J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

Enhancing Model-based Fault Traceability by Using Similarity between Bug and Commit Information

정 동 주[1] 민 경 식[1] 이 정 원[2] 이 병 정[1*]
Dongju Jung Kyeongsic Min Jung-Won Lee Byungjeong Lee

ABSTRACT

As software development technology evolves, the quality of software has increased. But software created through sophisticated technology is still defective. The developer will be aware of the defect through a bug report and the reported defect must be fixed as soon as possible for the software to function correctly. It is important to know which component of the program is related to the reported defect and should be fixed. However, even though the developer understands the developed software, the task of tracing faults is a time-consuming task and requires effort. Therefore, if there is a way for developers to support tracing faults, they could fix defects more quickly. Because fixing defects rapidly is a factor of software reliability, fault traceability is essential and an effective method is needed. Therefore, in this paper, we propose a model-based fault traceability enhancement technique by using bug report and commit information and verify the effectiveness of the proposed technique.

☞ keyword : Fault traceability, Bug Report, Commit Information

1. Introduction

The modern software has high quality through sophisticated software development techniques and processes. However, even though developed through a sophisticated process, many defects are found in most software. Defects are critical to software, and how to manage defects reported to developers after software development is always a major issue. Even if a developer understands the developed software, in the case of large projects, it is difficult to predict in a short time what components of the software defects are associated with and which components to modify when the defect is reported[1]. The procedure of tracing these faults consumes additional human and temporal resources. Therefore, it is necessary to trace the location of the defect-related resources as well as how to correct the defects. For supporting this, in this paper, we propose and verify a enhancing model-based fault traceability technique using bug report and commit information of software VCS. The contribution of this study is as follows.

- We extract keywords from VCS commits descriptions, source code changes, and bug report, and uses them as criteria for selecting commits associated with the reported bug report.
- After finding similar commit information, we use the behavior model to trace resources related to the origin of the fault.

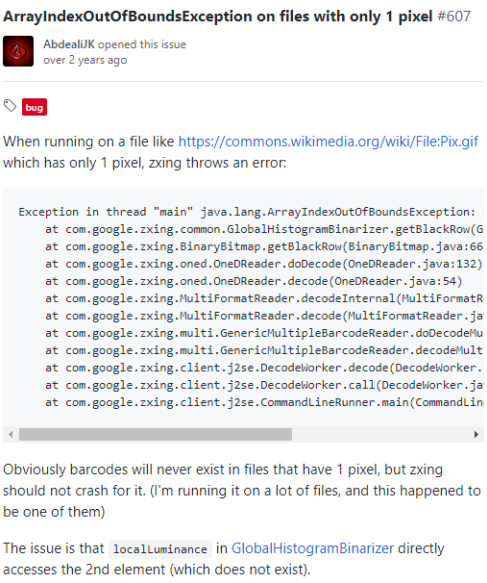
An order of rest of this paper is as follows. We first describe the background knowledge in Chapter 2. Chapter 3 introduces related works, and Chapter 4 describes our proposed technique. And we do case study in Chapter 5, and verify our technique through experiments using open source project in Chapter 6. After that, we discuss our research in Chapter 7. At last, in Chapter 8, we conclude our paper.

1 Department of Computer Science, University of Seoul., Seoul, 02504, Korea.
2 Department of Electrical and Computer Engineering, Ajou University., Suwon, 16499, Korea.
* Corresponding author (bjlee@uos.ac.kr)
[Received 31 December 2018, Reviewed 23 January 2019, Accepted 28 February 2019]
☆ This research was supported by Next-Generation Information Co mputing Development Program (NRF-2014M3C4A7030504) and by Basic Science Research Program (NRF-2017R1A2B4009937) through the National Research Foundation of Korea (NRF) fund ed by the Ministry of Science, ICT, and Future Planning.
☆ A preliminary version of this paper was presented at APIC-IST 2018 and was selected as an outstanding paper.

