Improving Fault Traceability of Web Application by Utilizing Software Revision Information and Behavior Model

Seungsuk Baek[1], Jung-Won Lee[2], Byungjeong Lee[1]
[1]Department of Computer Science, University of Seoul
Seoul, 02504 - South Korea
[e-mail: {baekss0409, bjlee}@uos.ac.kr]
[2]Department of Electrical and Computer Engineering, Ajou University
Suwon, 16499 - South Korea
[e-mail: jungwony@ajou.ac.kr]
*Corresponding author: Byungjeong Lee

Received October 15, 2017; accepted December 10, 2017; published February 28, 2018

Abstract

Modern software, especially web-based software, is broadly used in various fields. Most web applications employ design patterns, such as a model-view-controller (MVC) pattern and a factory pattern as development technology, so the application can have a good architecture to facilitate maintenance and productivity. A web application, however, may have defects and developers must fix the defects when a user submits bug reports. In this paper, we propose a novel approach to improving fault traceability in web application by using software revision information and software behavior model to reduce costs and effectively handle the software defect. We also provide a case study to show effectiveness of our approach.

Keywords: Fault Traceability, Software Revision Information, Behavior Model, Design Pattern, Web Application

A preliminary version of this paper was presented at APIC-IST 2017, and was selected as an outstanding paper.
This work was supported by the 2015 Research Fund of the University of Seoul for Byungjeong Lee. Also, this work was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2014M3C4A7030504) for Jung-Won Lee.

http://doi.org/10.3837/tiis.2018.02.016 **ISSN : 1976-7277**

1. Introduction

In order to effectively provide various services to many users, web-based software, which has the advantages of both accessibility and extendibility, is broadly used in various fields. For example, through a web application we can buy food, clothes, and other things from anywhere; moreover, a web application can provide more products by extending customer services. Generally, these applications have a lot of functionality in order to provide the various services and experiences, and most web applications employ a design pattern, such as a model-view-controller (MVC) pattern and a factory pattern as development technology, so the application can have a good architecture to facilitate maintenance and productivity [1].

On the other hand, a web application may have defects [2] and developers must fix the software defects discovered through system testing and users' bug reports. At present, if there is traceability information between source code related to an application's defects, developers can reduce the effort to find large amounts of source code to fix the defects. Therefore, with the bug report, it is important to provide fault traceability links between source code and the software bug. In this paper, we propose a novel approach to traceability links from bug report to source code based on understanding design patterns like the MVC pattern, and we improve the traceability links by utilizing a software behavior model and software revision information.

This study provides the following contributions.

- We extract faults from bug reports by using the design pattern applied to the software in order to fix the defects.
- We utilize a software behavioral model to find relevant source code for a defect's behavior.
- We improve fault traceability through software revision information to identify additional source code to fix defects.

The remaining sections of this paper are as follows. Section 2 provides background to this study. Section 3 presents related works. We present our approach in Section 4, a case study and results in Section 5, and we discuss our study in Section 6. Finally, we conclude the paper and suggest future work in Section 7.

2. Background

2.1 Software Testing for Monitoring Software R&D Project

This project has a testing objective to support monitoring software R&D project through content-based testing, and we have some roles in order to achieve the objective in the project [3]. For example, we have to provide an integrated test process model, test-case generation from a software model, and software-defect traceability from a bug report with a framework. This study aims at software-defect traceability from a bug report, as already mentioned in the introduction, and a case study will be conducted with the project.

2.2 MVC Pattern

A model-view-controller pattern is used in most web applications for software maintenance. If you apply an MVC pattern to your application, you can see benefits. **Fig. 1** shows the MVC pattern–based web application. In detail, the software architecture is divided into three components. M (for model) denotes the data of the application, and the behavior logic of the application (e.g. data insertion, updating, and deleting) is processed in M. V (view) denotes the output of the processed data in the model to present to the user. C (controller) is located between model and view; it handles the client's requests and the data of model to present it to the user in any view. Thus, with the MVC pattern, each component has loose coupling between it and the others.

