the workload assigned to A_1 is completed, its execution ends. *Input product* (= $(P_0, 7.0)$) represents that the product P_0 is given to A_1 as an input product, and the workload, which is equivalent to seven times as much as the size of the product P_0, is assigned to A_1. *Output product* (= $(P_1, 0.3), (P_2, 0.5), (P_3, 0.2)$) shows that A_1 develops three products P_1, P_2 and P_3, and the increase or decrease of the size and faults is distributed to P_1, P_2, and P_3 in a three-five-two ratio, respectively.

Workforce shows that developer M_1 engages in A_1 at the full rate of his/her business hours. *Deadline*(the appointed date for the completion of the activity) shows that the deadline of A_1 is specified to be 20 days after the beginning of the project.

The description of product P_0 shows that the size of P_0 is 8 pages, no fault exists in it, and development of P_0 is fully completed, that is, there are no omission in the description.

The descriptions of developers M_1, M_2, and M_3 show that their experience levels are 3(expert), 2(standard), and 1(novice), respectively.

Note that *workloads* of activities $A_1, \cdots, A_3$ and the attributes of all products except for the initial input product P_0 are determined during the execution of the model. Thus, they are not yet specified in Figure 3.3.

3.1.5 Activity model

An activity model is prepared for each type of activity such as design, coding, review, test, and debug. The descriptions of the Activity models are given using an extended GSPN. In this study, we only used the notation of GSPN, and thus we did not use the GSPN's computational power very much. Figure 3.4 shows an example of the description of the design activity. In the extended GSPN, a token has three attributes: the product size s, the number of faults f, and the consumed workload w. These attributes are used to represent the current status of development, which varies over the execution of each Activity model. This extension of GSPN is mainly for simplicity of description.

Transitions used here are timed transitions. The firing delay of each transition is exponentially distributed and the average firing delay of a transition is specified by a firing rate assigned to it. In Figure 3.4, the firing rate r_{cm} of transition t_1 means that the average firing delay of transition t_1 is $1/r_{cm}$.

In addition, each transition has a function (called the execution function) to be evaluated on its firing. Executing the function updates the attribute values of the token. Intuitively speaking, each transition corresponds to a developer behavior

such as thinking, writing, or communicating, or an event that occurs during execution of the activity. Places correspond to waiting states for occurrences of behaviors or events.

[Example 3] Figure 3.4 shows a description of the design activity. Here, we consider three kinds of developer behaviors in the design activity: communicating among developers, thinking of a problem solution, and writing down the solution in documents. Transitions t_1, t_2, and t_3 in Figure 3.4 correspond to communicating, thinking and writing and are given the firing rates r_{cm}, r_{th}, and r_{wr}, respectively.

The firing rates of the transitions are formulated by the following ten functions f_{cm}, f_{th}, f_{wr}, f_{pr}, f_{rd}, f_{dt}, f_{md}, f_{ps}, f_{lc}, and f_{in}. These functions should be actually specified based on the properties of the target project.

In the following, M is the number of the developers who engage in the activity, L is developer's experience level, ΣL is the sum of each developer's experience, S is the total size of the input products, R is the completion rate of the input products, F is the number of faults in the input products, D is the number of the days from the current date to the deadline of the activity. K_{cm}, K_{th}, K_{wr}, and K_{in} are parameters given to each activity and specify the communicating, thinking, writing and fault injection rate, respectively*.

(1) Communicating rate r_{cm}

$$r_{cm} = f_{cm}(M, \Sigma L, R)$$

(2) Thinking rate r_{th}

$$r_{th} = f_{th}(M, \Sigma L)$$

(3) Writing rate r_{wr}

$$r_{wr} = f_{wr}(M, \Sigma L)$$

*The parameters K_{cm}, K_{th}, K_{wr}, and K_{in} should be given actual values based on the characteristics of the project to which they are applied.

(4) Preparing rate r_{pr}

$$r_{pr} = f_{pr}(M, \Sigma L, S)$$

(5) Reading rate r_{rd}

$$r_{rd} = f_{rd}(M, \Sigma L)$$

(6) Fault detecting rate r_{dt}

$$r_{dt} = f_{dt}(M, \Sigma L, S, F)$$

(7) Fault modifying rate r_{md}

$$r_{md} = f_{md}(M, \Sigma L)$$

(8) Test case passing rate r_{ps}

$$r_{ps} = f_{ps}(M)$$

(9) Fault localizing rate r_{lc}

$$r_{lc} = f_{lc}(M, \Sigma L, S, F)$$

These parameters make it possible to dynamically determine the frequency of communication or the difficulty in thinking and writing depending on the number of developers, experience levels of developers and/or completion rates of input products.

In addition, the increase in product size s at every firing of writing transition t_3 and the consumption of workload at every firing of thinking transition t_2 are described by the corresponding execution functions. At each firing of the transition, the values of the token's attributes can be changed by evaluating its execution function.

The activity model handles fault injections in the design activity as stochastic events whose occurrences depend on the fault injection rate p_{in}. In general, p_{in} is formulated by the following function:

(10) Fault injection rate p_{in}

$$p_{in} = f_{in}(M, \Sigma L, R)$$

This function enables handling of a dynamic effect on the fault injection rate caused by completion rate of input products and by developers' experience levels.

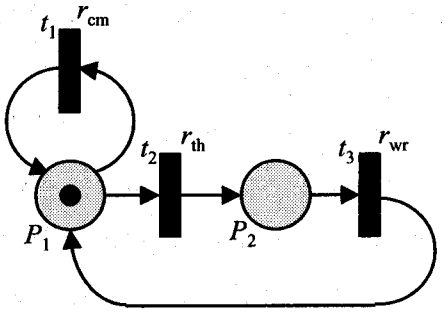
Though the functions (1)–(3) are used in Figure 3.4, the rest (4)–(9) are not included in Figure 3.4. Figure 3.4 shows an example of design and coding activities and any further improvement in the model should be conducted through case studies. But we also modeled other activities(review, test, and debug), in which the functions (4)–(9) are used.

[Example 4] In the design Activity model depicted in Figure 4, for example, transitions t_1 and t_2 which represent communicating and thinking behavior, respectively, can fire when a token exists in the place P_1. If the communicating transition t_1 fires, it has no effect on the attributes values, and the token returns to the place P_1 and only time elapses by the firing delay. On the other hand, if the transition t_2 fires by evaluating its execution function, then the consumed workload w is increased by one, and the token moves to the place P_2. When the token exists in the place P_2, only the transition t_3 which represents writing behavior is enabled. If the transition t_3 fires, then product size s is increased by one, and the number of faults f are increased according to the fault injection rate p_{in}. After the firing of t_3, the token moves back to the place P_1.

3.2 Examples of modeling

3.2.1 Design and coding activities

Figure 3.5 shows a description of the design activity. Here, we consider three kinds of developers' behaviors in the design activity: communicating among developers, thinking of problem solutions, and writing for putting down the solution in documents. Transitions t_1, t_2, and t_3 in Figure 3.5 correspond to communicating, thinking and writing and are given the firing rates r_{cm}, r_{th}, and r_{wr}, respectively. The firing rates of the transitions are formulated by the following



Transition	t_1	t_2	t_3
Execution function	-	$w=w+1$	$s=s+1$ $f=f+1$ (p_{in})

Figure 3.5: Design/coding activity model

functions:

(1) Communicating rate r_{cm}

r_{cm} = f_{cm}(the number of developers,
experience levels of developers,
completion rates of input products)

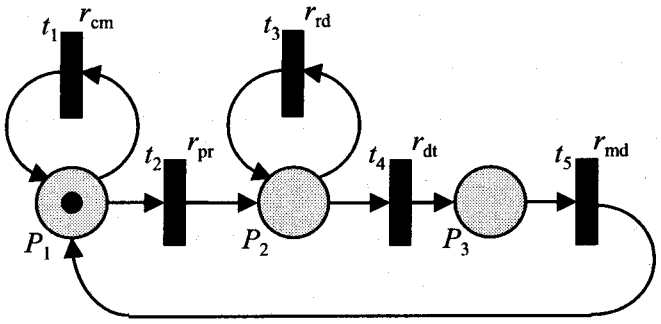
(2) Thinking rate r_{th}

r_{th} = f_{th}(the number of developers,
experience levels of developers)

(3) Writing rate r_{wr}

r_{wr} = f_{wr}(the number of developers,
experience levels of developers)

These functions make it possible to dynamically determine the frequency of communications or the difficulty in thinking and writing according to the number of developers, experience levels of developers and/or completion rates of input products.



Transition	t_1	t_2	t_3	t_4	t_5
Execution function	-	-	w=w+1	-	f=f-1 update(r_{dt})

Figure 3.6: Review activity model

Increases in product size s at every firing of writing transition t_3 and consumption of workload at every firing of thinking transition t_2 are described by the corresponding execution functions. At each firing of the transition, the values of token's attributes can be changed by evaluating its execution function.

The Activity model handles fault injections in the design activity as stochastic events whose occurrences depend on the fault injection rate p_{in}. In general, p_{in} is formulated by the following function:

(4) Fault injection rate p_{in}

$p_{in} = f_{in}$(the number of developers,
experience levels of developers,
completion rates of input products)

This function enables the handling of dynamic effects on the fault injection rate caused by incomplete input product and developers' experience levels.

3.2.2 Review activity

Figure 3.6 shows a description of the review activity. The review activity includes five kinds of developer behavior: communicating among developers, preparing

for the review, reading the documents to check them, detecting faults in the documents, and modifying the detected fault. Transitions t_1, t_2, t_3, t_4, and t_5 in Figure 3.6 correspond to communicating, preparing, reading, detecting and modifying are given the firing rates r_{cm}, r_{pr}, r_{rd}, r_{dt}, and r_{md}, respectively. The firing rates of the transitions are formulated by the following functions:

(1) Communicating rate r_{cm}

$r_{cm} = f_{cm}$(the number of developers,
experience levels of developers,
completion rates of input products)

(2) Preparing rate r_{pr}

$r_{pr} = f_{pr}$(the number of developers,
experience levels of developers,
size of input products)

(3) Reading rate r_{rd}

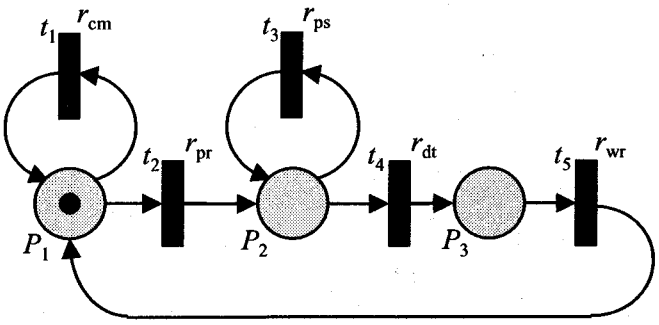
$r_{rd} = f_{rd}$(the number of developers,
experience levels of developers)

(4) Detecting rate r_{dt}

$r_{dt} = f_{dt}$(the number of developers,
experience levels of developers,
size and the number of faults of
input products)

(5) Modifying rate r_{md}

$r_{md} = f_{md}$(the number of developers,
experience levels of developers)



Transition	t_1	t_2	t_3	t_4	t_5
Execution function	-	-	$w=w+1$	update(r_{dt})	$s=s+1$

Figure 3.7: Test activity model

3.2.3 Test activity

Figure 3.7 shows a description of the test activity. The test activity includes three kinds of developers' behaviors: communicating among developers, preparing for test, and writing down a detected failure in report documents, and two events during the test activity: passing the test case without error and detecting a failure during the test execution.

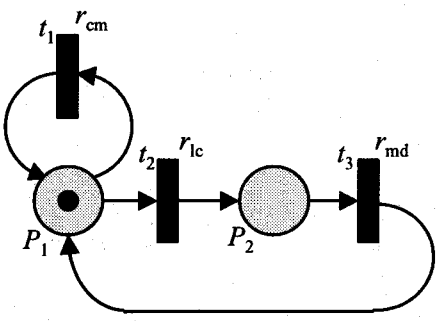
Transitions t_1, t_2, t_3, t_4, and t_5 in Figure 3.7 correspond to communicating, preparing, passing, detecting and writing are given the firing rates r_{cm}, r_{pr}, r_{ps}, r_{dt}, and r_{wr}, respectively. The firing rates of the transitions are formulated by the following functions:

(1) Communicating rate r_{cm}

$r_{cm} = f_{cm}$(the number of developers,
experience levels of developers,
completion rates of input products)

(2) Preparing rate r_{pr}

$r_{pr} = f_{pr}$(the number of developers,



Transition	t_1	t_2	t_3
Execution function	-	$w=w+1$ (update r_{lc})	$f=f-1$

Figure 3.8: Debug activity model

experience levels of developers,
size of input products)

(3) Passing rate r_{ps}

$r_{ps} = f_{ps}$(the number of developers)

(4) Detecting rate r_{dt}

$r_{dt} = f_{dt}$(the number of developers,
experience levels of developers,
size and the number of faults of
input products)

(5) Writing rate r_{wr}

$r_{wr} = f_{wr}$(the number of developers,
experience levels of developers)

3.2.4 Debug activity

Figure 3.8 shows a description of the debug activity. The debug activity includes three kinds of developers' behaviors: communicating among developers, localizing a fault associated with the reported failure and modifying the localized fault.

