Application of Generalized Stochastic Petri-net to Quantitative Evaluation of Software Process

Osamu MIZUNO, Yuji HIRAYAMA, Shinji KUSUMOTO and Tohru KIKUNO
Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560, Japan
E-mail: o-mizuno@ics.es.osaka-u.ac.jp

1. Introduction

As the complexity and size of software increase, project management becomes important to develop reliable software cost-effectively within specified time constraints. In order to manage a software project effectively, it is necessary to establish a control method based on the objective and quantitative evaluation data on the current status of the software project.

We have already presented quantitative evaluations of the software development process described by Generalized Stochastic Petri-net (GSPN)[3]. However, the evaluations in [3] cannot deal with the development period sufficiently because of restrictions on the GSPN model.

This paper defines a new hierarchical model for software project by extending the previous GSPN model and by introducing the concept of "workload", and proposes a method to evaluate the software process from the viewpoints of the quality, cost and delivery date. As the result, the new model can take the influence of human factors into account, and thus can evaluate the dynamic aspect of software project. Finally, the results of the case study of the proposed method provide us with some implications on software project management.

2. Fundamental Model

2.1 Extended GSPN [3]

A Petri-net is one of the most powerful models for representing and analyzing concurrent processes such as those occurring in software development process. A stochastic Petri-net is an extended model of a Petri-net by incorporating timing information. It has two types of transitions: timed transition and immediate transition. An immediate transition is the transition whose firing delay is zero. A timed transition needs specified time to fire.

Marsan et al. have proposed a generalized stochastic Petri-net (GSPN) [8] whose transition has exponential distributed firing rate. A GSPN has been used extensively to model and analyze dynamic behaviors of systems and software processes. However, a GSPN cannot totally evaluate the important factors of software project.

So, we have extended a GSPN in order to quantitatively evaluate software processes from the viewpoints of quality, cost and delivery. In the extended GSPN, a token has some attributes, which are useful for activity evaluation. Additionally, each transition of the extended GSPN has an execution function which is evaluated in time of firing. The value of each attribute of the token is updated by evaluating the execution functions.

2.2 Problems of extended GSPN

The extended GSPN model[3] did not include the following factors which affect the behaviors of the developers:

1. Communication overheads ··· Communication overheads are recognized to be serious in the software development. For example, it is impossible for one thousand developers to complete the software in a month that completed for one thousand man-months[1]. The major reason is that the increase of developers induces the increase of overheads for communications among the developers.

2. Difference of experience ··· The capability of each developer is strongly related to the productivity and quality of software. It is reported that there are large individual differences in programming performance and developers with high performance tend to develop programs with fewer faults[9][11].

3. Confusion by incompleteness ··· The design and coding activities mainly include translation work from input products into output products. For example, the coding activity is described as translation work from the design specifications into the source code. In such translation activities, if there are some omissions of required description on the input products, it needs corresponding inquiries, and then communication overheads increase[2].

4. Stress by delivery date ··· Usually, a delivery date or a deadline is fixed for each activity on the development plan. As the deadline closes in, stress on developers gets serious and could affect their performance negatively. It is reported that stress caused by the deadlines or short development time incurs the high incidence of errors and faults[4].

In a practical development, since these factors change the development period dynamically, it is very difficult to estimate the period precisely. The method in [3] simplifies the estimation by assuming that the value of the development period always becomes constant. Thus the period cannot be evaluated sufficiently in the previous model.

3. Key Idea of Solution

In order to solve the problems, this paper proposes a new concept of "workload" which describes fluctuation of the development period and the product size. Next, three factors: developers' experience level, completion rate of products and deadline for activities, are incorporated to take the dynamic influence of human factors into account. (About these factors, subsection 4.1 will describe in detail.)

Generally speaking, an 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 mandays. 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 reasons is that time to communicate among the developers increases as the number of developers increases.

Here, we newly 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, 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 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 Now we 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.

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).

In the proposed model, the workload is assigned to each activity depending on the input products for the activity. For example, for the coding activity, the larger the size of design specifications, the larger workload is assigned. For the debug activity, the more the number of faults contained in input products, the larger workload is assigned. 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 the values of the size or the number of faults in the output product.

4. New Hierarchical Model

The proposed model is a hierarchical model which consists of Project Model and Process Model. Figure 1 shows the outline of the hierarchical model.

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

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.

