A Method of Fault Fix Priority Identification for Open Source Project

Hironobu Sone, Yoshinobu Tamura, Shigeru Yamada

Abstract*: Open source software are adopted as embedded systems, server usage because of quick delivery, cost reduction and standardization of systems. Many open source software are developed under the peculiar development style known as bazaar method. According to this method, faults are detected and fixed by developers around the world, and the fixed result will be reflected in the next release. Also, the fix time of faults tends to be shorter as the development of open source software progresses. However, several large-scale open source projects have a problem that faults fixing takes a lot of time because the faults corrector cannot handle many faults reports quickly. In this paper, we aim to identify the fix priority of newly registered faults in the bug tracking system by using random forest, and we make an index to detect the faults that require high fix priority and long fault fixing time when faults are reported in specific version of open source project. The index is derived and identified by using open source project data obtained from bug tracking system. In addition, we try to improve the detection accuracy of the proposed index by learning not only the specific version but also the fault report data of the past version by using random forest considering the characteristic similarities of faults fix among different versions. As a result, the detection accuracy has highly improved comparing with using only specific version data and using logistic regression.

***Keywords*: Open source software, fault identification, random forest, software effort, fault big data**

I. INTRODUCTION

Source codes of open source software are freely available for use, reuse, fix and re-distribution by the users. Many open source software is known for their high performance and reliability although they are free of charge. Also, many IT companies often develop the open source software as commercial use. In particular, open source software is developed in the bazaar method [1], the source code is implemented in public through the Internet, and the design and implementation of software is promoted by an unspecified number of users and developers. The bug tracking system is also known as one of the systems used to develop open source software. Many fault information such as fix status, their details, and fix priorities are registered through the bug tracking system. Although open source software has been actively developed and used in recent years, there are problems in promoting open source projects. There are over 100 faults reported per a day in massive open source projects [2]. Massive open source projects have a large number of fault reports. Then, it is difficult to quickly fix the faults [3,4].

From the above problems, several researchers have been published papers to predict the fault fixing time for open source projects. For example, there are several papers in terms of the prediction whether the fault fixing time will end within a specific time [5,6,7]. However, there are no research paper in terms of the fault identification method considering the change of fault fixing environment in the open source project, e.g. fault fixing priority, number of fault occurrences and number of faults fixing assignees. In this paper, we discuss the prediction methods of faults that should be fixed preferentially in newly reported faults in open source projects by using the random forest and available data obtained from the bug tracking system. In addition, we discuss the improvement of prediction accuracy and use logistic regression analysis for comparison. This paper also shows several numerical examples by using real fault big data obtained from the bug tracking system.

II. RELATED RESEARCH

In massive open source projects, there is a problem that the fault fixing time is prolonging. It is necessary to prevent the fault fixing time from prolonging [8,9], because the open source software is socially important software. There are several previous researches focus on the fault fixing and reporting times by using the bug tracking system. In the software development using a bug tracking system, it is necessary to consider the priority and severity of fault in order to report the fault. However, there is research reports that the high priority faults do not have a significant effect on fault fixing time, while several high severity faults tend to contribute to shortening fault fixing time [3,10].

In this paper, we focus on a new index called "*fix priority*" by considering the fault fixing time and fault severity. We also evaluate newly reported faults associated with the previously fixed faults with *fix priority*.

III. FAULT IDENTIFICATION METHOD CONSIDERING HIGH FIX PRIORITY

The purpose of this paper is to identify the serious fault. This serious fault is more serious than newly reported fault with *fix priority*. Therefore, the fault data for identification is used as the test data.

Revised Manuscript Received on November 15, 2019.
***** Correspondence Author

Hironobu Sone*, Graduate School of Integrative Science and Engineering, Tokyo City University, Tokyo, Japan. Email: g1881827@tcu.ac.jp

Yoshinobu Tamura, Department of Intelligent Systems, Faculty of Knowledge Engineering, Tokyo City University, Tokyo, Japan. Email: tamuray@tcu.ac.jp

Shigeru Yamada, Graduate School of Engineering, Tottori University, Tottori-shi, Japan. Email: yamada@tottori-u.ac.jp

Then, the learning data is the fault data used for comparison. The process of identification of *fix priority* is follows:

1. We make an evaluation index by using fault fix time and fault severity obtained from bug tracking system.
2. We make label "*High*" or "*Low*" to the evaluation index in terms of the value of the evaluation index.
3. We use the random forest to predict *High* and *Low* in test data. The using data for prediction is obtained from bug tracking system.

