Evaluating Automated Program Repair Using Characteristics of Defects

Haruki Yokoyama, Yoshiki Higo and Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University, Japan
Email: {y-haruki, higo, kusumoto} @ist.osaka-u.ac.jp

***Abstract*—Debugging is a cost-consuming activity and reducing debugging cost is mandatory. Automated program repair (APR) is a debugging support technique which fixes a defect from a buggy program and a test suite. Recently, many APR techniques have been proposed. The performance of APR techniques are mainly evaluated by the number of fixed defects. Although the difficulties of fixing defects differ from defects, there are few evaluations using characteristics of defects. In this research, we extracted characteristics of defects from defect reports such as priority and isReopened, and we evaluated the performance of three APR tools against 138 defects in open source Java projects. The investigation revealed that jGenProg and Nopol were able to fix defects with high priority at a high rate. In addition, we evaluated the performance of the APR tools from various viewpoints such as fixing time and fixing LOCs.**

I. Introduction

In software development, debugging is a cost-consuming activity. There is a report that an estimated debugging cost is approximately 300 billion US dollars a year in the world, and about half of programming time is spent on debugging [1]. Thus, reducing debugging cost is mandatory. In recent years, research on automated program repair (APR) has been actively carried out [2]–[9].

APR takes a buggy program and a test suite as its inputs and generates a fixed version of the program, which passes all the given tests. The performance of APR is mainly evaluated by the number of fixed defects. Additional evaluations such as fixing time [2], readability [10], or patch simplicity [11] have been carried out. Although the difficulties of fixing defects differ from defects, there are few evaluations using characteristics of defects.

In this research, we evaluate three APR tools using the characteristics of defects. We use the following characteristics obtained from defect reports in JIRA[1] and Github[2].

- *priority*
- *isReopened*

When a defect has *high priority*, developers need to fix it as soon as possible. *Reopened* means that fixing the defect has failed at least once. If the APR tools can fix *high-priority* and/or *reopened* defects, it will significantly contribute to developers.

We manually identified *priority* and *isReopened* for 138 defects, which are included in Defects4J [12] dataset, by investigating their defect reports in JIRA and Github. Furthermore, we associated the defects with their success/failure of three APR tools: jGenProg [5], jKali [5], and Nopol [6]. We investigated the relationship between the performance of the APR tools and the characteristics of defects. The main contributions of this research are as follows.

- We revealed that jGenProg and Nopol fixed many *high-priority* defects.
- We revealed that Nopol was not able to fix *reopened* defects at all.
- We evaluated the performance of APR tools from various viewpoints such as fixing time and fixing LOCs.

The remainder of this paper is organized as follows: Section II introduces bug tracking systems and automated program repair as our research background, describes our research purpose, and shows our research questions. Section III describes the investigation method and our investigation targets. Section IV shows our findings and our answers to the research questions. Section V discusses the results of the investigation and describes additional investigation. Section VI describes the threats to internal and external validity in the investigation. Finally, we conclude this paper and describe our future works in section VII.

II. Background

A. Bug Tracking System

Bug tracking system (BTS) tracks debugging process of reported defects in software development. BTS allows unified management of defects, developer assignment to defects, and recording discussion for defects. In BTS, a defect is registered as a defect report. A defect report includes the following various records:

- a title of a given defect,
- detailed description of a given defect,
- a reporter of a given defect,
- developers assigned to a given defect,
- a life cycle of a given defect, and
- *priority* of a given defect.

When a given defect has *high priority*, developers need to fix it as soon as possible. A defect report has a life cycle, and all events related to a given defect are recorded in it. For example, events contain followings:

- changing status of the defect report,
- assigning developers to the defect, and

[1]https://ja.atlassian.com/software/jira
[2]https://help.github.com/articles/about-issues

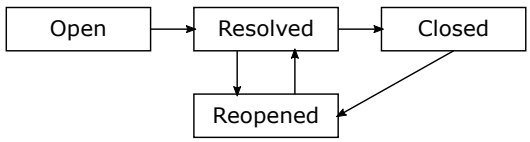

Fig. 1. A Simplified Life Cycle of a Refect Report on JIRA

- recording commits for the defect.

