2014 6th International Workshop on Empirical Software Engineering in Practice

A Gamified Tool for Motivating Developers to Remove Warnings of Bug Pattern Tools

Satoshi Arai
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
www.31o4-xy@fuji.waseda.jp

Kazunori Sakamoto
National Institute of Informatics
Tokyo, Japan
exkazuu@nii.ac.jp

Hironori Washizaki
and Yoshiaki Fukazawa
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
{washizaki, fukazawa}@waseda.jp

***Abstract*—Static analysis tools such as bug pattern tools are useful to detect bugs early in software development. However, existing tools sometimes yield so many warnings that developers tend to ignore such warnings.**

To deal with this problem, we propose a gamified tool for motivating developers to remove such warnings. Our tool employs the gamification technique that calculates points by counting removed warnings with respect to each developer and each team. The points give developers feedback and urge them to compete with each other. We confirmed that developers removed about 150% warnings with our tool in comparison with the case where they did not use our tool through an experiment.

I. Introduction

Bug pattern tools detect code fragments that seem to be faulty or to cause faults as warnings by analyzing source code. The automated detection can reduce costs of code inspection, and it can improve code quality [1]. There are many existing tools, for example, FindBugs [2] and PMD [1] are bug pattern tools supporting Java.

Although the effectiveness of bug pattern tools has already been evaluated in various research studies [3]–[5], existing tools sometimes yield so many warnings that developers tend to ignore all the warnings [6], [7]. To deal with this problem, some researchers devote to improve the precision of the detection [6], [8], [9]. However, it is very difficult to remove all false positives.

As another approach, we propose a gamified tool, named Game-based Bug Catcher (GBC), for motivating developers to remove such warnings. GBC makes it more fun to remove warnings of FindBugs with a gamification technique. Gamification is defined as a technique to apply game design techniques to non-game experiences [10], [11]. GBC calculates points on the basis of the number of warnings developers removed. The points give developers feedback and urge them to compete with each other. Because remaining warnings lead developers to ignore warnings, it is very important to remove all warnings including false positives. Thus, GBC does not distinguish real faults from false positives.

We investigate two research questions (RQs) as follows:

- RQ1: Can we apply the gamification technique to existing bug pattern tools?
- RQ2: How effectively GBC can reduce warnings reported by FindBugs?

The contributions of this paper are:

- A gamified tool that motivates developers to remove warnings.
- The results of an experiment that indicate that our tool overall reduced warnings.

II. Problems in Existing Bug Pattern Tools

In this section, we introduce two bug pattern tools: FindBugs and PMD explaining one of the most significant problem, too many warnings, in existing bug pattern tools.

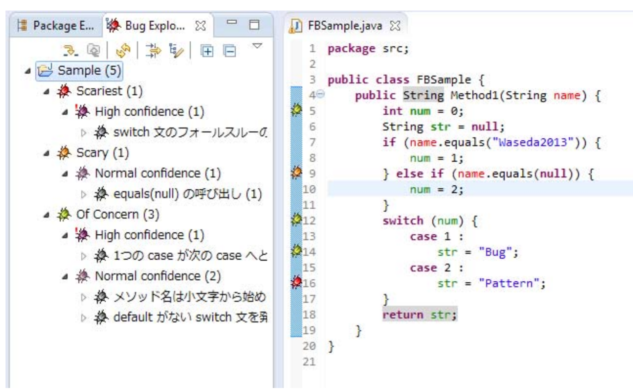


Fig. 1. Screenshot where FindBugs detected five warnings

GBC uses FindBugs that is a bug pattern tool supporting Java [2]. FindBugs is so famous that many studies refer to FindBugs [6], [9], [12]. There are FindBugs plug-ins for Eclipse and Jenkins. Figure 1 shows a screenshot of the FindBugs plug-in for Eclipse. The screenshot shows source code with five warnings reported by FindBugs. FindBugs reports warnings that consist of the name, description, category, and rank.

The source code consists of 17 lines and surprisingly contains a warning per 3.4 lines. These warnings are able to be both true positives and false positives. When warnings occur in this frequency, a large project may bother with too many warnings. The source code is not real code, and this calculation

[1] PMD, http://pmd.sourceforge.net/.