The details of proposed method are following section.

A. Evaluation Index

We propose the measure as *fix priority* using the fault fixing time and severity of learning and test data as follows:

$$T_i = t_{standard_i} \times severity_i, \quad (1)$$

where i is the fault number, and severity is the fault severity. Also, the i-th fault data in Eq. (1) is the test data, and the others are learning data. In the other words, the fix priority of one data is predicted using i-1 data. In particular, severity is registered when a fault is registered in the bug tracking system, and the range of possible values is 1 to 5. Table I shows the values of the severity changed from qualitative index for deriving T_i in this paper. Also, t is a normalized value of the fault fixing time for i-th fault data. Then, the standardized value of t is given as follows:

$$t_{standard_i} = \frac{t_i - \min(t)}{\max(t) - \min(t)}, \quad (2)$$

where t_i is the fault fixing time for i-th fault. Especially, the maximum and minimum values at $t_{standard\_i}$ are the values used in the i-1 learning data.

Table Ⅰ The scored fault severity.

Severity	Score
Blocker	5
Critical	4
Major	3
Normal	2.5
Minor, Enhancement	2
Trivial	1

B. Labelling with Evaluation Index

We can label the learning and test data in terms of *High* or *Low* based on the value of the evaluation index T_i created with i fault data. In this paper, we assume three patterns in terms of the labelled learning and test data as follows:

1) if $\mu \leq T_i$ then *High* else *Low*
2) if $\mu + \sigma \leq T_i$ then *High* else *Low*
3) if $\mu + 2\sigma \leq T_i$ then *High* else *Low*

where μ is the mean value of i fault data, and σ the standard deviation. In particular, *High* means that the fixing time of fault is long, and its severity is high in the i fault data. Also, we can judge that faults recognized as *High* under condition 3) are serious condition in all of the fixing time and severity, because the criteria for *High* is the most severe of the three condition. Then, we should fix the fault identified as *High* as soon as possible. Then, we predict these labels by using random forest and logistic regression.

C. Prediction Method with Random Forest

In this paper, we use the random forest to predict *High* and *Low* in test data. The random forest is a method of group learning using regression trees [11]. We show the scheme of random forest in Fig.1. The Repeated random sampling (reconstruction extraction) is performed on the data set for the model construction. Then, the multiple regression trees are created from the obtained sample group. The final prediction result is obtained by the majority of the output in each regression tree. In the conventional group learning, all explanatory variables are used in the model construction. Then, the selected randomly explanatory variables are used in the method of random forest.

In this paper, we use two types of learning data and compare their accuracy. One is the case that only the version containing the test data (the current version) is used as learning data. The other case is that both the past and current versions are used as learning data, because the distribution of the value of index T_i is similar to other versions.

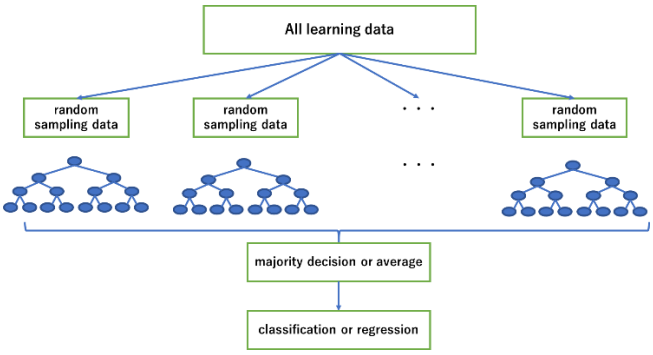

Fig. 1 Scheme of random forest.

In learning case of the past versions, we only include the faults that have been identified to be *High* in the learning data. As shown in Fig. 2, the number of data to be learned equals to the number of learning data in the current version.

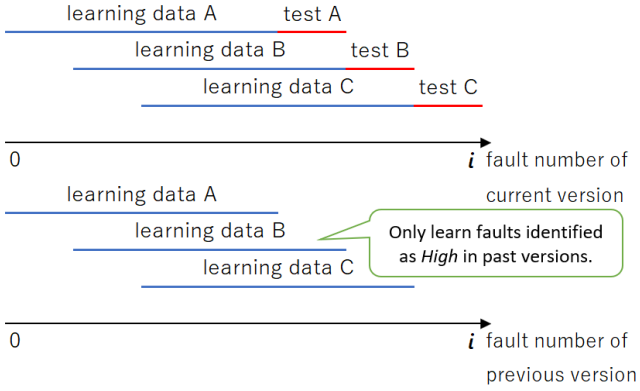

Fig. 2 Learning data with past version.