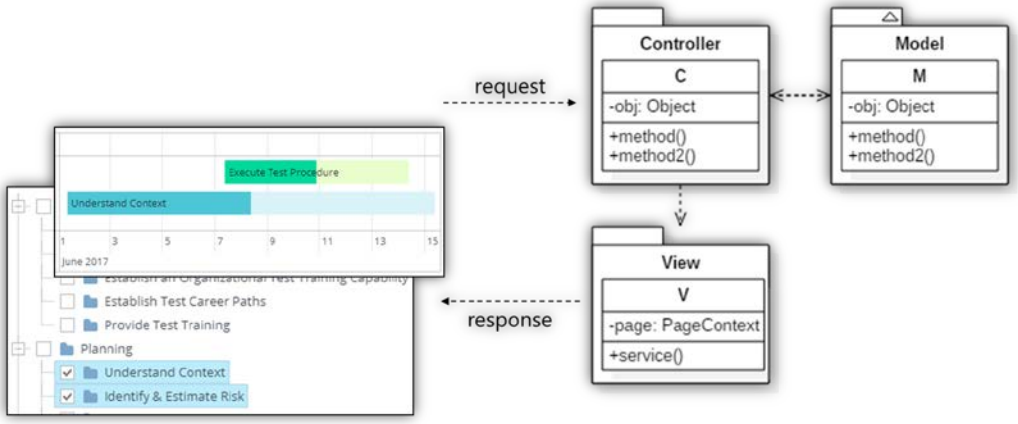

Fig. 1. MVC pattern-based web application

2.3 Software Revision Control

Software revision control [4] is a very important activity in software engineering, because software evolves continuously [5] through software bug fixing or software requirement changes. Most software development groups utilize a software revision control system to manage their software. When software has a new revision, the history of the evolution from previous revisions is recorded through a software revision control system, such as Git and Subversion [6, 7]. For example, one revision has source code files that are handled by the developer for bug fixing or enhancing software behavior. Therefore, the source code files in the same revision are relevant to one another.

3. Related Work

Previous studies introduced approaches to traceability links between software artifacts. Tsuchiya et al. introduced an approach to recovering traceability links between source code and requirements by using a configuration management log and a vector space model [8]. They used the configuration management log written by a developer to match software requirement specifications, and in their study, if the contents of the configuration management log and the contents of software requirement specifications result in relevant content by similarity calculations in a vector space model, they are relevant artifacts. They also improved

their approach through user feedback by using call relationships [9].

McMillan et al. presented a technique for indirectly recovering traceability links in requirements documentation [10]. In order to recover traceability links between source code and documentation, they used latent semantic indexing (LSI) [11] by applying singular value decomposition [12] to determine textual similarities among words and documents. After that, they used a structural analysis tool for finding source-to-source edges. Finally, they suggested an indirect traceability link throughout the documentation through LSI and method call relationships. Van Rompaey et al. demonstrated a method for identifying a tested object and a tested method through last call before assert (LCBA) [13]. They analyzed the unit under testing from a test case and tried to find an assert statement; generally, the target of the test is located before the assert statement, so they could find the target of the test and then they could find the actual source code of the tested object. Baek et al. proposed an approach to enhance traceability from test to source code [14]. In detail, they first depicted unit test code as an abstract syntax tree (AST), and then identified the tested object by the assert statement [13]. **Table 1** show some assert statements they used in their study, where the first column of **Table 1** denotes the assert statements for unit testing, and the second column of **Table 1** provides a description of each assert statement.

Table 1. Assert statements

Assert Statement	Description
Assert Equals	Test that two objects are equal
Assert False	Test that a condition is false
Assert True	Test that a condition is true
Assert Null	Test that an object is null
Assert fail	Test that whether exception is generated or not

Next, they enhanced traceability from test to source code by using a sequence diagram, which identified the object in the previous step because a sequence diagram presents interactions between objects through messages.

Most previous studies proposed an approach to recovering traceability links, such as source code to requirement, or test to source code. In the next Section, we introduce an approach to traceability links from software defect to source code.

