Vol. 42 No. 3 IPSJ Journal Mar. 2001

IPSJ 40th Anniversary Award Paper

Software Project Simulator for Effective Process Improvement

Shinji Kusumoto,[†] Osamu Mizuno,[†] Tohru Kikuno,[†]
Yuji Hirayama,[††] Yasunari Takagi[††] and Keishi Sakamoto[†††]

In this paper, we propose a new model for describing software processes and an estimation method for the quality, cost and delivery date of a software project. The model was developed based on the experience and the data from the software development process at OMRON Corporation. 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. Process model includes a set of Activity models, each of which specifies design, coding, review, test, and debug activities respectively using GSPN (Generalized Stochastic Petri-Net). The new model can take the influence of human factors into account by introducing the concept of "workload." Next, we develop a simulator which supports description of the process and executes the process described by the Activity model. As the result of its execution, we get the estimated values for the quality, cost and delivery date of the target process. Then, we apply the simulator to real software processes at OMRON Corporation and compare these estimated values with actual data. The experimental results show the applicability of the proposed simulator to improve real software process.

1. Introduction

Software quality is a major problem facing software engineering. It is very important to establish effective procedures, methods, notations, tools, and practices for promoting software quality. Moreover, quality affects productivity, since products with higher quality require less rework and maintenance. It is generally said that in order to improve productivity and quality of software products, the software development process has to be improved. That is, to consistently improve products, the process used for developing them should be understood, defined, measured, and progressively improved[18].

There are numerous studies and reports regarding the improvement of software development processes[4],[9],[12]. The importance of the following two key activities is commonly stressed for the improvement of software development processes: (1) to understand and analyze the current status of the software development process, and (2) to construct and execute the improvement plan of the process based on the analysis results.

We have described an actual experience of software process improvement[22]. In order to improve the software process, we have proposed a process improvement procedure which describes the current process using Petri-net and estimates the benefits gained by the improvement. In the procedure, the current software process is described accurately and in detail, and then a feasible action plan is presented to developers. Also, benefit estimation is performed to evaluate the impacts of an action plan before the action plan is actually implemented. In Ref. 22), we applied the proposed procedures to a practical project. We estimated that by applying the action plan to the next project, 10% of the total effort/KLOC would be reduced at test phases. Then, actually, the estimated effort reduction had been attained. Then, our research objective shifted to execute the process improvement, especially benefit estimation, systematically from the viewpoint of quality, cost and delivery date.

This paper proposes a new model based on Generalized Stochastic Petri-net (GSPN)[7] 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 GSPN. The model can take the influence of human factors into account by introducing the concept of "workload." The workload of an activity is defined as total time needed for a developer with the average capability to com-

† Graduate School of Engineering Science, Osaka University
†† OMRON Corporation
††† SPI consultant

plete the activity. The workload can reflect the necessity of communication and performance of CASE tools, and thus the simulator can evaluate the dynamic aspect of the project. Additionally, so-called parallel executions of several activities (for example, design and coding) can be easily specified in the Process model to reflect the reality of the software project.

Next, in order to support description of the process and execute the process described by an Activity model, we develop a software project simulator. The kernel part of the simulator is developed using C language and the display and the editor are developed using Tcl/Tk. As the results of simulation, we get the estimated values for the quality, cost and delivery date of the target process. Then, we conduct empirical evaluation. In the experiment, we apply the simulator to real software projects at OMRON Corporation 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 project in the future.

The rest of this paper is organized as follows: Section 2 reviews the related works and our past experience. Next, Section 3 proposes the new project model. Section 4 describes the project simulator based on the proposed model. Then, Section 5 applies the simulator to the several software projects in OMRON Corporation and evaluates the usefulness of the proposed model. Section 6 discusses the applicability of the proposed method and future research works. Finally, Section 7 concludes this paper.

