A Replicated Study on Relationship Between Code Quality and Method Comments

Yuto Miyake, Sousuke Amasaki, Hirohisa Aman and Tomoyuki Yokogawa

Abstract *Context*: Recent studies empirically revealed a relationship between source code comments and code quality. Some studies showed well-written source code comments could be a sign of problematic methods. Other studies also show that source code files with comments confessing a technical debt (called self-admitted technical debt, SATD) could be fixed more times. The former studies only considered the amount of comments, and their findings might be due to a specific type of comments, namely, SATD comments used in the latter studies. *Objective*: To clarify the relationship between comments other than SATD comments and code quality. *Method*: Replicate a part of the latter studies with such comments of methods on four OSS projects. *Results*: At both the file-level and the method-level, the presence of comments could be related to more code fixings even if the comments were not SATD comments. However, SATD comments were more effective to spot fix-prone files and methods than the non-SATD comments. *Conclusions*: Source code comments other than SATD comments could still be a sign of problematic code. This study demonstrates a need for further analysis on the contents of comments and its relation to code quality.

Keywords Source code comment · Software quality · Self-admitted technical debt

Y. Miyake · S. Amasaki (✉) · T. Yokogawa
Faculty of Computer Science and Systems Engineering,
Okayama Prefectural University, Soja 719-1197, Japan
e-mail: amasaki@cse.oka-pu.ac.jp

Y. Miyake
e-mail: cd28043k@cse.oka-pu.ac.jp

T. Yokogawa
e-mail: t-yokoga@cse.oka-pu.ac.jp

H. Aman
Center for Information Technology, Ehime University, Matsuyama 790-8577, Japan
e-mail: aman@ehime-u.ac.jp

R. Lee (ed.), *Applied Computing and Information Technology*,
Studies in Computational Intelligence 695,
DOI 10.1007/978-3-319-51472-7_2

1 Introduction

The success of software development depends on many factors. The quality management of source code is one of the crucial activities to the success. Software developers dedicate much effort to retain the quality of source code from a spectrum of perspectives.

The comprehensibility of source code is essential for software developers working for a project where they maintain and evolve the source code cooperatively. For this purpose, the coding standards such as MISRA C [11] and Google Java Style [7] help to read source code and to prevent from faults though they provide no guide for code construction.

Comments in source code are useful documentation for developers and help their comprehension for the logic, architecture, and limitations of the code nearby. In fact, comments are widely known as effective entities [4, 14]. On the other hand, Fowler's book on refactoring [6] indicated that the presence of many comments is one of "code smells." Code smells are typical signs of poor design (poor quality parts of the code) to be refactored. Comments themselves have no direct effect on the design, but they often work as deodorants for smells. From a view of refactoring, comments thus play dual roles: well-written comments increase readability of programs, but they can also be signs of poor quality code.

Recent studies empirically revealed a relationship between source code comments and code quality. Comments themselves are harmless to code. Nevertheless, those studies both indicated some types of comments might be a useful sign for code quality improvement. Aman et al. focused on the relationship between the amount of code comments and code quality [1–3]. These analyses reported that well-written comments in a method body could be a sign of its fault-proneness. The relationship between the content of code comments and code quality was also studied [12, 13, 18]. These analyses focused on specific comments called *self-admitted technical debt* (SATD). SATD is an apparent quality risk confessed in code comments by developers. They first conducted manual examination for SATD [12, 13], and then revealed that SATD-related changes were difficult [18].

The studies [1–3] focused on the amount of comments and did not look into the contents of comments. The comments they analyzed contain the SATD comments, and it has been unknown that how much the SATD comments affected their results. A question thus arises whether the relationship between comments and code quality is viable for comments other than SATD comments. On the other side, the studies [12, 13, 18] were conducted on a bit rough-grained units, namely, source files. Comments usually help to understand a code fragment close to them, and a finer grained empirical study at the method level can show a detailed relationship between SATD and code quality.

In this study, we focused on a relationship between code quality and comments of methods. We extracted methods of classes and compared fix-proneness between the methods with comments other than SATD comments and those without any comments. We also conducted comparisons of fix-proneness between the methods with SATD comments and the other methods.

This paper is organized as follows. In Sect. 2, we present the design of our experiments. In Sect. 3, we show the results and their discussion. Related work is shown in Sect. 4. We conclude the paper in Sect. 5.

