Journal of the Korea Academia-Industrial cooperation Society
Vol. 15, No. 9 pp. 5652-5659, 2014

http://dx.doi.org/10.5762/KAIS.2014.15.9.5652
ISSN 1975-4701 / eISSN 2288-4688

Performance Evaluation of Multi-Module Software System with Imperfect Debugging and Module Dependency

U-Jung Kim[1], Chong Hyung Lee[2*]

[1]**College of General Education, Hallym University**

[2]**Department of Hospital Management, Konyang University**

모듈의존성을 갖는 불완전수리 다항모듈 소프트웨어의 성능평가에 관한 연구

김유정[1], 이종형[2*]

[1]한림대학교 기초교육대학, [2]건양대학교 병원경영학과

Abstract The purpose of this study was to introduce a software task processing evaluation model that considers the following situations: i) a software system is integratedly composed of several number of modules, ii) each modules has its corresponding module task, iii) all module tasks are tested simultaneously, and iv) the processing times of the module tasks are mutually dependent. The software task completion probability with the module dependency was derived using the joint distribution function of Farlie [11]. The results showed that the task completion probability of software increases with increasing module dependency parameter.

요 약 소프트웨어를 구성하는 모듈들은 각 모듈에 주어지는 업무들이 동시에 처리될 수 있도록 멀티태스킹이 가능하도록 개발되며, 또한 처리중인 업무들은 완전처리된 업무들과 처리중 모듈고장으로 완전처리 되지 않는 불완전 처리업무로 세분화한다. 이러한 경우 여러 모듈에 동시에 업무가 주어졌을 때, Farlie [11]의 결합확률분포를 기반으로 모듈간의 의존성을 고려하여 업무의 완전처리확률을 평가할 수 있는 모형을 제안하며, 이를 통하여 모듈의존성 모수 값이 커질수록 소프트웨어에 주어진 업무의 완전처리확률은 점점 커짐을 보이고자 한다.

Key Words : Module dependency, Module task, Multi-module software, Software task, Task completion probability

1. Introduction

Many researches for software performance evaluation have proposed software availability models based on the Markov process which can represent software debugging times as software failures occur. Shooman and Trivedi [1] propose a software availability model assuming that software failures are perfectly debugged. Tokuno and Yamada [2] derive an availability model with two kinds of restoration actions: When a software failure occurs, the restoration action with the debugging activity is performed with probability p, and the restoration action without the debugging activity is performed with probability $1-p$.

Lee and Park [3] have proposed a Markovian imperfect debugging model for which the software failure is caused by two types of faults, one which is easily detected and the other which is difficult to detect. Tokuno and Yamada [4] consider a model for the multi-task processing system and provides a task

*Corresponding Author : Chong Hyung Lee(Konyang Univ.)
Tel: +82-42-600-6525 email: chlee@konyang.ac.kr
Received June 5, 2014 Revised (1st July 28, 2014, 2nd August 6, 2014) Accepted September 11, 2014

completion probability, which is the probability that the process of a task arrived up to a time period is completed. But, these researches have considered the one-module software whereas the software used in real fields consist of several modules to execute complex demands.

The structure-based software evaluation considers a software as a collection of several modules, and module coupling and module dependency represent the degree of interaction among multiple modules. Gokhale and Lyu [5] develope the simulation procedures to assess the impact of individual module on the reliability of a structure-based software. Recently, Yu et al [6] analyze the difference between module coupling and module dependency and introduces the metrics to quantify both notions regarding the module. Melo et al [7] present a dependability model based on stochastic Petri nets for probabilistic evaluation of risks regarding the turnover of team members and requirement implementation in software development projects. Also, Pitakrat et al [8] propose an architectural model which captures relevant properties of hardware and software components as well as dependencies among them and analyze impacts of architectural system changes for proactive failure management.

Lee et al [9] suggest the concept of module dependency that the processing times of module tasks in a software task are mutually dependent and extends the task processing model of Tokuno and Yamada [4] to the model of a structure-based software with perfect debugging. The result shows that the task completion probability with positive(negative) module dependency is larger(smaller) than the task completion probability without module dependency when the number of modules within software is even. But, the result is reversed when the number of modules within the software is odd. Therefore, it is difficult to use the result of Lee et al [9] when software developers newly upgrade the software by adding a new module with the module dependency that is the same as that of the existing software.