2. Preliminaries

2.1 Past Experience at OMRON Corporation

OMRON Corporation consists of several headquarters. One of them is the Electronic Fund Transfer Systems H.Q. (shortly called *EFTS*). The EFTS consists of several divisions including the Development Division and several business divisions. The SEPG (Software Engineering Process Group) is one of the groups within the Development Division. The SEPG was established in 1992 to improve the software development process of the development departments, each of which belongs to each business division. Moreover, the SEPG has been cooperating with several universities for process improvement.

In Ref. 22), we reported an actual experience of software process improvement at OMRON Corporation. For effective technology transfer, the SEPG had set three principal goals as follows: (1) motivate developers to improve on their process, (2) describe and define current software 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 the improvement.

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

Firstly, the SEPG held a series of interviews with developers and depicted a flow map in the form of a Petri net 19) which describes the current software process. Secondly, the group constructed an action plan from the in-depth analysis of the current flow map, and estimated the benefits obtained if this plan were to be rigorously followed. As a result, both the action plan and the benefit estimation were agreed by the developers as a feasible action plan. Furthermore, by applying the action plan to a practical project, it was confirmed that, compared to a similar project, approximately 10% of the total effort/KLOC was reduced at test phases. Then, we could suggest that the principal goals and the proposed procedure are effective to reduce the development effort at OMRON Corporation.

Then, our research objective shifted to develop an adequate process description language for a real project and then to execute the process improvement, especially benefit estimation, systematically from the viewpoint of quality, cost and delivery date.

2.2 Related Works

Several methods have already been proposed to model and evaluate the software development process. Kellner has proposed the evaluation method of software processes described by the modeling tool STATEMATE 13). This method has demonstrated how process models could be applied for software project management. Next, Lee and Murata have proposed a β-distributed stochastic Petri-net model for software project management[14]. This model is an integrated model of program evaluation and review technique (PERT) and Petri-net, and suitable to deal with the uncertainty and concurrency problems of large software project management. FUNSOFT Nets and SPADE are also the model based on Petri-net[2),3)] and are more oriented to process enactment than to process analysis and simulation. They mainly focus on the evaluation of the time constraints of the process.

Furthermore, Tvedt and Collofello have evaluated the effectiveness of process improvement on software inspections by using the system dynamics model[24]. This method makes it possible to predict the impact of process improvements through the cause-effect relationships on software development.

These methods are based on evaluation from the viewpoints of only the cost and delivery date. Thus they cannot totally evaluate 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[20]. This method, however, has not been applied to real software development processes.

3. Proposed Model

3.1 Overview

It is necessary to evaluate the software processes from the viewpoints of quality, cost and delivery date. At first, following the policy in Ref. 22), we decide to develop the model based on Petri-net. Among many kinds of Petri-net model, we select the Generalized Stochastic Petri-net (GSPN)[7].

Fundamental activities in the software development processes in Ref. 22) can be described by introducing the probability of the injection and removal of a fault as the firing rate of the transitions in order to evaluate the number of residual faults in the products.

Then, the concept of "workload" is considered in order to describe the fluctuations of the development period and product size. The interpretations of workload will be given in the next subsection.

There also exist several dynamic factors, which affect the behaviors of the developers, in the software development as follows:

(1) Communication overheads[5],
(2) Difference of experience[15),21)],
(3) Confusion by incompleteness[6],
(4) Stress by delivery date[8].

In the practical development, since these factors change the development period dynamically, it is very difficult to estimate the development period precisely. So, three attributes: developers' experience level, completion rate of products and deadline for activities, are incorporated to take the dynamic influence of human factors into account. Especially, "completion rate" of products makes it possible for developers to start concurrently the successive (and thus the next) activity based on incomplete documents developed by current activity. The degree of incompleteness is controlled by completion rate. All these attributes will be taken into a Project model later.

3.2 Key Concept "Workload"

Generally speaking, effort is used to measure the amount of the activity. But, the effort doesn't become clear until the activity is over, and thus the effort is not suitable to determine the amount of uncompleted activity. Additionally, the effort includes not only the amount of work needed for purely 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 this activity is performed by a developer in 10 days, it could not be performed by 10 developers in a day. One of the main reason is that time to communicate among the developers increases as the number of the developers increases.