4. Improving Fault Traceability of Web Application

In this Section, we introduce our approach, with **Fig. 2** showing an overview. Our approach has a process in four sequential steps to achieve our objective. First, we try to extract the fault from a well-written bug report, which means the bug report has detailed content, so we know what kind of defect has happened. After that, we try to find all classes related to the defective behavior by using a software behavioral model. Next, we try to find unit-testing class of the objects that we found in the second step through a software behavioral model, such as a sequence diagram. Finally, we try to find other source code through software revision information. We describe the details of our approach according to the steps, as follows. We assume that

$$W = M \cup V \cup C \tag{1}$$

where W is a set of classes in a web application and M, V, and C denote a set of the classes as Model, View, and Controller in a web application W, respectively.

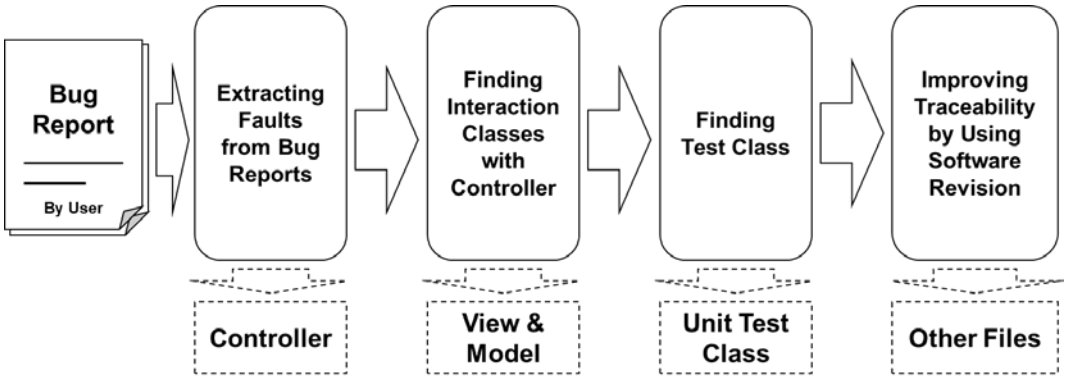

Fig. 2. Overview of our approach

4.1 Extracting Faults from Bug Reports

First, we try to find the web application's defect from a submitted bug report. In general, a bug report has a description of when the user encountered the problem and what kind of problem it was. In our study, we use a well-written bug report.

Fig. 3 shows the identified controller from the bug report. In detail, a bug report has URI information, and we can find a controller from the URI, because the URI written in the bug report matches the RequestMapping of the method in the controller. In other words, as introduced in Section 2.2, in an MVC pattern–based web application, the controller has the role of entry point for the client's request, and therefore, it is important that the bug report provides the request information to the developer so the developer can identify the controller by the URI.

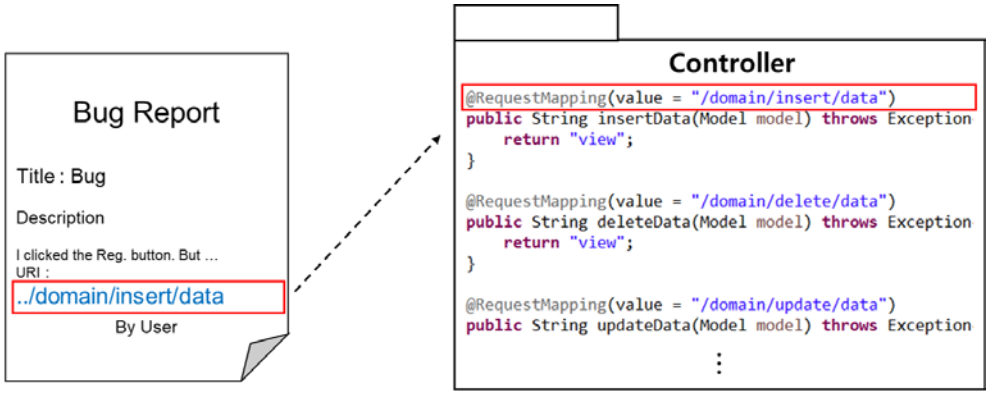

Fig. 3. Bug report and controller

This step is as follows. Initially, we set a set T for fault traceability to empty.

$$T = \emptyset \tag{2}$$

where T is a set of classes for fault traceability. If $c \in C$ is identified by a URI in a bug report, we add class c to set T.

$$T = T \cup \{c\} \tag{3}$$

4.2 Finding Interaction Classes with Controller

After Step 1, we utilize a sequence diagram to find classes through messages [14].