Figure 1 shows a simplified life cycle of a defect report in JIRA. Usually, a defect report is opened by a reporter or developers, resolved in several ways such as "Fixed", "Duplicate", and "Won't Fix", and closed when resolving is confirmed by the reporter or the developers. However, if there is a problem in resolving or if the same problem is also found again after getting closed, the defect report gets *reopened*. If a defect report gets *reopened*, developer's workloads get increased and some users of the software may become untrusting it [13]. There is a research study that *reopened* defect reports have a longer time to be closed than *non-reopened* ones [14].

B. Automated Program Repair

Automated program repair (APR) is a promising way to support debugging, and many APR techniques have been proposed [2]–[9]. There are also techniques to predict whether APR should be used [15] and empirical studies for APR techniques which use program synthesis [16]. APR is a generic term for techniques of inputting a buggy program, fixing source code, and outputting a fixed program.

Before now, APR techniques have been evaluated with metrics such as the number of fixed defects, the fixing time, and the fixing quality. Martinez et al.'s research [17] showed the number of fixed defects and the fixing time by running the APR tools: jGenProg [5], jKali [5], and Nopol [6] against Defects4J [12], a dataset of defects in open source Java projects. Table I shows the number of fixed defects in the Apache Math Project[3] in Defects4J.

C. Research Motivation

Some defects should be fixed immediately or need additional debugging due to incomplete fixing. If given software is managed with JIRA, we can extract *priority* and *isReopened* from its defect reports. *Priority* in JIRA is in order of

$$\text{Blocker} > \text{Critical} > \text{Major} > \text{Minor} > \text{Trivial}.$$

If the status of the defect report becomes "Reopened" at least once, the *isReopened* of the defect report is Yes. In our pilot investigation, we mapped *priority* and *isReopened* for each of the defects in the Apache Math Project, which is managed with JIRA, to their success/failure of the APR tools including jGenProg, jKali, and Nopol.

Table II shows the number of fixed defects in the Apache Math Project from the viewpoints of *priority* and *isReopened*. Focusing on the *priority*, we can see that jGenProg and jKali succeeded in fixing a Blocker defect. jGenProg and jKali have also fixed a *reopened* defect. In this way, it is possible to evaluate APR tools by using the characteristics of defects such as *priority* and *isReopened*. However, the above pilot investigation is an analysis with small data, and investigations with larger data are necessary to make a reasonable evaluation.

The purpose of this research is to evaluate APR tools using characteristics of defects. As a way of obtaining characteristics of defects, we use defect reports in BTS. In particular, as characteristics of defects, we use the following characteristics:

- *priority*
- *isReopened*

If the APR tools can fix *high-priority* defects that require immediate fixing, it will significantly contribute to developers. If the APR tools can fix *reopened* defects, the APR tools have ability to fix defects which are developers fail to fix.

D. Research Questions

In this research, we evaluate APR tools using characteristics of defects by answering the following two research questions.

RQ-1: Can APR tools fix *high-priority* defects?
RQ-2: Can APR tools fix *reopened* defects?

In RQ-1, we investigate whether APR tools can fix *high-priority* defects at a high rate. In this investigation, we regard a defect as a *high priority* defect when its *priority* level is Critical or Blocker in JIRA. If the APR tools can fix *high-priority* defects, it will be significantly helpful to developers.

In RQ-2, we investigate whether APR tools can fix *reopened* defects at a high rate. In this investigation, we regard a defect as *reopened* if the status of a given defect report has become "Reopened" at least once. If the APR tools can fix *reopened* defects, fixing time for *reopened* defects will get shorter.

III. Methodology

A. Investigation Method

Figure 2 shows our investigation method in this research. The investigation consists of the following two steps.

[3]http://commons.apache.org/proper/commons-math

TABLE I
The Number of Fixed Defects in the Apache Math Project

	jGenProg	jKali	Nopol	# defects
Fixed defects	18	14	21	106

TABLE II
The Number of Fixed Defects in the Apache Math Project by Priority and isReopened

	jGenProg	jKali	Nopol	# defects
Blocker	1	1	0	1
Critical	2	0	2	8
Major	11	11	15	63
Minor	4	2	4	32
Trivial	0	0	0	1
Non-priority	0	0	0	1
Reopened	1	1	0	7
Non-reopened	17	13	21	99