Here, we define the term "workload" of an activity as total time needed for a developer who has the average capability to complete the activity. The similar concept to workload has been presented in Ref. 1). Our "workload" could be considered as an actual instance of the concept in Ref. 1). Additionally, an 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 the developers, the necessity of communication and performance of CASE tools. Then, 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 the 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 attained workload becomes 18 (= 10 × 2 × 0.9).

Case 2: Four developers, with the 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 attained workload becomes 16 (= 5 × 4 × 0.8).

If we get the workload of an activity, then we can estimate the development time appropriate for specific several 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, workload of design activity (W_{design}) is defined as the following formula:

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

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

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

3.3 Structure of New Model

The proposed model consists of a Project model and a Process model. **Figure 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 **Fig. 2**.

The Process model includes a set of Activity models which include specifications of design, coding, review, test, debug activities, and so on.

3.4 Project Model

The Project model focuses on three key components: activities, products and developers, and attaches several attributes to each of them (See Fig. 2).

An activity has eight kinds of attributes, which are *type*, *entry/exit conditions*, *input/output products*, *workforce*, *deadline* and *workload*. (1) *Type* shows which the activity corresponds to and describes currently one of design, coding, review, test and debug. (2) *Entry condition* and (3) *exit condition* specify conditions for beginning and ending the activity, respectively. (4) *Input products* describe the products given to the activity as the input products and the degree of contribution of each input products to determine workload of the activity. (5) *Output products* describe the output products that are developed in the activity and the weight assigned to each product. The variation of the product size and that of the number of faults are distributed to the products according to weights. Thus, the sum of each weight must be one. (6) *Workforce* specifies tuples of the developers who engage in the activity and the ratio of time in which each developer can engage in the activity to his or her business hours. (7) *Deadline* represents the appointed date for the completion of the activity, which is fixed on

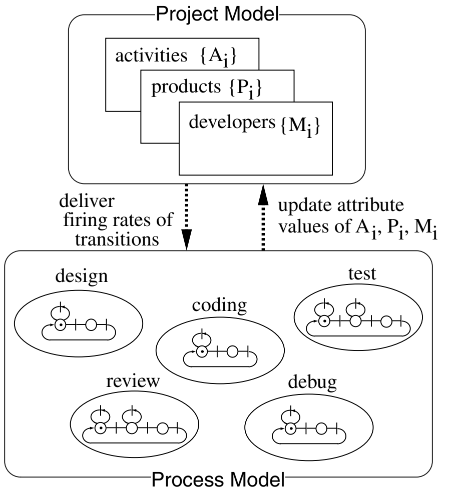


Fig. 1 Structure of the proposed model.

Attributes of activity A_i
type
entry condition
exit condition
input products
output products
workforce
deadline
workload

Attributes of product P_i
size
number of faults
completion rate

Attribute of developer M_i
experience level

Fig. 2 Project template.

the development plan. (8) *Workload* represents a tuple of the workload assigned to the activity and consumed amount of it.

Next, 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 consumed workload to the assigned workload.

Then, a developer has an attributes *experience level* which 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 discrete values 1, 2 and 3, respectively.

[Example 2] **Figure 3** shows an example of the Project model description. This project is composed of three activities (A_1, A_2, 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 c.* (the condition for starting the activity) represents that only if A_1 has not been started, it can be started at any time. *Exit c.* (the condition for ending the activity) represents that if all of the workload assigned to A_1 are consumed, its execution ends. *Input p.* and *output p.* show input and output products of the activity, respectively. *Input p.* $(= (P_0, 7.0))$ represents that the product P_0 is given to A_1 as an input product, and the workload equivalent to seven times as much as the size of the product P_0 is assigned to A_1. *Output p.* $(= (P_1, 0.3), (P_2, 0.5), (P_3, 0.2))$ represents 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* represents that the 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) represents 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 in terms of excluding omissions from 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 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 Fig. 3.