In this paper, we propose a task processing evaluation model for the software system with module dependency and imperfect debugging. Throughout this model, we can provide the consistent result that regardless of whether the number of modules is even or odd, the task completion probability with positive(negative) module dependency is always larger(smaller) than the completion probability without module dependency.

2. Notations and Assumptions

2.1 Notations

r	number of modules in a software
N	number of initial faults
i	number of removed faults
p	probability of perfect debugging
j	fault type, $j=1,2$.
α	module dependency parameter, $-1 \le \alpha \le 1$
$p_{r,\alpha,p}(t)$	probability that the process of a software task is completed

2.2 Assumptions

1. Software testing period is classified as two periods, module testing period and multi-module software testing period.
2. During a module testing period, each of modules is separately developed and tests given module tasks repeatedly until a required performance level is satisfied. The modules satisfying the level are integrated as a multi-module software.
3. During a multi-module software testing period, each of modules has its corresponding module task, and all the module tasks are tested simultaneously. A software task, a set of all the module tasks tested simultaneously, is completed if all module tasks are successfully executed.
4. The number of software tasks that a software system can process are sufficiently large and follows a homogeneous Poisson process with λ.
5. The processing times of module tasks in a

software task are assumed to be dependent.

6. When a software failure occurs, a processed software task is cancelled and a software system starts to be restored. After the debugging is complete, software system can operate and process another software task.

3. Imperfect debugging models

We briefly describe the LP model in subsection 3.1 and extend LP model to incorporate a number of modules in a software in subsection 3.2.

3.1 LP model

Consider a stochastic process $\{X(t), t \geq 0\}$ which represents both the total number of removed faults up to time t and the state of software at a time t, which is classified as working or nonworking during its testing and operation period. Nonworking state can be further classified into two types. One type is caused by a fault that is easily detected and the other is caused by a fault that is difficult to detect. The former type is referred to as fault type 1 and the latter is referred to as fault type 2. Debugging process starts immediately when a failure occurs and the perfect debugging with probability p removes exactly one fault for either fault type. When an imperfect debugging for a fault of type 1(2) is performed, the fault of type 1(2) remains the same type. The probability that two or more software failures occur simultaneously is negligible. Then, the state of software is defined by $X(t) = (x_1(t), x_2(t))$, where $x_1(t)$ is a total number of faults removed during a time interval $(0,t]$ and at time t, $x_2(t)$ has 0 if working, 1(2) if nonworking by fault type 1(2). Let $T_{i,1}$ and $T_{i,2}$ be the random variables of the waiting times elapsed for the occurrence of fault types 1 and 2, respectively, for the software system which is just returned to the working state after removing the ith fault. We assume that $T_{i,1}$ and $T_{i,2}$ follow the exponential distributions with means, $1/\mu_{i,1}$ and $1/\mu_{i,2}$, respectively, and they are mutually independent. Let $T_{i,j}$, $j = 1,2$, be the random variable representing the lengths of time needed to remove a fault of type j from the software, which had i faults removed previously. We assume that they follow the exponential distributions with means of $1/\theta_{i,j}$. For each j, $\mu_{i,j}$ and $\theta_{i,j}$ are assumed to be the decreasing and increasing functions of i, the number of removed faults, respectively. Let $F_{A,B}(t)$ be one step transition probability that $\{X(t), t \geq 0\}$ in state A will be in state B after time t and $F_{A,B,p}(t)$ be the $F_{A,B}(t)$ that represents a perfect debugging with p. Then, the probabilities are obtained as

$$F_{(i,0),(i,j)}(t) = \frac{\mu_{i,j}(1-\exp(-(\mu_{i,1}+\mu_{i,2})t))}{\mu_{i,1}+\mu_{i,2}},$$
$$F_{(i,j),(i,0),p}(t) = q(1-\exp(-\theta_{i,j}t)),$$
$$F_{(i,j),(i+1,0),p}(t) = p(1-\exp(-\theta_{i,j}t))$$

where $F_{(i,j),(i,0),p}(t)$ represents the imperfect transition probability with $1-p$.