Transitions t_1, t_2, and t_3 in Figure 3.8 corresponding to communicating, localizing and modifying are given the firing rates r_{cm}, r_{lc}, and r_{md}, respectively. The firing rates of the transitions are formulated by the following functions:

(1) Communicating rate r_{cm}

$r_{cm} = f_{cm}$(the number of developers,
experience levels of developers,
completion rates of input products)

(2) Localizing rate r_{lc}

$r_{lc} = f_{lc}$(the number of developers,
experience levels of developers,
size and the number of faults of
input products)

(3) Modifying rate r_{md}

$r_{md} = f_{md}$(the number of developers,
experience levels of developers)

3.3 Simulation Environment

In order to quantitatively evaluate software processes described by the proposed model, a simulation environment that executes the process automatically is indispensable. We have designed and implemented a simulator that supports description of the target process, executes the processes described by the activity model, and analyzes the simulation results statistically. In this Section, we give

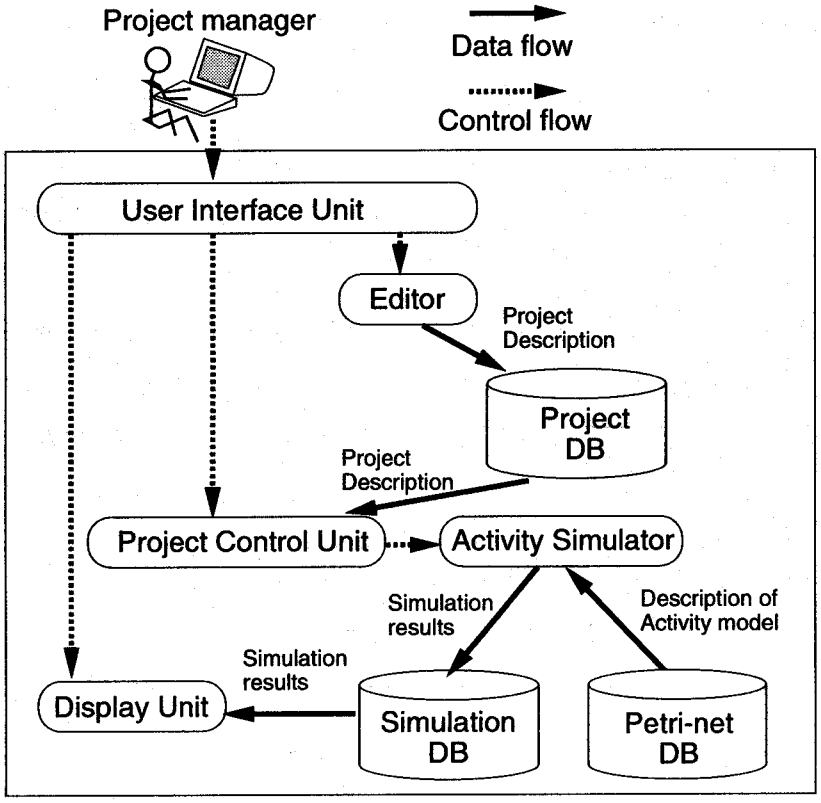


Figure 3.9: System architecture

an overview of the simulator. The experimental evaluation of the simulator will be described in Section 3.4.

3.3.1 System architecture

Figure 3.9 shows the system architecture of the simulator. The system consists of five functional units: project control unit, activity simulator, user interface unit, display unit and editor. In Figure 3.9, the solid lines represent data flows and the dotted lines represent control flows, respectively.

The function of each unit is as follows:

(1) Project control unit: The project control unit decides which Activity model is to be simulated by the activity simulator in accordance with the project

description. It defines the relationship among activities, products and developers of the Project model, and sets the values for the attributes of each activity, product and developer. Next, it delivers the name of the Activity model and the values of parameters to the activity simulator. When it receives the results of simulation, it updates the values of the attributes.

(2) Activity simulator: The activity simulator simulates activities specified by the Activity model using data such as the name of the Activity model and the parameters given from project control unit. First, it gets an Activity model with the same name from the Petri net database. Next, it simulates activities — specified by the model using the given parameter values. The results of simulation are returned to project control unit at regular intervals and are stored in the simulation database.

(3) User interface unit: The user interface unit manages exchanges of data or commands between the user and the system (editor, project control unit, and display unit).

(4) Display unit: The display unit displays the data received from the activity simulator. It can also provide data on the previous simulation results stored in the simulation database. The data include graphical information and statistical analysis of the simulation results.

(5) Editor: The editor supports the creation of the project description that is input to the project control unit. This editor enables us to describe a project and set up all parameters needed in the proposed model. The output of the editor is stored in the project database.

Since high-speed computation is necessary for the project control unit and activity simulator, we implemented them using C language. Conversely, the display unit and the editor, for which user-friendliness is strongly desired, are implemented using Tcl/Tk [45]. The program size is about 3500 lines (C language: 1000 lines, Tcl/Tk: 2500 lines).

3.3.2 Behavior of the simulator

Simulations proceeds at the intervals of unit time[†]. First, the project control unit determines the activities to be executed, based on the current status of the simulation and the *entry/exit conditions* of each activity. Next, for each executable activity, the project control unit delivers the parameters to the activity simulator and directs it to execute the activities for a day. The activity simulator then executes all of the activities specified by the project control unit, using the provided parameters and the extended GSPN. The execution of an activity is expressed by the consumption of its workload. When an activity consumes all of assigned workload, the activity is regarded as completed.

The execution of a simulation can be suspended or restarted at any time. Intermediate simulation results can be stored in the simulation database at every unit time in the simulation. The intermediate simulation results are stored in the same format of the original project description. It is thus possible to restart the simulation using the intermediate simulation results as input. It is also possible to change the values of parameters (attributes of project) at any time during the simulation. For example, we can modify the number of developers at any time during the simulation and simulate that change immediately.

Figure 3.10 shows an example of a simulation execution that is in progress. (This figure shows a situation near the end of the coding activities.) In the main window, we can see the progress of the simulation. We can also get various information from the window such as control flow between activities, consumed/assigned workload for each activity, developers assigned to an activity, the size of the product, the number of residual faults in the product, and so on. This information supports an investigation of the progress of simulation in various ways. In Figure 3.10, another window graphically shows the change in the number of residual faults.

[†]Currently, one unit time is a day (8 hours).

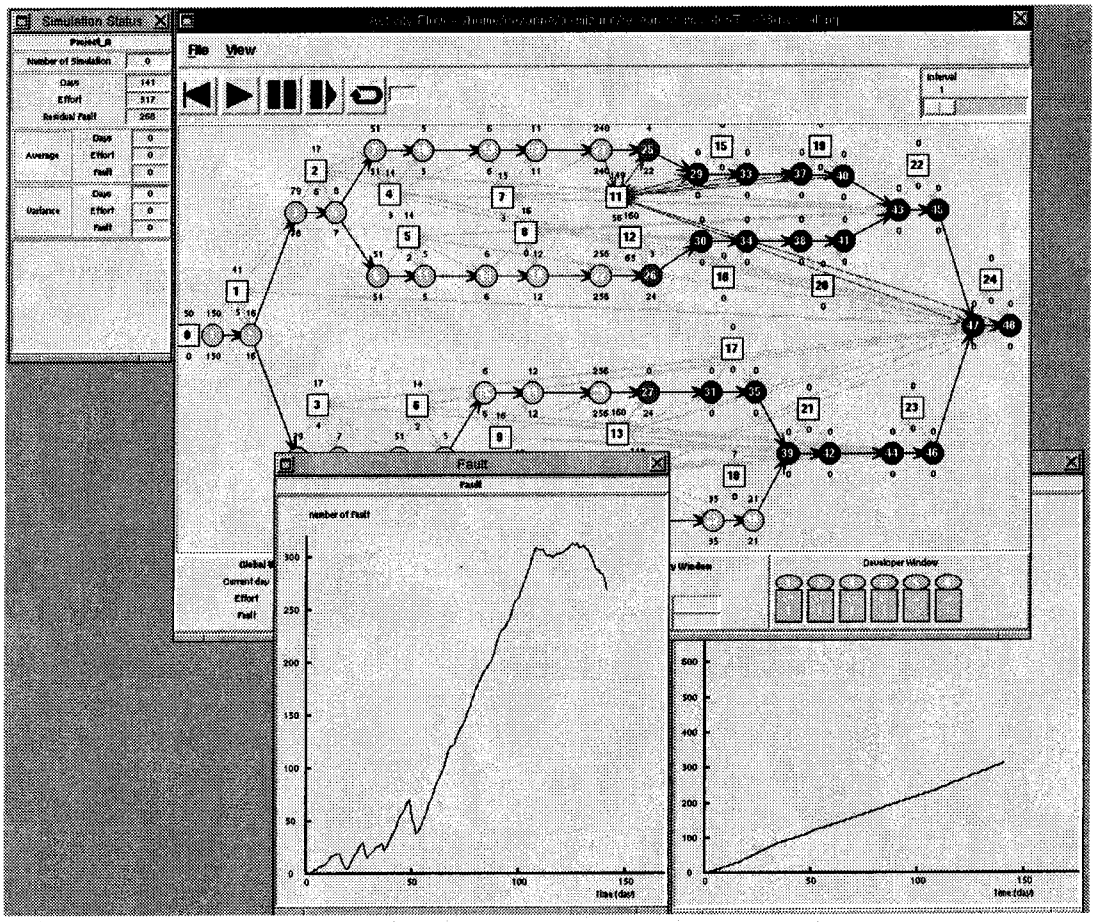


Figure 3.10: Example of simulation

3.4 Experimental Evaluation

In order to evaluate the usefulness of the proposed method, we conducted an experimental evaluation. In the experiment, we applied the simulator to three similar software development projects PR_1, PR_2, and PR_3 in Company A.

3.4.1 Assumptions

We formulated each firing rate and fault injection rate in the design and coding Activity models on the following assumptions (H1)–(H3). In the following formulas, M is the number of the developers engaged in the activity, ΣL is the sum

of each developer's experience level, and R is the completion rate of the input products. K_{cm}, K_{th}, K_{wr}, and K_{in} are parameters given to each activity and define the communicating, thinking, writing, and fault injection rate, respectively.

(H1) The communicating rate r_{cm} is proportional to the square of the number of developers and inversely proportional to the developers' experience levels and the completion rates of the input products.

$$r_{cm} = K_{cm} \times \frac{M^2}{\Sigma L \times R}$$

The validity of (H1) comes from the following: (1) The number of communication paths among M developers is $M(M-1)/2$, and a novice developer needs more communication because of his/her immature knowledge, and (2) incomplete input products induce frequent inquiries about the omissions in the description [11].

(H2) The thinking rate r_{th} and writing rate r_{wr} are proportional to the average developer's experience and the number of developers. This assumption is based on the assertion that the individual capability of the developer is strongly related to productivity in the software development process [37,54].

$$r_{th} = K_{th} \times \frac{\Sigma L}{M} \times M = K_{th} \times \Sigma L$$

$$r_{wr} = K_{wr} \times \frac{\Sigma L}{M} \times M = K_{wr} \times \Sigma L$$

(H3) The fault injection rate p_{in} is proportional to the number of developers and inversely proportional to the average experience level of the developers and the completion rate of the input products.

$$p_{in} = K_{in} \times \frac{M}{\Sigma L \times R} \times M$$

The validity of (H3) comes from the following: (1) The individual capabilities of developers are also related to the quality of software [37, 54], and

(2) incompleteness of products prevents developers from performing their work correctly.

In a similar way, each firing rate used in other activity models such as review, test and debug are formulated based on the following assumptions (H4)–(H7). In the following formulas, S is the total size of the input products, F is the number of faults in input products, and K_{pr}, K_{rd}, K_{md}, K_{ps}, K_{dt}, and K_{lc} are parameters given to each activity and define the preparing, reading, modifying, passing, fault detecting, and fault localizing rate, respectively.

(H4) The preparing rate r_{pr} used in the review and test activities is proportional to the average developer's experience, number of developers, and the total size of input products. (Preparation time for review or test is expected to increase according to the size of input products.)

$$r_{pr} = K_{pr} \times S \times \frac{L}{M} \times M = K_{pr} \times S \times L$$

(H5) The reading rate r_{rd} and the modifying rate r_{md} are proportional to the average developer's experience and the number of developers as well as the thinking and writing rates.

$$r_{rd} = K_{rd} \times \frac{L}{M} \times M = K_{rd} \times L$$

$$r_{md} = K_{md} \times \frac{L}{M} \times M = K_{md} \times L$$

(H6) The passing rate in the test activity is proportional to the multiple degree of the test execution, which is assumed to be the same as the number of developers who engage in the test activity. (In a real situation, the multiple degree of the test involves the number of hardware systems executing the programs.)

$$r_{ps} = K_{ps} \times M = K_{ps} \times M$$

(H7) The detecting rate r_{pr} in the review and test activity and the localizing rate r_{lc} in the debug activities are proportional to the average developer's experience, number of developers, and the fault density in the products that are objects in the activity. (The frequency of the detection of failure or faults and localization of faults increases in accordance with the number of faults per unit size of the product.)

$$r_{dt} = K_{dt} \times \frac{F}{S} \times \frac{L}{M} \times M = K_{dt} \times \frac{L \times F}{S}$$

$$r_{lc} = K_{lc} \times \frac{F}{S} \times \frac{L}{M} \times M = K_{lc} \times \frac{L \times F}{S}$$

3.4.2 Characteristics of the target projects

The main characteristics of the projects are summarized as follows:

(1) Development effort is 170–330 person-days per project.

(2) The size of the system is about 15KLOC.