A_1	
type	FD
entry condition	(A_1, non-executed)
exit condition	(A_1, consumed)
input products	(P_0, 7.0)
output products	(P_1, 0.3), (P_2, 0.5), (P_3, 0.2)
workforce	(M_1, 1.0)
deadline	20

A_2	
type	PG
entry condition	(A_1, done)
exit condition	(A_2, consumed)
input products	(P_1, 1.2)
output products	(P_4, 1.0)
workforce	(M_2, 1.0), (M_3, 1.0)
deadline	35

A_3	
type	PG
entry condition	(A_1, done)
exit condition	(A_3, consumed)
input products	(P_2, 1.1)
output products	(P_5, 1.0)
workforce	(M_1, 1.0)
deadline	35

P_0	
size	8
number of faults	0
completion rate	1.0

M_1	
experience level	3

M_2	
experience level	2

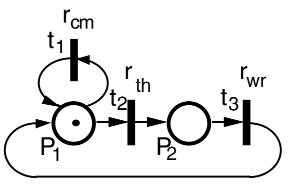
M_3	
experience level	1

Fig. 3 Example of project description.

Parallel execution of several activities can be easily defined by utilizing both attributes entry condition of activity and completion rate of product. With respect to example of the parallel execution, please refer to Ref. 10).

3.5 Activity Model

An activity model is prepared for each type of activities such as design, coding, review, test, debug and so on. The descriptions of the Activity models are given using an extended GSPN. **Figure 4** shows an example of the description of the design activity. In the extended GSPN, a token has three attributes: product size s,



transition	t_1	t_2	t_3
execution function	-	w=w+1	s=s+1 f=f+1 (p_{in})

Fig. 4 Activity model.

number of faults f and consumed workload w. These attributes are used to represent the current status of development that varies over the execution of each Activity model.

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 Fig. 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 execution function) to be evaluated on its firing. The execution of the function updates the attribute values of the token. Intuitively speaking, each transition corresponds to the developers' behavior such as thinking, writing and communicating or an event which occurs during execution of activity. Places correspond to waiting states for occurrences of behaviors or events.

[**Example 3**] Figure 4 shows a description of the design activity. Here, we consider three kinds of developers' behaviors in the design activity: communicating among developers, thinking for problem solution and writing for putting down the solution in documents. Transitions t_1, t_2 and t_3 in Fig. 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 concretely specified based on the property 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 of 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 concerned with 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}
$r_{wr} = f_{wr}(M, \Sigma L)$
(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) Testcase 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 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.

Moreover, the increase of 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 token's attributes can be changed by evaluating its execution function.

The activity model handles fault injections in the design activity as the 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}
$p_{in} = f_{in}(M, \Sigma L, D, R)$

By using this function, it is possible to take account of dynamic influence on the fault injection rate caused by the stress from deadline of

[☆] The functions f_{cm}, f_{th}, f_{wr} and f_{in} should be concretely given based on the characteristics of the project to be applied.

the activity or developers' experience levels.

Though the functions (1)–(3) are used in Fig. 4, the rests (4)–(9) are not included in Fig. 4. Figure 4 shows an example of design and coding activities and thus the improvement of model should be conducted through case studies. Besides we also modeled other activities (review, test and debug), in which the functions (4)–(9) are used. The detail of the activity models is shown in Ref. 11).

[**Example 4**] In the design Activity model depicted in Fig. 4, for example, transitions t_1 and t_2 which represent communicating and thinking behavior, respectively, are enabled to 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 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 could be increased according to the fault injection rate p_{in}. After the firing of t_3, the token moves back to the place P_1.

3.6 Simulation by Model

The development process specified by the model is carried out by repeating the interaction between the Project model and the Activity models at specified intervals. Each interaction cycle consists of the following three steps:

Step 1. Based on values of attributes of activities, products and developers, the Project model computes the firing rates of transitions of the Activity models.

Step 2. The Project model delivers the firing rates to the corresponding activity model, and then the Process model executes the process described by GSPN.