D. Input Data as the Explanatory Variable

As shown in Table II, we have used the data obtained from the bug tracking system as explanatory variables. In particular, the variables of Component, OS, Reporter, Hardware and Assignee are converted to the appearance rates in order to use as the learning data. Also, the range of possible values for Priority is 1 to 5.

Table Ⅱ Input data as explanatory variable.

Explanatory Variable	Scale	Description
Component	ratio scale	Component name where the fault occurred
Words Count of Summary	ratio scale	Number of words of detailed information in the reported fault
OS	ratio scale	Operating Sytem where the fault occurred
Weekday	nominal scale	Fault report day (weekday, weekend)
Reporter	ratio scale	Developer name who reported the fault
Type	nominal scale	Content of the fault report(fault,Enhancement)
Opened Interval	interval scale	Elapsed days since last fault report
Opened Month	interval scale	Month the fault was reported
Hardware	ratio scale	Hardware on which the fault occurred
Opened (day)	interval scale	Elapsed days since the first fault was reported
Priority	nominal scale	Fix priority for reported fault
Assignee	ratio scale	Developer name who fixed the fault

Table III Accuracy of the prediction model for OpenStack[12] and Eclipse[13] project.

	threshold	OpenStack			Eclipse		
		Recall	Precision	F1 measure	Recall	Precision	F1 measure
A) Random forest using only current version	μ	0.583	0.467	0.519	0.386	0.320	0.350
	$\mu+\sigma$	0.165	0.235	0.194	0.160	0.203	0.179
	$\mu+2\sigma$	0.048	0.065	0.055	0.037	0.062	0.046
B) Logistic regression	μ	0.558	0.430	0.486	0.325	0.288	0.305
	$\mu+\sigma$	0.182	0.149	0.164	0.162	0.167	0.164
	$\mu+2\sigma$	0.193	0.035	0.060	0.070	0.035	0.047
C) Random forest using current and past version	μ	0.724	0.438	0.546	0.593	0.283	0.383
	$\mu+\sigma$	0.456	0.176	0.254	0.294	0.163	0.210
	$\mu+2\sigma$	0.198	0.036	0.060	0.064	0.047	0.054
Improvement rate from A) to C)	μ	24.15%	-6.30%	5.17%	53.80%	-11.50%	9.60%
	$\mu+\sigma$	176.62%	-25.06%	31.11%	84.18%	-19.45%	17.59%
	$\mu+2\sigma$	310.00%	-44.93%	9.31%	72.97%	-24.52%	16.53%
Improvement rate from B) to C)	μ	29.73%	1.83%	12.34%	82.46%	-1.60%	25.51%
	$\mu+\sigma$	150.86%	17.78%	54.85%	81.47%	-1.93%	27.88%
	$\mu+2\sigma$	2.50%	0.47%	0.78%	-8.57%	32.99%	15.49%

IV. NUMERICAL ILLUSTRATIONS

In order to evaluate the performance of the proposed method, we focus on version 17 of OpenStack [12] and versions 4.7 of Eclipse [13] in this paper. Each data set is obtained from a bug tracking system, Bugzilla operated by the Apache Software Foundation. Also, we use 2284 fault report data for each version in OpenStack, and 1800 fault report data for each version in Eclipse. In this paper, 200 learning data are used for preparing one test data. In order to predict the value of T_i after the first 200 fault data, we move the learning data and the test data one by one as shown in Fig. 2.

A. Model Evaluation Criteria

In this paper, we use *Recall*, *Precision*, *and* F_1 *measure* [14] as evaluation criteria. *Recall* means the proportion of correctly predicted faults among the faults identified as *High*. Also, *Precision* is the proportion of correctly predicted faults among the *High* predicted faults. In this paper, we use the F_1 *measure*, which is the harmonic mean of *Recall* and *Precision* as the evaluation criterion. In particular, we can use F_1 *measure* to consider the balance between *Recall* and *Precision*. The F_1 *measure* is defined as

$$F_1\ measure = \frac{2 \times Precision \times Recall}{Precision + Recall}. \quad (3)$$

All three evaluation criteria take the range [0,1]. The prediction accuracy is high if this value becomes large, and the prediction accuracy is low if this value becomes small.