Let $G_{(i,0),(n,0),p}(t)$ be the distribution function of the first passage time of the software with i faults already removed until the number of faults removed reaches n, where $i < n$. The distribution function can be expressed as

$$G_{(i,0),(n,0)}(t) \equiv \Pr\{T_{(i,0),(n,0)} \leq t\}$$
$$= \sum_{j=1}^{2} F_{(i,0),(i,j)} * F_{(i,j),(i,0)} * G_{(i,0),(n,0)}(t)$$
$$+ \sum_{j=1}^{2} F_{(i,0),(i,j)} * F_{(i,j),(i+1,0)} * G_{(i+1,0),(n,0)}(t),$$

where $i = 0, 1, \cdots, n-1$, $n = 1, 2, \cdots, N$ and $*$ symbolizes the Stieltjes convolution. Also, the working probability that the software is in state $(n,0)$ at time t on condition that the software was in state $(i,0)$ at time 0 can be obtained as

$$p_{(i,0),(n,0),p}(t) = G_{(i,0),(n,0),p} * p_{(n,0),(n,0),p}(t),$$

where $i,n = 0,1,...,N,\ i \le n$ and

$$p_{(n,0),(n,0),p}(t) = \exp(-(\mu_{n,1}+\mu_{n,2})t) + \sum_{j=1}^{2} F_{(n,0),(n,j)} * F_{(n,j),(n,0),p} * p_{(n,0),(n,0),p}(t),$$

for $n = 0,1,\cdots,N-1$. As $n = N$, $p_{(N,0),(N,0),p}(t) = 1$. Observe that $p_{(0,0),(N,0),p}(t)$ is simply equal to $G_{(0,0),(N,0),p}(t)$ for all $t \ge 0$ and thus, it is obvious that $p_{(0,0),(N,0),p}(t) \to 1$ as $t \to \infty$. More discussions for $G_{(i,0),(n,0),p}(t)$ and $p_{(i,0),(n,0),p}(t)$ can be found in [4].

3.2 Imperfect debugging model of a multi-module software

Let k represent kth module in a multi- module software system and M_k be the total number of faults removed from the kth module during the module testing period. Also, let $T_{k,i,j}$, $k = 1,2,\cdots,r$, be the random variable representing the waiting time to the failure of kth module after removing i faults and follow the exponential distributions with means of $1/\mu_{k,i,j}$. Then, the last failure rate of kth modules, $\mu_{k,M_{k,j}}$, becomes the initial failure rate of kth module in a software when a software is integrated with the modules. We consider that the waiting time to the first failure of the software is equal to the the minimum of the first failure times of all modules. Thus, the waiting time to the first software failure with a fault type j follows an exponential distribution with a mean of $1/\sum_{k=1}^{r}\mu_{k,M_{k,j}}$ under the assumption that the waiting time to the first type j failure of the kth module in a software follow the distribution of $T_{k,M_{k,j}}$.

Let s_j be the decreasing amount of failure rate of software whenever a fault is removed from software, and we assume that the failure rate of software is geometrically decreased by as the number of fault removals is increased[10]. Then, if n is the number of faults removed from the software, the type j failure rate of the software, denoted by $\mu_{n,j}$, has the form of

$$\mu_{n,j} = \sum_{k=1}^{r} \mu_{k,M_{k,j}} \times s_j^n \qquad (1)$$

for $0 < s_j < 1$ and $j = 1,2$.

The imperfect debugging model of multi-module software can be obtained when the rate in equation (1) is considered as the type j failure rate of LP model and the same procedures for obtaining $G_{(i,0),(n,0),p}(t)$ and $p_{(i,0),(n,0),p}(t)$ is applied.