Step 3. The Process model returns the execution results to the Project model. Then, the Project model updates relevant attributes of activities and products based on the returned values.

4. Simulation Environment

In order to quantitatively evaluate software processes described by the proposed model, a simulation environment which executes the process automatically is indispensable. We have designed and implemented a simulator which supports description of the target process, executes the process described by the activity model and analyses the simulation results statistically. In this Section, we explain the overview of the simulator. The experimental evaluation of the simulator will be described in Section 5.

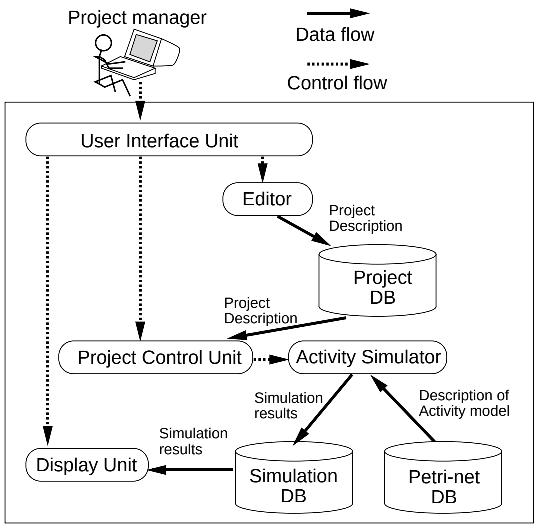


Fig. 5 System architecture.

4.1 System Architecture

Figure 5 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 Fig. 5, solid lines represent data flows and dotted lines represent control flows, respectively.

The followings are the function of each unit.

(1) Project control unit: Project control unit decides which Activity model is to be simulated by activity simulator according to the project description. It defines the relationship among activities, products and developers of the Project model, and sets the values of 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 attributes.

(2) Activity simulator: Activity simulator simulates activities specified by the Activity model using data such as name of the Activity model and parameters given from

project control unit. At first, it gets an Activity model with the same name from Petri-net database. Next, it simulates activities specified by the model using 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: User interface unit manages exchanges of data or commands between user and system (editor, project control part and display part).

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

(5) Editor: Editor supports to create the project description which is an input of project control unit. By using this editor, we can describe a project and set up all parameters needed in the proposed model. The output of editor is stored in the project database.

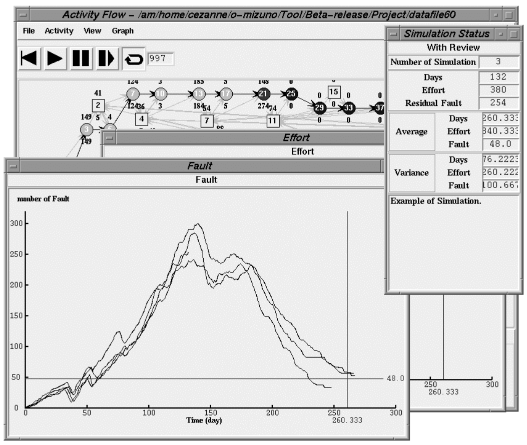
Since high speed computation is necessary for the project control unit and activity simulator, we implemented them using C language. On the other hand, the display unit and the editor, for which user-friendliness is strongly desirable, are implemented using Tcl/Tk. The program size becomes about 3500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

4.2 Behavior of Simulator

Simulations proceeds at the intervals of unit time☆. At first, project control unit determines activities to be executed, based on the current status of the progress and *entry/exit conditions* of each activities. Next, for each executable activity, project control unit delivers the parameters to activity simulator and directs it to execute activities for a day. Then, activity simulator executes all of the activities, which are directed to execute by project control unit, using given parameters and 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 to be completed.

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

☆ Currently, one unit time is a day (8 hours).



With Review		
Number of Simulation		3
Days		132
Effort		380
Residual Fault		254
Average	Days	260.333
	Effort	340.333
	Fault	48.0
Variance	Days	76.2223
	Effort	260.222
	Fault	100.667

Example of Simulation.