2 Methodology

2.1 Self-Admitted Technical Debt

Technical debts are introduced into source code by developers. Some technical debts are intentionally buried due to several reasons such as time pressure. A part of intentional technical debts are admitted in source code comments. Such technical debts are called *Self-Admitted Technical Debt* (SATD) [12]. An example of such comments is as follows:

```
if (implementation == null) {
  // No proper implementation
  // FIXME: We should log a warning, shouldn’t we ?
  implementation = new AnyLanguage(language);
}
```

Here, a developer used a typical SATD keyword *FIXME* to share anxiety for logging option. In [18], such SATD keywords were used for specify SATDs.

2.2 Research Questions

This study aims at clarifying the relationship between comments other than SATD comments and code quality. SATD comments were rarely found in comments, and it was difficult to use its amount for experiments. Therefore, we decided to investigate a relationship between the presence of comments other than SATD comments and code quality. We used the fix-proneness to measure code quality as well as the studies [2, 18].

The studies [2, 18] conducted at different module levels: method and class. Comments usually describe a code fragment close to them, and the presence of a problematic fragment would also be signified by a comment near to it. This study thus focused on comments of methods but compared the fix-proneness at both the file level and the method level.

Table 1 OSS projects analyzed in the empirical work

Project	HEAD	KLOC	# of files	# of methods	# of commits
PMD	4e5e0187	90852	1048	6828	8698
SQuirreL SQL Client	7d867694	367732	2293	17802	7340
FreeMind	cc9cca40	113275	519	5631	1061
Hibernate ORM	ef46293c	453556	3326	28460	7083

We designed our empirical research on the following two research questions:

RQ1: Are files containing comments other than SATD comments more fix-prone than files without any comments?

RQ2: Are methods containing SATD comments more fix-prone than the other methods? Are methods containing comments other than SATD comments more fix-prone than methods without any comments?

2.3 Datasets

Our experiments required projects managed by a version control system because the fix-proneness required access to change histories of files. This study was conducted on four OSS projects: PMD,[1] SQuirreL SQL Client,[2] FreeMind,[3] and Hibernate ORM.[4] They all ranked in the top 50 popular Java products at SourceForge.net, are developed in Java, and their source files are maintained with Git. These projects were used in the past studies [2, 3]. Analyses on projects used in [18] are future work.

We selected Java files and focused on source code comments. The other files like XML were excluded from this study. Furthermore, our experiments compared the fix-proneness at two levels of source code, namely, file-level and method-level. Therefore, files not having any method were also excluded to keep the same files among experiments on the two levels.

Table 1 summarizes some statistics of project characteristics. The number of files counts the number of Java source code files which have at least one method. Table 1 shows a variety of projects in code size and the number of methods.

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

[2] http://squirrel-sql.sourceforge.net.

[3] http://freemind.sourceforge.net/wiki/index.php/Main_Page.

[4] http://hibernate.org/orm.

2.4 Target Comment Types

There are various types of comments described in source code [16]. This study includes experiments on the relationship between comments and code quality at the method-level, and we focused on comments related to methods: member comment and inline comment.

According to [16], member comments describe the functionality of method/field, being located either before or in the same line as the member definition. In our case, member comments are ones located ahead of a method. Member comments for fields were ignored. Member comments help developers understanding a usage or a specification of the method. Although Javadoc is a typical member comment, we analyzed member comments other than Javadoc. Because Javadoc rarely gives quality related descriptions and often ignored [3, 13].

Inline comments are located within a method body. Inline comments often describe code fragments nearby. Implementation decisions are also described by inline comments [16], they were candidates for a deodorant of code smells [3]. Sometimes code comments (commented-out code) were found within a method body. The code comments were irrelevant to our scope and ignored as well as the past studies.

2.5 Comments Extraction and Labeling

For each OSS project, we cloned the git repository. Table 1 shows HEAD sha of the projects. We then extracted methods and comments of them. For this purpose, we made a tool with Eclipse Java development tools (JDT).[5]

We first selected target files to be analyzed. The target files were source code written in Java. Files related to testing were excluded. The tool was applied to extract member and inline comments from the target files in each project. The tool also extracted method signatures and the range of method bodies. According to that information, the extracted comments were linked to the methods and classified into member comments, inline comments, and the others. Only the methods having member comments or inline comments were supplied for experiments.