4. Task processing evaluation model

Let $\{N(t), t \ge 0\}$ be the random variable representing the number of tasks arriving at the integrated software system up to time t, and let $\{Z(t), t \ge 0\}$ denote the cumulative number of tasks which can be completed out of $N(t)$ tasks. Then, the distribution function of $Z(t)$ can be written as

$$\Pr\{Z(t) = z\} = \sum_{l=0}^{\infty} \Pr\{Z(t) = z \mid N(t) = l\} \times \Pr\{N(t) = l\}. \qquad (2)$$

The probability that z out of l tasks is completed by the kth module can be calculated as

$$\Pr\{Z(t) = z \mid N(t) = l\} = \binom{l}{z} [p_{r,\alpha,p}(t)]^z [1 - p_{r,\alpha,p}(t)]^{l-z} \qquad (3)$$

where α is the module dependency parameter, $-1 \le \alpha \le 1$, p is a perfect debugging probability, and $p_{r,\alpha,p}(t)$ is the probability that the process of a task is completed by the software system with r modules.

To incorporate the concept of dependency among the module processing times, we adopt the following type of dependency discussed in [11]. Let Y_k be the length of processing time for a module task by kth module and $F_{Y_k}(t)$ be the distribution function of Y_k. Also, let Y_u be the length of processing time for the task by the software system. Then, Y_u is equal to the maximum

of $Y_1, Y_2, \cdots, Y_r$. The joint probability density function(pdf) of $Y_1, Y_2, \cdots, Y_r$ is assumed to be as

$$f(y_1, y_2, \cdots, y_r; r, \alpha)$$
$$= \prod_{k=1}^{r} f_{Y_k}(y_k)\{1+\alpha \prod_{k=1}^{r}[1-2F_{Y_k}(y_k)]\}, \quad (4)$$

where $0 \le y_k < \infty$ and $f_{Y_k}(y_k) = dF_{Y_k}(y_k)/dy_k$. When $\alpha = 0$, the joint pdf indicates that the processing times of module tasks by r modules are mutually independent. Based on the joint pdf of (3), the pdf of Y_u can be expressed as

$$f_{Y_u}(y; r, \alpha) = \sum_{k=1}^{r}[f_{Y_k}(y) \prod_{\substack{j=1 \\ j \ne k}}^{r} F_{Y_j}(y)]$$
$$+\alpha \sum_{k=1}^{r}[f_{Y_k}(y)(1-2F_k(y)) \prod_{\substack{j=1 \\ j \ne k}}^{r}[F_{Y_j}(y)(1-F_{Y_j}(y))]].$$

Let X_n denote the elapsed time of the software system between the nth and the $(n+1)$th fault. The probability that the process of an arbitrary task is completed on condition that $\{X(t) = (n,0)\}$ can be calculated as

$$\beta_{n,r,\alpha} = \Pr\{ Y_u < X_n \mid X(t) = (n,0)\}$$
$$= \Pr\{ Y_u < \min(T_{n,1}, T_{n,2}) \mid X(t) = (n,0)\}$$
$$= \int_0^{\infty} f_{Y_u}(y; r, \alpha) \exp(-(\sum_{j=1}^{2}\sum_{k=1}^{r} \mu_{k,M_{k,j}} \cdot s_j^n) y) dy, \quad (5)$$

where $\beta_{n,r,\alpha} = 1$ for $n = N$. Note also that the arrival times of all tasks up to time t for the software system are distributed uniformly over the time interval[12]. It follows that $p_{r,\alpha,p}(t)$ can be written as

$$p_{r,\alpha,p}(t) = \int_0^t \sum_{n=0}^{N} \Pr\{X(y) = (n,0)$$
$$\times \Pr\{ Y_u < X_n \mid X(t) = (n,0)\} \frac{dy}{t}$$
$$= \frac{1}{t} \sum_{n=0}^{N} [\beta_{n,r,\alpha} \int_0^t p_{(0,0),(n,0),p}(y) dy]. \quad (6)$$

Let l be the total number of tasks being arrived for processing by the software. Then, similar to equation (2), the distribution function of $Z(t)$ can be obtained as

$$\Pr\{Z(t) = z\}$$
$$= \sum_{l=0}^{\infty} \binom{l}{z} [p_{r,\alpha,p}(t)]^z [1-p_{r,\alpha,p}(t)]^{l-z} \times \frac{(\lambda t)^l \exp(-\lambda t)}{l!}$$
$$= \exp(-\lambda t \cdot p_{r,\alpha,p}(t)) \frac{[\lambda t \cdot p_{r,\alpha,p}(t)]^z}{z!}.$$