2. Background Knowledge

2.1 Bug Report

A bug report describes the defect or functional enhancement that the user found during the test phase or using the deployed software [2, 3]. The bug report describes the details of the defect and provides a variety of information so that the developer can figure out the reason of the defect.



(Figure 1) A Bug Report

Figure 1 is an example of a bug report, one of the bug reports from zxing , an open source project. The bug report of github , the repository of git, contains information such as title, bug report content, related images, and date.

2.2 Version Control System

The Version Control System (VCS) is a system for assigning versions of changes of software source code and software output, storing the versions, and controlling it.

When there are changes of the source code and output, the developer commits it and reflects it in VCS. A single commit has information such as a description of the commit and source code changes. Generally, a commit description describes source code changes or modified features to indicate changes and allow for commits to be traced and recovered throughout the project [4]. Typically, a unit of commit is a group of changes with the same purpose, such as bug fixes or functional enhancements. That is, changes contained in a single commit are considered to be related resources.

3. Related Works

There are many existing studies using a variety of methodologies for fault traceability that trace related resources in the event of a software fault. In the previous studies, S. Back et al. [5] used behavior model, commit information of VCS, and web application specified bug report form for web applications developed on the basis of MVC patterns. In the step of tracking the source code from the bug report, [5] used the method of tracing the entry point of the controller according to the user's request through the URI information of the bug report if there is URI information in the bug report. Otherwise, they used the similarity between the bug report and the source code of the controller. And [5] used only source code changes as commit information to improve fault traceability, and considered the source code changes in a single commit as associative resources. C. Youm et al. [6] proposed a methodology to improve fault traceability by utilizing bug report, structured source code change history, and stack trace information based on information retrieval method. L. Moreno et al. [7] determined whether the bug report is related to the source code information such as class name, method name, and argument through the information retrieval method, and proposed a methodology of tracing the source code from the bug report.

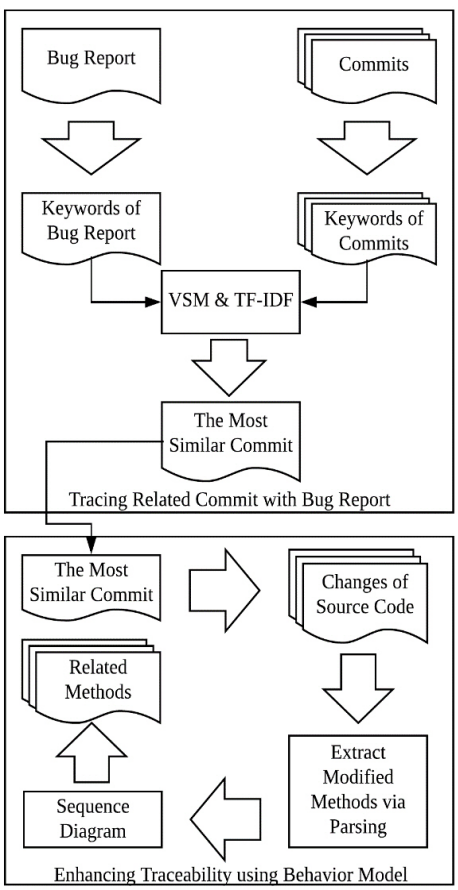
Existing fault traceability studies mainly used information retrieval methods, and some studies utilized behavior model [8-12]. However, since the level of traceability of most of the studies is a source code file level, it is hard to provide detailed resources related to the defect to the developer. And in [5], the trace level is a method level, but it is impossible to apply it to general projects because the proposed technique is restricted to web applications using MVC pattern. But, for useful defect tracing techniques, the trace level should be as

detailed as a method level[13] and general versatility must be ensured so that it can be applied to general projects.

4. Fault Traceability Enhancement Technique

4.1 Overview

The overview of the fault traceability enhancement technique proposed in this paper is shown in Figure 2. The fault traceability technique is largely a two-step process. The first step is finding the most related commit with the reported bug report. gram as a behavior model. A detailed description of each step is given in Sections 4.2 and 4.3.