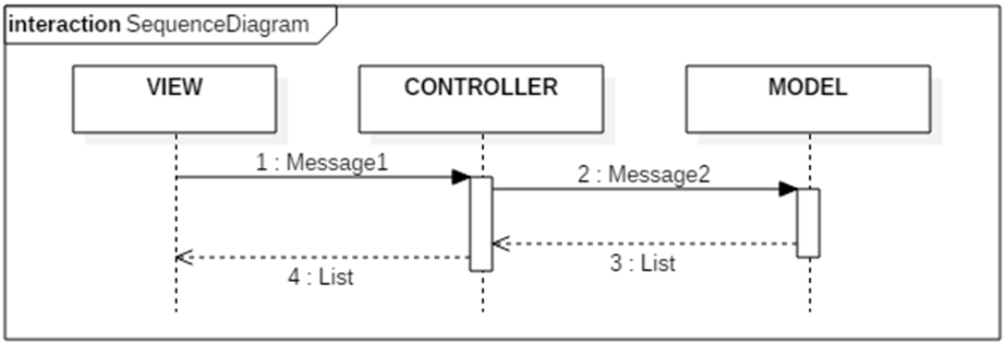

Fig. 4. Sequence diagram

Fig. 4 is a sequence diagram that shows how the controller identified in the previous step interacts with other classes. In detail, the View calls message1 from the controller and the controller calls message2 from the Model. Thus, we can find and obtain other classes with the controller through the sequence diagram. In particular, the model's classes will be fixed, because in an MVC pattern, the model is in charge of the behavior of the software, and the defect is unexpected behavior. This step is as follows. If a controller class $c \in T$ sends or receives a message to or from View or Model class o, we add class o to set T.

$$T = T \cup \{o\} \tag{4}$$

where class o interacts with controller $c \in T$.

4.3 Finding Test Class

Next, we obtain unit testing source code of the objects that we found from the process described in Section 4.2. The objects should have regression testing to protect against an unexpected defect that did not exist previously when a method has a rebuilt state from refactoring, bug fixing, and any other procedures. Due to this method (especially methods of the model), classes will be fixed to remove the software bug, and their unit testing must be performed with a unit testing tool [15] in order to check whether the method performs correctly or not after rebuilding. Therefore, we need to find the unit testing source code of the object.

In our study, we use an approach proposed by Van Rompaey et al. [13] to obtain the unit testing source code. For example, we can gain unit testing source code for model, view, and controller objects according to the following steps. First, we find the assert statement in the testing source code, and next, we find an object located before the assert statement. This step is as follows. If a test class tc tests an class $c \in T$, we add test class tc to set T.

$$T = T \cup \{tc\} \tag{5}$$

where tc is a test class which tests class $c \in T$.

4.4 Improving Traceability by Using Software Revision

Finally, we have to improve the fault traceability links through software revision information. As mentioned in Section 2.3, all source code files in the same revision are relevant to one another. Therefore, we investigate the revision that includes the classes obtained in the previous step. If any revision contains the class (*c*, *o*, and *tc*), we improve fault traceability collecting other files through its revision. This step is as follows. If an class *o* and any $t \in T$ appear in the same revision r_i (i = 1, … N), we add class *o* to set *T*.

$$T = T \cup \{o\} \tag{6}$$

where *N* is the total number of revisions in a web application *W*.

Thus, we can improve the fault traceability of the web application by using software revision information.

5. Case Study

We present a case study based on the approach proposed in Section 4 to verify our study. We used a bug report and source code files of our software R&D project.