978-1-4799-6666-0/14 $31.00 © 2014 IEEE
DOI 10.1109/IWESEP.2014.17

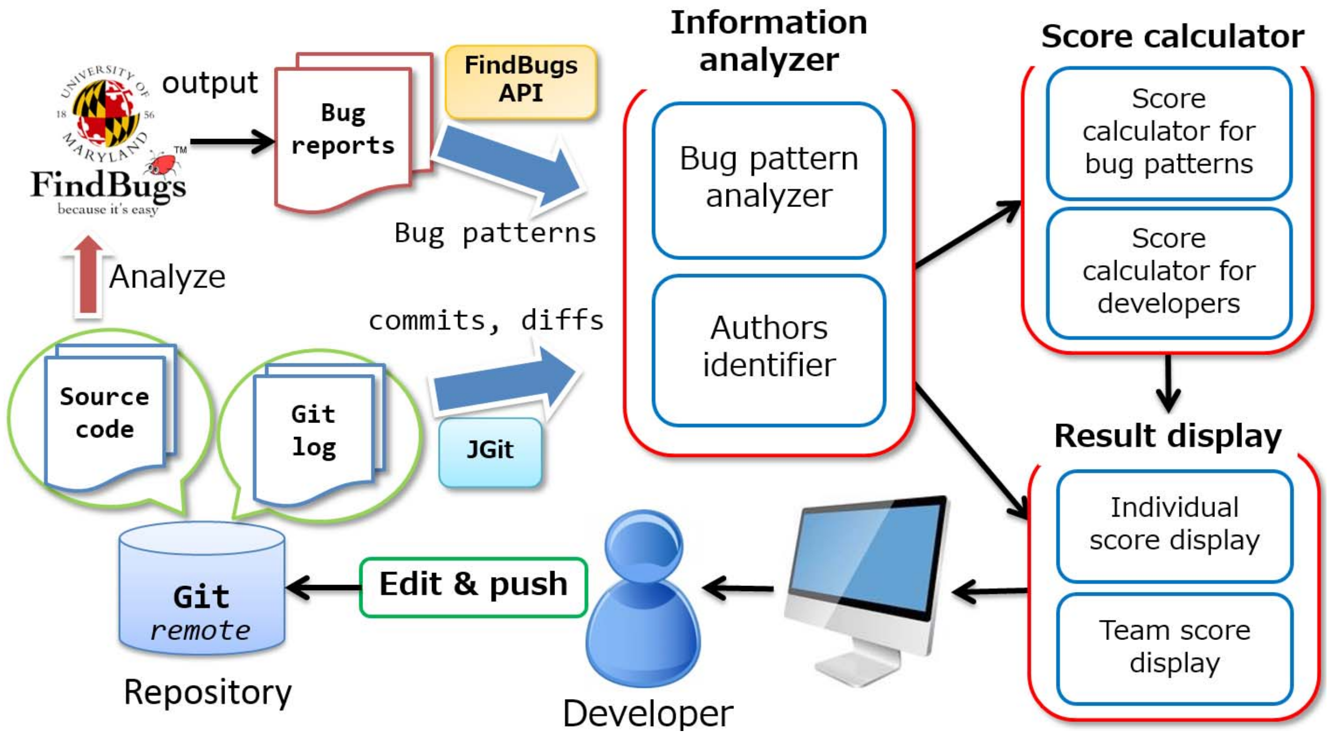


Fig. 3. Overview of GBC

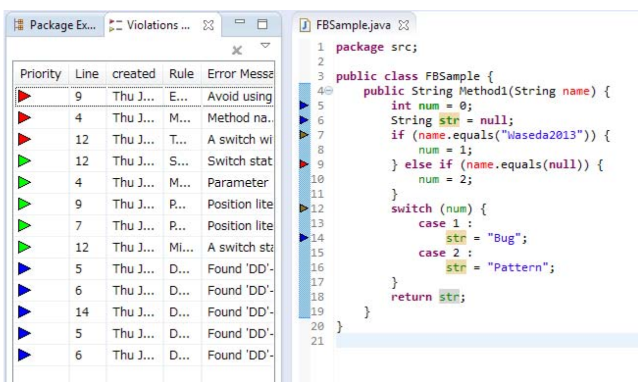


Fig. 2. Screenshot where PMD detected five warnings

cannot be directly applied to real software development, but it is still not a trivial example.