(Figure 2) An Overview of Enhancing Fault Traceability Technique

4.2 Tracing a Related Commit with the Bug Report

This step is to trace the commit that has the most related commit information with the bug report. This step takes advantage of the nature of VCS's commits. Fist of all, a commit description describes the changes made by the commit. Also, changes contained in a single commit are resources that are related to each other. Therefore, we use VSM(Vector Space Model) and TF-IDF techniques to calculate the similarity between the bug report and commit information in order to find the commit related with the bug report [14]. The detailed steps are as follows.

1. Extract keywords from the bug report and commit information
2. Calculate similarity between bug report and commit information using VSM
3. Extract information of the commit with the highest similarity score

Step 1 removes stopwords from natural language and program code and then extracts keywords. The target of the keyword extraction is the description of the commit and the source code changes. In the case of source code changes, we extended the above and below three lines of source code based on the added, changed, and deleted codes. This is a heuristic method, in which some source codes in the modified method are subjected to keyword extraction so that it is possible to extract the more accurate keyword. In this study, we used the stopword module of Node.js for Step 1. In addition, in the source code changes of commits, we improved the accuracy of Step 3 by removing some reserved words such as public, void, string, and int as well as stopword.

Step 2 is a step of calculating the similarity between the bug report and the commit information. VSM is used to calculate the similarity between the commit information and the bug report, and the similarity expression can be expressed as follows.

$$\text{similarity}(d_i, \text{q}) = \frac{\overrightarrow{V_{d_i}} \cdot \overrightarrow{V_q}}{\|\overrightarrow{V_{d_i}}\| \|\overrightarrow{V_q}\|}$$

$\overrightarrow{V_{d_i}}$ *is vector of term weights of document* d_i

$\overrightarrow{V_q}$ is *vector of term weights of query* q

In the similarity formula, q and d refer to the query and i-th document in the D, respectively. And $\overrightarrow{V_{d_i}}$ is the term weight vector of the i-th document, and $\overrightarrow{V_q}$ is the weight vector of the query term respectively. Each term weight vector is calculated using the term frequency (TF) and the inverse document frequency (IDF), and its calculation formula is as follows.

$$\mathrm{TF}(t,d) = \frac{f(t,d)}{\#\ of\ terms}$$
$$\mathrm{IDF}(t,D) = \log\left(\frac{\#\ of\ D}{n}\right)$$
$$\vec{V} = [w_t] = [\mathrm{TF}(t,d) \times IDF(t,D)],$$
where $\vec{V}$ *is vector of* w_t,
where w_t *is weight of a term* t

In the word frequency formula, t and d are terms and documents, respectively $f(t,d)$, is the number of occurrences of term t in document d, and # of terms is the total number of terms in document d. In the inverse document frequency formula, t and D mean term and whole document respectively, # of D means the total number of documents, and n means the number of documents containing term t. Also, $\overrightarrow{V}$ is the weight vector of term t and the weight of term t is the product of TF and IDF.

As a final step, Step 3 ranks commits based on the similarity score calculated using VSM and extracts the information of the commit with the highest similarity score.

4.3 Enhancing Fault Traceability using Behavior Model

In this study, a behavior model is used for improving fault traceability. A behavior model represents how each component behaves to a function that meets a specific requirement. Generally, the behavior model is represented as a sequence diagram [15] and can trace the resources associated with the component [16]. We enhance fault traceability by using the behavior model toward the information of the commit which is most similar to the bug report found through the method The detailed steps of this method are as follows.

1. Extract source code changes from commit information
2. Extract methods modified by source code changes
3. Trace related resources using sequence diagrams

In the first step, only source code changes are extracted from the information of the commit with the highest similarity score. Step 2 is the step of extracting the method modified by the source code changes. The source code changes, which is the information of the commit, save only which lines of source code have been added, modified and deleted. It is necessary to determine the modified lines belong to which method and class. Therefore, we parse the modified source files to extract the method that the source code changes belongs to.

As a final step, fault tracing is performed through the component flow of the sequence diagram for the extracted methods. The reason for this procedure is that the methods that need to be modified to fix the bug may be in other functionally related resources, rather than the source code changes of the commit that most similar with the bug report. In this paper, we define traceability set using enhanced fault traceability as follows.