5.1 Extracting Faults from Bug Report

We used a well-written bug report. **Fig. 5** shows a bug report, and this bug report has '/s2/testplan/schedule/insertschedule' as URI information with a description. We can find a controller from the URI, because the URI written in the bug report matches RequestMapping of the method in TestPlanController.

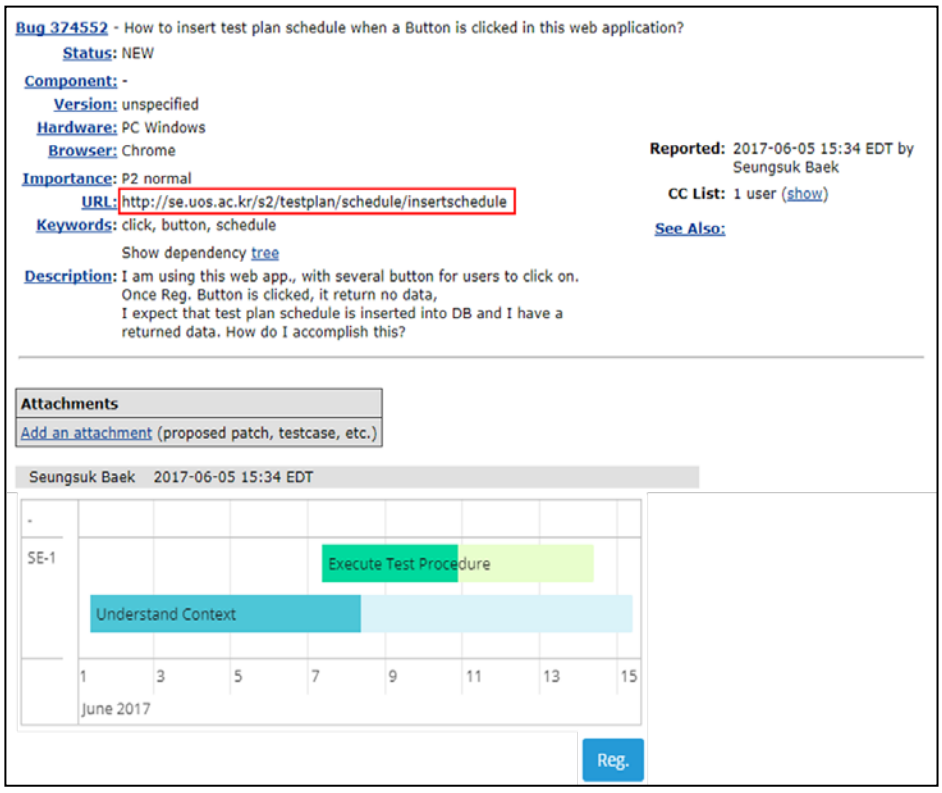

Fig. 5. An example of bug report

Fig. 6 shows TestPlanController and its method. As shown in **Fig. 6**, the insertSchedule method has RequestMapping, which matches the URI written in the bug report. Therefore, we could identify the defective method in TestPlanController from the bug report. If no URI information exists in a bug report, we can use an attached file such as a captured screen image or an otherwise occurring event (e.g. button click or key press), because it is not a problem for developers to extract the URI from any event function or captured screen image. Therefore, it is important that the bug report provide rich information for finding the fault.

```
@RequestMapping(value = "/testplan/schedule/insertschedule", method = {RequestMethod.POST,RequestMethod.GET})
@ResponseBody
public ModelMap insertSchedule(HttpServletRequest request) throws Exception {
    String userID          = request.getParameter("user");
    int taskID             = Integer.parseInt(request.getParameter("name"));
    String startDate       = request.getParameter("start");
    String endDate         = request.getParameter("end");
    double workRate        = Double.parseDouble(request.getParameter("value"));
    String taskPhase       = request.getParameter("testLevel");
    String qualityAttribute = request.getParameter("testNonFunc");
    String description     = request.getParameter("task");
    int projectID          = 1;
    String updateFlag      = request.getParameter("updateFlag");
```

Fig. 6. TestPlanController's method

5.2 Finding Interaction Classes with Controller

In this step, we use a sequence diagram (**Fig. 7**) by which we obtain the classes that interact with TestPlanController through messages in the sequence diagram.