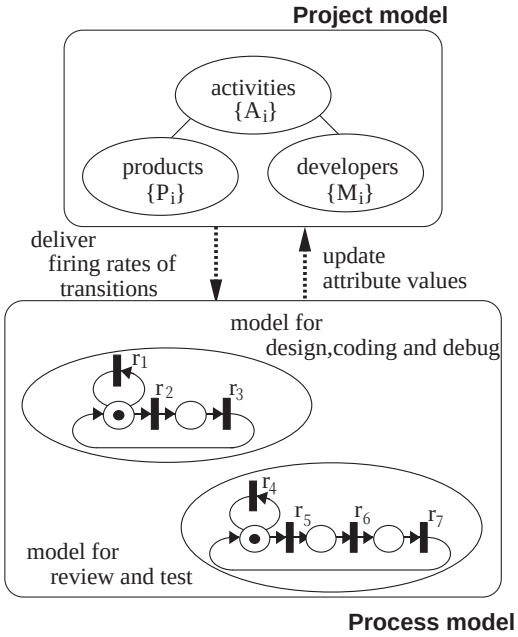


Figure 1: Outline of hierarchical model

4.1 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 condition*, *input/output product*, *workforce*, *deadline* and *workload*. (1) *Type* shows the type of the activity, such as design, coding, review, test, debug and so on. (2) *Entry condition* and (3) *exit condition* specify conditions for beginning and ending the activity,

Table 1: Project template

Attributes of activity A_i
type
entry condition
exit condition
input product
output product
workforce
deadline
workload
Attributes of product P_i
size
number of faults
completion rate
Attribute of developer M_i
experience level

respectively. (4) *Input product* describes a set of products given to the activity as the input products and the degree to which each of input products affects the determination of workload assigned to the activity as tuples. (5) *Output product* describes a set of output products developed in the activity and the weight assigned to each of the products. The variations of product size and number of faults are distributed to the products according to each weight. Thus, the sum of each weight is required to be one. (6) *Workforce* specifies the developers who engage in the activity and the ratio of time in which each of developers can engage in the activity to his or her business hours as tuples. (7) *Deadline* represents the appointed date for the completion of the activity fixed on the development plan. (8) *Workload* represents the workload assigned to the activity and consumed amount of it as a tuple.

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 extent to which omissions or obscurities are excluded from the product as a rate.

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 level. They are quantified as discrete values 1, 2 and 3, respectively.

Example 2 Table 2 shows an example of Project model description. This project is composed of five activities $(A_1, \cdots, A_5)$, seven products and three developers. Note that *workloads* of activities $A_1, \cdots, A_5$ and attributes of all products except for initial input product P_0 are determined during the execution of the model. Thus, they are not yet specified in Table 2.

Parallel execution of several activities can easily be defined by utilizing both entry condition of activity and completion rate of product. For example of the parallel execution, refer to [6].

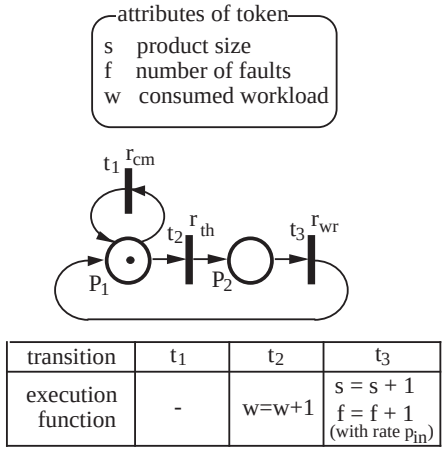
4.2 Activity model

Activity model is prepared for each type of activities such as design, coding, review, test and debug. The descriptions of Activity models are given using extended GSPN. Figure 2 shows the description of the design activity. In the extended GSPN, a token has three attributes: product size s, the 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.

Each transition corresponds to the developers' behavior such as thinking, writing and communicating or event which occurs during an activity. Places correspond to waiting states for occurrences of behaviors or events.

In addition, each transition has a function (called execution function) to be evaluated on its firing. The execution of the functions updates the attribute values of the token.

Example 3 Figure 2 shows a description of the design activity. Here, we consider three kinds of developers' behaviors in the design activity: communicating among developers, thinking needed for problems solution and writing for putting down the solution in documents. Transitions t_1, t_2 and t_3 in Figure 2 correspond to communicating, thinking and writing, and are given the firing rates r_{cm}, r_{th} and r_{wr}, respectively. The firing rate r_{cm} of transition t_1 means that the average firing delay of transition t_1 is $1/r_{cm}$.



