Estimating the Number of Faults using Simulator based on Generalized Stochastic Petri-Net Model

Osamu MIZUNO†, Shinji KUSUMOTO†, Tohru KIKUNO†,
Yasunari TAKAGI‡ and Keishi SAKAMOTO‡
† Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University
‡ Development and Production H.Q., Social Systems Business Group,
OMRON Corporation, Japan

Abstract

In order to manage software projects quantitatively, we have presented a new model for software project based on Generalized Stochastic Petri-net model which can take influence of human factors into account, and we have already developed software project simulator based on GSPN model. This paper proposes methods for calculating model parameters in the new model and estimating the number of faults in the design and debug phases of software process. Then we present experimental evaluation of proposed method using a data of actual software development project on a certain company. As the result of case study, we confirmed its effectiveness with respect to estimating the number of faults in the software process.

1 Introduction

Increasing the productivity and quality of software development has been an important research objective in software engineering. It is reported that most companies spend between 50-80% of their development cost on test process [2]. Thus, putting more emphases on test process (testing and debugging) has been recognized to be one of the most effective approaches.

Software Reliability Growth Model (SRGM) is one of the most well-known models for quantitatively evaluating test process [8]. SRGM gives us useful information, e.g., the number of residual faults in the software or MTBF, for deciding the shipping date. On the other hand, program slicing[10] has been proposed and applied to localize faults in the program. By using slicing, debugger can efficiently determine the faulty statements in the program. However, the test process is very complicated consisting of: planning the general approach, finding resources and scheduling, determining features to be tested, designing the set of tests, implementing the plan and design, executing the test procedures, checking for termination, and evaluating the test effort and unit[11]. Thus, only using the SRGM or slicing is not sufficient to improve the test process.

On the other hands, there are numerous studies and reports [1][5] regarding the improvement of software development processes, including SEI Capability Maturity Model (*CMM*) proposed by Humphrey [5]. Based on a *process maturity model*, SEI Self-Assessment builds a consensus view of an organization's maturity and the key issues facing the organization. Then ultimately it presents an improvement plan for software development process endorsed by general management.

In order to attain the improvement of software development process, we have proposed process improvement procedure which describes the software development process using Generalized Stochastic Petri-net(GSPN) and estimates quality, cost and delivery date of it using the project simulator based on GSPN [6]. In [6], we showed that the simulator could estimate the value of the development duration and development effort.

In this paper, we show the usefulness of the simulator in estimating the quality of the product (e.g. the number of residual faults in the final product, the number of detected faults during reviews and debugs). That is, by using the simulator with the actual data of the previous or the past project A and development plans of the target project B, we propose a method to estimate the quality of the product developed in the project B.

In the proposed method, at first, we determine several parameters in the GSPN using the actual data of the previous project A. Next, using the development plans of the target project B, we define the parameters about workload. Workload is the key factor of

the proposed model and will be explained in subsection 3.1. Finally, using these parameters, we simulate the target project B and estimate the numbers of detected and residual faults in each debug activity of the project B.

2 Preliminary

We assume that software development process consists of two successive phases: design and debug phases (See Figure 1).

As shown in Figure 1, we introduce 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 is calculated by Σe_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 is calculated by r_{design}-d_{debug}.

However, as a matter of fact, we cannot collect all values of the above parameters. Thus, in this paper, we aim to estimate the value of d_{design}, d_{debug} and r_{debug}. In Section 5, we evaluate the usefulness of the proposed quality estimation method by comparing the estimated values of d_{design}, d_{debug} and r_{debug} and the actual ones.

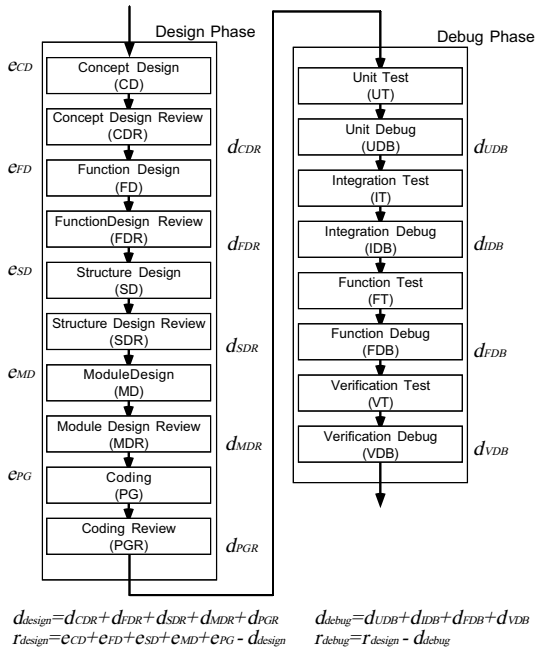


Figure 1: Software development process

3 GSPN Model and Simulator

3.1 Workload

In [6], we define the term "workload" of an activity as the total time needed for a developer who has the standard capability to complete the activity. 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.