(3) Project members are almost unchanged throughout all projects.

(4) Each project uses the standard waterfall model shown in Figure 3.11.

3.4.3 Outline of the experiment

In order to execute the process specified by the process description, it is necessary to determine the values of parameters for every activity based on the collected data of projects PR_1 and PR_2. Since we could not obtain all of the necessary data, we initiated the simulation of the development process using virtual data for some parameters. For example, with respect to the parameters K_{cm}, K_{th}, K_{wr}, and K_{in} of the design activity, we assigned reasonable values at first and then changed the values so that the simulated results of the design activity at PR_1 and PR_2 matched as the actual data for that activity. We also calculated the value of

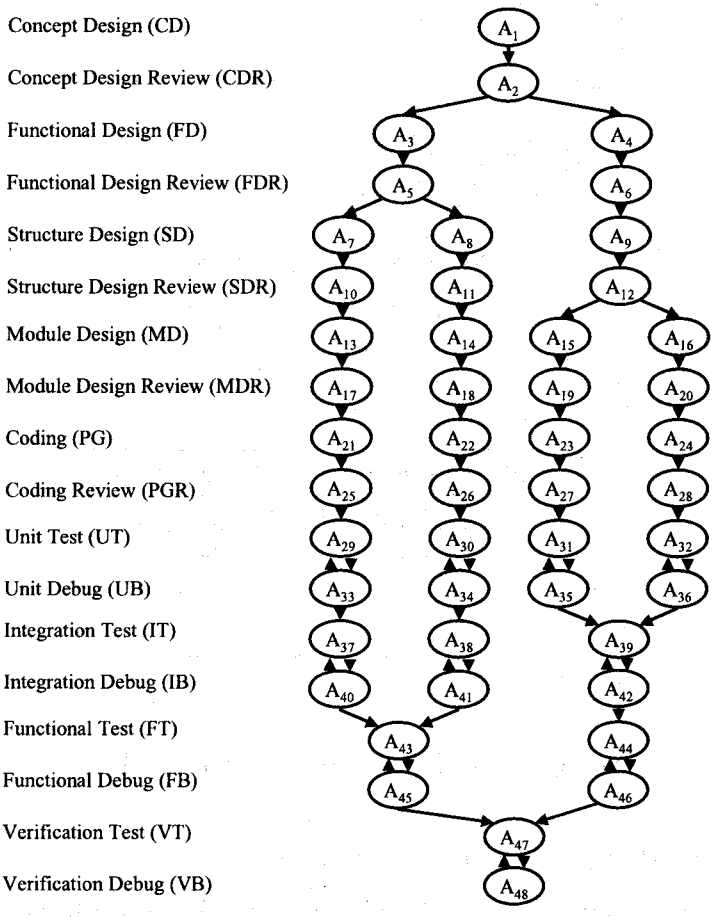


Figure 3.11: Development process of target projects

some parameters deterministically using actual input data. For example, we can calculate the *input product rate* parameter for each activity using the actual data for effort and product size in PR_1 and PR_2. We then get a common project description for projects similar to PR_1 and PR_2.

After calibrating the parameters, we described the project PR_3 by adding to the common project description some attributes that are unique to PR_3 (e.g. the number of developers). We simulated it repeatedly on the simulator and got the estimated values for PR_3 with respect to the development duration, development effort, and residual faults. In the case study, we repeated the simulation one hundred times and calculated the average values for the development duration,

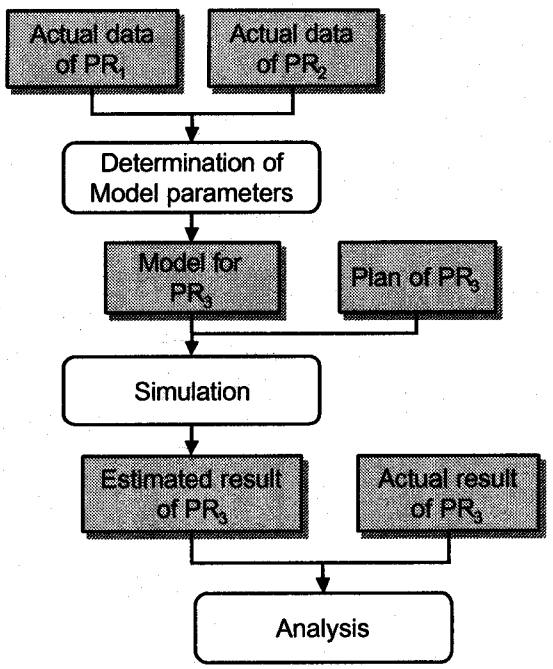


Figure 3.12: Outline of experiment

development effort, and residual faults of PR_3.

Finally, we compare these estimated values for PR_3 with the actual values for PR_3. The outline of experiment is also shown in Figure 3.12.

3.4.4 Simulation results

Table 3.1 shows both the estimated and the actual values for project PR_3. In Table 3.1, the estimated values for development duration, development effort, and the number of residual faults are 242 (days), 312 (person-days), and 15, respectively. On the other hand, the actual values for them are 248 (days), 329 (person-days), and 26, respectively.

For the development duration and development effort, the estimated values are quite close to the actual values. On the other hand, for the number of residual faults, the difference (= 11) is about three times as large as the standard deviation (= 4.21).

Table 3.1: Analysis result of simulation

	Development duration	Development effort	Residual faults
Simulation	242	312	15
(Std. dev.)	(6.54)	(12.08)	(4.21)
Actual value	248	329	26

Thus, we investigated the reason why an error occurs in estimating the residual faults. As a result of examining the data of PR_1, PR_2, and PR_3, we found that for projects PR_1 and PR_2 the average number of developers allocated to the test and debug activities was 10.5. On the other hand, for project PR_3, the average number of developers in the activities was 20. We consider that this developer allocation plan induced the error in estimating the residual faults. Though we discussed the bad effect of communication overhead in the earlier Sections, the good effect of increasing the number of developers in test and debug had a remarkable effect on the simulation results. We expect that the accuracy of the estimation can be improved by revising the equation of the fault injection rate. This revision is an important area for future research.

3.5 Discussions

3.5.1 Analytic solutions of GSPN model

Generally, the original GSPN model [36] has both high computation capability and high description capability. Its computational capability is equivalent to the Markov chain. The expected value for each resultant metric (such as effort, duration, and the number of residual faults) can be calculated analytically. However, in this model, we did not compute expected values analytically but by simulation.

The main reason was to avoid a state explosion problem. In the proposed model, the descriptions of activities are simple. However, the descriptions will become more complex as the details of an activity are investigated. A huge number of states will have to be enumerated to calculate the result of a simulation.

Another reason is related to extension of the simulator. We do not have to persist in the GSPN model. If we find a reason that the firing rate of developers' behavior is not exponentially distributed, the analytic solution of GSPN model is no longer used. The simulator, however, can be used by changing firing rules of transitions.

Thus, at this stage, we chose the simulation technique. The possibility of analytic calculation remains as future area of research.

3.5.2 Parameter setting of the proposed model

Before simulating the target project, we had to customize the simulator by tuning the values of the parameters so that each activity in the project can simulate the actual situation. It is generally very hard to determine these parameters, however, since they are tightly interrelated. Therefore, in this paper, we used heuristic values in Section 3.4. It is necessary to develop a systematic method or algorithms to determine the parameters. We feel that the parameter values can be determined by a stepwise method. In [39], we empirically found certain relationships between the parameters. We then chose several projects for the parameter determination, and determined the values of parameters for each project so that the results of the simulation became the same as the actual results. We are going to generalize the stepwise method to efficiently determine the parameters for the proposed model.

3.5.3 Tailoring the proposed model to other software organizations

Currently, the proposed model has been built for the software development process in Company *A*. Based on the data and experience from Company *A*, we determined the details of the model (For example, the various attributes of the project template in Figure 3.2 and the description of the activity model in Figure 3.4). In order to apply the model to software development processes in other organizations, we have to tailor the model. For example, we prepared five kinds of the activity models (design, coding, review, testing, and debug). If necessary, we should reconstruct them and add other kinds of the activity models. Also, in the activity models, the firing delay of each transition is exponentially distributed. Although this is a property of the GSPN model, our simulator does not utilize the analytic power of the GSPN model as explained above. It might be appropriate to modify the distribution to a normal distribution or some other distributions.

Chapter 4

Improvement in Parallel Execution

4.1 Parallel Execution of Activities

The standard waterfall process model presents each activity as usually being executed sequentially one after another. It makes the specification of each activity clear. If an activity has to start before the products from a previous activity are sufficiently completed, the following problems may occur:

- Usually, since incomplete specifications include many inconsistencies and mistakes, they are more difficult to read and understand than complete ones. Thus, the more incomplete specifications are, the more effort needed to understand them.

- Furthermore, such incomplete specifications may not be well reviewed. So, they may include a great many faults. Faults included in specifications may cause an additional injections of faults in successive activities.

Thus, in the waterfall model, activities in a process should be executed successively, not in parallel.

However, based on the data collected from not only the target project but also most other previous projects in Company *A*, it seems that there are times when some activities are executed in parallel with others.

One of the major reasons for parallel execution is insufficient development time. Software projects are usually executed on tight schedules, and thus the easiest way to recover the delay in a schedule is parallel execution of activities.

Basically, developers and managers do not want to perform parallel execution at the beginning of an activity. The parallel execution usually occurs when an activity should to finish in a few days, but the deadline is today!

Such a situation can be avoided by making an adequate and flexible plan. To do so, we have to improve the developers' awareness of parallel execution. So, we tried to show the developers the flaws in parallel execution by project simulation.

4.2 Case Study

4.2.1 Two cases for parallel execution

We considered the following two cases that are likely in the same target project:

- Case 1: Activities in the process are executed sequentially one after another except for the test and debug activities. This is an ideal execution of the process according to the waterfall model.

- Case 2: Each coding activity in the process is forced to start its executing if 70% of the previous design activity is completed as shown in Figure 4.1 (Other conditions are the same as Case 1). This is an exceptional case that could occur because of a severe workforce and time schedule.

In Case 2, each coding activity is started with an incomplete design specification, and executed in parallel with remaining 30% of the previous design activity. These cases are constructed based on interviews with actual developers in Company *A*. So, the degree of parallelization, 70%, is usual value in this company.

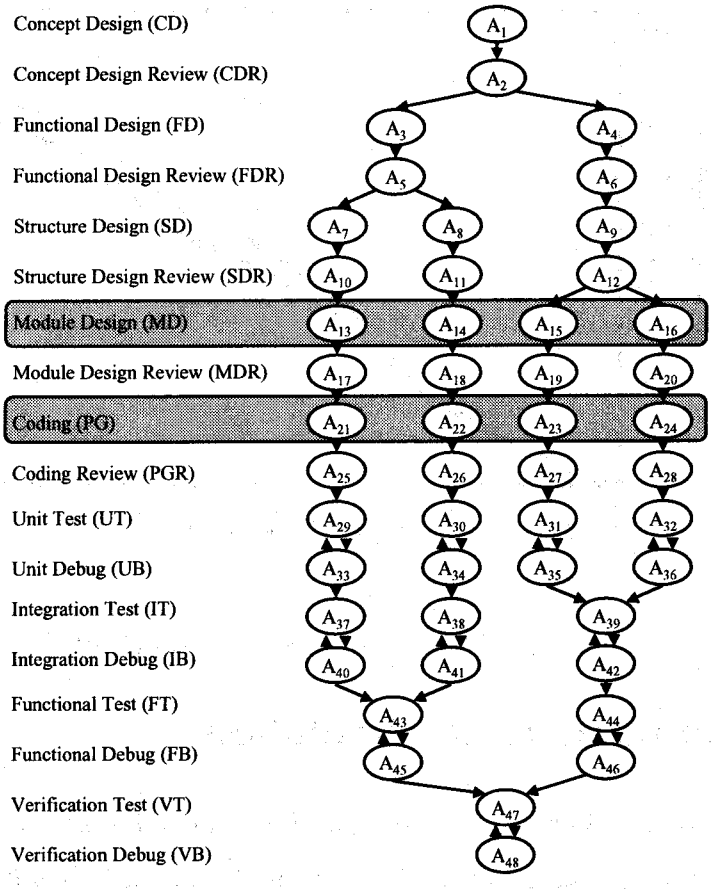


Figure 4.1: Parallel execution in the development process

4.2.2 Target project

The project targeted for the evaluation is one of a series of embedded software development projects in Company *A*. The main characteristics of the projects are summarized as follows:

1. The development effort is 5–50 (30 on average) person-months per project.
2. The development duration is 5–15 (7 on average) months per project.
3. The project members are almost unchanged throughout the all projects.
4. Each project uses a standard waterfall model.