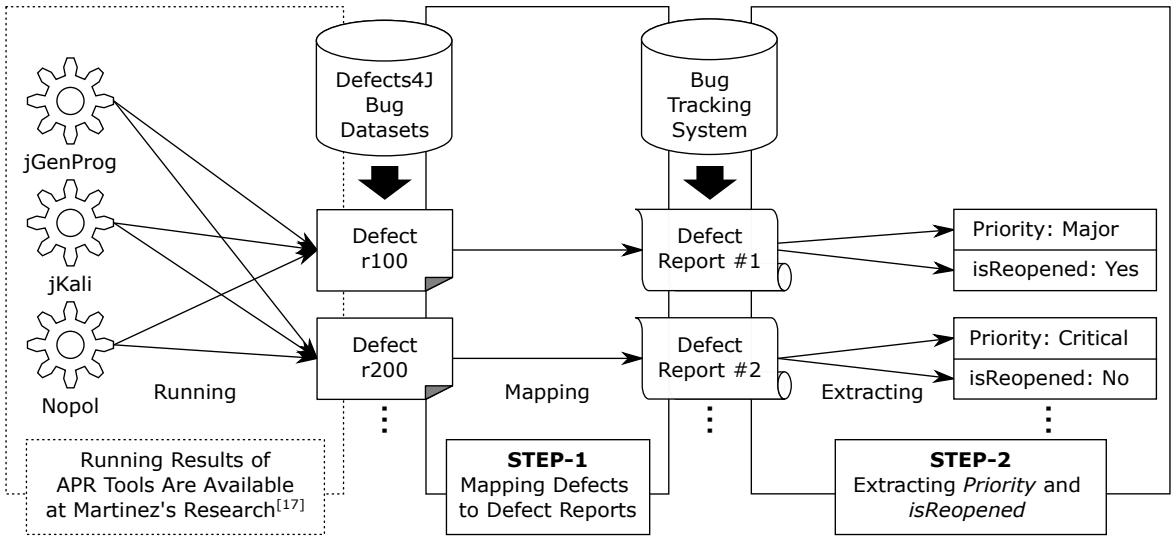

Fig. 2. The Overview of Investigation Method

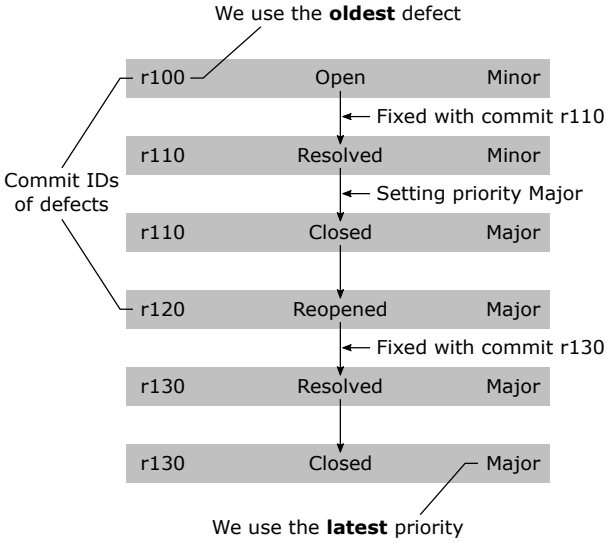

Fig. 3. An Example of Multiple Defects in A Defect Report

STEP-1: mapping defects to defect reports.
STEP-2: extracting *priority* and *isReopened* from defect reports.

In STEP-1, we match defects in the Defects4J dataset with their corresponding JIRA/Github defect report. We assume that each defect is extracted from version control system (VCS), and a commit log in VCS can be referenced from a defect report. Under the assumption, we can obtain a commit ID (e.g. r100) of a defect and we can map the defect to its defect report. However, in general, a defect report and commit IDs of defects are in a one-to-many relationship because a complex debugging process or a *reopened* defect may result in multiple commits. In this investigation, when multiple defects correspond to a certain defect report like Figure 3, we use only the oldest one. Using a new defect is not appropriate for the investigation because the defect was generated after *reopened* and the defect itself was not *reopened*.