$\mathrm{TS} = \{\}$
$\mathrm{TS} = \mathrm{TS} \cup \{m_c()\}$,
where $m_c()$*is method of changed code by commit*
$\mathrm{TS} = \mathrm{TS} \cup \{m_r()\}$,
where $m_r()$ *is related method with* $m_c()$

Traceability set TS is a set of methods. m_c is a method modified by a commit similar to the bug report, and it is added to the TS. m_r is an related method called by m_c, which adds it to the TS.

5. Case Study

In this chapter, we will apply the examples to the methodology presented above to the zxing project.

5.1 Tracing a Related Commit with the Bug Report

We first extract keywords from the bug report of Figure 1 in Chapter 2, and remove stop words. The total number of terms extracted from the bug report through this procedure is 70. After that, keywords excluding stop words and reserved

words are extracted from the description and the source code changes of the whole commit of the project. We then treat the bug report as a query and the commit information as a document, respectively. And then TF-IDF and VSM are applied to them. Table 1 shows the results of calculating similarity scores between the bug report and commits.

(Table 1) TF-IDF & VSM Result(Top 5 Commits)

Commit ID	# of terms	# of identical terms	VSM Score
84a3b27	11	93	0.5017
d1ef2a9	16	398	0.4889
90e1822	12	263	0.4822
02d3697	14	168	0.4810
23893a0	14	183	0.4737

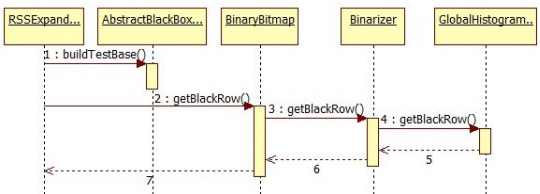
Comparing the similarity scores calculated by the VSM results, we can see that the commit (84a3b27...) with a similarity score of 0.5017 is the most similar commit with the bug report. Therefore, we extract the information of the commit and use it in the next step.

5.2 Enhancing Fault Traceability by Using Behavior Model

The source code changes of the commit extracted from the previous step include lines 239-245 in RSSExpanded Image2binaryTestCase.java, lines 91-97 in RSSExpanded Image2resultTestCase.java and so on. First, we analyze the structure of the method by parsing the modified Java files to figure out which method the modified line belongs to. Through the analyzed method structure, we can trace lines 239-245 in RSSExpandedImage2binaryTestCase.java and lines 91-97 in RSSExpandedImag2resultTestCase.java modified and belongs to assertCorrectImage2binary() method and assertCorrectImage2result() method, respectively. We add these traced methods to the traceability set TS.

As a final step, the behavior model, sequence diagram, improves fault traceability. Figure 3 shows part of the flow of the assertCorrectImage2result() method in the sequence diagram of RSSExpandedImage2resultTestCase.java. According to the sequence diagram in RSSExpandedImage2 resultTestCase.java, assertCorrectImage2result() calls getHeight() and getBlackRow(). Therefore, these methods are added to the traceability set TS, since these methods are also resources associated with the fault.

In the zxing project, in reality, the developer applied a commit (6cdc749...) to fix the bug described in the bug report of the case study. The source code changes of this commit are line 65, lines 69-86. These source code changes are included in the getBlackRow() method in the GlobalHistogramBinarizer.java file and this method exists in the traceability set TS created in the case study. That is, when the proposed technique is applied, it can be seen that the defective method has been successfully traced.



(Figure 3) Sequence Diagram

6. Experiment and Evaluation

For verifying and seeing the effectiveness of the enhanced fault traceability approach proposed in this paper in case of a general program, we apply our approach to the open source project. The subject of the experiment is the zxing project used in case studies.