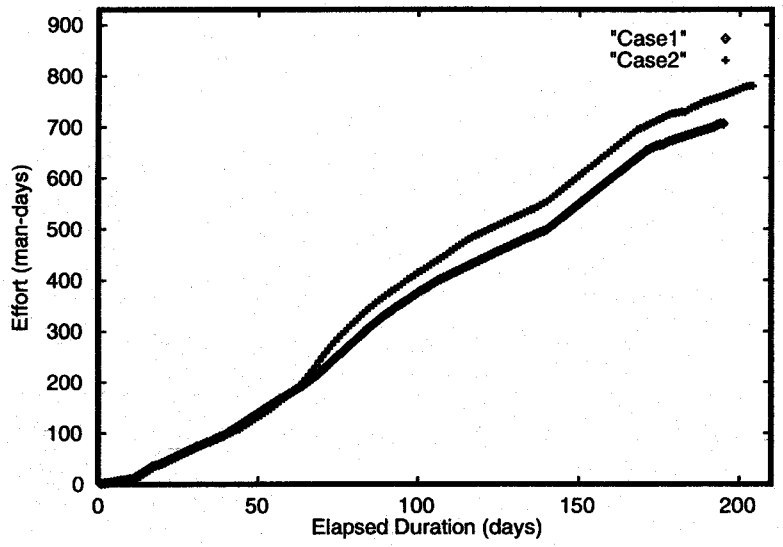


Figure 4.2: Variations of cumulative efforts

4.3 Experiment using Simulation

The simulations were iterated 100 times, and we tracked their execution. We first showed the result of a typical execution in order to see the behaviors of simulated projects in both cases. We then showed the statistical results of 100 executions.

4.3.1 Result of the simulation

Figures 4.2 and 4.3 show the results of simulating two cases in the same process described in the proposed model. The simulation results of Case 1 and Case 2 are plotted out with diamonds and pluses, respectively.

The variation in the cumulative efforts is shown in Figure 4.2. At the starting of the parallel execution of both coding and module design activities in Case 2, the difference of effort between Case 1 and Case 2 appears and the result is that the development duration of Case 2 is 9 days longer than that of Case 1 and the total effort of Case 2 is 74 person-days larger than that of Case 1.

The variation of the residual faults is shown in Figure 4.3. Just as for the variations in the cumulative efforts, the difference of residual faults in the two

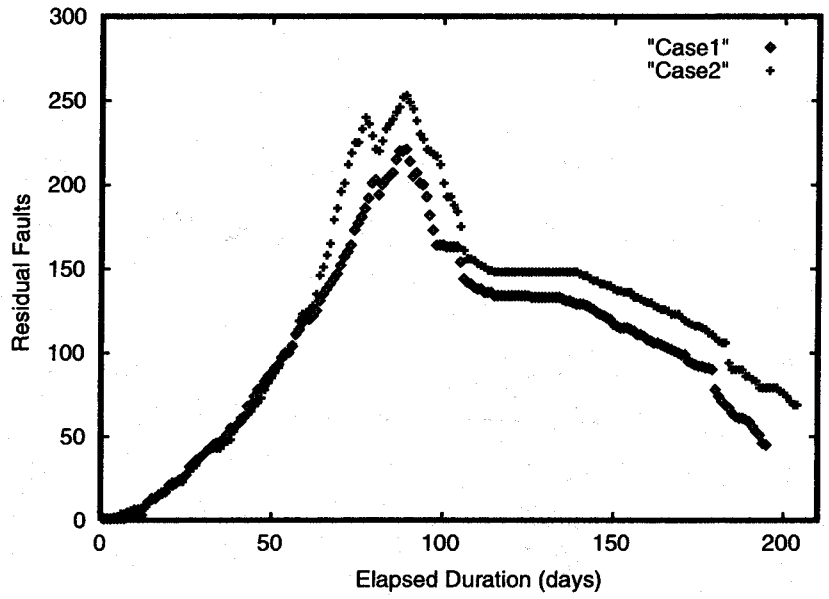


Figure 4.3: Variations of residual faults

cases appears at the start of the parallel execution in Case 2, and the number of residual faults in Case 2 is more than those of Case 1 ever after. At the point of process completion, a difference of 23 faults remains.

The causes of these results are as follows: The incomplete input products of the coding activities in Case 2 induces (1) an increase in injected faults in source code (as a result, the total workload of the activities in the lower stream of the process increases), and (2) an increase in communication overheads in the coding activities.

As mentioned above, the proposed model makes it possible to quantitatively evaluate a real software development process with severe constraints on resources, from the viewpoints of the quality, cost, and duration.

4.3.2 Statistical analysis

In order to verify that the proposed model definitely handles the effect of parallel execution, this section shows statistical analyses of the simulation results. We con-

firmed that the model handles the effect of incomplete products in the following way.

1. We confirmed that the difference between Case 1 and Case 2 is not seen until parallel execution of both design and coding activities start in Case 2.

2. We examined the difference in the cumulative efforts, project duration, and the number of residual faults between Case 1 and Case 2 at the point of process completion.

Table 4.1: Comparison of simulation results of before parallel execution (100 simulations)

Point of before parallel execution	Case 1	Case 2
Average duration	84	84
Standard deviation of durations	3.4	2.9
Average Effort	202	201
Standard deviation of efforts	7.7	7.5
Average # of Faults	89	91
Standard deviation of # of faults	8.7	10.0

Table 4.1 shows that there is no difference in simulation results between Case 1 and Case 2 until the parallel executions start in Case 2. (The statistical test shows that there are no significant differences in both average periods and average efforts between Case 1 and Case 2.) Next, Table 4.2 shows that there exists a difference in simulation results between Case 1 and Case 2 at the point of process completion. (The differences are shown to be significant by the statistical test.)

Note that "Effort" in the simulator means "the time in which developers actually work." So, the results in Table 4.2 do not imply that additional person-power was invested in Case 2.

Table 4.2: Comparison of simulation results at the point of process completion (100 simulations)

Point of process completion	Case 1	Case 2
Average duration	212	201
Standard deviation of durations	9.4	6.5
Average Effort	751	771
Standard deviation of Efforts	22.0	24.8
Average # of Faults	49	48
Standard deviation of # of Faults	7.5	5.7

Since the parallel executions of both design activities ($A_7, \cdots, A_{10}$) and coding activities ($A_{11}, \cdots, A_{14}$) are the only difference in process description between Case 1 and Case 2, we can consider that the difference in the simulation results of Case 1 and Case 2 are caused by the effect of the parallel executions, that is, by the incomplete input products.

4.4 Discussions for Practical Use (Another Experiment)

Finally, another simulation result for another project is shown here. In this experiment, a project with a rather small product size was chosen (The size of product was about 10KLOC and development effort was 68 person-days.). Simulations were performed by changing the degree of parallelization c (such as $c = 70\%$ in Case 2). We changed the value from $c = 100\%$ (it is in Case 1) to $c = 10\%$ (very heavy parallel execution). Figure 4.4 shows variations of effort and durations for these cases.

Figure 4.4 shows an interesting finding. As shown in the graph, the project duration drastically decreases in cases of rather "lightly" parallel executions(such as $60\% \leq c < 100\%$). However, for cases of "heavily" parallel execution (such

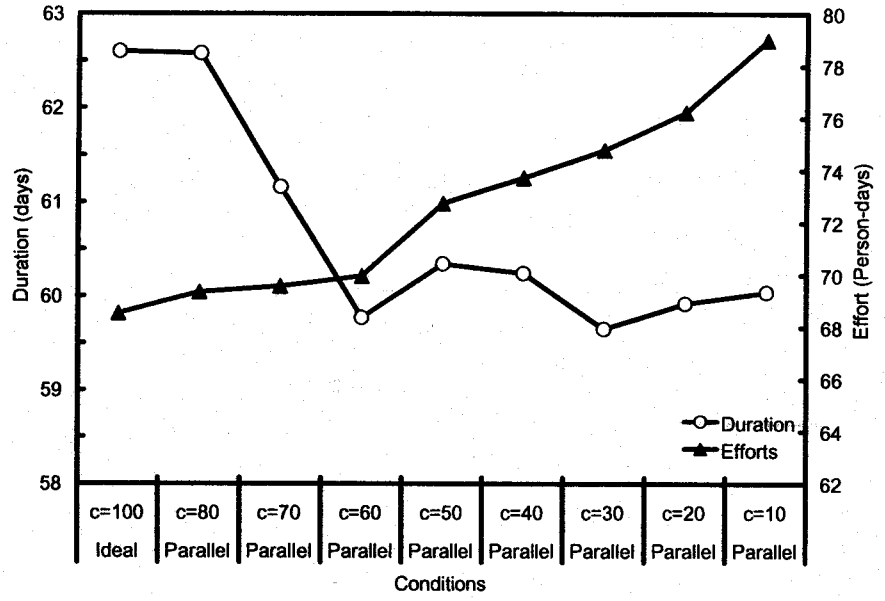


Figure 4.4: Variations of efforts and durations according to degree of parallelization.

as $10\% \leq c \leq 50\%$), durations do not decrease. The increase in effort for lightly parallel execution is rather small. It drastically increases in heavily parallel execution.

The initial objective of this experiment was to avoid the parallel executions in Company *A*. However, this result implies that there is a possibility of utilizing our simulator for more practical use. That is, we might find an optimal degree of parallel execution by simulations (that is, a relatively low increase in effort and high reduction in duration.).

In order to use the simulator for such practical work, there are many problems remaining at this point (mainly related to the accuracy, validity, and so on). However, this topic continues as an area for future work.

Chapter 5

Improvement in Test Process Planning

5.1 Dynamic Project Control

Generally speaking, a real software development process is a concurrent process in the sense that many activities are executed in parallel by team members. For example, (as was explained in Chapter 4) both the module design activities for subsystems and the coding and the coding review are executed concurrently. The former is for efficiency, but the latter may be due to a slippage in the schedule.

Therefore, it is not easy to manage the progress of a development project with such concurrent processes [24]. In particular, it is very difficult to control the process of development using a development plan initially constructed on insufficient data for the target project. It is also difficult to estimate the development cost. Many typical failures are found in so-called death march projects [65].

The test phase, especially, has the following problems or difficulties in controlling the progress of the test phase [22, 44].

- Determining the amount of effort for the test phase
- Estimating the quality of the delivered code

These problems are tightly coupled. If the effort is too small, the quality will become very poor. Even an excessively large effort may not improve the quality very much.

Based on these observations, we proposed a new method for controlling the progress of software development. The key idea is to update or modify the initial plan at an intermediate stage in the development, and to apply the updated plan to the succeeding stages. We then proposed to control the progress of the design and coding activities using the initial plan, and then control the progress of the test and debug activities using the updated plan. We call this method two-phase project control.

The plan update should reflect the results of development using the initial plan. For instance, the plan must consider the number of faults introduced into the product and the number of residual faults in the design and coding activities.

In this chapter, we confirm the usefulness of the two-phase project control using the project simulator proposed in Chapter 3. Additionally, in the evaluation we use real data in the simulation, which was collected from actual software development projects in Company *A*.

5.2 Software Process Model

5.2.1 Actual software process

An example of the actual software process was shown in Figure 3.11. That process is currently used in Company *A*. However, this actual development process is too complicated for further discussions in this chapter. We use instead a simple process model, shown in Figure 5.1, which maintains the fundamental flow structure in Figure 3.11. In this simplification, for example, the four module design activities A_{13}, A_{14}, A_{15}, and A_{16} in Figure 3.11 are merged into one module design activity (MD) in Figure 5.1. (The simple process in Figure 5.1 is only used to clearly define the proposed methods. The actual simulation in Section 5.4 uses

the actual process in Figure 3.11.)

Additionally, we logically divided the whole process into two phases from the viewpoint of faults: a design phase and a debug phase. The reason is that the activities in the design phase are primarily oriented toward increasing the size of the product and thus introducing faults, but the activities in the debug phase are for detecting and removing faults from the product.

5.2.2 Simple software process model

We assumed that software development process consists of two successive phases: a design phase and a debug phase. The design phase includes ten activities: (Concept Design (CD), Concept Design Review (CDR), Function Design (FD), Function Design Review (FDR) Structure Design (SD), Structure Design Review (SDR), Module Design (MD), Module Design Review (MDR), Coding (PG) and Code Review (PGR)). The debug phase includes eight activities: (Unit Test (UT), Unit Debug (UDB), Integration Test (IT), Integration Debug (IDB), Function Test (FT), Function Debug (FDB), Verification Test (VT) and Verification Debug (VDB)).

As shown in Figure 5.1, we introduced several parameters to make the discussions clear. Let e_i be the number of faults introduced into the product developed in design/coding activity i. Let d_j be the number of detected and removed faults in reviews/debug activity j. Let d_{design} be the total of d_{CDR}, d_{FDR}, d_{SDR}, d_{MDR}, and d_{PGR}. Let r_{design} be the number of residual faults in design phase and be calculated by $\Sigma e_i - d_{design}$. Also, let d_{debug} be the total of d_{UDB}, d_{IDB}, d_{FDB}, and d_{VDB}. Let r_{debug} be the number of residual faults in debug phase and be calculated by $r_{design} - d_{debug}$.

Strictly speaking, official acceptance testing and debugging are executed after VDB. Since these are not activities of the development team, we omit them from Figure 3.11 and Figure 5.1. Let r_{last} be the number of residual faults in the acceptance test and debug. Thus we can consider that r_{last} is the number of residual faults in the last product(usually program codes).