Fig. 6 Example of simulation.

Figure 6 shows the execution example of simulation which is in progress. 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 of each activity, assigned developers on a activity, size of product, the number of residual faults in product and so on. These information supports to investigate the progress of simulation in various ways. In Fig. 6, another window graphically shows the change of residual faults.

5. Empirical Evaluation

In order to evaluate the usefulness of the proposed method, we conduct an empirical evaluation. In the experiment, we apply the simulator to three similar software development projects PR_1, PR_2 and PR_3 in OMRON Corporation.

5.1 Assumptions

Here, we formulate 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, R is the completion rate of 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 concerned with communicating, thinking, writing and fault injection rate, respectively.

(H1) Communicating rate r_{cm} is proportional to the squared number of developers and inversely proportional to the developers' experience levels and 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 followings: (1) The number of communication paths among M developers is $M(M-1)/2$, and a novice developer needs more communications because of his/her immature knowledge, and (2) Incomplete input products induce frequent inquiries about the omission of the description [6].

(H2) 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 the productivity of software [15],[21].

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

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

(H3) Fault injection rate p_{in} is proportional to number of developers and inversely proportional to the average experience level of the developers, completion rate of the input products and the number of the days from the current date to the deadline of the activity.

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

The validity of (H3) comes from the followings: (1) The individual capabilities of developers are also related to the quality of software [15],[21], (2) Incompleteness of products prevents developers from performing their work correctly, and (3) The analysis in Ref. 8) shows that mental stress by deadlines is the largest cause of generated faults.

As spaces are limited, we omit the description of the Project model, formulas of firing rates, and parameters used by other activity models.

5.2 Characteristics of 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 15K steps.
(3) Project members are almost unchanged through all projects.
(4) Each project uses a standard waterfall model.

5.2.1 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. Now, however, we cannot obtain all of the necessary data, and thus we must initiate 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 assign the suitable values at first and, then, changed the values so that the simulated results of the design activity at PR_1 and PR_2 become the same as the actual data of it. Also, we calculated the value of some parameters deterministically using actual input data. For example, with respect to the parameter *Input product rate* of each activity, we can calculate it using the actual data of effort and product size of PR_1 and PR_2. Then, we get a common project description for those similar projects PR_1 and PR_2.

Then, we describe the project PR_3 by adding some attributes, that is peculiar to PR_3 (e.g., the number of developers), to the common project description. Successively, we simulate it on the simulator and get the estimated value of 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 of the development duration, development effort and residual faults of PR_3.

Finally, we compare these estimated values of PR_3 with the actual values of PR_3.

5.2.2 Simulation Results

Table 1 shows both the estimated and the actual values of project PR_3. In Table 1, estimated values of development duration, development effort and the number of residual faults are 242 (days), 312 (person-days) and 15, re-

Table 1 Analysis result of simulation.

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

spectively. On the other hand, actual values of 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).

Here, we investigate the reason why the error occurs in estimating the residual faults. As the results of examining the data of PR_1, PR_2 and PR_3, it is 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 induces the error in estimating the residual faults. Though we discussed the bad influence of communication overhead in the earlier sections, the good influence of increasing the number of developers appeared in the simulation results remarkably. We expect that the accuracy of the estimation can be improved by revising the equation of the fault injection rate. It is one of the important future research works.

6. Discussions/Future Works

6.1 Parameter Setting of the Proposed Model

Before simulating the target project, we must customize the simulator by tuning up the values of parameters so that each activity in the project can simulate actual situation. But, it is generally very hard to determine parameters, since these are tightly related each other. Therefore, in this paper we used heuristic values in Section 5. It is necessary to develop the systematic method or algorithms to determine the parameters. We consider that the values of parameters can be determined by stepwise method. In Ref. 16), 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 simulation became the same as the actual results. We are going to generalize the stepwise method to efficiently determine the parameters of the proposed model.