This equation is equivalent to the NHPP with the mean value function, $\lambda \cdot t \cdot p_{r,\alpha,p}(t)$. Therefore, the expected number of tasks which can be completed out of the tasks arriving up to time t can be obtained as

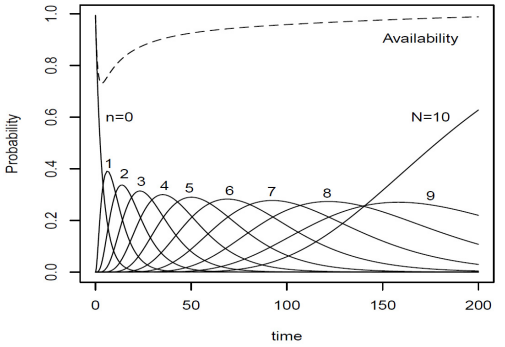
$$E[Z(t)] = \lambda \sum_{n=0}^{N} [\beta_{n,r,\alpha} \int_0^t p_{(0,0),(n,0),p}(y) dy].$$

5. Numerical examples

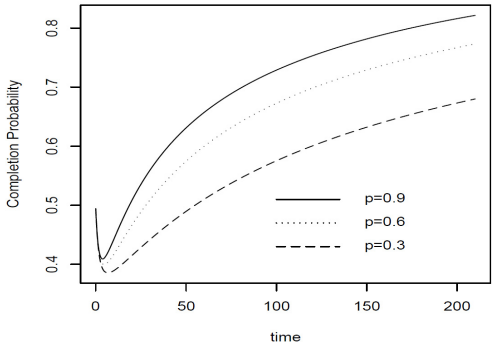
In this section, we compare the patterns of working probability, availability and completion probability of software for various choice of p, r and α. When the ith perfect debugging is completed, we assume that $\theta_{i,j}$ is geometrically increasing in i and the debugging time to remove the fault of type j decreases as the number of previous fault removals gets larger. Under this assumption, we choose $\theta_{i,j}$, as $\theta_{i,j} = \theta_{0,j}[1+[1-\exp(-\sqrt{i})]l_j]$ where $\theta_{0,j}$, $l_j > 0$. Here, l_j is a learning factor which affects the probability of perfect debugging.

As a distribution of the processing time for each software task, we consider a gamma distribution with shape parameter of 2 and scale parameter of 0.5. That is, the pdf of Y_k is assumed to be as $f(y_k) = 0.5^2 y_k \exp(-0.5 y_k)$ for $k = 1, 2, \cdots, r$. Also, we suppose that the fault of type 2 occurs less frequently than the fault of type 1, and the mean debugging time is shorter for the fault of type 1 than for the fault of type 2. Thus, we set $N = 10$, $\mu_{0,1} = 0.2$, $\mu_{0,2} = 0.1$, $\theta_{0,1} = 0.8$, $\theta_{0,2} = 0.5$, $l_1 = 0.2$, $l_2 = 0.1$, $s_1 = 0.8$, $s_2 = 0.5$, $\lambda = 0.99$, and $\alpha = 0.9$.

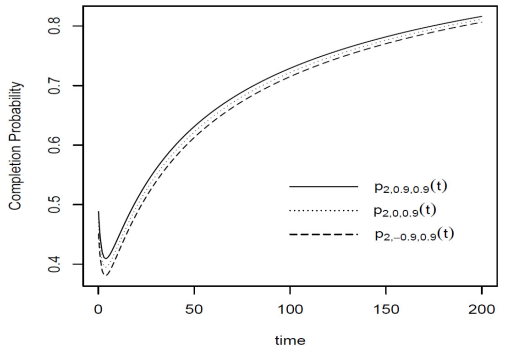
Fig. 1 shows the working probability and the availability of the software, $p_{(0,0),(n,0),p}(t)$ and $A_p(t)$, for various n's with $p = 0.9$ and $r = 2$. The working probability of the software during the initial testing period which needs to remove the greater number of faults is smaller. On the other hand, the probability of