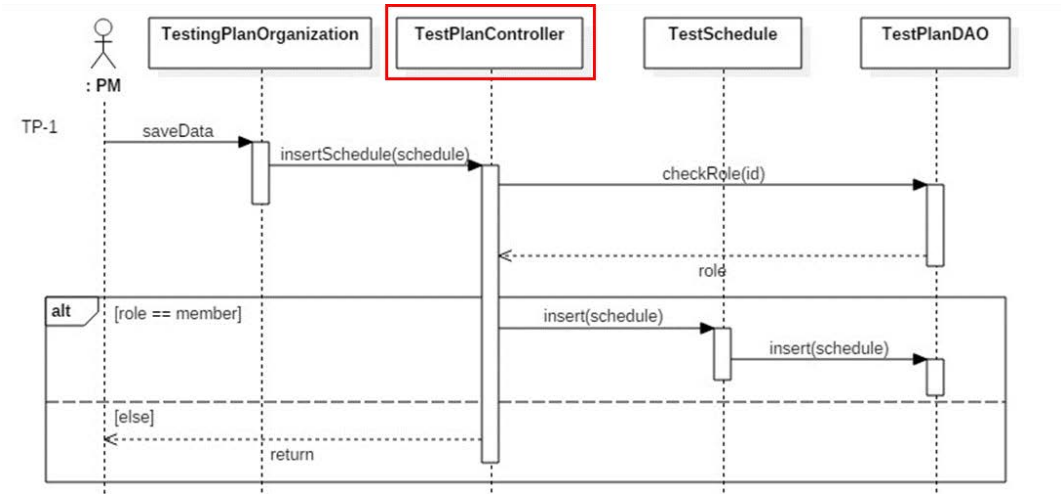

Fig. 7. Interaction of classes with the controller

As shown in **Fig. 7**, TestPlanController interacts with other classes. In detail, we obtain TestingPlanOrganization as V (View) and TestSchedule and TestPlanDAO as M (Model) because they interact with TestPlanController by messages.

5.3 Finding Test Class

We tried to find unit testing source code for the TestingPlanOrganization, TestSchedule, TestPlanDAO, and TestPlanController objects to perform regression testing after fixing the

defect. First, we find the assert statement in the testing source code, and after that, we find an object located before the assert statement. If the object was one of them, we took the unit testing source code.

Table 2. Unit testing source codes

Source Code	Description
TestScheduleInsertingTest.java	These are testing source codes of TestSchedule for insertSchedule, deleteSchedule, updateSchedule, and selectSchedule methods, respectively.
TestScheduleDeletingTest.java	
TestScheduleUpdatingTest.java	
TestScheduleSelectingTest.java	
TestPlanDaoInsertingTest.java	These are testing source codes of TestPlanDAO for insertSchedule, deleteSchedule, updateSchedule, and selectSchedule methods, respectively.
TestPlanDaoDeletingTest.java	
TestPlanDaoUpdatingTest.java	
TestPlanDaoSelectingTest.java	

Table 2 shows that we obtained unit testing source code using an approach proposed by Van Rompaey et al. [13], along with their descriptions. As shown in **Table 2**, we could gain unit testing source code for the TestSchedule and TestPlanDAO objects, but there was no unit testing source code for the other objects in our software R&D project.

5.4 Improving Traceability by Using Software Revision

Finally, we tried to improve the fault traceability links through software revision information.