However, as a matter of fact, we could not collect all the values for the above

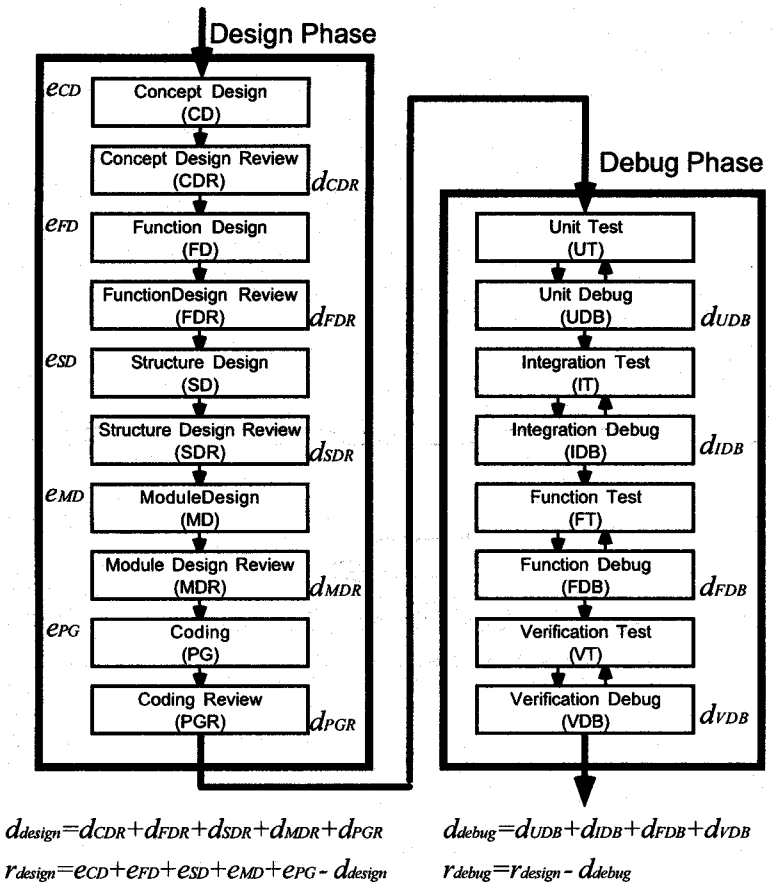


Figure 5.1: Simple development process

parameters. Thus, in this paper, we estimate the values for d_{design}, r_{design}, d_{debug}, r_{debug}, and r_{last}. In Section 5.4, we evaluate the usefulness of the proposed project control method by comparing the estimated values of d_{design}, r_{design}, d_{debug}, r_{debug}, and r_{last} with the actual ones.

5.2.3 Project plan

The initial project plan consists of the assignment of both developers and schedule of each activity. The assignments are determined at the beginning of project based on documents such as WBS(Work Breakdown Structure) charts, organization charts for the project, PERT charts, and a list of software products to

be developed. Simultaneously, the initial plan also reflects the experience and knowledge of the project manager.

First, we performed an estimation using the initial plan. However, some difference usually occur between the initial plan and the actual progress of the project. If the developers follow the initial plan regardless of the existence of these differences, then fatal confusion may be caused. On the contrary, if we can take the progress into consideration and reconstruct a new plan, then we should get a more accurate estimation. In Section 5.3, we will propose such a new project control method.

5.3 Project Control Methods

In this Section, we explain about proposed project control methods and project plan. The details of the project plan will be explained in subsection 3.1.4.

5.3.1 Single-phase control

Using single-phase control of a software development process, we constructed the initial project plan p at the beginning of project, and executed whole project under the initial project plan p (See Figure 5.2).

In the following, p_{design} and p_{debug} denote the project plan for the design and debug phases, respectively. We often use $p = (p_{design}, p_{debug})$ to denote a whole plan for the project.

Let the p^{init}_{design} and p^{init}_{debug} denote the initial plans of design and debug phases, respectively. We assume that both p^{init}_{design} and p^{init}_{debug} are constructed at the beginning of the project. The project plan of the single-phase controlled project is described as follows:

$$p = (p^{init}_{design}, p^{init}_{debug})$$

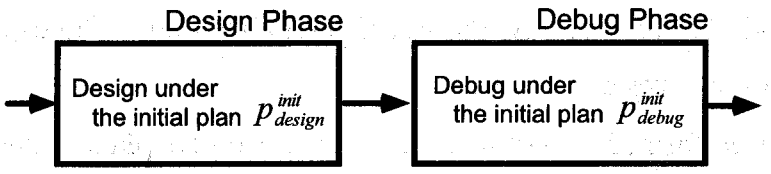


Figure 5.2: Single-phase control

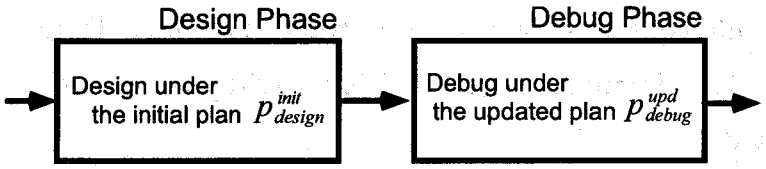


Figure 5.3: Two-phase control

5.3.2 Two-phase control

Next, using two-phase control of software development process, we constructed two versions of the project plans, $p = (p^{init}_{design}, p^{init}_{debug})$ and $p' = (p^{init}_{design}, p^{upd}_{debug})$. At the beginning of the project, we constructed the project plan $p = (p^{init}_{design}, p^{init}_{debug})$ and executed the design phase under p^{init}_{design}. We then stopped the execution of the project at the end of design phase, and updated the plan for the debug phase, p^{init}_{debug}, into a new plan, p^{upd}_{debug}, taking the development situation into account. After that, we continued the project under the updated new project plan p^{upd}_{debug} (See Figure 5.3).

For simplicity, we represented the project plan of the two-phase controlled project as follows:

$$p = (p^{init}_{design}, p^{upd}_{debug})$$

As already mentioned, in the project plan p for two-phase control, p^{init}_{design} is the initial project plan for design phase and p^{upd}_{debug} is the updated plan for the debug phase, which is constructed at the end of design phase.

5.4 Case Study

In order to evaluate the usefulness of the proposed project control method, we applied the simulator to similar software development projects, which were constructed based on the actual project data in the Company *A*. In the following, we call all of software development projects thus constructed imaginary software development projects.

5.4.1 Characteristics of target project

The main characteristics of the target project (it was rather a small project) are summarized as follows:

(1) Development effort of the project was 62 (man-days).

(2) The size of the system developed in the project was about 6.9 (KLOC).

(3) The project uses a standard waterfall model.

5.4.2 Experimental projects

Next, we considered five development projects: Project 0, Project 1, Project 2, Project 3, and Project 4, based on the development data of the actual project. Table 5.1 summarizes the characteristic attributes of these projects. Project 1 and Project 2 correspond to the project that executes debug phase under initial plan. Project 3 applies the proposed approach and Project 4 follows a uniform plan.

Project 0 is the only actual project; Project 1 through Project 4 are imaginary projects. All imaginary projects are executed by the project simulator mentioned in Chapter3. Since we evaluate all imaginary projects by comparing them to Project 0(for which we have data collected from actual project in Company *A*), we assume that the design phase of any imaginary project (except for Project 1) is executed under the actual plan P_{design}^{actual}. With this assumption, we can get highly

Table 5.1: Target projects

	Type of Project	Project Plan
Project 0	Actual	$p_1 = (P^{actual}_{design}, P^{actual}_{debug})$
Project 1	Imaginary	$p_0 = (P^{init}_{design}, P^{init}_{debug})$
Project 2	Imaginary	$p_2 = (P^{actual}_{design}, P^{init}_{debug})$
Project 3	Imaginary	$p_1 = (P^{actual}_{design}, P^{actual}_{debug})$
Project 4	Imaginary	$p_3 = (P^{actual}_{design}, P^{uniform}_{debug})$

accurate simulation results for the debug phase, and can compare in detail the resulting data among the five development projects.

Actual development

This is the actual development executed in Company A. Let P^{init}_{design} and P^{init}_{debug} denote the initial project plans for the design and debug phases, respectively, which were actually constructed at the beginning of the project in Company A. On the other hand, let P^{actual}_{design} and P^{actual}_{debug} denote specific project plans for the design and debug phases, which we constructed based on the records and the data collected at the completion of the target project.

In summary, the project was originally planned to be executed under the following initial plan p_0:

$$p_0 = (P^{init}_{design}, P^{init}_{debug})$$

Using the actual data resulting from the project, we can interpret that the project was executed under the following project plan p_1:

$$p_1 = (P^{actual}_{design}, P^{actual}_{debug})$$

Thus, we call this actual project (which was executed by the developers under the project plan p_1) as Project 0.

Imaginary projects(single-phase control)

Here, we defined two imaginary development projects that were executed under single-phase controlled plan.

At first, we defined an imaginary project, Project 1. In Project 1, the design and debug phases are executed under the initial plan P^{init}_{design} and P^{init}_{debug}, respectively. That is, the whole project is executed by project simulator under the initial plan p_0.

Next, we defined an imaginary project, Project 2, in which design phase is executed under the actual plan P^{actual}_{design} and debug phase is executed under the initial plan P^{init}_{debug}. This pattern enables us to create a case in which the plan for the design phase in Project 2 was not updated (which is the difference from Project 0).

Thus, Project 2 is executed under the following plan p_2 of single-phase control:

$$p_2 = (P^{actual}_{design}, P^{init}_{debug})$$

Note that two projects Project 1 and Project 2 have the same project plan P^{init}_{debug} for the debug phase, and that they have the different plans for the design phase.

Imaginary projects(two-phase control)

Now, we defined two imaginary development projects Project 3 and Project 4, under the two-phase project control method. As mentioned before, we assume that the design phase of Project 3 and Project 4 is executed under the actual plan P^{actual}_{design}.

First, we designed an imaginary project, Project 3, in which the design phase is executed under the actual plan P^{actual}_{design}, and its debug phase is also executed under the actual plan P^{actual}_{debug}. That is, Project 3 is executed under the project plan p_1.

Note that Project 3 is very close to Project 0, since Project 3 is entirely simulated by using actual data of Project 0. The difference is that Project 0 was executed by the developers, while Project 3 is executed by the project simulator.

Table 5.2: Assignment of workload

	UT & UDB	IT & IDB	FT & FDB	VT & VDB
P^{init}_{debug}	44	44	80	56
P^{actual}_{debug}	45	27	96	64
$P^{uniform}_{debug}$	52	52	52	52

Finally, we designed another imaginary project, Project 4, in which design phase is executed under the actual plan P^{actual}_{design}, and its debug phase is executed under the virtual uniform plan $P^{uniform}_{debug}$. We constructed the uniform plan $P^{uniform}_{debug}$ as follows: First, we obtained the total time to debug from the debug plan data. Next, each debug activity in this project was uniformly assigned the same amount of workload to be consumed. The values for all other attributes used in $P^{uniform}_{debug}$ were derived from P^{init}_{debug}. Thus, Project 4 is executed under the following uniform project plan p_3.

$$p_3 = (P^{actual}_{design}, P^{uniform}_{debug})$$

5.5 Experiment using Simulation

5.5.1 Preparation

Next, we assigned workload to each activity. As we mentioned before, we used five project plans, P^{init}_{design}, P^{actual}_{design}, P^{init}_{debug}, P^{actual}_{debug}, and $P^{uniform}_{debug}$. Table 5.2 shows a part of these project plans and summarizes the assignment of workload to each activity(UT and UDB, IT and IDB, FT and FDB, and VT and VDB)(see Figure 5.1). For P^{init}_{debug}, the workload is obtained from the plan of the project. For P^{actual}_{debug}, the workload is obtained from the actual project data. For $P^{uniform}_{debug}$, the same amount of workload is assigned uniformly to each activity.

Table 5.3: Values of quality metrics

	Executor	Plan	Design Phase		Debug Phase		Acceptance Test
			d_{design}	r_{design}	d_{debug}	r_{debug}	r_{last}
Project 0	developers	$p_1 = (P^{actual}_{design}, P^{actual}_{debug})$	19	—	16	—	1
Project 1	simulator	$p_0 = (P^{init}_{design}, P^{init}_{debug})$	22.80	25.39	17.97	7.42	0.82
Project 2	simulator	$p_2 = (P^{actual}_{design}, P^{init}_{debug})$	19.14	19.83	13.79	6.04	0.94
Project 3	simulator	$p_1 = (P^{actual}_{design}, P^{actual}_{debug})$	19.23	19.71	15.02	4.69	1.16
Project 4	simulator	$p_3 = (P^{actual}_{design}, P^{uniform}_{debug})$	19.58	19.87	12.90	6.97	1.13

5.5.2 Results of simulation

Using the project simulator, we iterated the simulations 100 times for each project and calculated the average value of the quality metrics.

Project 0 (actual development) had no information about residual faults at the end of the design/debug phase. We can find the value for the residual faults in the last product(r_{last}) from faults detected during the acceptance test, we cannot obtain data for the residual faults in other products from Company *A*.

As for Project 1, r_{debug} became 7.42. It was the worst value among all the projects. One of the reason is that Project 1 was executed using the initial project plan p_0 with no check points. (In some sense, Project 1 is essentially the same as Project 0. However Project 0 can be interpreted as if a check point was prepared at the end of the design phase.)

In Project 2, the design phase is executed under the actual plan P^{actual}_{design}, and the debug phase is executed under the initial plan P^{init}_{debug}. The result $r_{debug}(= 6.04)$ is better than that of Projects 1 and 4, but worse than that of Project 3. Due to the definition of Project 2, the project plan P^{init}_{debug} never reflects the situation at the end of the design phase, and thus it could not remove faults sufficiently.

The value of $r_{debug}(= 4.69)$ in Project 3 is the best in this experiment. Regarding

r_{debug}, we can say that the debug phase works very effectively under the project plan P_{debug}^{actual}. Since the plan for the debug phase was updated from the initial one in consideration of the situation at the end of the design phase, the updated plan seems to give adequate control for the succeeding debug phase (as shown in Table 5.2), and improve the value of r_{debug} compared with Project 2.