6.1 Experimental Environment and Procedure

Before proceeding with the experiment, We collected bug reports from the issues listed in github's zxing project, with the exception of bug reports that is simple questions or have lacked information. In addition, if the bug report includes an image, the natural language portion excluding the image was used as the bug report information. Also, in order to verify the accuracy of the proposed method, it is necessary to confirm that the tracked resources are related to actual defects [17]. That is the commit that fixes the defect described by the bug report should be clear to verify the success of the test results. Therefore, only the bug reports

that are connected with commits and can confirm the success of the experiment results were selected and became subject of the experiment. Also, if the commit that most similar to the bug report is a commit connected to the bug report, this commit is exactly a commit for fixing the defect, so that it is unacceptable fault traceability. Therefore, this kind of commits, in this case, were excluded from the dataset of fault traceability.

The experiment is divided into two steps. In the first step, the VSM and the TF-IDF were used to select the commit with high similarity to the bug report. Thereafter, the methods including the source code changes of the selected commit were extracted and added to the traceability set TS. In the second step, the behavior model was used to enhance fault traceability. We analyzed the behavioral model to trace the methods called by the methods extracted in the previous step and added those methods to the TS. In this study, we confirmed the improvement effect of fault traceability through these two steps.

6.2 Experiment Result

(Table 2) Experiment Results

Bug Reports	File Trace	Method Trace
B_1	1/1	0/1
B_2	0/1	1/1
B_3	0/1	1/1
B_4	1/1	1/1
B_5	0/1	1/1
B_6	0/1	1/1
B_7	1/1	1/1
B_8	1/1	1/1
B_9	1/1	1/1
B_{10}	1/8	1/10
B_{11}	0/3	0/4
B_{12}	0/3	0/1
B_{13}	0/1	0/1
B_{14}	0/3	0/2
B_{15}	1/1	0.1

Table 2 shows the results of the experiment. We experimented 15 bug reports set B in this paper. First, the first step used VSM evaluates whether we can directly find the file contains the fault. In this case, the average traceability accuracy was about 41%. In case of traceability set TS consist of only directly modified methods by the selected commit in step 1, result of the fault traceability did not show a meaningful value. Therefore, in the first step, we did not track up to the method unit. In order to extend it to the method unit, fault tracing was performed through the second step using behavior model and its tracing accuracy was about 54% in method unit. In addition, through the second step, it was possible to trace even the fault that failed to trace the file in the first step, and it was confirmed that tracing using the behavior model can improve the fault traceability. However, if the number of files included in the commit is small and the commit information is reliable and it was easy to distinguish the specificity and purpose of the commit, high traceability was obtained. In contrast, if the number of files is large or the commit information is uncertain, the tracing fault was failed or tracing only some of the resources. That is if the commit contains the large scale of function or bug, or source code changes are not closely related to each other, it becomes difficult to distinguish the specificity and purpose of the commit. So, it led to decreasing fault traceability precision.

7. Discussion

The feature of the proposed approach and related works is to trace the resources related to the reported bug report by using the existing project's commit history and behavior model. Using this technique, it is possible to identify defective resources and to fix them with minimized the developer's time and effort. In this chapter, we compare the fault traceability enhancement method of this study, which was verified through previous case studies and experimental results, with the existing studies and then discuss the limitations.

7.1 Comparison with Existing Study

Table 3 shows the qualitative comparison of the features of this study and previous studies. All of the previous studies, including this study, used techniques such as NLP (Natural Language Processing) of bug reports, ie tokenization and stop word removal. Also, in all the studies,

VSM was used to convert the term of the bug report to vector for calculating similarity. However, in this paper, we used commit information including source code changes and description for calculating similarity with bug reports, while existing studies only used the source code of a software for the similarity. Commit information has been used in some existing studies, but its purpose is to use changed filenames and changed method names. Therefore, the purpose is different from the purpose of commit information in this study. And the behavior model was used in some previous studies as well as in this study. A fault traceability level of the studies was method unit. However, our proposed method can be applied to a general program, while the program that can be traced by the method of [5] is limited to the web application developed based on the MVC design pattern.