6.2 Application to Other Projects

In the actual process improvement activity, usually several action plans are appeared as candidates. In such case, it is necessary to identify the most appropriate plan for the current software development. In such situation, by using our simulator, we can choose the most appropriate one. In Ref. 17), we used the proposed simulator to compare two strategies of project planning in software development process. One is to construct the initial project plan at the beginning of project, and execute whole project under the initial project plan. The other is to construct the initial project plan at the beginning of project and execute the project under it until the end of design phase, then, construct the revised plan based on the data from design phase and execute the rest of the project base on the revised plan. Clearly, the latter strategy is more appropriate than the former. However, it needs a great deal of effort to collect the data from the design phase and revise the plan. So, the managers generally don't want to adopt the latter strategy. If it is necessary for the managers to adopt the latter strategy, we must show the effect of it. From the results of project simulation, we have confirmed the effectiveness of the latter strategy. We are going to use our simulator in actual software process improvement activities in OMRON Corporation.

6.3 Tailoring the Proposed Model to Other Software Organizations

Currently, the proposed model has been built for the software development process in OMRON Corporation. Based on the data and experience from OMRON Corporation, we determined the details of the model (For example, the several attributes of the project template in Fig. 2 and the description of the activity model in Fig. 4). In order to apply the model to the 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, 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. It may be appropriate to modify the distribution to normal distribution and so on.

7. Conclusion

We have proposed a new model for software project which can evaluate the software process from the viewpoints of the quality, cost and development period. In the new model, by introducing the concept of workload and the attribute of completion rate, it is possible to evaluate the dynamic aspect of software project.

Next, we have developed an integrated simulator to support the estimation of software project based on the proposed model. Finally, we have applied the simulator to real software projects and compare the estimated values with actual data. The experimental results show the applicability of the proposed simulator to manage real software project.

Up to the present, numerous studies in software engineering develop new methods, tools, or techniques to improve some aspect of software development or maintenance. However, it has been very difficult to introduce them to the actual software development. One of the reason is that relatively little evidence has been gathered on which of these new developments are effective[23]. We consider that the collaboration research between industry and academia in software engineering may be one solution to above problem and our results show a good suggestion to efficiently introduce the software engineering technique to the actual software organization. We would like to continue the collaboration research to develop a framework for effective technology transfer.

References

1) Abdel-Hamid, T.K.: The dynamics of software project staffing: A system dynamics based simulation approach, *IEEE Trans. Softw. Eng.*, Vol.15, No.2, pp.109–119 (1989).

2) Armenise, P., Bandinelli, S., Ghezzi, C. and Morzenti, A.: Software processes representation languages: Survey and assessment, *Proc. 4th Conf. Software Engineering and Knowledge Eng.*, pp.455–462 (1992).

3) Bandinelli, S.C., Fuggetta, A. and Ghezzi, C.: Software process model evolution in the SPADE Environment, *IEEE Trans.Softw.Eng.*, Vol.19, No.12, pp.1128–1144 (1993).

4) Basili, V.R. and Rombach, H.D.: The TAME project: Towards improvement-oriented software environment, *IEEE Trans. Softw. Eng.*, Vol.14, No.6, pp.758–773 (1988).

5) Brooks, Jr., F.P.: *The Mythical Man-Month*, Addison-Wesley (1975).

6) Curtis, B., Krasner, H. and Iscoe, N.: A field study of the software design process for large systems, *Comm. ACM*, Vol.31, No.11, pp.1268–1287 (1988).

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

8) Furuyama, T., Arai, Y. and Iio, K.: Fault generation model and mental stress effect analysis, *Journal of Systems and Software*, Vol.26, pp.31–42 (1994).

9) Genuchten, M.V.: Why is software late? An empirical study of reason for delay in software development, *Trans. IEEE Softw. Eng.*, Vol.17, No.8, pp.582–590 (1991).

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

11) Hirayama, Y.: Quantitative evaluation of software process based on hierarchical project management model, Master Dissertation, Osaka University (1996).

12) Johnson, A.: Software process improvement experience in the DP/MIS function, *Proc. ICSE16*, pp.323–329 (1993).