To tell the truth, we predicted that Project 4 would not show a good performance because we constructed the debug plan without considering the situation of at the end of the design phase. The result, $r_{debug} = 6.97$, is better than that of Project 1, but is worse than that of Projects 2 and 3.

As explained in subsection 5.4.2, the design phases of Projects 2, 3 and 4 were executed under the same actual project plan. That is, the values of d_{design} and r_{design} in Table 5.3 are almost the same for Projects 2, 3, and 4.

From the viewpoint of residual faults in all products (that is, the value of r_{debug}) after the debug phase, Project 3 has the least faults, and is superior to the Projects 1, 2 and 4. But, judging from residual faults in the last product (the value of r_{last}) after acceptance testing and debugging, there is no essential difference among the four projects.

5.5.3 Discussions

We investigated the reason why differences appear in the five experimental projects. From further investigations of the actual development data of Project 0, we conducted that the time assignment in the project plan P_{debug}^{actual} for the debug phase was rather reasonable. In particular, the data of Project 0 implies that if the concept design takes longer than the estimated time in the initial plan, then developers need to spend longer time in the verification test and debug.

Of course, there are many other factors, such as the developers' skill and the fault density, that affect the fault injection and removal in the projects. So we cannot conclude that the time assignment (that is, workload assignment) to each activity in debug phase is the key factor affecting to the number of detected/residual

faults. However, as far as the simulation results are concerned updating the time assignment in the project plan for the debug phase will effectively improve the number of detected/residual faults and quality of the products at the end of debug phase.

Chapter 6

Extension for Risk Prediction

6.1 Application to Risk Management

We have statistically analyzed the project data collected from a number of projects. During the analysis, we have noticed that there were projects that appeared to be out of control from the project managers' viewpoint. We called such projects "risky projects" [65]. We then tried to develop a risk predicting system that calculates the risk probabilities based on risk questionnaires for project managers and finds such risky projects based on those probabilities. An experimental application showed that most of risky projects can be detected by those probabilities [41]. The project managers were, however, still skeptical because the probability itself does not provide concrete proofs why a project becomes risky. The managers in Company *A* wanted such concrete proofs.

In this chapter, we try to estimate development costs, which are an important metric to determine risky projects. To do so, the project simulator must have capability to deal with risk factors in the risk questionnaire and to estimate development costs.

It is impossible for the initial simulator (proposed in Chapter 3) to simulate a project under risks, because the previous simulator cannot represent the risk factors. In the extended simulator, we implemented a mechanism that adjusts

parameters to deal with the influence of the risk factors. As a result, the following typical disorders in risky projects can be represented in the simulator: disorder caused by fluctuation in developer's skill level and disorder caused by deadline pressure [19].

Finally, we performed a case study to confirm whether a risky project can be simulated. The results show that the extended simulator can estimate the development costs for both an ideal case and a risky case. As a result, we confirmed that the simulator shows how much the development cost of a risky project exceeds the estimate.

6.2 Needs for Extension

6.2.1 Cost estimation of risky project

As mentioned in Chapter 2, the prediction of the final status of a project (risky or not) using the probability was successfully applied to actual projects. However, the project managers were still skeptical because they were provided with only the probabilities of risk. They actually wanted to know exactly what will happen in such risky projects.

We therefore tried to provide more concrete proof of risky projects. In more detail, we estimated the development cost of such risky projects, since the development cost is one of the important criteria to determine risky projects.

One of the possible ways to estimate the cost is to execute the project simulator under software risks. However, the simulator shown in Chapter 3 did not take the influence of software risks into account. Thus, in order to estimate the cost of risky projects, we extended the project simulator so that the software risks can be considered.

6.2.2 An approach for cost estimation considering risks

As a result of our investigation, we found that the following two disorders, which are closely related to the risk factors in Figure 2.1, will have a effect on the cost.

Disorder caused by a risky project: In a risky project, developers cannot bring their ability into full play [65]. That is, in a risky project, the skill level of a developer is reduced from his/her original level. It is known that the skill level of a developer significantly influences the cost of software projects [37, 54]. In the extended simulator, the developer's skill level can fluctuate according to the risks.

Disorder caused by deadline pressure: The deadline pressure causes many injected faults [19, 20], and thus increases the cost of the latter phases of a project. In a risky project, since the schedule is frequently delayed [65], the influence of the deadline pressure increases greatly. If the development duration of an activity is 90% completed and the assigned workload is not completed 90%, then the developers feel deadline pressure. In the extended simulator, the deadline pressure is represented as a parameter in the fault injection rate.

As will be described in the next section, these situations are implemented by adjusting the parameters in the project simulator according to evaluation results for risk factors. As a result, we expect that the cost of risky projects can be estimated more correctly.

6.3 Extension of Project Simulator

In this section, we try to extend the previous simulator so that the simulator can deal with the risk factors.

Activities' Definition

ID	Type	Assigned workload	Consumed Workload	Deadline	Injected faults	Detected faults	Input product(s)	Output product(s)	Developer(s)	Current condition	Start condition
0	FD	128	0	24	0	0	0	1	0,1	WAIT_START	0-NOT_START
1	FDR	16	0	0	0	0	1	1	0,1,2	NOT_START	0-FINISHED
2	PG	72	0	37	0	0	1	2	0,2	NOT_START	1-FINISHED
3	PGR	9	0	0	0	0	2	2	0,1,2	NOT_START	2-FINISHED
4	MT	48	0	45	0	0	1,2	3	0	NOT_START	3-FINISHED
5	MDB	0	0	45	0	0	3	1,2	0	NOT_START	4-FINISHED
6	FT	40	0	51	0	0	2	4	0	NOT_START	5-FINISHED
7	FDB	0	0	51	0	0	4	1,2	0	NOT_START	6-FINISHED

Products' Definition

ID	Size	Injected faults	Detected faults	Completion
0	10	0	0	1.0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

Developers' Definition

ID	Name	Skill	WorkTime
0	Willium	1	0
1	Richard	1	0
2	John	1	0

Figure 6.1: Project description for the simulator

6.3.1 Selected risk factors

Based on Figure 2.1, we selected risk factors to be added to the simulator. We identified and deleted several risk factors for which there are was insufficient data for analysis. However, since important risk factors may be included in these deleted factors, we consider further analysis of them as an important future work. As a result, 15 risk factors in Table 6.1 remain to be selected and added to the extended simulator. In the new questionnaire, responses should be filled in either binary, ordered, or absolute scales.

6.3.2 Parameters to represent the risk factors

In order to represent the risk factors in Table 6.1, we used the parameters in the project simulator. In the project simulator, the following parameters are used in the firing rate functions:

- The skill level of developers (L)
- The mental stress caused by deadline pressure (D)
- The completion rate of the input products (R)

Table 6.1: Risk factors to be used in extended simulator

	Risk factors	Evaluation
Requirements	Requirements specification review	Done | Not
	Number of requirements changes	#
Project Plan	Resultant products in the project plan	Described | Not
	Project plan review	Done | Not
	Project plan review by the manager	Done | Not
Estimation	Optimism for the technical issues	Exists | Not
	Appropriate estimation	Done | Not
	Needed time to make estimate	Days
Project Management	Unclear project management method	Unclear 3 - 0 Clear
	Review effort ratio	%
Developers	The morale of developers	Strong 3 - 0 Weak
	Experience of the requirement describer	High 3 - 0 Low
	Average experience of the developer	High 3 - 0 Low
	Responsive person for each activity in the WBS	Specified | Not
External Risks	Untechnical pressure	Exists | Not

- The number of faults in the input products (F)
- The number of developers in an activity (M)
- The constant for each developer's behavior (K_*)
- The assigned workload by the development plan (WL)

For example, the firing rate functions of transitions in Figure 6.2 are as follows*:

$$r_{cm} = K_{cm} \times \frac{M \times (M-1)}{L \times R}$$

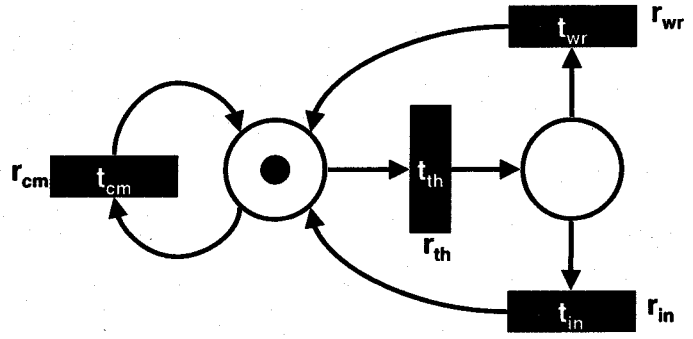
$$r_{th} = K_{th} \times \frac{L}{D}$$

$$r_{wr} = K_{wr} \times L \times D$$

$$r_{in} = K_{in} \times \frac{F+1}{R \times D}$$

The initial values of these parameters must be given in a project description. An example is shown in Figure 6.1. It shows the specified values for the activities such as FD, FDR, PG, and so on (As shown in Figure 5.1). The number of developers(M) for Activity 0 is 2, the initial skill levels(L) of all developers in this

*The definitions of firing rate functions are slightly extended from that of Chapter 3.



	t_{cm}	t_{th}	t_{wr}	t_{in}
Meaning	Communication	Think	Write	Write with a fault
Execution function	-	w = w + 1	s = s + 1	s = s + 1 f = f + 1

s : size of product
w : consumed workload
f : number of faults

Figure 6.2: Extended description of activity model

project are 1, the completion rate(R) of Product 0 is 1.0, and the workload(WL) for the Activity 0 is 128.

In the extended simulator, the values of parameters are adjusted according to the given software risks. (The details for the adjusting parameters will be explained in Section 6.4.) In the following subsections 6.3.3 and 6.3.4, we will explain how the risky situations in subsection 6.2.2 are represented by adjusting the parameters.

6.3.3 Confusion by fluctuating skill level

In risky projects, the skill of a developer can fluctuate (for instance, it falls down from 1.0 to 0.7). However, in the previous simulator, the value of the skill level L was a constant determined before a simulation begins. In the extended simulator, we adjust the value of L to represent a risky situation in a project.

Let us explain an example of a risky situation in a design activity. As shown in Figure 6.2, transitions t_{cm} and t_{th} fire selectively according to the firing rates

r_{cm} and r_{th}. Consider the following two cases. Assume that $K_{cm} = 0.1$, $K_{th} = 0.2$, $M = 2$, $R = 1$, and $D = 1$. Assume that the assigned workload is 80 hours.

Case 1 (Ideal Case): Consider a case of no risk, in which the skill level $L = 1$. Firing rates r_{cm} and r_{th} are calculated as follows:

$$r_{cm} = 0.1 \times \frac{2 \times 1}{1 \times 1} = 0.2$$

$$r_{th} = 0.2 \times \frac{1}{1} = 0.2$$

These two equations simulate a situation where communication occurs as frequently as the thinking ($r_{cm}/r_{th} = 0.2/0.2 = 1.0$).

In our simulator, the communication among developers is treated as a verbose behavior in the activity. So, communication is modeled as a wasting of time, that is, it does not consume any workload but only takes time. The cost of an activity increases according to the increase of time.

From the definition of Activity model in Figure 6.2, t_{th} fires 80 times during the activity, because the execution of an activity finishes when $w = 80$. Thus, t_{cm} also fires 80 times. Since the time needed for a communication behavior is 5 minutes, it takes 400 minutes (= 6.6 hours) more than expected.

Case 2 (Risky Case): Consider a case of a risky project, where the skill level of developers becomes $L = 0.7$. r_{cm} and r_{th} are then calculated as follows:

$$r_{cm} = 0.1 \times \frac{2 \times 1}{0.7 \times 1} = 0.29$$

$$r_{th} = 0.2 \times \frac{0.7}{1} = 0.14$$

In this case, the frequency of communication is about 2.1 times ($r_{cm}/r_{th} = 0.29/0.14 = 2.1$) more frequent than thinking. Thus, t_{cm} fires 168 times, and it takes 840 minutes (= 14 hours) more than expected.

As a result, since there are 2 developers($M = 2$) in this activity, the cost of Case 2 is $(14 - 6.6) \times 2 = 14.8$ person-hours more than Case 1.

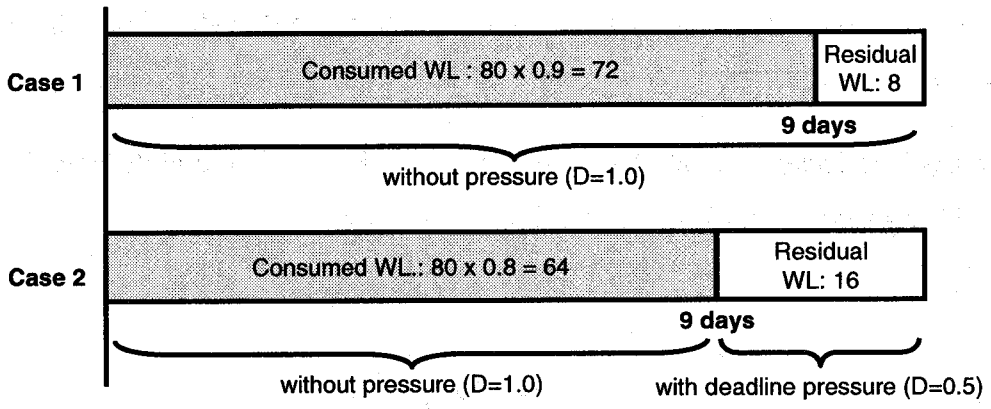


Figure 6.3: Example of deadline pressure

6.3.4 Confusion by the deadline pressure

As mentioned earlier, in risky projects, the influence of deadline pressure may increase greatly. The value of D in the previous simulator was also a constant determined when a simulation begins. In the extended simulator, we adjust the value of D to represent deadline pressure.