PMD is another bug pattern tool supporting Java, which is famous similarly to FindBugs. FindBugs detects warnings from Java bytecode, in contrast, PMD detects them from Java source code. Figure 1 shows a screenshot of the PMD plug-in for Eclipse. The screenshot shows source code with thirteen warnings reported by PMD. PMD tends to detect more warnings due to a different algorithm and rule set from them of FindBugs.

The problems of static analysis tools including bug pattern tools are found by various researchers. One of the most significant problems is too many warnings due to false positives as the example [6], [7], [13]–[16]. To deal with the problem, there already exists various approaches such as an improvement in the detection accuracy by enhancing algorithms or by combining multiple tools [13]. Although these approaches partially alleviate the problem, it is very difficult to remove false positives completely.

III. Gamified Tool for Motivating Developers to Remove Warnings

We developed GBC for motivating developers to remove warnings as another approach to deal with the problem. Figure 3 shows the overview of GBC. GBC finds who wrote the code fragments corresponding to each warning and who fixed the code fragments to remove corresponding warnings. GBC calculates points by counting removed or added warnings with respect to each developer and team.

GBC consists of three components: the information analyzer, score calculator, result display. The information analyzer consists of the bug pattern analyzer and author identifier. The bug pattern analyzer analyzes warnings reported by FindBugs and understands which bug patterns each warning corresponds to. The author identifier identifies who wrote or fixed the code fragment corresponding to each bug pattern. The score calculator consists of the two score calculators for bug patterns and developers. The score calculators calculate the point of each warning and the total point of each developer, respectively. The result display consists of the individual score display and team score display. The score displays show the point of each developer and the point of each team, respectively. Figures 4 and 5 show the individual score and group score displays.

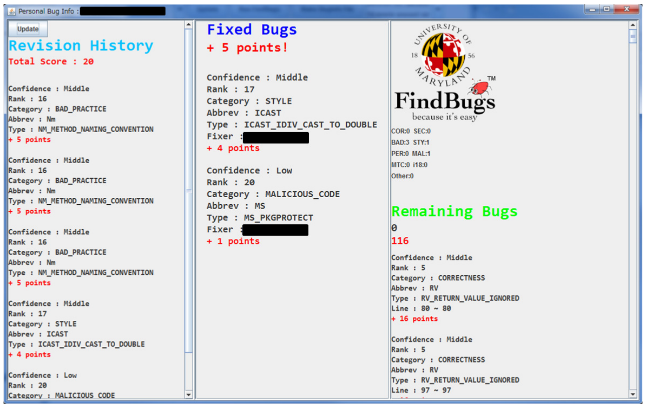


Fig. 4. Screenshot of individual score display

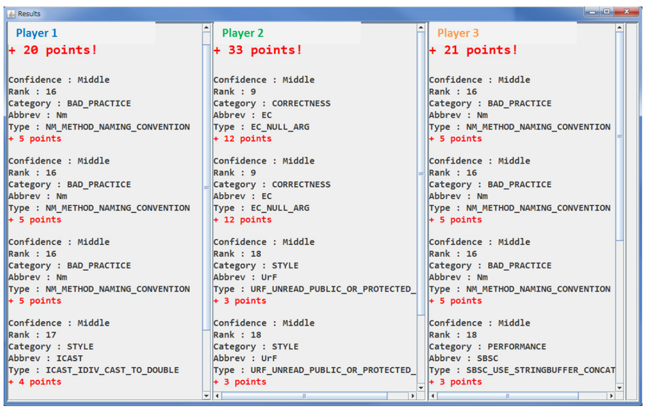


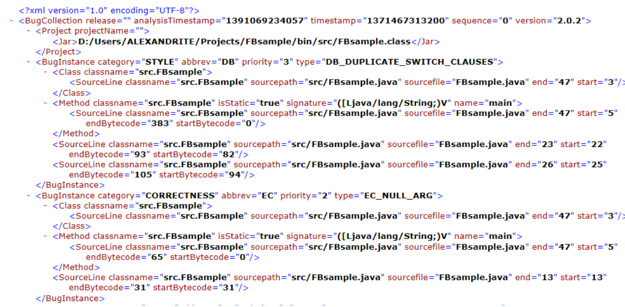
Fig. 5. Screenshot of group score display

