Helping Testers by Fault-Prone Functionality Prediction

Keiichi TABATA, Haruto TANNO, Morihide OINUMA
Software Innovation Center
Nippon Telegraph and Telephone
Kounan 2–13–34, Minato, Tokyo 108–0038
Email: {tabata.keiichi haruto.tanno oinuma.m}@lab.ntt.co.jp

***Abstract*—In this paper, we propose a technique to predict fault-prone software functionality, instead of fault-prone module. The granularity of prediction is not a line number nor a function name in a source code, but a software functionality from point of view of testers who are dedicated to software testing. Our approach makes it possible for testers to find faults efficiently and effectively on test case creation and test case execution. We applied the proposed technique to an open source project on github. The result graphically suggests that we can predict fault-prone software functionality using source code repository mining.**

I. Introduction

To ensure software quality, software testing forms an important part of software development project. However, any software development project has limited amount of time and resource for software testing. Therefore, it is required for testers to find faults efficiently and effectively.

In practical software testing, projects have testers who are dedicated to software testing. Testers first read and understand requirement specification of system under test. Then testers construct test cases, which include pre-requisite, testing steps and post-requisite. Figure 1 is an example test case spreadsheet. Finally, testers run test cases by their hands.

Through this typical testing process, testers, in most cases, are not aware of program structure such as file, class, method and line number.

While many studies have been conducted to propose how to find more faults on software testing, such techniques are mainly focused on unit testing which is done by programmers. Unlike these studies, our effort is focused on system testing done by testers.

Our proposed technique takes advantage of fix information from source code repository such as git[1]. In other words, we extract hints of software testing from source code repository which contains evolutionary history of each development project.

Project	Some Web System	Design date	14, Feb, 2015
System	User management	Executed by	John

Pre-requisite	After login by user name "user", click "personal information" tab.
Post-requisite	User name has changed to "user2" successfully.

Test case #	Step description	Result #1	Result #2
10	Enter "new user name" textbox to "user2".		
10	Click "change" button.		

Fig. 1. Example test case spreadsheet

II. Previous studies

Studies on fault-prone module prediction[2][3] can be fallen into two categories based on prediction scope. One is inter-version prediction, and the other is inter-program prediction. In either case, the grain of prediction is premised on program structure such as file, class, method and line number.

Fix cache[4] is an variation of fault-prone prediction. It predicts possible source code location to be fixed. It is based on a heuristic that “a source code location which was changed recently will be changed in the near future”.

While the previous studies showed how to predict fault-prone locations in source codes, typical testers are not aware of source code structure in programming language. Testers read and understand requirement specification documents in natural language.

To bridge this gap, we focused on logical coupling[5][6]. Logical coupling is a heuristic that “a group of source code locations with co-occuring changes is likely to have high association in functionality”.

III. Proposed Technique

Our proposed technique achieves both fix cache and logical coupling at same time using graph clustering. It predicts fault-prone software functionality, such as grain of items described in requirement specification documents. It has three steps.

First, create an undirected graph, namely G. Each node of graph G represents single source code location, and each edge represents what two source code locations have changed at same time.

Second, apply graph clustering algorithm such as markov clustering [7] to graph G, and inquire cluster set, namely C_n.

Property	Description
Name	ucc-c-compiler
Purpose	C compiler
Language	C
Started	July 2011
VCS	git
LOC	59488
Committers	1
Evaluated	October 2013
Commits	3814

TABLE I. EVALUATED OPEN SOURCE REPOSITORY

nodes	clusters
2-5	76
6-10	16
11-50	19
51-100	5

TABLE II. RESULTED CLUSTERS

Each cluster in cluster set Cn represents functionally related source code positions.

Finally, for each cluster in cluster set Cn, estimate suitable software functionality described in specification documents.

At the present, we execute the final step manually. Thus evaluation is limited to first and second step. We discuss about the final step in section V.

IV. EVALUATION

We applied the proposed method to an OSS source code repository from github. Table I is the detailed information of the repository.

To confirm the effectiveness of proposed method qualitatively and visually, we created a graph adjacency matrix of graph G, namely M. Figure 2 shows the heat map of co-occurring fixes. Here, we assumed "co-occurring" as changes in a single commit, and also assumed "fix" as changes in a commit which has string "fix" in its commit message[8].

After creating matrix M, we applied Markov clustering algorithm to it. Figure 3 is the result graph of Markov clustering algorithm, namely H. Table II is the detailed cluster information.

In the context of logical coupling, graph H reveals functional relations between fixed source code positions. In addition, according to fix cache studies, clusters in graph H are likely to be changed in the near future because clusters are consist of recent changes. Therefore, we can predict fault-prone functionality using this graph.