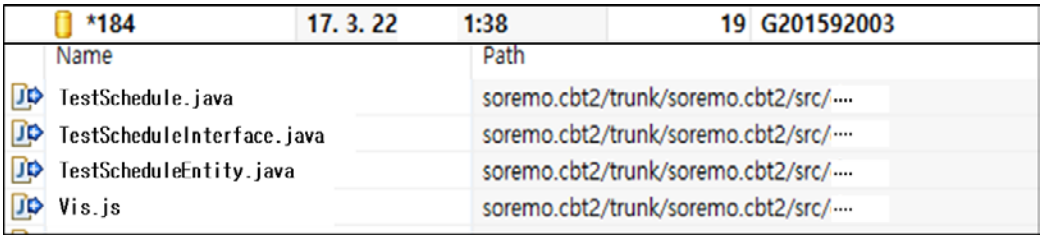

Fig. 8. Information on a revision

Fig. 8 shows revision number 184 as an example, and we could gain additional source code from revision number 184 because it contained TestSchedule class that we already found in Section 5.2 and other files were in the same revision with the TestSchedule class. So, as mentioned in sections 2.3 and 4.4, they are relevant to one another. Next, we tried to investigate the additional source code.

Table 3 shows that we obtained additional source code and file. We could gain various kinds of source code and files. In detail, we could gain interface source code of the classes obtained in the previous step, and we could gain source code such as TestScheduleEntity. TestScheduleEntity has the role of an entity bean for Object-Relational Mapping (ORM), because we use an ORM framework in our project [16].

Finally, due to the revision number 275 which contained TestPlanDAO class that we already found in Section 5.2, we also could gain configuration files such as persistence-context.xml and servlet-context.xml as well as classes.

Table 3. Additional files

File	Description	Rev. No.
TestScheduleInterface.java	Interface of TestSchedule	184
TestPlanDaoInterface.java	Interface of TestPlanDAO	
TestingPlanOrganizationInterface.java	Interface of TestingPlanOrganization	
TestScheduleEntity.java	Entity bean for Object-Relational Mapping from TestScheduleTable of DB	
Vis.js	Script file for dynamic representation of inserted schedule contents	
Soremo.css	Css file for control attribution such as color, position, and etc. of view	
persistence-context.xml	Config file for connecting to DB	275
servlet-context.xml	Config file for framework of web development	

6. Discussion

We propose an approach to improving fault traceability of web applications by utilizing software revision information and a behavior model, and we offered a case study according to the steps introduced in Section 4. Our case study showed that source code is found increasingly with each step, and our approach showed the improved fault traceability to reduce the developer's efforts. In addition, we conducted a case study with another bug report and controller in the same manner.

Table 4. Comparison with other studies

Approach	Statement Analysis	Behavioral Model	Revision Information	# of classes	# of files
C. McMillan et al. [10]	O	X	X	8	0
B. Van Rompaey et al. [13]	O	X	X	12	0
S. Baek et al. [14]	O	O	X	20	0
Our Study	O	O	O	28	4

Table 4 shows a comparison with other studies. Our approach uses statement analysis, a software behavioral model, and software revision information. However, the other works do not use software revision information. Our approach can find more classes than the other

works and can also find resource files as well as classes. Therefore, our approach improves fault traceability links better than the other works.

However, we have to apply it to other projects to verify whether our approach is a useful approach or not, and we must more closely investigate the bug reports because it may be that we can use other attributes in the bug report.

7. Conclusion

Web-based software, which has the advantages of both accessibility and extendibility, is broadly used in various fields. However, it is software, so it can have defects, and the defects must be fixed by developers. In this study, we provided an approach to improving fault traceability in web applications by utilizing software revision information and a behavior model to reduce the developer's efforts. In detail, we first tried to extract the fault from a bug report. With the request URI in the bug report matched to the method of the controller, we could find the controller and method related to the defect's behavior. Next, we tried to obtain interaction classes with the controller from a sequence diagram. In general, a sequence diagram provides interactions between classes, so we could find and obtain other classes that interact with the controller through messages. After that, we tried to obtain a unit test case for regression testing, and finally, we showed the improved fault traceability by using software revision information.

As a result, our approch outperforms other work, and we expect our approach can be a useful one for improving fault traceability. In the future, we will provide a framework for software defect fixing [17] as well as for supporting fault traceability links.