In STEP-2, we extract *priority* and *isReopened* from each defect report. *Priority* is described in a defect report. However, developers may change the *priority* of the defect report like Figure 3. In this investigation, we use the latest *priority* described in the defect report because we think that the latest *priority* most reflects the *priority* decided by developers. *isReopened* can be decided from the life cycle in the defect report. If *"Reopened"* is prepared as the status of the defect report as in JIRA, it is easy to decide whether the defect is *isReopened*. On the other hand, if *"Reopened"* is not prepared in advance as in Github, we regard a defect report which has multiple "Closed" as *reopened*.

B. Subject systems

Our subject systems are open source Java projects in Defects4J dataset including defects. We use three projects, the Apache Lang[4] (hereinafter called "Lang"), the Apache Math (hereinafter called "Math"), and the Joda-Time[5] (hereinafter called "Time"), which were also used in the investigation of Martinez et al [17]. The subject systems in this research are as shown in Table III. In Time project, since there is no *priority* item in Github's defect report, *priority* cannot be extracted.

[4]http://commons.apache.org/proper/commons-lang
[5]http://www.joda.org/joda-time

TABLE III
SUBJECT SYSTEMS

Project	# Defects	BTS	Priority	isReopened
Lang	65	JIRA	Available	Available
Math	106	JIRA	Available	Available
Time	12	Github	Unavailable	Available

In our investigation, we did not use JFreeChart[6] and Closure Compiler[7] in Defects4J as subject systems. In JFreeChart, we were not able to match the defect reports in the BTS with the commits of the version control system. Since the test cases attached to Closure Compiler are not for JUnit, they can not be handled by current jGenProg, jKali, and Nopol [17].

IV. Investigation Results

Table IV shows the results of classifying the number and the success rate(%) of fixed defects of each tool by *priority* and *isReopened*. Success rate means the percentage of the number of fixed defects in the total number of defects for each cell of Table IV. Furthermore, we show the success rate of fixing defects by *priority* and *isReopened* in Figure 4. In our definition, when the success rate of fixing *specific-priority* defects is higher than of fixing overall defects, the APR tools can fix *specific-priority* defects at a high rate. When the success rate of fixing *reopened* defects is higher than of fixing *non-reopened* defects, the APR tools can fix *reopened* defects at a high rate.

Please note that the success rate of fixing defects for Blocker and Critical defects was higher than for overall defects. However, jKali failed to fix any Critical defects. Therefore, our answer to the RQ1 is YES for jGenProg and Nopol, but NO for jKali. In addition, Trivial defects were not fixed by any tools.

On the other hand, the success rate of fixing defects did not change very much from the viewpoint of *isReopened*. Nopol has not been able to fix any *reopened* defects. Therefore, our answer to the RQ2 is NO for any tools.

In addition to the above, we investigated the fixing time when each tool succeeded in fixing the defect. Figure 5 shows the mean time of fixing defects from the viewpoint of *isReopened*. We calculated the mean time for each cell of Table IV. For example, in the cell of Nopol and Non-reopened, the mean time was calculated from 29 defects. From these results, in all the tools, we can see that the mean time of fixing defects for Minor is longer than the overall mean. In jGenProg and jKali, the mean time of fixing defects for *reopened* is longer than that for *non-reopened*. Furthermore, Nopol was not able to fix any *reopened* defects.

This investigation differs from our pilot investigation in the following points:

- increasing the number of subject systems from one (Math) to three (Lang, Math, and Time), and
- evaluating the APR tools using success rate instead of the number of fixed defects.

V. Discussion

As the results of our investigation, We found that jGenProg and Nopol fixed many *high-priority* defects, but all the APR tools were not able to fix many *reopened* defects. In particular, Nopol was not able to fix any *reopened* defects. From the viewpoint of *fixing time*, all the APR tools take much time to fix Minor or *reopened* defects.

In addition, we focused on the amount of change in source code in case of successful fixing. When an APR tool succeeds to fix a defect, we can obtain a part of fixing as a difference of the source code. The source code difference is obtained in the Unix Diff format, and it consists of added lines and deleted lines. We counted added/deleted lines separately. Figure 6 shows the mean added/deleted lines of each tool. We calculated the mean added/deleted lines for each cell of Table IV. From Figure 6, we can see that the mean added/deleted lines of fixing defects for Minor is higher than the overall mean. The fact that added/deleted lines of Minor defects are large can be a factor of its low success rate and long fixing time. In *reopened* defects, the added lines of jKali are much larger than others. From these results, an increase in the added/deleted lines may lead to making the fixing time longer.