B. Prediction Results

Table III shows the accuracy of the prediction model using random forest and logistic regression for the Eclipse and OpenStack projects. There are two types of learning data in case of using random forest, i.e., the first case is using the current version and the other case is using the current and past versions. Moreover, these results show the mean value in 10 times calculations.

Table III shows that the value of Recall is almost improved by learning both the current version and the past version in both OpenStack and Eclipse. On the other hand, in both cases, the value of Precision is almost becoming worse, but the value of F_1 measure is improved. In conclusion, we have found that it is better to use the past version as well as the current version for learning data.

Also, we list the variable importance of the prediction model using random forest with the highest F1measure and the threshold μ, $\mu+\sigma$, and $\mu+2\sigma$ in Tables IV, V and VI, respectively. Moreover, we list in Tables IV, V, and VI as the variable importance, respectively. Then, we have learned 200

Table IV Variable important threshold μ.

explanetory variable	scale	OpenStack				Eclipse			
		1-	501-	1001-	1501-	1-	501-	1001-	1501-
Component	ratio scale	9.413	11.329	**12.637**	9.389	**8.212**	**10.299**	**14.504**	9.993
Words Count of Summary	ratio scale	10.123	**11.795**	10.474	**14.735**	4.605	9.037	9.858	**11.180**
OS	ratio scale	**17.637**	4.463	7.476	6.498	7.210	8.562	**10.203**	6.304
Weekday	nominal scale	0.155	0.964	1.029	0.598	0.261	0.717	0.976	1.101
Reporter	ratio scale	8.672	**13.379**	8.709	11.196	**8.930**	**9.719**	9.783	**11.872**
Type	nominal scale	0.565	0.026	0.133	3.595	1.198	0.605	0.635	0.568
Opened Interval	interval scale	**15.942**	**18.848**	**17.232**	**11.773**	**13.371**	**14.670**	**14.149**	10.210
Opened Month	interval scale	**15.659**	**16.356**	**16.808**	**11.473**	5.789	5.615	3.541	**15.776**
Hardware	ratio scale	**15.072**	10.083	7.504	8.180	**8.071**	**12.108**	**13.239**	8.838
Opened (day)	interval scale	**30.698**	**32.835**	**38.634**	**29.332**	**47.035**	**25.780**	**35.889**	**33.477**
Priority	nominal scale	5.777	8.373	5.577	7.809	0.056	1.475	0.447	0.400
Assignee	ratio scale	9.413	9.967	**11.500**	**11.292**	5.761	9.187	8.863	**12.229**

Table V Variable important threshold $\mu + \sigma$.

explanetory variable	scale	OpenStack				Eclipse			
		1-	501-	1001-	1501-	1-	501-	1001-	1501-
Component	ratio scale	4.570	7.794	**6.791**	5.539	**4.981**	4.247	**8.239**	**9.897**
Words Count of Summary	ratio scale	4.172	**7.942**	**7.836**	**8.320**	1.141	5.758	5.753	7.296
OS	ratio scale	**10.235**	4.199	3.831	6.455	**5.000**	**6.258**	**9.225**	4.116
Weekday	nominal scale	0.424	0.616	0.829	1.508	0.312	0.417	0.473	1.457
Reporter	ratio scale	4.698	**8.098**	6.082	6.359	2.375	6.025	5.689	**10.477**
Type	nominal scale	0.772	0.011	0.029	1.703	0.392	0.246	0.797	0.120
Opened Interval	interval scale	**9.638**	**11.945**	**11.787**	**8.913**	**7.348**	**11.066**	**10.978**	**8.490**
Opened Month	interval scale	**8.983**	**14.950**	**15.361**	**12.550**	3.544	4.605	2.447	**7.227**
Hardware	ratio scale	**10.246**	7.928	4.247	**7.907**	**5.091**	**8.844**	**12.736**	4.449
Opened (day)	interval scale	**21.965**	**28.540**	**29.372**	**24.311**	**38.016**	**18.756**	**28.154**	**20.589**
Priority	nominal scale	2.725	3.713	3.748	5.195	0.098	1.789	0.798	0.325
Assignee	ratio scale	4.335	5.912	5.737	6.057	2.556	**6.834**	6.059	6.552

Table VI Variable important threshold $\mu + 2\sigma$.