For analyzing a relationship between comments and code quality, we have two options. The first one is to use the number of comment lines as well as Aman et al. [1–3]. The other is to use the presence of comments. This study focused on SATD comments and comments other than the SATD comments, and SATD studies [12, 13, 18] only considered the presence of SATD. Furthermore, SATD comments were rarely found in comments and it was difficult to use its amount for experiments. In this study, we thus decided to use the presence of SATD comments and the other comments for simplicity.

[5] http://www.eclipse.org/jdt.

Following [18], we first labeled the extracted methods as SATD or non-SATD with keyword matching. The keywords provided online by [12, 18] were used for the labeling. A method was labeled as SATD if any of the keywords appeared in member or inline comments of the method. Otherwise, the method was labeled as non-SATD. The non-SATD methods may contain comments other than SATD comments. A file was labeled as SATD if any of its methods were SATD.

We then labeled the non-SATD methods according to the presence of comments. A method was labeled as *commented* if the method has any comments. Note that no SATD keyword appeared in these comments. A file was commented if any of its methods were commented.

2.6 Fix-Proneness Identification

The fix-proneness were considered as a valuable measure for assessing code quality and maturity level [2, 18]. This study thus adopted the fix-proneness as code quality.

The definition of the fix-proneness defined as follows:

$$\frac{\#\,\text{of fix commits}}{\#\,\text{of commits}}$$

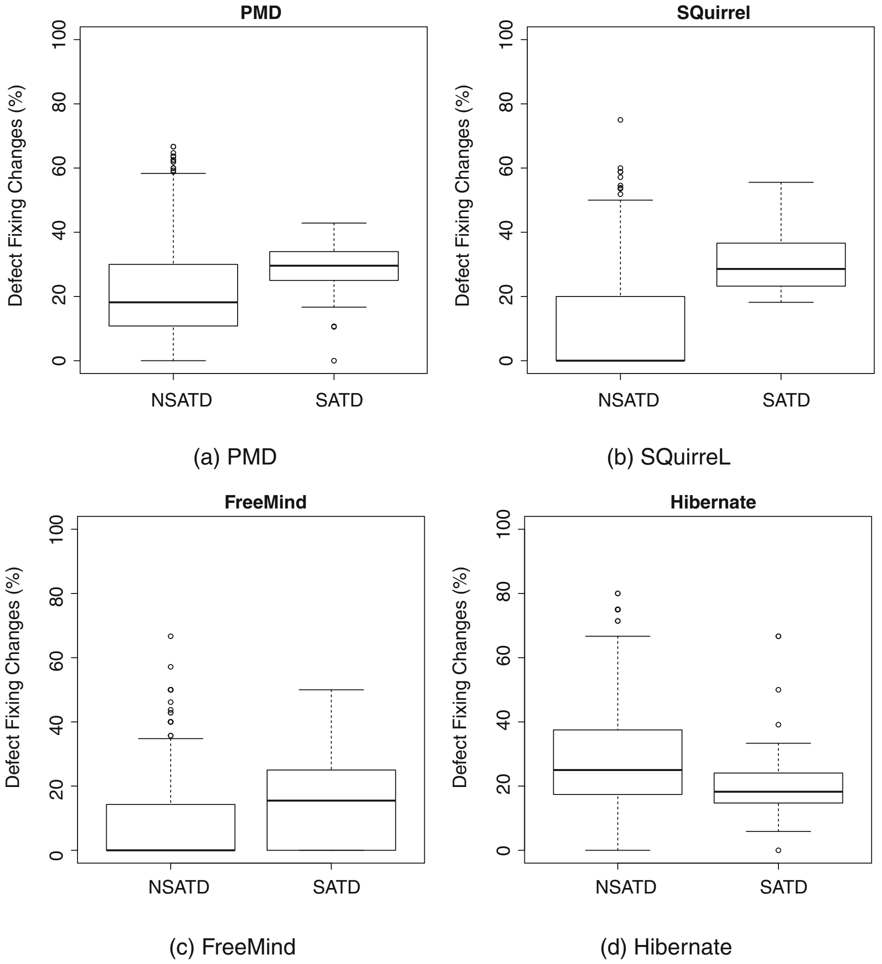
For each target file, all commits related to the file were obtained from a git repository. A commit was considered as a fix commit if any of bug-fixing-related keywords were found in a summary or a message of the commit. We used the following keywords shown in [18]: fixed issue #ID, bug ID, fix, defect, patch, crash, freeze, breaks, wrong, glitch, proper. All commits related to the target file were traversed, and the fix-proneness of the file was recorded.