We evaluated the APR tools with characteristics of defects. Using characteristics of defects is a new viewpoint of evaluating APR techniques. We suggest that researchers use not only the number of fixed defects but also characteristics of defects when evaluating APR techniques. Using characteristics of defects makes their evaluations more multidirectional. We are add investigation targets and consider more characteristics of defects to make our findings more general.

VI. Threats to Validity

A. Threats to internal validity

In this research, we used the execution time of the APR tools to evaluate fixing time, but we did not consider the execution time for failed defects. Please note that the execution time for failed defects is affected by the tool's timeout setting. If we do not set the timeout limitation, there are tools which do not stop the operation within realistic time in some cases [2], [5]. For this reason, we were aware of the fact that we cannot consider the exact execution time of the APR tools, and we only used the execution time when fixing was succeeded.

In this research, characteristics of the defects are extracted from defect reports. Since defect reports and their *priority* differ by reporters who have different understanding of *priority*, there is a possibility that *priority* is mislabeled in some defect reports.

In our evaluation of fixing time and fixing LOCs, because of small sample size, we carried out direct comparison of average values instead of statistical comparison. For statistical comparison, it is necessary to increase defects.

B. Threats to external validity

In our investigation target, Major defects occupied 55% or more. On the other hand, the ratio of Critical or Blocker defects is about 6%, and the number of *high-priority* defects was insufficient. Therefore, it is necessary to investigate more *high-priority* defects.

The ratio of *reopened* defects was about 6%. The results support Mi et al.'s findings [14] against Bugzilla[8] that the

[6]http://jfree.org/jfreechart

[7]https://developers.google.com/closure/compiler

[8]https://www.bugzilla.org

TABLE IV
THE NUMBER AND THE SUCCESS RATE(%) OF FIXED DEFECTS BY PRIORITY AND ISREOPENED

	Blocker	Critical	Major	Minor	Trivial	Non-priority	Reopened	Non-reopened	Total
jGenProg	1 (33%)	2 (25%)	11 (11%)	4 (9%)	0 (0%)	2 (17%)	1 (9%)	19 (11%)	20 (11%)
jKali	1 (33%)	0 (0%)	11 (11%)	2 (5%)	0 (0%)	2 (17%)	1 (9%)	15 (9%)	16 (9%)
Nopol	1 (33%)	2 (25%)	21 (21%)	4 (9%)	0 (0%)	1 (8%)	0 (0%)	29 (17%)	29 (16%)
# Defects	3	8	101	44	1	12	11	172	183