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:

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

Here, s_{design} denotes the size of input product of design activity and R_{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.

3.2 Outline of the Model

The proposed model consists of Project model and Process model. Figure 2 shows the outline of the proposed model.

Project model includes three key components: activities, products and developers. Some attributes are attached to each of them, as shown in Tables 1.

Process model includes a set of Activity models which include specifications of design, coding, review, test, debug activities, and so on. Activity model is described by an extended GSPN, as shown in Figure 3.

3.3 Project Model

Project model focuses on three key components: activities, products and developers, and attaches several attributes to each of them (See Table 1).

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.

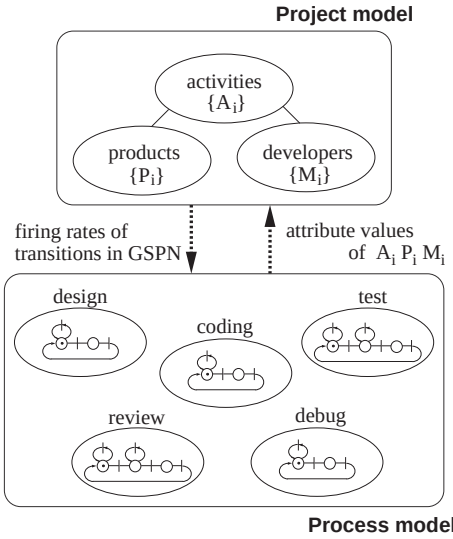


Figure 2: Outline of the proposed model

Table 1: Project template

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

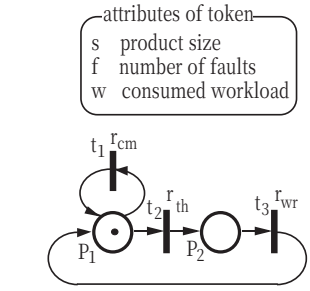
(2) *Entry condition* and (3) *exit condition* specify conditions for beginning and ending the activity, respectively. (4) *Input products* describes the products given to the activity as the input products and the workload parameter, that is the degree of contribution of each input products to determine workload of the activity. (5) *Output products* describes 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 the development plan. (8) *Workload* represents a tuple of the workload assigned to the activity and consumed amount of it.

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.

A developer has an attribute *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.

3.4 Activity Model for Process

Activity model is prepared for each type of activities such as design, coding, review, test, debug and so on. The descriptions of Activity models are given using an extended GSPN. Figure 3 shows an example of the description of the design activity. In the extended GSPN, a token has three attributes: product size s, 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.



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

Figure 3: 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 Figure 3, 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 oc-

curs during execution of activity. Places correspond to waiting states for occurrences of behaviors or events.

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 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 and D is the number of the days from the current date to the deadline of the activity. Activity model parameters (e.g., K_{cm}, K_{th}, K_{wr}, and so on) are given to each activity and concerned with the developers' behavior. For example, K_{cm}, K_{th} and K_{wr} correspond to communicating, thinking and writing respectively.

(1) Communicating rate: $r_{cm} = f_{cm}(K_{cm}, M, \Sigma L, R)$

(2) Thinking rate: $r_{th} = f_{th}(K_{th}, M, \Sigma L)$

(3) Writing rate: $r_{wr} = f_{wr}(K_{wr}, M, \Sigma L)$

(4) Preparing rate: $r_{pr} = f_{pr}(K_{pr}, M, \Sigma L, S)$

(5) Reading rate: $r_{rd} = f_{rd}(K_{rd}, M, \Sigma L)$

(6) Fault detecting rate: $r_{dt} = f_{dt}(K_{dt}, M, \Sigma L, S, F)$

(7) Fault modifying rate: $r_{md} = f_{md}(K_{md}, M, \Sigma L)$

(8) Testcase passing rate: $r_{ps} = f_{ps}(K_{ps}, M)$

(9) Fault localizing rate: $r_{lc} = f_{lc}(K_{lc}, M, \Sigma L, S, F)$

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.