A. Bug Pattern Analysis using FindBugs and Git

We employed FindBugs to detect code fragments corresponding to bug patterns. The reason why we selected FindBugs is because FindBugs can show warnings with their kind and importance which are useful to calculate scores. Moreover, Thung et al. showed that FindBugs had the high reliability of detecting warnings [6]. FindBugs can show results on Eclipse such as Figure 1 and can output them as XML files.

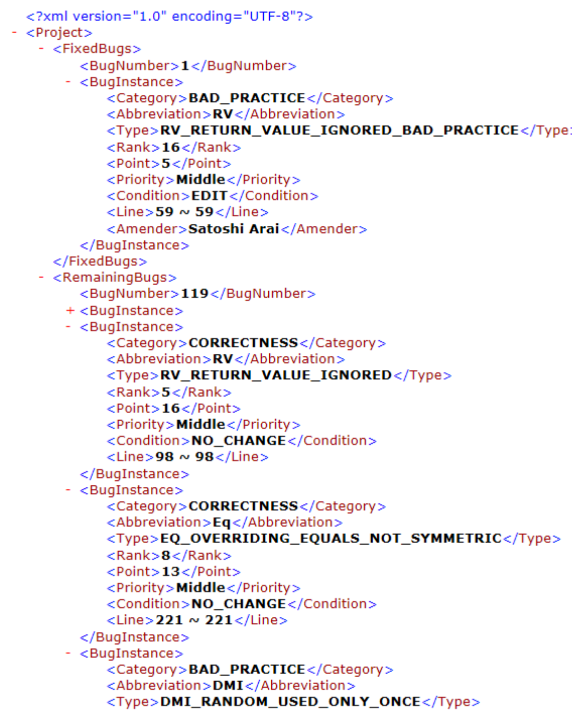
Figure 6 shows a fragment of the XML file generated by FindBugs. The XML file contains the category of a warning (the `category` attribute), the name of a warning (the `type` attribute), the location information of a warning in source code (the `start` and `end` attributes) and the relative path of the source code (the `sourcefile` attribute). GBC parses the XML file generated by FindBugs and retrieve information required to calculate scores. Moreover, GBC also retrieves other information which does not appear in the XML file from FindBugs API [2].

After parsing the XML file such as Figure 6, GBC generates a new XML file such as Figure 7. The new XML file can be easily manipulated by GBC. This new XML file contains analysis results of Git, for example, developer information who write code related to a warning (the `Amendar` attribute). This



```
<?xml version="1.0" encoding="UTF-8"?>
- <BugCollection release="" analysisTimestamp="1391069234057" timestamp="1371467313200" sequence="0" version="2.0.2">
  - <Project projectName="">
      <Jar>D:/Users/ALEXANDRITE/Projects/FBsample/bin/src/FBsample.class</Jar>
    </Project>
  - <BugInstance category="STYLE" abbrev="DB" priority="3" type="DB_DUPLICATE_SWITCH_CLAUSES">
    - <Class classname="src.FBsample">
        <SourceLine classname="src.FBsample" sourcepath="src/FBsample.java" sourcefile="FBsample.java" end="47" start="3"/>
      </Class>
    - <Method classname="src.FBsample" isStatic="true" signature="([Ljava/lang/String;)V" name="main">
        <SourceLine classname="src.FBsample" sourcepath="src/FBsample.java" sourcefile="FBsample.java" end="47" start="5"
          endBytecode="383" startBytecode="0"/>
      </Method>
      <SourceLine classname="src.FBsample" sourcepath="src/FBsample.java" sourcefile="FBsample.java" end="23" start="22"
        endBytecode="93" startBytecode="82"/>
      <SourceLine classname="src.FBsample" sourcepath="src/FBsample.java" sourcefile="FBsample.java" end="26" start="25"
        endBytecode="105" startBytecode="94"/>
    </BugInstance>
  - <BugInstance category="CORRECTNESS" abbrev="EC" priority="2" type="EC_NULL_ARG">
    - <Class classname="src.FBsample">
        <SourceLine classname="src.FBsample" sourcepath="src/FBsample.java" sourcefile="FBsample.java" end="47" start="3"/>
      </Class>
    - <Method classname="src.FBsample" isStatic="true" signature="([Ljava/lang/String;)V" name="main">
        <SourceLine classname="src.FBsample" sourcepath="src/FBsample.java" sourcefile="FBsample.java" end="47" start="5"
          endBytecode="65" startBytecode="0"/>
      </Method>
      <SourceLine classname="src.FBsample" sourcepath="src/FBsample.java" sourcefile="FBsample.java" end="13" start="13"
        endBytecode="31" startBytecode="31"/>
    </BugInstance>
```