References

[1] E. Freeman, B. Bates, K. Sierra, and E. Robson, "Head First Design Patterns: A Brain-Friendly Guide," *O'Reilly Media*, 2004. Article (CrossRef Link)

[2] G. Yang, T. Zhang, and B. Lee, "Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-feature of Bug Reports," in *Proc. of Computer Software and Applications Conference (COMPSAC)*, 2014. Article (CrossRef Link)

[3] A. Dashbalbar, S. Song, J. Lee, and B. Lee, "Towards Enacting a SPEM-based Test Process with Maturity Levels," *KSII Transactions on Internet and Information Systems*, vol. 11, no. 2, pp. 1217-1233, 2017. Article (CrossRef Link)

[4] B. Alwis, and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?," in *Proc. of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pp. 36-39, 2009. Article (CrossRef Link)

[5] T. Mens, and S. Demeyer, "Software Evolution," *Springer Publishing Company*, 2008. Article (CrossRef Link)

[6] J. Loeliger, and M. McCullough, "Version Control with Git: Powerful tools and techniques for collaborative software development," *O'Reilly Media*, 2012. Article (CrossRef Link)

[7] C. Michael Pilato, B. Collins-Sussman, and B. Fitzpatrick, "Version Control with Subversion: Next Generation Open Source Version Control," *O'Reilly Media*, 2008. Article (CrossRef Link)

[8] R. Tsuchiya, H. Washizaki, Y. Fukazawa, T. Kato, M. Kawakami, and 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. Article (CrossRef Link)

[9] R. Tsuchiya, H. Washizaki, Y. Fukazawa, K. Oshima, and R. Mibe, "Interactive Recovery of Requirements Traceability Links Using User Feedback and Configuration Management Logs," in *Proc. of the International Conference on Advanced Information Systems Engineering*, pp. 247-262,

2015. Article (CrossRef Link)
[10] C. McMillan, D. Poshyvanyk, and 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*, 2009. Article (CrossRef Link)
[11] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the American Society for Information Science*, vol. 41, pp. 391-407, 1990. Article (CrossRef Link)
[12] G. Salton, and M. McGill, "Introduction to Modern Information Retrieval," *McGraw-Hill*, 1983. Article (CrossRef Link)
[13] B. Van Rompaey, and S. Demeyer, "Establishing traceability links between unit test cases and units under test," in *Proc. of European Conference on Software Maintenance and Reengineering*, pp. 209-218, 2009. Article (CrossRef Link)
[14] S. Baek, J. Lee, and B. Lee, "Utilizing Software Behavioral Model to Enhance Traceability from Test to Source Code," in *Proc. of the 19th Korea Conference on Software Engineering*, 2017.
[15] P. Tahchiev, F. Leme, V. Massol, and G. Gregory, "JUnit in Action," *Manning Publications*, 2010. Article (CrossRef Link)
[16] C. Babu, and G. Gunasingh, "DESH: Database evaluation system with hibernate ORM framework," in *Proc. of International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2016. Article (CrossRef Link)
[17] H. Yokoyama, Y. Higo, K. Hotta, T. Ohta, K. Okano, and S. Kusumoto, "Toward improving ability to repair bugs automatically: a patch candidate location mechanism using code similarity," in *Proc. of the 31st Annual ACM Symposium on Applied Computing*, pp. 1364-1370, 2016. Article (CrossRef Link)

Seungsuk Baek, He received his B.S. degree from Eulji University, Seongnam, Korea, in 2012. He is currently studying toward the M.S. degree in Computer Science at University of Seoul, Korea. His research areas of interest include software engineering and software fault.

Jung-Won Lee, is an associate professor of the Department of Electrical and Computer Engineering at Ajou University, Korea. She received her PhD. Degree in Computer Science and Engineering from Ewha Womans University, Korea, in 2003. She was a researcher of LG Electronics and did an internship in the IBM Almaden Research Center, USA. Her areas of research include context-aware, embedded software and software engineering.

Byungjeong Lee, He received the B.S., M.S., and Ph.D. degrees in Computer Science from Seoul National University in 1990, 1998, and 2002, respectively. He was a researcher of Hyundai Electronics, Corp. from 1990 to 1998. Currently, he is a professor of the Department of Computer Science and Engineering at the University of Seoul, Korea. His research areas include software engineering and web science.