(Table 3) Qualitative Comparison of Our Study and Existing Studies

		[7]	[5]	This Study
Bug Report NLP	Tokenize Stopword	O	O	O
VSM		O	O	O
Commit Information	Changes	X	O	O
	Description	X	X	O
Behavior Model		X	O	O
Fault Trace Level		Class	Method	Method
Application Domain		General	Web	General

7.2 Limitation

1. Commit Information : This study begins by looking for similar commit information with the reported bug report. But it has a limitation that existing commits must have a commit corresponding to the bug report. In addition, a certain amount of commits are required for valid similarity calculation for using TF-IDF. In other words, there is a limitation that commits of the project occurs more than a certain level and also the description of the commits must be described well in order to use the technique proposed in this study.

2. Behavior model : The second step of our technique is fault traceability enhancement using behavior model. The behavior model allows you to identify the methods associated with the object. However, if the project to which our technique is applied has a high coupling, there would be too many associated components, and too many trace results would be added to the traceability set TS. This can be ineffective or adversely affecting developers who want to use software fault traceability method to save time and effort.

8. Conclusion

In this paper, we proposed a fault traceability enhancement technique for the reported bug report. The similar commit was traced through the computation of similarity between the bug report and the commit information. Next, behavior model was used for the similar commit to improving fault traceability. Compare with previous studies, this study can be applied to general software and also tracing faults to method level at the same time. However, it is possible that the behavior model leads to a situation where too many trace results belong to the traceability set. Therefore, in the future, it would be more effective to rank the traceability results according to how much the result in the traceability set is associated with the fault. Also, in order to more accurately find a commit that is similar to bug reports, we will develop techniques to improve fault traceability by continuing to study additional applicable elements such as commit author information other than commit descriptions and source code changes.

References

[1] D. Baek, B. Lee, J. Lee, "Content-based Configuration Management System for Software Research and Development Document Artifacts," KSII Transactions on Internet and Information Systems, Vol. 10, No. 3, pp.1404-1415, 2016.
http://dx.doi.org/10.3837/tiis.2016.03.027

[2] S. Kim, T. Zimmermann, E. Whitehead, A. Zeller, "Predicting Faults from Cached History," in Proc. of 29th International Conference on Software Engineering (ICSE), pp.489-498, 2007.
http://dx.doi.org/10.1109/ICSE.2007.66

[3] H. Zhang, "An Investigation of the Relationships between Lines of Code and Defects," in Proc. of 2009 IEEE International Conference on Software Maintenance (ICSM), pp.274-283, 2009.
http://dx.doi.org/10.1109/ICSM.2009.5306304

[4] S. Wang, D. Lo, "Version History, Similar Report, and Structure: Putting Them Together for Improved Bug Localization," in Proc. of the 22nd International Conference on Program Comprehension(ICPC), pp.53-63, 2014.
http://dx.doi.org/10.1145/2597008.2597148

[5] S. Baek, J. Lee, B. Lee, "Improving fault traceability of web application by utilizing software revision information and behavior model," KSII Transactions on Internet and Information Systems, Vol. 12, No. 2, pp.817-828, 2018.
http://doi.org/10.3837/tiis.2018.02.016

[6] C. Youm, J. Ahn, J. Kim, E. Lee, "Bug localization based on code change histories and bug reports," in Proc. of Asia-Pacific Software Engineering Conference (APSEC), pp.190-197, 2015.
http://doi.org/10.1109/APSEC.2015.23

[7] L. Moreno, W. Bandara, S. Haiduc, A. Marcus, "On the Relationship between the Vocabulary of Bug Reports and Source Code," in Proc. of International Conference on Software Maintenance(ICSM), pp.452-455, 2013.
http://dx.doi.org/10.1109/ICSM.2013.70

[8] R. Tsuchiya, H. Washizaki, Y. Fukazawa, K. Oshima, R. Mibe, "Interactive Recovery of Requirements Traceability Links Using User Feedback and Configuration Management Logs," in Proc. of International Conference on Advanced Information Systems Engineering, pp.247-262, 2015.
http://dx.doi.org/10.1007/978-3-319-19069-3_16