Fig. 6. Fragment of XML file generated by FindBugs



```
<?xml version="1.0" encoding="UTF-8"?>
- <Project>
  - <FixedBugs>
      <BugNumber>1</BugNumber>
    - <BugInstance>
        <Category>BAD_PRACTICE</Category>
        <Abbreviation>RV</Abbreviation>
        <Type>RV_RETURN_VALUE_IGNORED_BAD_PRACTICE</Type>
        <Rank>16</Rank>
        <Point>5</Point>
        <Priority>Middle</Priority>
        <Condition>EDIT</Condition>
        <Line>59 ~ 59</Line>
        <Amender>Satoshi Arai</Amender>
      </BugInstance>
    </FixedBugs>
  - <RemainingBugs>
      <BugNumber>119</BugNumber>
    + <BugInstance>
    - <BugInstance>
        <Category>CORRECTNESS</Category>
        <Abbreviation>RV</Abbreviation>
        <Type>RV_RETURN_VALUE_IGNORED</Type>
        <Rank>5</Rank>
        <Point>16</Point>
        <Priority>Middle</Priority>
        <Condition>NO_CHANGE</Condition>
        <Line>98 ~ 98</Line>
      </BugInstance>
    - <BugInstance>
        <Category>CORRECTNESS</Category>
        <Abbreviation>Eq</Abbreviation>
        <Type>EQ_OVERRIDING_EQUALS_NOT_SYMMETRIC</Type>
        <Rank>8</Rank>
        <Point>13</Point>
        <Priority>Middle</Priority>
        <Condition>NO_CHANGE</Condition>
        <Line>221 ~ 221</Line>
      </BugInstance>
    - <BugInstance>
        <Category>BAD_PRACTICE</Category>
        <Abbreviation>DMI</Abbreviation>
        <Type>DMI_RANDOM_USED_ONLY_ONCE</Type>
```

Fig. 7. Fragment of XML file generated by GBC

new XML file allows other developers to create other gamified tools for motivating users to remove warnings.

We also employed Git [3] to identify who wrote or fixed the code fragments. Git is a famous version control system that stores edit histories. GBC identifies an author of each commit by analyzing the edit histories. Figure 8 shows the result of `git log` command which describes the edit histories. GBC collects each set of warnings in two versions using FindBugs and detects which warnings are added or removed by analyzing the difference of the two sets of warnings.

B. Introduction of Gamification Technique

Gamification techniques can motivate users by adding game elements in non-game contexts. For example, Foursquare [4] uses gamification techniques. Foursquare provides a feature

[2]FindBugs API, http://findbugs.sourceforge.net/api/.

[3]Git, http://git-scm.com/.

[4]Foursquare, https://ja.foursquare.com/.