explanetory variable	scale	OpenStack				Eclipse			
		1-	501-	1001-	1501-	1-	501-	1001-	1501-
Component	ratio scale	1.615	2.131	1.906	1.295	**3.736**	2.467	**5.089**	**5.035**
Words Count of Summary	ratio scale	1.317	2.152	**3.503**	1.674	0.915	3.916	3.803	**5.239**
OS	ratio scale	2.583	0.830	1.333	**4.168**	**3.149**	**5.061**	**5.986**	2.351
Weekday	nominal scale	0.086	**0.252**	0.045	0.289	0.104	0.194	0.244	0.547
Reporter	ratio scale	1.652	2.076	**2.782**	1.512	1.455	3.604	3.659	3.804
Type	nominal scale	**2.812**	0.000	0.001	0.152	0.260	0.089	0.023	0.046
Opened Interval	interval scale	**4.290**	**4.118**	**3.369**	**3.361**	**4.755**	**6.743**	**5.370**	**5.333**
Opened Month	interval scale	**4.137**	**5.632**	**8.171**	**9.340**	**2.439**	**5.015**	1.295	**5.048**
Hardware	ratio scale	**3.517**	**3.994**	1.094	**4.021**	1.675	**8.486**	**10.075**	3.380
Opened (day)	interval scale	**7.562**	**10.821**	**12.875**	**12.263**	**27.241**	**13.347**	**13.859**	**15.654**
Priority	nominal scale	1.616	0.979	1.186	0.972	0.103	1.015	0.098	0.089
Assignee	ratio scale	1.399	1.277	1.767	1.802	1.821	4.350	3.238	2.481

each fault from the first fault, the 501st fault, the 1001st fault, and the 1501st fault. In particular, the bold indicate that variable importance is the top five highest in prediction model.

Tables IV, V, and VI show that Opened Interval and Opened (day) have high variable importance in both Open Stack and Eclipse. In other words, we can understand that "time" is greatly influenced in terms of the prediction of T_i. Also, variable "Opened Month" has consistently high variable importance in OpenStack. On the other hand, Eclipse depends on learning data.

V. CONCLUSION

In many open source projects, the bug tracking system is an important system for developing open source software. Open source software has been actively developed and used by a lot of people in recent years, so many faults in open source software have been reported. As a result, the developers sometimes take a long time to fix a fault, and sometimes miss a fix for faults.

In this paper, we aim to identify the fix priority of newly registered faults in the bug tracking system, and we have proposed the fix priority index considering the faults fixing time and severity. Also, we have compared with the most recently fixed faults to predict if the new reported fault's fix priority is high or not. Also, we have found that the transition of the value of fix priority T_i shows the same tendency as different versions for the same open source software. In addition, we have found that it is possible to improve the prediction accuracy by learning not only the version of the fault for prediction but also the past version. From the above, we have considered that the proposed method using past version data is a practical method in order to identify the high fix priority of large open source project.

REFERENCES

1. E. S. Raymond, *The Cathedral and the Bazzar*: *Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly and Associates, Sebastopol, California, 1999.
2. C. Sun, D. Lo, X. Wang, J. Jiang, and S.C. Khoo, "A discriminative model approach for accurate duplicate," *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010, pp. 45-54.
3. M. Nurolahzade, S.M. Nasehi, S.H. Khandkar, and S. Rawal, "The role of patch review in software evolution: an analysis of the mozilla firefox," *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops*, 2009, pp. 9-18.
4. P. Hooimeijer and W. Weimer, "Modeling bug report quality," *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*, 2007, pp. 34-43.
5. P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?," *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 207-210.
6. A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," *Proceedings of the International Conference on Software Engineering*, 2010, pp. 1-10.
7. Akbarinasaji. S, Caglayan. B, and Bener. A, "Predicting Bug-Fixing Time: A Replication Study Using An OSS Project," *Syst*, Softw, vol.136, 2018, pp.173–186.
8. N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," *Proceedings of the 16th International Symposium on Foundations of Software Engineering*, 2008, pp. 308-318.
9. S. Just, R. Premraj, and T. Zimmermann, "Towards the next generation of bug tracking systems," *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, 2008, pp. 82-85.
10. L. Marks, Y. Zou, and A.E. Hassan, "Studying the fix-time for bugs large open source projects," Proceedings of the 7th International Conference on Predictive Models in Software Engineering, 2011.
11. L. Breiman, "Random forests," *in Machine Learning*, Springer, vol.45, 2001, pp. 5-32.
12. The OpenStack Foundation, The OpenStack project, http://www.openstack.org/
13. The Eclipse Foundation, The Eclipse Project, http://www.eclipse.org/
14. J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *Transactions on Information Systems*, vol.22, No. 1, pp. 5-53, 2004.