[9] R. Tsuchiya, H. Washizaki, Y. Fukazawa, T. Kato, M. Kawakami, K. Yoshimura, "Recovering Traceability Links between Requirements and Source Code Using the Configuration Management Log," IEICE Transactions on Information and Systems, Vol. 98, No. 4, pp.852-862, 2015.
http://dx.doi.org/10.1587/transinf.2014EDP7199

[10] C. McMillan, D. Poshyvanyk, M. Revelle, "Combining textual and structural analysis of software artifacts for traceability link recovery," in Proc. of ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp.41-48, 2009.
https://doi.org/10.1109/TEFSE.2009.5069582

[11] B. Van Rompaey, S. Demeyer, "Establishing Traceability Links between Unit Test Cases and Units under Test," in Proc. of 13th European Conference on Software Maintenance and Reengineering, pp.209-218, 2009.
https://doi.org/10.1109/CSMR.2009.39

[12] X. Ye, R. Bunescu, C. Liu, "Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation," IEEE Transactions on Software Engineering, Vol. 42, No. 4, pp.379-402, 2016.
https://doi.org/10.1109/TSE.2015.2479232

[13] H. Choi, J. Lee, B. Lee, "Supporting Systematic Software Test Process in R&D Project with Behavioral Models," Journal of Internet Computing and Services(JICS), Vol. 19, No. 2, pp.43-48, 2018.
http://dx.doi.org/10.7472/jksii.2018.19.2.43

[14] R. Saha, M. Lease, S. Khurshid, D. Perry, "Improving Bug Localization using Structured Information Retrieval," in Proc. of 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp.345-355, 2013.
http://dx.doi.org/10.1109/ASE.2013.6693093

[15] J. Rumbaugh, I. Jacobson, G. Booch, Unified Modeling Language Reference Manual, Pearson Higher Education, 2004.

[16] Le, T.B., Oentaryo, R.J., Lo, D., "Information Retrieval and Spectrum Based Bug Localization: Better Together," in Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering(ESEC/FSE), pp.579-590, 2015.
http://dx.doi.org/10.1145/2786805.2786880

[17] Herzig, K., Just, S., Zeller, A., "It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction," in Proc. of the 2013 International Conference on Software Engineering(ICSE), pp.392-401, 2013.
http://dx.doi.org/10.1109/ICSE.2013.6606585

◐ 저 자 소 개 ◑

정 동 주(Dongju Jung)
2018년 서울시립대학교 컴퓨터과학부(학사)
2018년~현재 서울시립대학교 컴퓨터과학부 석사과정
관심분야 : 소프트웨어테스트, 소프트웨어공학
E-mail : jdj700@uos.ac.kr

민 경 식(Kyeongsic Min)
2018년 서울시립대학교 컴퓨터과학부(학사)
2018년~현재 서울시립대학교 컴퓨터과학부 석사과정
관심분야 : 소프트웨어테스트, 소프트웨어공학, 블록체인, 스마트 컨트랙트
E-mail : ksmin1710@uos.ac.kr

이 정 원(Jung-Won Lee)
1993년 이화여자대학교 전자계산학과(학사)
1995년 이화여자대학교 전자계산학과(석사)
1995년~1997년 LG종합기술원 주임연구원
2003년 이화여자대학교 컴퓨터학과(박사)
2003년~2006년 이화여자대학교 컴퓨터학과 BK교수, 전임강사(대우)
2006년~현재 아주대학교 전자공학과 교수
관심분야 : Embedded Software, Automotive Software, Bio·Medical Data Modeling
E-mail : jungwony@ajou.ac.kr

이 병 정(Byungjeong Lee)
1990년 서울대학교 계산통계학과(학사)
1990년~1998년 (주)하이닉스반도체 연구원
1998년 서울대학교 전산과학과(석사)
2002년 서울대학교 전기·컴퓨터공학부(박사)
2002년~현재 서울시립대 컴퓨터과학부 교수
관심분야 : 소프트웨어테스트, 소프트웨어 진화, 소프트웨어공학
E-mail : bjlee@uos.ac.kr