Consider the following two cases in a design activity. Assume that $K_{wr} = 1.0$, $K_{in} = 0.1$, $R = 1$, $F = 0$, and $L = 1$. Assume that assigned workload WL is 80 hours, and scheduled duration to complete the activity is 10 days.

Case 1 (Ideal Case): Assume that 90% of workload(WL) is consumed when 9 days elapse. Since consumption of WL is on schedule, deadline pressure does not happen(See Case 1 in Figure 6.3). So, $D = 1.0$ holds during the activity.

In this case, firing rates for transitions t_{wr} and t_{in} are calculated as follows:

$$r_{wr} = 1.0 \times 1 \times 1 = 1.0$$

$$r_{in} = 0.1 \times \frac{0+1}{1 \times 1} = 0.1$$

According to Figure 6.2, the number of faults injected in current product is calculated as follows:

$$f = 80 \times \frac{0.1}{0.1 + 1.0} = 7.3$$

Case 2 (Risky Case): Assume that 80% of *WL* is consumed when 9 days elapse. In this case, consumption of *WL* is behind schedule, and thus deadline pressure occurs[†] (See Case 2 in Figure 6.3). To represent the deadline pressure, the value of *D* is decreased from 1.0 to 0.5. During the consumption of the residual 20% of *WL*, the firing rates are changed as follows:

$$r_{wr} = 1.0 \times 1 \times 0.5 = 0.5$$

$$r_{in} = 0.1 \times \frac{0+1}{1 \times 0.5} = 0.2$$

The number of faults injected in the current product is calculated as follows:

$$\begin{aligned} f &= (80 \times 0.8) \times \frac{0.1}{0.1+1.0} \\ &+ (80 \times 0.2) \times \frac{0.2}{0.2+0.5} = 10.4 \end{aligned}$$

Faults will be removed in successive review or debug activities. If faults are detected in the review activity, it will take extra 30 minutes to remove each fault. That is, additional 1.5 person-hours are needed to remove the faults.

Furthermore, suppose that such faults are not detected in the review, and instead are detected in the test and debug activities. The activities are modeled to take 1 workload to detect a fault, and 1 workload to remove a fault. This implies that more than 6 person-hours (2 workloads × 3 faults) are needed to detect and remove such faults.

6.4 Implementation of Extended Simulator

6.4.1 Adjusting parameters for risk factors

In order to determine how to adjust the parameters, we interviewed the developers in Company *A*. As a result of the interviews, we finally determined the

[†]The reason why such delay occurs is out of the scope of this example, but we can assume that a certain risk has already existed in the activity.

Table 6.2: Adjustment of parameters according to evaluation result

Risk factors			Adjustments of parameters
	Description of risk factors	Evaluation results	
Requirements	Requirements specification review	Not done	Deadline pressure(D): decrease by 0.1. Completion rate(R) for initial product: decrease by 0.05. Injected faults(F) in initial product: increase by 2.
	Number of requirements changes	# of changes (α)	Workload for the project: increase by $w_1(\alpha)$ [max. 10%]. Deadline pressure(D): decrease by $d_1(\alpha)$ [max. 0.1]. Completion rate(R) for initial product: decrease by $r_1(\alpha)$ [max. 0.05]. Injected faults(F) in initial product: increase by $f_1(\alpha)$ [max. 3].
Project Plan	Resultant products in the project plan	Not described	Completion rate(R) for initial product: decrease by 0.05.
	Project plan review	Not done	Completion rate(R) for initial product: decrease by 0.05.
	Project plan review by the manager	Not done	Completion rate(R) for initial product: decrease by 0.05.
Estimation	Optimism for the technical issues	Exists	Workload for the project: decrease by 5%. Deadline pressure(D): increase by 0.1.
	Appropriate estimation	Not done	Communication rate(K_{cm}) in the design and coding: increase by 0.05. Thinking rate(K_{th}): decrease by 0.05.
	Needed time to make estimate	days (β)	Workload for the project: decrease by $w_2(\beta)$ [max. 5%]. Communication rate(K_{cm}) in design & coding: increase by $r_2(\beta)$ [max. 0.05]. Deadline pressure(D): decrease by $d_2(\beta)$ [max. 0.1].
Project Management	Unclear project management method	Unclear 3 - 0 Clear (δ)	Communication rate (K_{cm}) in all activities: increase by 0.01 x δ. Thinking rate(K_{th}): decrease by 0.01 x δ.
	Review effort ratio	Ratio in % (χ)	Workload for the review: set to χ% of corresponded activity's effort.
Developers	The morale of developers	Strong 3- 0 Weak (ε)	Skill level(L) of all developers: decrease by 0.03 x (3-ε).
	Experience of the requirement describer	High 3 - 0 Low (ϕ)	Skill level(L) of requirement describer: decrease by 0.03 x (3-ϕ). Completion rate(K_{cm}) for initial product: decrease by 0.02 x (3-ϕ). Injected faults(F) in initial product: increase by 1 if ϕ = 3.
	Average experience of the developer	High 3 - 0 Low (γ)	Skill level(L) of all developers: decrease by 0.03 x (3-γ).
	Responsive person for each activity in the WBS	Not specified	Skill level(L) of all developers: decrease by 0.05.
External Risks	Untechnical pressure	Exists	Workload for the project: decrease by 10%. Deadline pressure(D): decrease by 0.2.

parameters to represent the risk factors. Table 6.2 shows how the risk factors are implemented by the adjustment of parameters.

The left side of Table 6.2 shows the risk factors, and the right side shows the adjustment of parameters in the simulator to represent corresponding risk factors.

Most adjustments are specified by differences (plus or minus) with respect to the current values.

(1) For risk factors with a binary evaluation, we changed the values for the parameters according to one of the evaluations. For example, if the evaluation of "Requirement specification review" is "Not done", the parameters for 'deadline pressure' and 'completion rate' are decreased by 0.1 and 0.05, respectively, and 'injected fault' is increased by 2.

(2) For risk factors with an ordinal evaluation, we changed the values of parameters according to the corresponding order. For example, if the evaluation of "The morale of developers" is ϵ ($0 \leq \epsilon \leq 3$), the skill level of developers are decreased by $0.03 \times (3 - \epsilon)$.

(3) For risk factors with evaluations using an absolute value, we changed the values according to step functions such as $w_1(\alpha)$, $d_1(\alpha)$, and so on. For example, if the evaluation of "Number of requirements changes" is α, the workload for the project is decreased by step function $w_1(\alpha)$ where $w_1(\alpha)$ generates values from 0 to 3 according to α.

The "review effort ratio" is the only exception. The value of the parameter "workload for the review activity" is set to χ% of the corresponding design/coding activity according to the evaluation. The value of a parameter is cumulatively increased or decreased if several risk factors influence the parameter.

Let us explain an implementation of a risk factor. If the risk factor "Requirement specification review" is "Not done," then the corresponding parameters in the simulator are changed. Because of the incompleteness of the requirements specification, the completion rate R is decreased by 0.05. The incompleteness of

the requirements also influences the deadline pressure because such incompleteness is usually revealed in the final phase of an activity. The value of deadline pressure D is also decreased by 0.1. Since the lack of review implies several residual faults, injected faults F in the initial product are increased by 2[‡].

6.4.2 Procedure of application

Here, we explain the procedure of application of the extended simulator to an actual project.

In order to estimate costs of projects, we prepared inputs for the extended simulator. The inputs for the simulator are a "project description" (See Figure 6.1) and "evaluations of risk factors." The project description is constructed from the plan of the target project. It specifies the initial assignment of workload, assignment of developers, initial parameters in the simulator, and so on.

The evaluations of risk factors are collected from project managers of the target projects. The parameters in the simulator are adjusted by the evaluations of risk factors according to Table 6.2. The simulator then calculates the resultant cost for the project under the risk factors. By comparing the resulting cost in a risky situation with an initial estimate, we can see how much the cost exceeds the estimate under specified risks.

6.5 Case Study

In order to show the effectiveness of the extended simulator, we made a case study. The objective of this case study is to show that the new simulator can estimate the cost according to the applied risk factors.

[‡]Note that the values shown in Table 6.2 are heuristic ones based on our experience in Company A. The generalization of such parameter setting remains as a future work.

6.5.1 Target projects

In the case study, we use three projects (PR_1, PR_2, and PR_3), each of which is a typical development of embedded software for ticket vending machines in Company *A*. They were executed in 1997. The development process of these projects was the standard waterfall model, as shown in Figure 5.1.

6.5.2 Estimation by previous simulator

First, we simulated the 3 projects with the previous simulator so that the estimated costs reflect the planned costs[§] as correctly as possible. The estimated costs, calculated by the simulator, for the projects PR_1, PR_2, and PR_3 are shown in Table 6.3. For comparison, the planned cost and the actual resulted cost for each project are also shown. By comparing the planned cost and estimated cost in Table 6.3, we can see that the simulator can estimate the development cost quite correctly in terms of the project plan for each project.

Table 6.3: Estimated costs of 3 projects (person-days)

		CD	FD–PG	MT–VDB	Total
PR_1	Planned cost	31	73	41	145
	Estimated cost	34	78	50	162
	Actual cost	31	84	46	161
PR_2	Planned cost	–	78	14	92
	Estimated cost	–	72	37	109
	Actual cost	–	99	19	118
PR_3	Planned cost	76	33	30	139
	Estimated cost	70	42	31	143
	Actual cost	108	81	57	246

[§]The planned costs were calculated using conventional cost estimating method, such as COCOMO [8] or FPA [2], by the developers.

Table 6.4: Two cases in the case study

	Risk factors	Ideal Case	Risky Case
Requirements	Requirements specification review	Done	Not done
	Number of requirements changes	0	10
Project Plan	Resultant products in the project plan	Described	Not described
	Project plan review	Done	Not done
	Project plan review by the manager	Done	Not done
Estimation	Optimism for the technical issues	Not Exists	Exists
	Appropriate estimation	Done	Not done
	Needed time to make estimate	5	0
Project Management	Unclear project management method	0 (Clear)	3 (Unclear)
	Review effort ratio	20	3
Developers	The morale of developers	3 (Strong)	0 (Weak)
	Experience of the requirement	3 (High)	0 (Low)
	Average experience of the developer	3 (High)	0 (Low)
	Responsive person for each activity in the WBS	Specified	Not specified
External Risks	Untechnical pressure	Not exists	Exists

6.5.3 Estimation by extended simulator

We regret to say that we do not have actual evaluations for the risk questionnaire for the 3 projects, PR_1, PR_2, and PR_3, since these projects had already been completed in 1997. We therefore tried to see the effect of the implemented risk factors by constructing virtual situations.

We constructed two cases of risks: "Ideal (no risk) Case" and "Risky Case." The evaluations of two cases are shown in Table 6.4. The evaluations for "Risky Case" are entirely bad, and for "Ideal Case" are entirely good. The probabilities of being risky are calculated by the risk prediction model [41] for the cases "Ideal Case" and "Risky Case." They are 0.8% and 95.6%, respectively.

6.5.4 Discussions

Table 6.5 shows the estimated costs calculated by the extended simulator. In Table 6.5, "Decision by [41]" shows the evaluation of being risky or not, based on the risky project's decision criteria in reference [41] for the resultant project documents. Since there are no serious problems in the documents for the projects PR_1 and PR_2, they are considered as "Non-risky." On the other hand, in the

Table 6.5: Comparison of development costs

	Decision by [41]	Actual cost	Estimated cost	
			Ideal Case	Risky Case
PR_1	Non-Risky	161	149	230
PR_2	Non-Risky	118	118	149
PR_3	Risky	246	123	207

resulting documents of PR_3, the following descriptions remain:

- At the beginning, a certain part of the system was expected to be reused from other systems. However, it became clear during the functional design that that part must be modified.
- An experienced developer who knows that part of the system well was not available.
- As a result, the planned delivery date of the system was delayed twice.

The problem clearly corresponds to the risk factors in Table 6.1. From these facts, we consider that PR_3 was a "Risky" project.

The results of the "Ideal Case" for projects PR_1 and PR_2 show that they are close to the actual costs. On the other hand, estimated costs are relatively high in the case of the "Risky Case." Among them, the results for the "Risky Case" of PR_3 (that is, 207) is close to the actual cost (that is, 246).

As a result of this case study, we validated the execution of the extended simulator under risks at Company *A*.

Chapter 7

Conclusion

7.1 Achievements

In this dissertation, we have proposed a new method to simulate a software development project in order to apply software improvement activities in a certain company. The simulation model was described using a GSPN model, and a software project simulator was developed based on the model. It can simulate software projects from three viewpoints: development cost, project duration, and residual faults. An experimental evaluation showed that the simulator has high accuracy in estimation of cost, duration, and residual faults.

We then tried to apply the simulator to actual process improvement activities. To show its effectiveness, we performed three experiments: (1) Investigation of parallel execution of activities in a development process, (2) Evaluation of dynamic updating of project plans, and (3) Application to prediction of risky projects.

First, we performed an experiment to investigate the effect of "parallel execution" of activities in a development process. We simulated two cases: a case that does not include any parallel execution and a case that includes parallel executions between module design and coding activities. The result of the simulation showed that parallel execution actually reduces development duration, but it also

increase the development effort.