The commits were then mapped to methods. We used `git diff` to identify lines added, modified, and deleted of a file. Matched the lines against the range of methods in the file, we identified methods changed by the commit. The fix-proneness of methods was obtained from the matching results.

3 Results

3.1 Are Files Containing Comments Other Than SATD Comments More Fix-Prone Than Files Without Any Comments?

Figure 1 shows boxplots of the percentage of defect fixing changes between files having and not having SATD methods for the four projects. This comparison works as a sanity check for [18] under a slightly different definition for SATD files and target



(a) PMD

(b) SQuirreL

(c) FreeMind

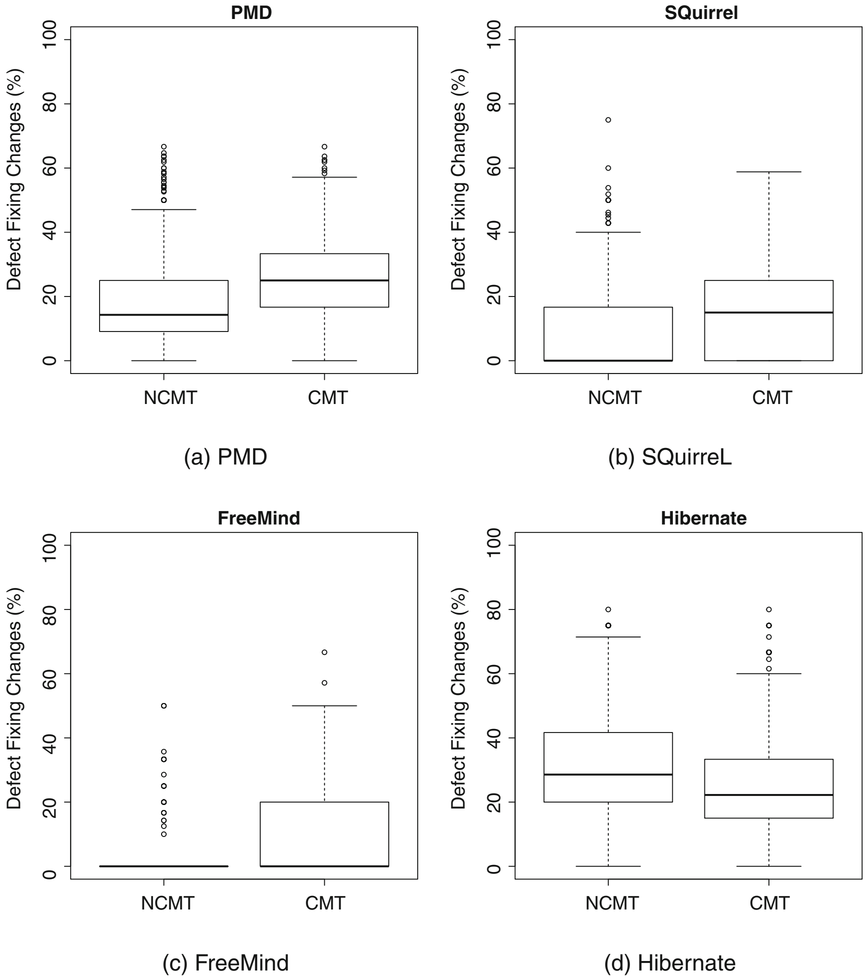
(d) Hibernate

Fig. 1 Percentage of defect fixing changes for SATD and NSATD files

projects. All projects except for Hibernate (Fig. 1d) show that files having SATD methods have a higher percentage of defect fixing changes. This result was consistent to [18] somewhat where files having SATD comments were more fixed in four out of the five projects they used.

Figure 2 shows boxplots of the percentage of defect fixing changes between files having comments other than SATD comments and files having no comment for the four projects. As same as the previous results, all projects but Hibernate (Fig. 2d) show that files having comments other than SATD comments have a higher percentage of defect fixing changes.