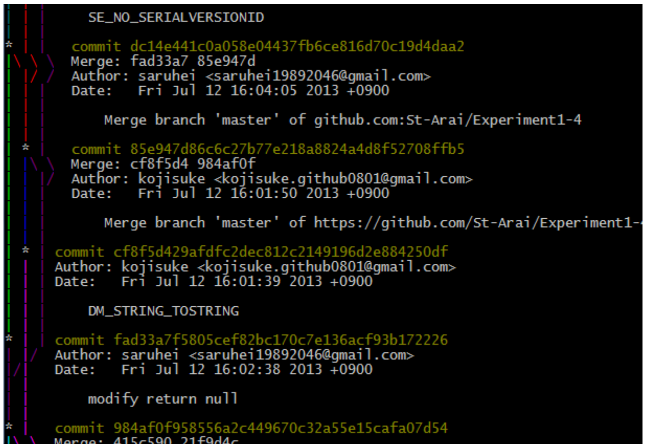


Fig. 8. Screenshot of the command `git log` on Git BASH

that shares the locations where users checked in. Foursquare gives users badges by referring to the number of the locations and the kinds of the locations. Users are motivated to use Foursquare frequently to collect badges.

Gamification techniques began to be applied to various fields including software engineering [17]–[19]. We implemented the calculator components that analyze the rank of bug patterns. FindBugs gives each bug pattern the rank that indicates how serious the bug pattern is. We define a point of a bug pattern as $21-rank$ because the rank is an integer between 0 and 21. When a developer removes a warning corresponding to a bug pattern, he/she gets the point of the bug pattern. The team point is the summation of the points of the team members. In this way, GBC encourages developers to compete with each other, thus, GBC motivates developers to remove warnings.

IV. Evaluation

To investigate RQ1 and RQ2, we conducted an experiment. We gathered six bachelor or master students who studied computer science.

A. Experimental Setup

The experiment uses two open source software (OSS) programs, namely, bukkit [5] ($S_{\rm A}$) and twitter4j-core [6] ($S_{\rm B}$), as subject programs. bukkit is a server program for a famous game software Minecraft. twitter4j-core is a core library for handling Twitter API from Java programs. bukkit and twitter4j-core have 26,917 and 18,123 lines of code, respectively.

We conducted the experiment with the following steps.

1) We divide the six students into two groups ($G_{\rm A}$ and $G_{\rm B}$).
2) We deploy the two OSS programs in our Git repository. The students acquire them and put them on their local machines.
3) $G_{\rm A}$ removes warnings in $S_{\rm A}$ with GBC within 30 minutes. We handle GBC to show the point information in another display. In contrast, $G_{\rm B}$ removes them without GBC within 30 minutes.
4) After the task for $S_{\rm A}$, $G_{\rm A}$ removes warnings in $S_{\rm B}$ without GBC within 30 minutes. In contrast, $G_{\rm B}$ removes them with GBC within 30 minutes.

[5]https://github.com/Bukkit/Bukkit
[6]https://github.com/yusuke/twitter4j

B. Results

Table I shows the results of the experiment. The columns of "Group", "OSS", "GBC", "Total", "Removed", and "Ratio" indicate that the group which removed warnings, the name of the OSS program where they removed warnings, whether they removed warnings with GBC or without GBC, the total number of the warnings that FindBugs reported, the number of the warnings removed by the corresponding group, and the ratio of the removed warnings.

Figures 9 and 10 show the numbers of removed warnings with respect to each group. We cannot compare the results of $G_{\rm A}$ and $G_{\rm B}$ because they have not the same skills. Thus, we compare the results in the same group.

TABLE I. Removed warnings of FindBugs with or without GBC

Group	OSS	GBC	Total	Removed	Ratio
$G_{\rm A}$	$S_{\rm A}$	Yes	168	26	15.48%
	$S_{\rm B}$	No	150	13	8.67%
$G_{\rm B}$	$S_{\rm A}$	No	168	41	24.40%
	$S_{\rm B}$	Yes	150	59	39.33%