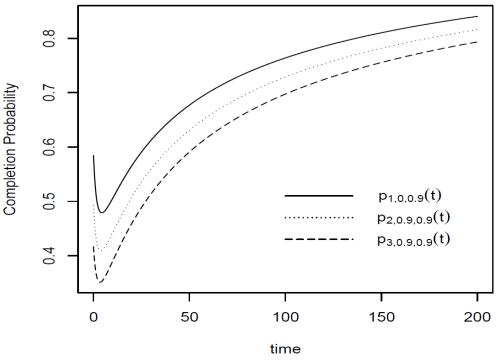
[Fig. 1] Working probability and availability of multi-module software, $p_{(0,0),(n,0),0.9}(t)$ and $A_{0.9}(t)$, for n's with $r=2$.



[Fig. 2] Completion probability of multi-module software, $p_{2,0.9,p}(t)$, for various p's.



[Fig. 3] Completion probability of multi-module software, $p_{2,\alpha,0.9}(t)$, for various α's.



[Fig. 4] Completion probability of multi-module software, $p_{r,0.9,0.9}(t)$, for various r's.

the software during the initial testing period which needs to remove the smaller number of faults is smaller. Also, software availability, defined as $A_p(t)=\sum_{n=0}^{N} p_{(0,0),(n,0),p}(t)$, decreases initially and then increases to 1 monotonically as t increases. This means the fact that the availability tends to decrease fast initially because the software failures may occur more frequently during the initial testing period. However, the perfect debugging is performed for the software failure detected in the early stage, the availability gradually increase later.

Fig. 2 considers the completion probability of a software system for $p=0.9, 0.6, 0.3$ when $r=2$ and $\alpha=0.9$. It shows that the completion probability increases as a debugging probability, p, increases. This is the reason that the larger p comes to have the shorter debugging period, and the longer operation period.

Fig. 3 illustrates the behaviors of the completion probability for $\alpha=0.9, 0, -0.9$ when $r=2$ and $p=0.9$. It is shown that the completion probability decreases for a initial period and then increases monotonically after the period. Also, the completion probability with $\alpha=0.9(\alpha=0)$ is higher than the completion probability with $\alpha=0(\alpha=-0.9)$. That is, task completion probability increases monotonically as α increases.

Fig. 4 compares the completion probabilities of one-module software and multi-module softwares with $\alpha=0.9$. It is shown that the completion probability decreases as the number of modules increases.

Table 1 considers the completion probability of a software task for $\alpha=0.9, 0.5, 0, -0.5, -0.9$ when

[Table 1] Completion probability of multi-module software, $p_{r,\alpha,0.9}(t)$, for various r's and α's.

		Software Testing Period (t)							
r	α	0+	1	3	5	30	100	300	700
2	0.9	0.4943	0.4437	0.4109	0.4114	0.5594	0.7288	0.8574	0.9096
	0.5	0.4858	0.4362	0.4043	0.4052	0.5546	0.7257	0.8557	0.9083
	0	0.4752	0.4268	0.3961	0.3975	0.5487	0.7218	0.8535	0.9068
	-0.5	0.4646	0.4173	0.3878	0.3898	0.5428	0.718	0.8513	0.9052
	-0.9	0.4561	0.4098	0.3812	0.3836	0.5381	0.7149	0.8495	0.904
3	0.9	0.4186	0.3764	0.3516	0.3554	0.5139	0.6974	0.8387	0.8959
	0.5	0.4169	0.3749	0.3502	0.3541	0.5129	0.6967	0.8384	0.8956
	0	0.4147	0.3729	0.3485	0.3525	0.5117	0.6959	0.8379	0.8953
	-0.5	0.4125	0.3709	0.3468	0.3509	0.5104	0.6951	0.8375	0.895
	-0.9	0.4107	0.3694	0.3454	0.3496	0.5095	0.6945	0.8371	0.8947

the number of modules in software system has 2 and 3. Lee et al [9] presents the result that the completion probability increases (decreases) as α increases from -0.9 to $+0.9$ if r is even(odd), regardless of the value of t. However, throughout Table 1, we show the consistent result that the task completion probability with a positive α is always higher than the probability with a negative α in all r's.