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

Figure 2: Design activity model

4.3 Firing rate of transitions

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 developers engaged in the activity, L is the 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}
$r_{cm} = f_{cm}(M, L, R)$

(2) Thinking rate r_{th}
$r_{th} = f_{th}(M, L)$

(3) Writing rate r_{wr}
$r_{wr} = f_{wr}(M, L)$

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

(5) Reading rate r_{rd}
$r_{rd} = f_{rd}(M, L)$

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

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

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

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

Here, we will explain the functions used in the design and coding activity. Functions f_{cm}, f_{th} and f_{wr} make it possible to dynamically determine the frequency of communications or the difficulty in thinking and writing according to the number of developers and the experience levels of developers or completion rates of input products.

Example 4 In the design activity, r_{cm} is concretely formulated as follows.

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

Next, r_{th} and r_{wr} are concretely formulated as follows.

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

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

On the other hand, $r_{cm}, r_{pr}, r_{rd}, r_{dt}$ and r_{md} are used in the review activity, $r_{cm}, r_{pr}, r_{ps}, r_{dt}$ and r_{wr} are used in the test activity and r_{cm}, r_{lc} and r_{md} are used in the debug activity.

4.4 Effects of transition firing

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

(10) Fault injection rate p_{in}
$p_{in} = f_{in}(M, L, R, D)$

Example 5 In the design activity, p_{in} is concretely formulated as follows.

$$p_{in} = K_{in} \times \frac{M}{LRD} \times M$$

By using this function, it is possible to take account of dynamic influence on the fault injection rate caused by the deadline of the activity or developers' experience levels.

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

Example 6 In the design activity model depicted in Figure 2, for example, transitions t_1 and t_2 which represent communicating and thinking, respectively, can fire when a token exists in place P_1. If communicating transition t_1 fires, it has no effect on the attributes values, and the token returns to place P_1 and just time corresponding to the firing delay elapses.

If transition t_2 fires, by evaluating its execution function, consumed workload w increases by one, and the token moves to place P_2. When the token exist in place P_2, only transition t_3 which represents writing behavior is enable. If transition t_3 fires, product size s increase by one, and the number of faults f could increase according to fault injection rate p_{in}. After the firing of t_3, the token moves back to place P_1.

5. Execution of Model

The development process specified by the hierarchical model is carried out by repeating the interactions between Project model and Activity models. In this section, we explain how this hierarchical model is carried out.

Example 7 We apply the proposed model to the software development process shown in Figure 3. Here, we explain the execution of the model by using Activity A_7. Activity A_7 is the module design activity. Activity A_7 is executed until it consumes all the assigned workload. Now, assume that the execution of A_7 has completed on the 59th day. The assigned and consumed workload of A_7 are 172 and 117, respectively. The input and output product of A_7 are P_4 and P_7, respectively.

On the 60th day, the following three steps are carried out:

Step 1. Based on the the values of the attributes of the activity, products and developers, Project model computes the firing rates of transitions of Activity models. In this case, project model sets the values of K_{cm}, K_{th} and K_{wr} as $0.1, 0.2$ and 0.2, respectively. In Figure 4(a), the firing rate of A_7 is calculated as follows:

$$r_{cm} = K_{cm} \times \frac{1^2}{1 \times 1} = 0.10$$

$$r_{th} = K_{th} \times 1 = 0.20$$

$$r_{wr} = K_{wr} \times 1 = 0.20$$

Step 2. Project model delivers the firing rates to the corresponding Activity model (GSPN model), and then Process model executes the model. In Figure 4(b), a design activity model is executed by Process model with the specified firing rate r_{cm}, r_{th} and r_{wr}. The GSPN computes the current size of product$(= 3)$ and consumed workload$(= 8)$.

Step 3. Process model returns the execution results to Project model. Then Project model updates relevant attributes of activities and products based on the returned results. In Figure 4(c), at the end of the 60th day, consumed workload of A_7 becomes 125, and the size of P_7 becomes 34.

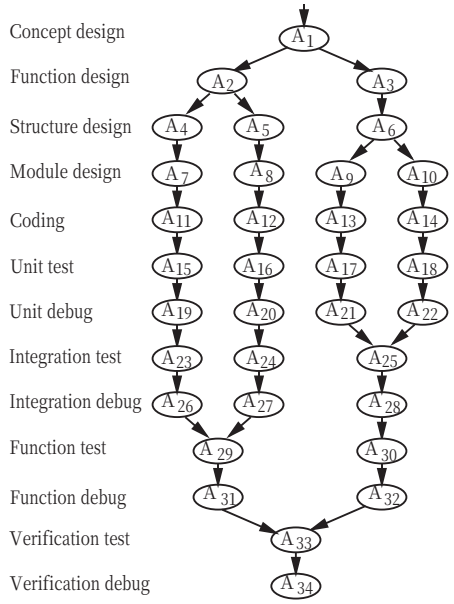


Figure 3: Development process flow

6. Conclusion

This paper has proposed a new hierarchical 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 by taking the influence of human factor into account, it is possible to evaluate the dynamic aspect of software project.