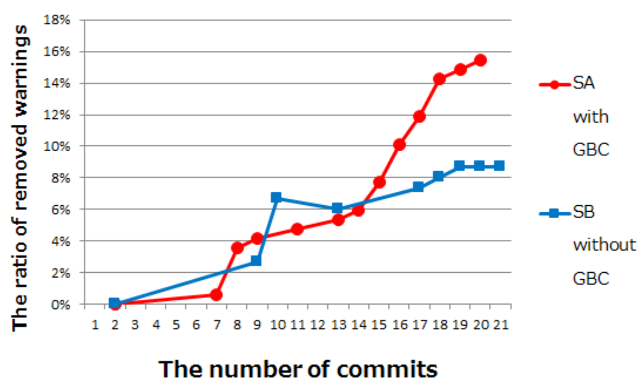


Fig. 9. Experiment results of $\rm G_A$

The results show that both $G_{\rm A}$ and $G_{\rm B}$ with GBC removed more warnings than the case where they removed warnings without GBC. They also show that the ratios of removed warnings with GBC are bigger than the case where they removed warnings without GBC. Note that we cannot compare $G_{\rm A}$ and $G_{\rm B}$ because students in $G_{\rm B}$ has better skills than them in $G_{\rm A}$ and then $G_{\rm B}$ always removed more warnings than $G_{\rm A}$. Thus, we can conclude that GBC motivated developers to remove warnings, and we can answer yes to RQ1.

We also collected information from the six students through interviews and questionnaires. Five of them said that removing warning with GBC is better than removing warning without GBC. In the interview, a student said that he tried to fix source code aggressively because GBC motivated him to remove

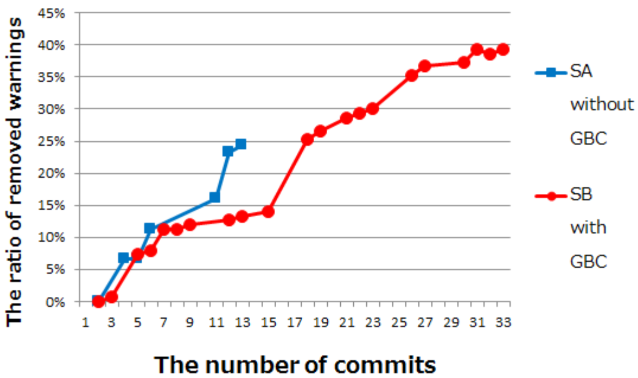


Fig. 10. Experiment results of G_B

warnings. Another student said that he prioritized the number of removed warnings rather than the correctness and accuracy because GBC urged to remove warnings quickly. For the case where they did not use GBC, a student said that he did not want to remove them because it was a tedious task to remove warnings. In contrast, another student said that he could concentrate on removing warnings because a factor such as GBC did not urge him to compete with others. Overall, GBC is effective to motivate developers to remove warnings and we can answer yes to RQ2, but we should keep in mind that our gamification technique were not effective to all students who do not like the competition.

V. Threats to Validity

Some students have not strong skills of FindBugs and Git. They may learn something through the experiment, and it may have an affect on the results. However, both groups removed more warnings when they used GBC. Thus, we can ignore such learning.

The experiment setting has two differences from real software development. 1) The examinees removed warnings in the OSS programs that they did not write. 2) The examinees removed warnings within only 30 minutes. To validate GBC more carefully, we plan to apply GBC to a long-term software development.

VI. Related work

Thung et al. compares three static analysis tools, FindBugs, PMD, and Jlint in terms of false negatives [6]. They applied the three tools to three OSS programs and found that FindBugs and PMD are more effective to prevent false negatives than Jlint. However, there are still false detection problems in all the three tools.

Nanda et al. developed a new tool that executes multiple static analysis tools to deal with the costs of selecting tools and the problem of too many warnings [13]. Their tool collects feedback from users to improve the detection accuracy. In contrast, our approach motivates developers to remove warnings instead of the improvement in the detection accuracy.

Singer et al. proposed a gamified tool that adds gamification techniques into Git to motivate developers to commit change history more frequently [17]. Their tool observes commits and reports the summary of the commits on Web pages and e-mails. They confirmed that their tool can successfully motivate developers due to the competition, but the competition bothered some developers. Our approach employs similar gamification techniques, and we got similar experimental results.

VII. Conclusion

Bug pattern tools are effective but have some problems. One of the most significant problems is that they yield sometimes too many warnings to make developers hesitate to remove them. To deal with the problem, we propose a gamified tool, named GBC, to motivate developers to remove warnings of FindBugs. GBC urges developers to remove warnings through the competition by counting the number of removed warnings with respect to each developer and team. We conducted the experiment and confirmed that developers removed more warnings when they used GBC.