6. Conclusion

We consider the multi-module software with module dependency and suggest a software task processing evaluation model which derives the availability of the software and the software task completion probability with module dependency. As the results of this paper, it is shown that the task completion probability of a software increases as α increases. Also, task completion probability with a positive α is always higher than the probability with a negative α in all r's.

References

[1] M.L. Shooman and A.K. Trivedi, "A many-state Markov model for computer software performance parameters", IEEE Transactions on reliability, R-25, pp. 66-68, 1976. DOI: http://dx.doi.org/10.1109/TR.1976.5214978

[2] K. Tokuno and S. Yamada, "Markovian software availability measurement based on the number of restoration actions", IEICE Transactions on Fundamentals, E83-A, pp. 835-841, 2000.

[3] C.H. Lee and D.H. Park, "Markovian imperfect software debugging model and its performance", Stochastic Analysis and Applications, 21(4), pp. 849-864, 2003. DOI: http://dx.doi.org/10.1081/SAP-120022866

[4] K. Tokuno and S. Yamada, "Stochastic performance evaluation for multi-task processing system with software availability model", Journal of Quality in Maintenance Engineering, 12, pp. 412-424, 2006. DOI: http://dx.doi.org/10.1108/13552510610705964

[5] S. Gokhale and M.R. Lyu, "A simulation approach to structure-based software reliability analysis", IEEE Transactions on Software Engineering, 31(8), pp. 643-656, 2005. DOI: http://dx.doi.org/10.1109/TSE.2005.86

[6] L. Yu, K. Chen and S. Ramaswamy, "Multiple- parameter coupling metrics for layered component based software", Software Quality Journal, 17, pp. 5-24, 2009. DOI: http://dx.doi.org/10.1007/s11219-008-9052-9

[7] A. Melo, E. Tavares, M. Marinho, E. Sousa, B. Nogueira and P. Maciel, "Development Risk Assessment in Software Projects Using Dependability Models", IEEE 16th International Conference on Computational Science and Engineering, pp. 260-267, 2013. DOI: http://dx.doi.org/10.1109/CSE.2013.49

[8] T. Pitakrat, A.V. Hoorn and L. Grunske, "Increasing Dependability of Component-Based Software Systems by Online Failure Prediction", 2014 European Dependable Computing Conference, pp. 66-69, 2014.

[9] C.H. Lee, Y.H. Kim and D.H. Park, "Evaluation of multi-tasking software system performance with consideration of module dependency", Journal of Software Maintenance and Evolution: Research and Practice, 23(5),

pp. 361-374, 2011.
DOI: http://dx.doi.org/10.1002/smr.514
[10] P.B. Moranda, "Event-altered rate models for general reliability analysis", IEEE Transactions on Reliability, R-28(5), pp. 376-381, 1979.
DOI: http://dx.doi.org/10.1109/TR.1979.5220648
[11] D.J.G. Farlie, "The performance of some correlation coefficients for a general bivariate distribution", Biometrika, 47, pp. 307-323, 1960.
DOI: http://dx.doi.org/10.2307/2333302
[12] S.M. Ross, Introduction to probability models(11th Edition), San Diego: Academic press, 2014.

U-Jung Kim [Regular member]

- Feb. 1994 : Dept. of Statistics Hallym Univ., MS
- Feb. 2005 : Dept. of Statistics Hallym Univ., PhD
- Sep. 2007 ~ current : Hallym Univ., College of General Education, Assistant Professor

<Research Interests>
Information, Design and Culture, General education

Chong Hyung Lee [Regular member]

- Feb. 2001 : Dept. of Statistics Hallym Univ., PhD
- Feb. 2001 ~ Feb. 2002 : SRCCS, Seoul National Univ., Post-Doctor
- Mar. 2002 ~ current : Konyang Univ., Dept. of Hospital Management, Professor

<Research Interests>
Software evaluation, System reliability, Hospital information and management