Experimental evaluations of the model by applying real software development data are currently under investigation[5].

Acknowledgement

The authors would like to thank Professor Ichiro Suzuki of the University of Wisconsin at Milwaukee for his careful reading of the earlier version of this paper and helpful comments.

References

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

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

[3] Furusawa K. et al.:"Modeling and quantitative evaluation of software process based on a Generalized Stochastic Petri-net", Proc. of 15th Software Reliability Symposium, pp.99-104 (1994)(in Japanese).

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

[5] Hirayama Y. et al.:"Hierarchical project management model based on quantitative evaluation of software process", Technical Report of IEICE. FTS95-74 (1995-12)(in Japanese).

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

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

[8] Marsan A., Conte G. and Balbo G.: "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).

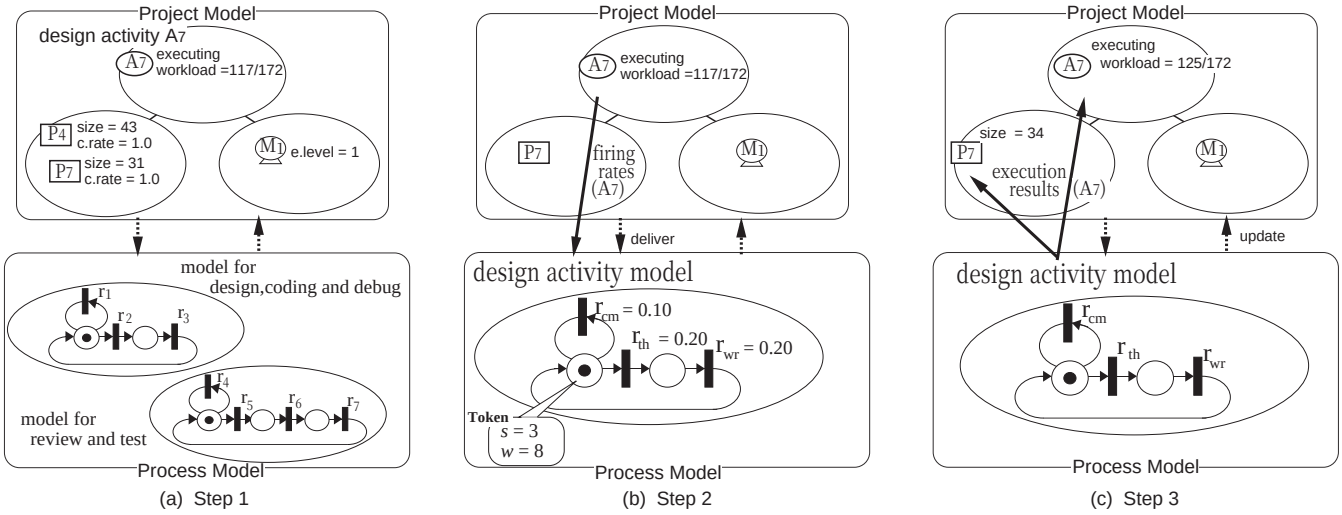


Figure 4: Execution on the 60th day

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

[10] Raffo D. M.: "Evaluating the impact of process improvements quantitatively using process modeling", Proc. of CASCON93, Vol.1, pp.290-313 (1993).

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

[12] Tvedt J. D. and Collofello J. S.: "Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modeling", Proc. of COMPSAC95, pp.318-325 (1995).

Table 2: Example of project description

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

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

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

A_4	
type	FT
entry c.	(A_2,done), (A_3,done)
exit c.	(A_4,consumed)
input p.	(P_3, 2.0), (P_4, 0.2), (P_5, 0.25)
output p.	(P_6, 1.0)
workforce	(M_2, 1.0)
deadline	60

A_5	
type	FTD
entry c.	(A_4,running)
exit c.	(A_4,done),(A_5,consumed)
input p.	(P_6, 2.4)
output p.	(P_4, 0.45), (P_5, 0.55)
workforce	(M_1, 1.0), (M_3, 1.0)
deadline	65

P_0	
size	8
fault	0
c. rate	1.0

M_1	
e. level	3

M_2	
e. level	2

M_3	
e. level	1