AUTHORS PROFILE

Hironobu Sone received his B.S.E. from Tokyo City University in 2018. His research interests included project management and open source software at Graduate School Integrative Science and Engineering Systems Information Engineering, Tokyo City University.

Yoshinobu Tamura received the B.S.E., M.S., and Ph.D. degrees from Tottori University in 1998, 2000, and 2003, respectively. From 2003 to 2006, he was a Research Assistant at Tottori University of Environmental Studies. From 2006 to 2009, he was a Lecturer and Associate Professor at Faculty of Applied Information Science of Hiroshima Institute of Technology, Hiroshima, Japan. From 2009 to 2017, he was an Associate Professor at the Graduate School of Sciences and Technology for Innovation, Yamaguchi University, Ube, Japan. Since 2017, he has been working as a Professor at the Department of Industrial & Management Systems Engineering, Faculty of Knowledge Engineering, Tokyo City University, Tokyo, Japan. His research interests include reliability assessment for open source software. He is a regular member of the Institute of Electronics, the Information and Communication Engineers of Japan, the Operations Research Society of Japan, the Society of Project Management of Japan, the Reliability Engineering Association of Japan, and the IEEE. He has authored the book entitled as OSS Reliability Measurement and Assessment (Springer International Publishing, 2016).

Dr. Tamura received the Presentation Award of the Seventh International Conference on Industrial Management in 2004, the IEEE Reliability Society Japan Chapter Awards in 2007, the Research Leadership Award in Area of Reliability from the ICRITO in 2010, and the Best Paper Award of the IEEE International Conference on Industrial Engineering and Engineering Management in 2012.

Shigeru Yamada received the B.S.E., M.S., and Ph.D. degrees from Hiroshima University, Japan, in 1975, 1977, and 1985, respectively. Since 1993, he has been working as a professor at the Department of Social Management Engineering, Graduate School of Engineering, Tottori University, Tottori-shi, Japan. He has published over 500 reviewed technical papers in the area of software reliability engineering, project management, reliability engineering, and quality control. He has authored several books entitled such as Introduction to Software Management Model (Kyoritsu Shuppan, 1993), Software Reliability Models: Fundamentals and Applications (JUSE, Tokyo, 1994), Statistical Quality Control for TQM (Corona Publishing, Tokyo, 1998), Software Reliability: Model, Tool, Management (The Society of Project Management, 2004), Quality-Oriented Software Management (Morikita Shuppan, 2007), Elements of Software Reliability Modeling Approach- (Kyoritsu Shuppan, 2011), Project Management (Kyoritsu Shuppan, 2012), Software Engineering: Fundamentals and Applications (Suurikougaku Publishing, 2013), Software Reliability Modeling: Fundamentals and Applications (Springer-Verlag, 2014), and OSS Reliability Measurement and Assessment (Springer International Publishing, 2016). Dr. Yamada received the Best Author Award from the Information Processing Society of Japan in 1992, the TELECOM System Technology Award from the Telecommunications Advancement Foundation in 1993, the Best Paper Award from the Reliability Engineering Association of Japan in 1999, the International Leadership Award in Reliability Engg. Research from the ICQRIT/SREQOM in 2003, the Best Paper Award at the 2004 International Computer Symposium, the Best Paper Award from the Society of Project Management in 2006, the Leadership Award from the ISSAT in 2007, the Outstanding Paper Award at the IEEE International Conference on Industrial Engineering and Engineering Management (IEEM208) in 2008, the International Leadership and Pioneering Research Award in Software Reliability Engineering from the SREQOM/ICQRIT in 2009, the Exceptional International Leadership and Contribution Award in Software Reliability at the ICRITO'2010, 2011 Best Paper Award from the IEEE Reliability Society Japan Chapter in 2012, the Leadership Award from the ISSAT in 2014, the Project Management Service Award from the SPM in 2014, the Contributions Award for Promoting OR from the ORSJ in 2017, and the Research Award from the ISSAT in 2017. He is a regular member of the IEICE, the Information Processing Society of Japan, the Operations Research Society of Japan, the Reliability Engineering Association of Japan, Japan Industrial Management Association, the Japanese Society for Quality Control, the Society of Project Management, and the IEEE.