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

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 activity or developers' experience levels.

[Example 1] In the design Activity model depicted in Figure 3, 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.

3.5 Simulator

We have designed and implemented a simulator which supports description of the target process, executes the process described by Activity model and analyses the simulation results statistically.

The system consists of five functional units: project control unit, activity simulator, user interface unit, display unit and editor.

Simulations proceed at the intervals of unit time[1] . 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

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

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.

4 Estimation Methods

Here, we propose two estimation methods for software faults. In the proposed methods, we consider two similar projects: the previous or the past project A (which has already been completed) and the target project B.

4.1 Method $M1$

In $M1$, the input data are actual development data of the previous project A (called D_A) and the development plan of the target project B (called P_B). Here, P_B consists of the estimated period of each activity and the assignment of developers. The output data are the values of quality metrics (d_{design}, d_{debug} and r_{debug}) at the design and debug phases in the project B.

$M1$ consists of the following four steps:

Step 1: Using D_A, calculate the activity model parameters.

Step 2: Using P_B, calculate the workload parameters in design and debug phases.

Step 3: Simulate the design and debug phases.

Step 4: Output the values of quality metrics.

4.2 Method $M2$

In $M2$, the input data are actual development data of the previous project A (called D_A), actual development data of design phase of the target project B(called D_B) and the development plan of the project B (called P_B). The output data are the values of quality metrics (d_{design}, d_{debug} and r_{debug}) at the design and debug phases in the project B.

$M2$ consists of the following four steps:

Step 1: Using D_A, calculate the activity model parameters.

Step 2: Using D_B, calculate the workload parameters in design phase. Also, using P_B, calculate the workload parameters in debug phase.

Step 3: Simulate the design and debug phases.

Step 4: Output the values of quality metrics.

5 Experimental Evaluation

In order to evaluate the usefulness of the proposed method, we apply the simulator to two similar software development projects A and B in an organization.

5.1 Characteristics of Target Projects

The main characteristics of the projects are summarized as follows:

(1)Development effort of the projects A and B are 152 (man-days) and 62 (man-days), respectively.

(2)The size of the system of the projects A and B are about 12.3 (Kstep) and 6.9 (Ksteps), respectively.

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

(4)Each project uses a standard waterfall model.

5.2 Assumptions

Here, we formulate each firing rate and fault injection rate in the design and coding Activity models. 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. On deciding these formula, we referred to past works such as [3][4][7][9].

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

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

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

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

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

5.3 Preliminary Experiment(Step 1 of $M1$ and $M2$)

Here, we execute the Step 1 of $M1$ and $M2$. That is, using D_A, calculate the activity model parameters. As described in Section 2, we assume the target process is shown in Figure 1. Actually, we have the data shown in Table 2. Thus, in order to determine the activity model parameters and workload parameters of the project A, we use all of d_i ($i = CDR, FDR, SDR, MDR, PGR, design$).

From now on, by using the concept design activity, we explain the way to determine the activity model

parameters. Before determining the activity model parameter, we have to determine the workload parameter. For the workload parameter, we use the formula shown in subsection 3.1. The actual value of the input product size of concept design activity is 50. The actual value of workload (the total amount of concept design time) is 121.5. Thus, workload parameter of concept design activity is $R_{CD} = 121.5/50 = 2.43$.

For the activity model parameters of concept design, we determine the values of K_{cm}, K_{th}, K_{wr} and K_{in}. We assign the suitable values at first and, then, changed the values so that the simulated results of the concept design activity at the project A become the same as the actual data of it.

Similarly, we obtained the workload parameters and the activity model parameters for all other activities of the project A. The resultant values of the activity model parameters are shown in Table 3. Concerning the values of the workload parameters we omit them, since they are nearly to secrets of the organization.

As a matter of course, the simulated values of d_{design}(=89) and d_{debug}(=32.1) are quite close to the actual d_{design}(=89) and d_{debug}(=30) of the project A.

Table 2: Real data of detected faults

d_{CDR}	d_{FDR}	d_{SDR}	d_{MDR}	d_{PGR}	d_{design}
23	21	16	12	15	89
	d_{UDB}	d_{IDB}	d_{FDB}	d_{VDB}	d_{debug}
	3	12	6	9	30

5.4 Experiment 1 (Step 2 through 4 of *M1*)

Next, we execute the Step 2 through 4 of $M1$. At Step 2, using P_B, we calculate the workload parameters in design and debug phases. Basically, we use the same way described in subsection 5.3 except using the values of workload and the input product size in the plan P_B instead of the actual values of workload and the input product size.