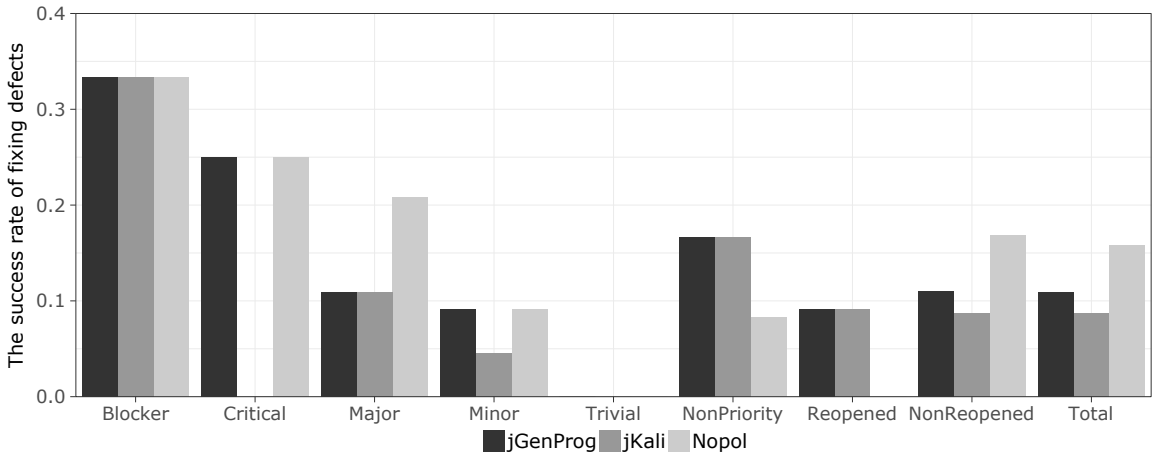

Fig. 4. The Success Rate of Fixing Defects by Priority and isReopened

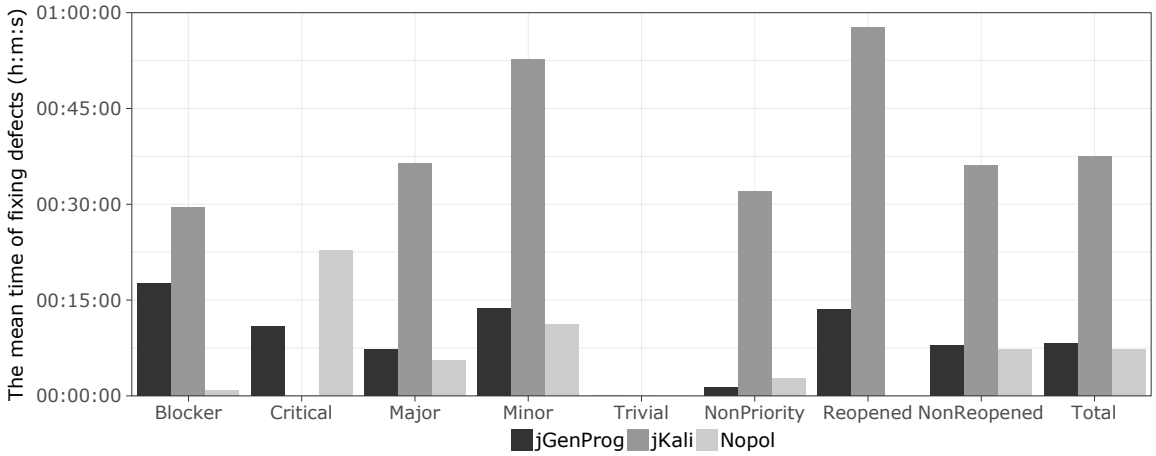

Fig. 5. The Mean Time of Fixing Defects by Priority and isReopened

percentage of *reopened* defects is 6∼10%. However, the number of *reopened* defects was insufficient. Therefore, it is necessary to investigate more *reopened* defects.

Our evaluation does not consider differences in projects. To evaluate by project, the number of defects in each category is too small. Therefore, it is necessary to increase defects.

Our investigation targets are Java defect dataset. Therefore, if someone conducts similar investigations on other programming languages, the investigation results may be different.

The characteristics of the defects used in this research are only *priority* and *isReopened*. By adding more characteristics, it is possible to clarify the relationship between the characteristics of defects and the performance of APR tools.

VII. CONCLUSIONS

In this research, we manually identified *priority* and *isReopened* for each of the 138 defects, which are included in open source Java projects, by investigating their defect reports in bug tracking system. Furthermore, we associated the defects with their success/failure of three APR tools: jGenProg, jKali, and Nopol. We investigated the relationship between the performance of the APR tools and the characteristics of defects.

As a result, we revealed that jGenProg and Nopol fixed many *high-priority* defects, but Nopol was not able to fix the *reopened* defects much. We also evaluated the performance of