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

14) Lee, G. and Murata, T.: A β-distributed stochastic Petri net model for software project time/cost management, *Journal of Systems and Software*, Vol.26, No.2, pp.149–165 (1994).

15) Matsumoto, K., Kusumoto, S., Kikuno, T. and Torii, K.: An experimental evaluation of team performance in program development based on model – Extension of programmer performance model, *Journal of Information Processing*, Vol.15, No.3, pp.466–473 (1992).

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

17) Mizuno, O., Kusumoto, S., Kikuno, T., Takagi, Y. and Sakamoto, K.: Experimental evaluation of two-phase project control for software development process, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E81-A, No.4, pp.605–

614 (1998).
18) Paulk, M.C., Humphrey, W.S. and Pandelios, G.J.: Software process assessments: Issues and lessons learned, *Proc. ISQE92*, pp.4B41–4B58 (1992).
19) Peterson, J.L.: *Petri Net Theory and the Modeling of Systems*, Prentice-Hall (1981).
20) Raffo, D.M.: Evaluating the impact of process improvements quantitatively using process modeling, *Proc. CASCON93*, Vol.1, pp.290–313 (1993).
21) Sackman, H., Erickson, W.J. and Grant, E.E.: Exploratory experimental studies comparing online and offline programming performance, *Comm. ACM*, Vol.11, No.1, pp.3–11 (1968).
22) Tanaka, T., Sakamoto, K., Kusumoto, S., Matsumoto, K. and Kikuno, T.: Improvement of software process by process description and benefit estimation, *Proc. 17th International Conference on Software Engineering*, pp.123–132 (1995).
23) Tichy, W.F., Harbermann, N. and Prechelt, L.: Future directions in software engineering, *ACM SIGSOFT, Software Engineering Notes*, Vol.18, No.1, pp.35–48 (1993).
24) Tvedt, J.D. and Collofello, J.S.: Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modeling, *Proc. COMPSAC95*, pp.318–325 (1995).

(Received June 30, 2000)
(Accepted September 27, 2000)

Shinji Kusumoto was born in 1965. He received the B.E., M.E. and D.E. degrees in information and computer sciences from Osaka University in 1988, 1990 and 1993, respectively. He is currently Associate Professor in the Department of Informatics and Mathematical Sciences, Graduate School of Engineering Science, Osaka University. His research interests are software metrics, software quality assurance technique. He is a member of the IEEE, IEICE, JFPUG and IPSJ.

Osamu Mizuno was born in 1973. He received B.E. and M.E. degrees in information and computer sciences from Osaka University in 1996 and 1998, respectively. He has been working for Osaka University since 1999. He is currently a Research associate in the Department of Informatics and Mathematical Science at Osaka University. His research interests include the software process and the software quality assurance technique. He is a member of the IEEE.

Tohru Kikuno was born in 1947. He received M.S. and Ph.D. degrees from Osaka University in 1972 and 1975, respectively. He joined Hiroshima University from 1975 to 1987. Since 1990, he has been a Professor in the Department of Informatics and Mathematical Science at Osaka University. His research interests include the quantitative evaluation of software development processes and the analysis and design of fault-tolerant systems. He served as a program co-chair of the 1st International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98) and of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA'98). He also served as a general co-chair of the 2nd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '99). He is a member of the IEEE, ACM, IEICE and IPSJ.

Yuji Hirayama was born in 1971. He received B.E. and M.E. degrees in information and computer sciences from Osaka University in 1994 and 1996, respectively. He has been working for OMRON Corporation.

Yasunari Takagi received B.E. degree in information and computer science, from Nagoya Institute of Technology in 1985. He has been working for OMRON Corporation.

Keishi Sakamoto received B.E. degree in electrical engineering, from Kobe University in 1969 and D.E. degree in information science, from Nara Institute of Science and Technology in 2000. He had been working for OMRON Corporation from 1969 to 2000. Currently, he is a Software Process Improvement (SPI) Consultant.