As the result, we obtained the values of the workload parameters, shown in Table 4, for all of the activities of the project B.

Then, in Steps 3 and 4, we simulate the project B and get the output values of quality metrics. The simulation results of the project B are shown in Experiment 1 of Table 5. The simulated values of d_{design}, d_{debug} and r_{debug} are 23.9, 14.9 and 1.1, respectively. Though the simulated values of d_{debug} and r_{debug} are close to the actual d_{debug}(=16) and r_{debug}(=1) of the project B, there is a relatively large difference between the simulated value of d_{design} and the actual one.

5.5 Experiment 2 (Step 2 through 4 of *M2*)

Now, we execute the Step 2 through 4 of $M2$ to reduce the difference of d_{design},d_{debug} and r_{debug}. At Step 2, using D_B, we calculate the workload parameters in design phases. As for the workload parameters in debug phase, we use the same one in Step 2 in Experiment 1. The updated values in Experiment 2 are shown in Table 4.

Then, in Steps 3 and 4, we simulate the project B and get the output values of quality metrics. The simulation results of the project B are shown in Experiment 2 of Table 5. The simulated values of d_{design}, d_{debug} and r_{debug} are 17.9, 15.8 and 0.7, respectively. These values are quite close to the actual values of d_{design}(=19), d_{debug}(=16) and r_{debug}(=1) of the project B.

Thus, with respect to the experiments, we can conclude that the proposed method can correctly estimate the numbers of faults in the design and debug phases.

Table 5: Result of experiments

	d_{design}	d_{debug}	r_{debug}
Real data	19	16	1
Experiment 1	23.9	14.9	1.1
Experiment 2	17.9	15.8	0.7

6 Conclusion

In this paper, we applied the simulator to the evaluation of the testing plan. In our proposed method, we determined the model parameter of GSPN model using the development data of the previous project A. Next, we determined the workload parameter of project model using the development plan of the project B, which is the target project. Then, we simulated the project B using the above parameters and estimated the number of detected and residual faults in design and test phases of the project B.

Finally, we experimentally evaluated the proposed method using the data of actual software development project on a certain company. Though it has some limitations, we confirmed the effectiveness of the proposed method.

References

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

[2] J. S. Collofello and S. N. Woodfield: "Evaluating the effectiveness of reliability-assurance

Table 3: Values of activity model parameters

	K_{cm}	K_{th}	K_{pr}	K_{lc}	K_{rd}	K_{ps}	K_{dt}	K_{wr}	K_{md}	K_{in}
design	0.10	0.20	—	—	—	—	—	0.20	—	15.5
coding	0.10	0.20	—	—	—	—	—	0.20	—	17.0
review	0.08	—	0.10	—	3.00	—	0.35	—	0.20	—
test	0.10	—	0.20	—	—	6.00	0.24	0.20	—	—
debug	0.08	—	—	0.12	—	—	—	—	0.10	—

Table 4: Values of workload parameters

	R_{CD}	R_{CDR}	R_{FD}	R_{FDR}	$\cdots$	R_{VDB}
Experiment 1	1.14	0.45	4.20	0.5	$\cdots$	2.0
Experiment 2	1.88	0.30	1.17	1.0	$\cdots$	2.0

techniques," *Journal of Systems & Software,* Vol.9, No.3, pp.191-195 (1989).

[3] 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).

[4] T. Furuyama, Y. Arai and K. Iio: "Fault generation model and mental stress effect analysis", *The Journal of Systems and Software*, Vol.26, pp.31-42 (1994).

[5] W. S. Humphrey: "Characterizing the software process: A maturity framework," *IEEE Software*, Vol.5, No.2, pp.73-79 (1988).

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

[7] 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", *Journal of Information Processing*, Vol.15, No.3, pp.466-473 (1992) (in Japanese).

[8] J.D. Musa, A. Iannino and K. Okumoto: *Software Reliability: Measurement, Prediction, Application,* McGraw-Hill (1987).

[9] 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).

[10] M. Weiser: "Programmers use slices when debugging", *Communications of the ACM*, Vol.25, No.7, pp 446-452 (1982).

[11] "IEEE Standard for Unit Testing", IEEE. Rep. IEEE-Std-1008-1987 (1987).