Next, we tried to apply our simulator to update a project plan. We proposed a two-phase project control method that examines the initial development plan at the end of design phase, updates it to the current status of the development process and executes the debug phase under the new plan. In order to show its usefulness, we define three imaginary projects based on actually executed projects in Company *A*: a project that executes the debug phase under the initial plan, a project that applies the proposed approach, and a project that follows a uniform plan. The result of this experiment showed that considering the actual situation of projects improves the resulting cost and quality.

Finally, we presented an extension of the software project simulator to perform risk prediction with a cost estimating capability. To consider the risk factors, we modified the previous simulator so that both the fluctuation of skill level and the deadline pressure can be represented by parameters in the simulator. By using a case study, we confirmed that the enhanced simulator can estimate the development cost under some typical risks. As a result, we confirmed that the simulator shows how much the development cost of a risky project exceeds an estimate.

Up to the present, numerous studies in software engineering have developed new methods, tools, or techniques to improve some aspect of software development or maintenance. However, it has been very difficult to introduce them into the actual software development. One of the reasons is that relatively little evidence has been gathered on which of these new developments are effective [61]. We consider that collaborative research between industry and academia in software engineering may be one solution to that problem and our results give a good suggestions for efficiently introducing software engineering techniques into an actual software organization. We would like to continue this collaborative research to develop a framework for effective technology transfer.

7.2 Future Work

Future works include the following:

(1) Generalization of the proposed model

This research was based on the data of Company *A*. We would like to apply the proposed simulator to other software developing organizations. To do so, a more general framework or environment should be prepared for the simulator. Also, the process model targeted in the simulator was a standard waterfall model. Other process models such as prototyping or incremental development, have recently been used in various software development fields. Thus we would like to investigate the applicability of the simulator to such process models.

(2) Analytic solution

As mentioned in Chapter 3, we did not utilize the analytic power of a GSPN model in the proposed simulator. In future research, we will try to investigate an analytic solution based on the original GSPN model. The state explosion problem in reachability analysis will be one of the most difficult problems to solve. If we can solve the GSPN model of process activity mathematically, many useful insights will be obtained from the analysis.

(3) Application to process patterns

In recent years, the notion of process patterns, which are identical to design patterns but applied to processes, has been introduced. The concept of the Activity model and Process model shown in Chapter 3 are exactly a pattern of processes and a database of patterns, respectively. We will investigate applicability of simulation technique to the field of process patterns.

Bibliography

[1] T. K. Abdel-Hamid, "The dynamics of software project staffing: A system dynamics based simulation approach," *IEEE Trans. on Software Engineering*, vol.15, no.2, pp.109–119 (1989).

[2] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation," *IEEE Trans. on Software Engineering*, vol.9, pp.639–648 (1983).

[3] M. Aoyama, "Agile software process model," *Proc. of 21th Annual International Computer Software and Applications Conference(COMPSAC97)*, pp.454–459 (1997).

[4] P. Armenise, S. Bandinelli, C. Ghezzi, and A. Morzenti, "Software processes representation languages: Survey and assessment," *Proc. of 4th Conference Software Engineering and Knowledge Engineering*, pp.455–462 (1992).

[5] S. C. Bandinelli, A. Fuggetta, and C. Ghezzi, "Software process model evolution in the SPADE Environment," *IEEE Trans. on Software Engineering*, vol.19, no.12, pp.1128–1144 (1993).

[6] V. R. Basili and H. D. Rombach, "The TAME project: Towards improvement-oriented software environment," *IEEE Trans. on Software Engineering*, vol.14, no.6, pp.758–773 (1988).

[7] V. R. Basili and J. D. Musa, "The future engineering of software: A management perspective," *IEEE Computer*, vol.14, no.9, pp.91–96 (1991).

[8] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall (1981).

[9] F. P. Brooks, Jr., *The Mythical Man-Month*, Addison-Wesley, MA (1975).

[10] A. M. Christie, "Simulation in support of CMM-based process improvement," *Journal of Systems and Software*, vol.46, pp.107–112 (1999).

[11] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol.31, no.11, pp.1268–1287 (1988).

[12] W. Deiters and V. Gruhn, "The funsoft net approach to software process management," *International Journal of Software Engineering and Knowledge Engineering*, vol.4, no.2, pp.220–256 (1994).

[13] M. Diaz and J. Sligo, "How software process improvement helped Motorola," *IEEE Software*, vol.14, no.5, pp.75–81 (1997).

[14] A. Drappa and J. Ludewig, "Quantitative modeling for the interactive simulation of software projects," *Journal of Systems and Software*, vol.46, pp.113–122 (1999).

[15] A. Drappa and J. Ludewig, "Simulation in software engineering training," *Proc. of 22nd International Conference on Software Engineering(ICSE2000)*, pp.199–208 (2000).

[16] N. E. Fenton and S. L. Pfleeger, *Software Metrics : A Rigorous & Practical Approach*, PWS Publishing (1997).

[17] A. Fuggetta, "Software process: A road map," *The Future of Software Engineering*, pp.27–34, Publication Dept. ACM (2000).

[18] K. Furusawa, Y. Hirayama, S. Kusumoto, and T. Kikuno, "Modeling and quantitative evaluation of software process based on a Generalized Stochastic Petri-net," *Proc. of 15th Software Reliability Symposium (in Japanese)*, pp.99–104 (1994).

[19] T. Furuyama, Y. Arai, and K. Iio, "Fault generation model and mental stress effect analysis," *Journal of Systems and Software*, vol.26, pp.31–42 (1994).

[20] M. V. Genuchten,"Why is software late? An empirical study of reason for delay in software development," *IEEE Trans. on Software Engineering*, vol.17, no.8, pp.582–590 (1991).

[21] N. Hanakawa, S. Morisaki, and K. Matsumoto, "A lerning curve based simulation model for software development," *Proc. of 20th International Conference on Software Engineering(ICSE98)*, pp.350–359 (1998).

[22] F. J. Heemstra, "Software cost estimation," *Information and Software Technology*, vol.34, pp.627–639 (1992).

[23] Y. Hirayama, O. Mizuno, S. Kusumoto, and T. Kikuno, "Hierarchical project management model for quantitative evaluation of software process," *Proc. of International Symposium on Software Engineering for the Next Generation*, pp.40–49 (1996).

[24] W. S. Humphrey, *Managing the Software Process*, Addison-Wesley, MA (1989).

[25] W. S. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, MA (1995).

[26] International Standards Organization, *ISO 9001: Quality Systems – Model for Quality Assurance in Design, Development, Production, Instllation and Serving*, International Standards Organization (1987).

[27] International Standards Organization, *SPICE Baseline Practice Guide, Product Description*, Issue 0.03 (Draft) (1987).

[28] A. Johnson,"Software process improvement experience in the DP/MIS function," *Proc. of 16th International Conference on Software Engineering(ICSE94)*, pp.323–329 (1993).

[29] M. I. Kellner, "Software process modeling support for management planning and control," *Proc. of 1st International Conference on Software Process*, pp.8–28 (1993).

[30] M. I. Kellner, R. J. Madachy, and D. M. Raffo, "Software process simulation modeling: Why? What? How?," *Journal of Systems and Software*, vol.46, pp.91–105 (1999).

[31] M. S. Krishnan and M. I. Kellner, "Measuring process consistency: Implications for reducing software defects," *IEEE Trans. on Software Engineering*, vol.25, no.6, pp.800–815 (1999).

[32] S. Kusumoto, O. Mizuno, Y. Hirayama, T. Kikuno, Y. Takagi, and K. Sakamoto, "A new software project simulator based on generalized stochastic petri-net," *Proc. of 19th International Conference on Software Engineering(ICSE97)*, pp.293–302 (1997).

[33] S. Kusumoto, O. Mizuno, T. Kikuno, Y. Hirayama, Y. Takagi, and K. Sakamoto, "Software project simulator for effective process improvement," *Trans. of Information Processing Society of Japan*, vol.42, no.3, pp.396–408 (2001).

[34] G. Lee and T. Murata, "A β-distributed stochastic Petri net model for software project time/cost management," *Journal of Systems and Software*, vol.26, pp.149–165 (1994).

[35] M. M. Lehman and J. F. Ramil, "The impact of feedback in the global software process," *Journal of Systems and Software*, vol.46, pp.123–134 (1999).

[36] M. A. Marsan, G. Balbo, and G. Conte, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Transactions on Computer System*, vol.2, no.2, pp.93–122 (1984).

[37] K. Matsumoto, S. Kusumoto, T. Kikuno, and K. Torii, "An experimental evaluation of team performance in program development based on model – Extension of programmer performance model," *Trans. of Information Processing Society of Japan*, vol.15, no.3, pp.466–473 (1992).

[38] R. McFeeley, "IDEAL: A User's Guide for Software Process Improvement," CMU technical report, CMU/SEI-96-HB-001 (1996).

[39] O. Mizuno, S. Kusumoto, and T. Kikuno, "Customization of software project simulator for improving estimation accuracy," *Proc. of 9th International Symposium on Software Reliability Engineering, vol.2*, pp.47–48 (1998).

[40] O. Mizuno, S. Kusumoto, T. Kikuno, Y. Takagi, and K.Sakamoto, "Experimental evaluation of two-phase project control for software development process," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E81-A, no.4, pp.605–614 (1998).

[41] O. Mizuno, T. Kikuno, Y. Takagi, and K. Sakamoto, "Characterization of risky projects based on project managers' evaluation," *Proc. of 22nd International Conference on Software Engineering(ICSE2000)*, pp.387–395 (2000).

[42] O. Mizuno, T. Kikuno, K. Inagaki, Y. Takagi, and K. Sakamoto, "Statistical analysis of deviation of actual cost from estimated cost using actual project data," *Information and Software Technology*, vol.42, pp.465–473 (2000).

[43] K. H. Möller and D. J. Paulish, *Software Metrics : A Practitioner's Guide to Improved Product Development*, IEEE Press, Chapman & Hall Computing (1993).

[44] J. D. Musa, A. Iannino, and K. Okumoto, *Software reliability: measurement, prediction, application*, McGraw-Hill (1987).

[45] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley (1994).

[46] M. C. Paulk, W. S. Humphrey, and G. J. Pandelios, "Software process assessments: Issues and lessons learned," *Proc. of International Software Quality Exchange*, pp.4B41–4B58 (1992).

[47] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall (1981).

[48] D. Pfahl and K. Lebsanft, "Integration of system dynamics modelling with descriptive process modelling and goal-oriented measurement," *Journal of Systems and Software*, vol.46, pp.135–150 (1999).

[49] D. Pfahl and K. Lebsanft, "Using simulation to analyse the impact of software requirement volatility on project performance," *Information and Software Technology*, vol.42, pp.1001–1008 (2000).

[50] A. Powell, K. Mander, and D. Brown, "Strategies for lifecycle concurrency and iteration – A system dynamics approach," *Journal of Systems and Software*, vol.46, pp.151–161 (1999).

[51] D. M. Raffo, "Evaluating the impact of process improvements quantitatively using process modeling," *Proc. of CASCON93, vol.1*, pp.290–313 (1993).

[52] D. M. Raffo, J. V. Vandeville, and R. H. Martin, "Software process simulation to achieve higher CMM levels," *Journal of Systems and Software*, vol.46, pp.163–172 (1999).

[53] I. Rus, J. Collofello, and P. Lakey, "Software process simulation for reliability management," *Journal of Systems and Software*, vol.46, pp.173–182 (1999).

[54] H. Sackman, W. J. Erickson, and E. E. Grant, "Exploratory experimental studies comparing online and offline programming performance," *Communications of the ACM*, vol.11, no.1, pp.3–11 (1968).

[55] K. Sakamoto, "A study of software process improvement and quality control based on analyses of actual project data," Doctor's thesis, Department

of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9861009 (2000).

[56] W. Scacchi, "Experience with software process simulation and modeling," *Journal of Systems and Software*, vol.46, pp.183–192 (1999).

[57] Software Engineering Institute, Carnegie Mellon University, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, MA (1995).

[58] I. Sommerville, *Software Engineering, 4th edition*, Addison-Wesley, MA (1992).

[59] Y. Takagi, T. Tanaka, N. Niihara, K. Sakamoto, S. Kusumoto, and T. Kikuno, "Analysis of review's effectiveness based on software metrics," *Proc. of 6th International Symposium on Software Reliability Engineering*, pp.34–39 (1995).

[60] T. Tanaka, K. Sakamoto, S. Kusumoto, K. Matsumoto, and T. Kikuno, "Improvement of software process by process description and benefit estimation," *Proc. of 17th International Conference on Software Engineering(ICSE95)*, pp.123–132 (1995).

[61] W. F. Tichy, N. Harbermann, and L. Prechelt, "Future directions in software engineering," *ACM SIGSOFT, Software Engineering Notes*, vol.18, no.1, pp.35–48 (1993).

[62] J. D. Tvedt and J. S. Collofello, "Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modeling," *Proc. of 19th Annual International Computer Software and Applications Conference*, pp.318–325 (1995).

[63] P. Wernick and M. M. Lehman, "Software process white box modelling for FEAST/1," *Journal of Systems and Software*, vol.46, pp.193-201 (1999).

[64] J. Williford and A. Chang, "Modeling the FedEx IT division: a system dynamics approach to strategic IT planning," *Journal of Systems and Software*, vol.46, pp.203-211 (1999).

[65] E. Yourdon, *Death March : The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*, Prentice-Hall Computer Books (1997).