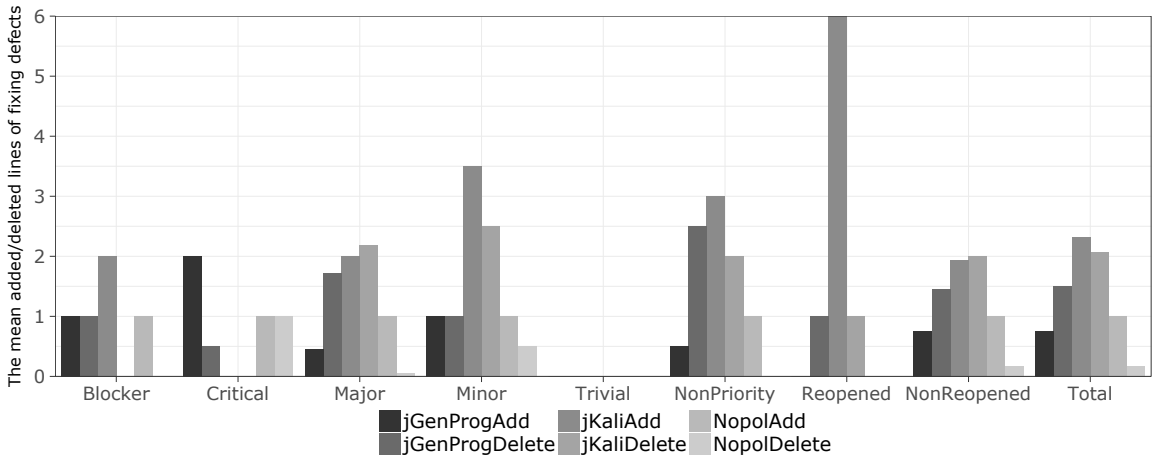

Fig. 6. The Mean Added/Deleted Lines of Fixing Defects by Priority and isReopened

the APR tools from the viewpoint of fixing time and fixing LOCs. We also manually inspected the commit in the Trivial defect reports which APR tools were not able to fix at all.

From the findings obtained from the investigation, we suggest that APR tools can be used in the following scenes of actual software development.

- When developers find a *high-priority* defect, they first run jGenProg or Nopol. The APR tools can fix the defect with high probability.
- Before developers commit their debugging work to version control system, if they see differences in their source code and an APR tool's one, they can get the opportunity of finding mistakes leading to *reopened*.

Our future research topics are as follows:

- using more *high-priority* and *reopened* defects for the same kinds of investigations,
- considering more characteristics of defects,
- conducting more experiment with different programming languages and APR tools, and
- discussing why a certain repair tool can fix a certain kind of defect.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP25220003.

REFERENCES

[1] T. Britton, L. Jeng, G. Carver, P. Cheak, and T. Katzenellenbogen, "Reversible debugging software," University of Cambridge, Judge Business School, Tech. Rep., 2013.

[2] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in *Proceedings of the 34th International Conference on Software Engineering*, 2012, pp. 3–13.

[3] F. Long and M. Rinard, "Staged program repair with condition synthesis," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 166–178.

[4] S. Mechtaev, J. Yi, and A. Roychoudhury, "Angelix: Scalable multiline program patch synthesis via symbolic analysis," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 691–701.

[5] M. Martinez and M. Monperrus, "Astor: A program repair library for java," in *Proceedings of the 25th International Symposium on Software Testing and Analysis, Demonstration Track*, 2016, pp. 441–444.

[6] J. Xuan, M. Martinez, F. DeMarco, M. Clement, S. L. Marcote, T. Durieux, D. L. Berre, and M. Monperrus, "Nopol: Automatic repair of conditional statement bugs in java programs," *IEEE Transactions on Software Engineering*, 2016.

[7] X. B. D. Le, D. Lo, and C. Le Goues, "History driven program repair," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, vol. 1, 2016, pp. 213–224.

[8] F. Long and M. Rinard, "Automatic patch generation by learning correct code," in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2016, pp. 298–312.

[9] X. B. D. Le, Q. L. Le, D. Lo, and C. Le Goues, "Enhancing automated program repair with deductive verification," in *2016 IEEE International Conference on Software Maintenance and Evolution*, 2016, pp. 428–432.

[10] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 802–811.

[11] S. Mechtaev, J. Yi, and A. Roychoudhury, "Directfix: Looking for simple program repairs," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. Proceedings of the 2015 International Conference on Software Engineering, 2015, pp. 448–458.

[12] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 437–440.

[13] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. i. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," in *2010 17th Working Conference on Reverse Engineering*, 2010, pp. 249–258.

[14] Q. Mi and J. Keung, "An empirical analysis of reopened bugs based on open source projects," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 37:1–37:10.

[15] X.-B. D. Le, T.-D. B. Le, and D. Lo, "Should fixing these failures be delegated to automated program repair?" in *2015 IEEE 26th International Symposium on Software Reliability Engineering*, 2015, pp. 427–437.

[16] X. B. D. Le, D. Lo, and C. Le Goues, "Empirical study on synthesis engines for semantics-based program repair," in *2016 IEEE International Conference on Software Maintenance and Evolution*, 2016, pp. 423–427.

[17] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, "Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset," *Springer Empirical Software Engineering*, pp. 1–29, 2016.