In future work, we will add other gamification techniques not related to the competition into GBC because there are some people who do not like the competition. We also plan to conduct a more realistic and long-term experiment similar to real software development to deal with threats to validity.

Acknowledgment

This material is based upon work partially supported by HAYAO NAKAYAMA Foundation for Science & Technology and Culture.

References

[1] S. Wagner, J. Jürjens, C. Koller, and P. Trischberger, "Comparing bug finding tools with reviews and tests," in *Testing of Communicating Systems*, ser. Lecture Notes in Computer Science, F. Khendek and R. Dssouli, Eds. Springer Berlin Heidelberg, 2005, vol. 3502, pp. 40–55. [Online]. Available: http://dx.doi.org/10.1007/11430230_4

[2] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *SIGPLAN Not.*, vol. 39, pp. 92–106, December 2004.

[3] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou, "Evaluating static analysis defect warnings on production software," in *Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, ser. PASTE '07. New York, NY, USA: ACM, 2007, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/1251535.1251536

[4] N. Rutar, C. Almazan, and J. Foster, "A comparison of bug finding tools for java," in *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, Nov 2004, pp. 245–256.

[5] S. Wagner, F. Deissenboeck, M. Aichner, J. Wimmer, and M. Schwalb, "An evaluation of two bug pattern tools for java," in *Software Testing, Verification, and Validation, 2008 1st International Conference on*, April 2008, pp. 248–257.

[6] F. Thung, Lucia, D. Lo, L. Jiang, F. Rahman, and P. T. Devanbu, "To what extent could we detect field defects? an empirical study of false negatives in static bug finding tools," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 50–59. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351685

[7] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 672–681. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486877

[8] M. G. Nanda and S. Sinha, "Accurate interprocedural null-dereference analysis for java," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 133–143. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070515

[9] F. Rahman, S. Khatri, E. T. Barr, and P. Devanbu, "Comparing static bug finders and statistical prediction," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 424–434. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568269

[10] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification. using game-design elements in non-gaming contexts," in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '11. New York, NY, USA: ACM, 2011, pp. 2425–2428. [Online]. Available: http://doi.acm.org/10.1145/1979742.1979575

[11] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: defining "gamification"," in *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, ser. MindTrek '11. ACM, 2011, pp. 9–15.

[12] N. Rutar, C. B. Almazan, and J. S. Foster, "A comparison of bug finding tools for java," in *Proceedings of the 15th International Symposium on Software Reliability Engineering*, ser. ISSRE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 245–256. [Online]. Available: http://dx.doi.org/10.1109/ISSRE.2004.1

[13] M. G. Nanda, M. Gupta, S. Sinha, S. Chandra, D. Schmidt, and P. Balachandran, "Making defect-finding tools work for you," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/1810295.1810310

[14] N. Ayewah, D. Hovemeyer, J. Morgenthaler, J. Penix, and W. Pugh, "Using static analysis to find bugs," *Software, IEEE*, vol. 25, no. 5, pp. 22–29, Sept 2008.

[15] D. Hovemeyer and W. Pugh, "Finding more null pointer bugs, but not too many," in *Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, ser. PASTE '07. New York, NY, USA: ACM, 2007, pp. 9–14. [Online]. Available: http://doi.acm.org/10.1145/1251535.1251537

[16] S. Kim and M. D. Ernst, "Which warnings should i fix first?" in *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC-FSE '07. New York, NY, USA: ACM, 2007, pp. 45–54. [Online]. Available: http://doi.acm.org/10.1145/1287624.1287633

[17] L. Singer and K. Schneider, "It was a bit of a race: Gamification of version control," in *Games and Software Engineering (GAS), 2012 2nd International Workshop on*, June 2012, pp. 5–8.

[18] N. Chen and S. Kim, "Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 140–149. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351697

[19] N. Tillmann, J. De Halleux, and T. Xie, "Pex4fun: Teaching and learning computer science via social gaming," in *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*, May 2011, pp. 546–548.