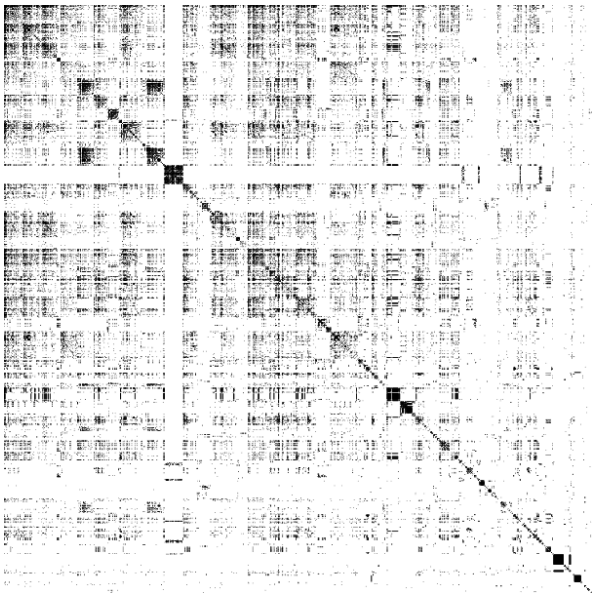

Fig. 2. Co-occurring fixes in matrix view

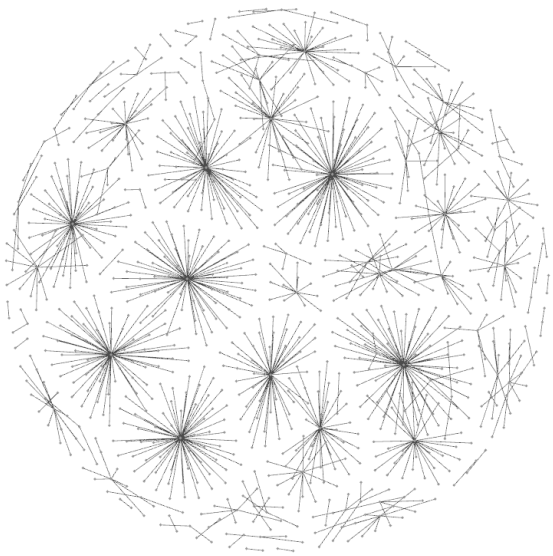

Fig. 3. Clustered co-occurring fixes in graph view

V. FUTURE WORK

The proposed method makes it possible for testers to know the presence of fault-prone software functionality. However, estimating related item in requirement specification for each fault-prone functionality remains a matter of debate.

Some logical coupling studies use change report analysis[9][10] to associate requirement specification and source code positions. In a similar way, we may be able to estimate items in requirement specification which must be associated for source code positions via analyzing commit log of source code repository.

If we apply change report analysis for commit log of source code repository, we need natural language processing approach

to inspect commit log. In that case, development process may require programmers to describe some formalized information in commit log.

VI. Conclusion

In this paper, we proposed a technique to predict fault-prone functionality instead of fault-prone module.

The proposed technique helps testers to know fault-prone software functionality in a grain of software specification.

After estimating items in requirement specification which must be associated for fault-prone software functionality, testers have two advantages on software testing.

First, testers can know hot spots of fault in advance of real test result. This is nearly predicting software quality in scope of functionality before testing.

Then, testers can prioritize test cases by fault-proneness and run test cases intensively on specific functionalities. Because any development project has limited time for software testing, prioritization of test cases is very important.

References

[1] D. Spinellis, "Git," *Software, IEEE*, vol. 29, no. 3, pp. 100–101, May 2012.

[2] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *Software Engineering, IEEE Transactions on*, vol. 22, no. 12, pp. 886–894, Dec 1996.

[3] H. Hata, O. Mizuno, and T. Kikuno, "Bug prediction based on fine-grained module histories," in *In Proceedings of 34th International Conference on Software Engineering, ICSE'12*, 2012, pp. 200–210.

[4] S. Kim, T. Zimmermann, E. Whitehead, and A. Zeller, "Predicting faults from cached history," in *In Proceedings of 29th International Conference on Software Engineering, ICSE '07*, 2007, pp. 489–498.

[5] H. Gall, M. Jazayeri, and J. Krajewski, "Cvs release history data for detecting logical couplings," in *In Proceedings of Sixth International Workshop on Principles of Software Evolution*, 2003, pp. 13–23.

[6] R. Robbes, D. Pollet, and M. Lanza, "Logical coupling based on fine-grained change information," in *Reverse Engineering, 2008. WCRE '08. 15th Working Conference on*, Oct 2008, pp. 42–46.

[7] E. AJ1, V. Dongen, and O. CA, "An efficient algorithm for large-scale detection of protein families," *Nucleic Acids Res.*, vol. 30, no. 7, pp. 1575–84, Apr 2002.

[8] A. Mockus and L. Votta, "Identifying reasons for software changes using historic databases," in *In Proceedings of International Conference on Software Maintenance 2000*, 2000, pp. 120–130.

[9] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *In Proceedings of International Conference on Software Maintenance, ICSM '98*, 1998, pp. 190–198.

[10] M. D'Ambros and M. Lanza, "Reverse engineering with logical coupling," in *In Proceedings of 13th Working Conference on Reverse Engineering, WCRE '06*, 2006, pp. 189–198.