Those differences were tested by Man-Whitney test [10] and quantified by the effect size with Cliff's delta [5] as well as [18]. All differences were statistically



(a) PMD

(b) SQuirreL

(c) FreeMind

(d) Hibernate

Fig. 2 Percentage of defect fixing changes for Commented (non-SATD) and non-Commented files

significant at $\alpha = 0.05$. Table 2 shows the effect-size for the four projects. Cliff's Delta ranges in the interval [-1, 1] and is considered negligible for $\|d\| < 0.147$, small for $0.147 \leq \|d\| < 0.33$, medium for $0.33 \leq \|d\| < 0.474$, and large for $\|d\| \geq 0.474$. The criteria indicate that the differences between SATD and non-SATD were medium or large for PMD, SQuirreL, and FreeMind. Only the difference on Hibernate was small. The differences between commented and non-commented files were all considered small.

Table 2 Cliff's Delta for SATD versus NSATD and Commented versus non-Commented files

Project	SATD versus NSATD	CMT versus NCMT
PMD	0.403	0.327
SQuirreL SQL Client	0.737	0.281
FreeMind	0.384	0.197
Hibernate ORM	−0.308	−0.192

These observations lead the following answer for RQ1:

- Where the presence of SATD comments increases defect fixing changes, the presence of comments other than SATD comments also increases defect fixing changes
- The difference of fix-proneness between commented and non-commented files was statistically significant but smaller than that between SATD and non-SATD files

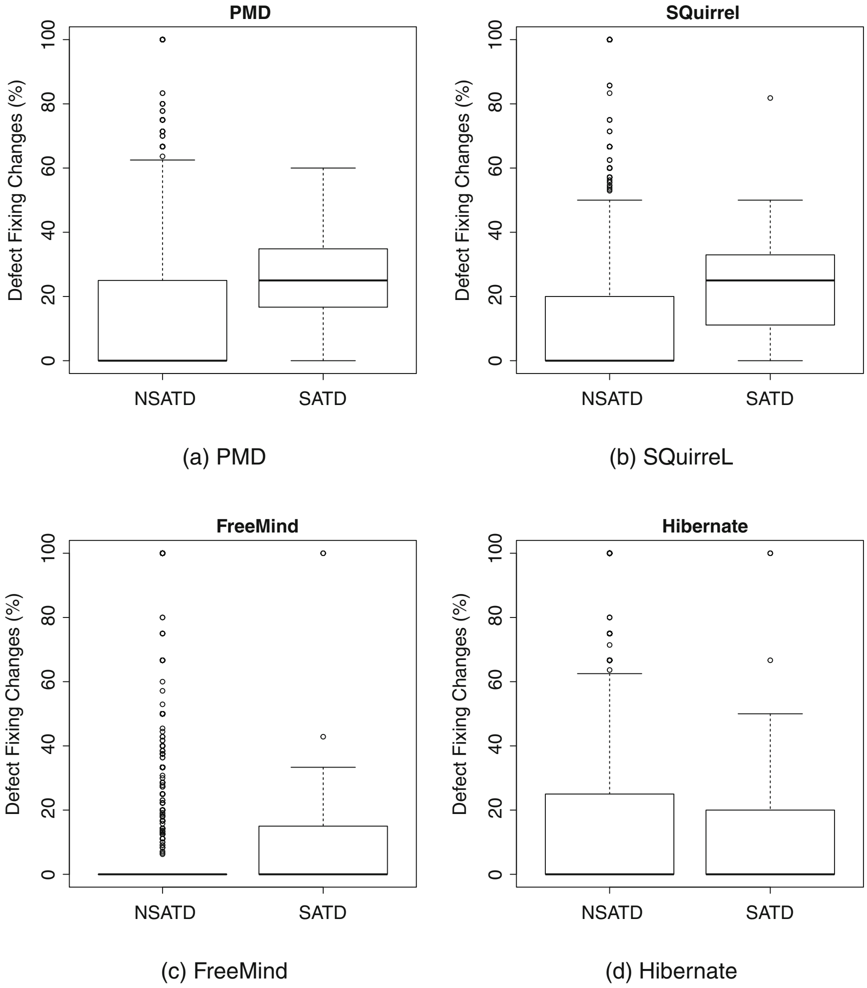
3.2 *Are Methods Containing SATD Comments More Fix-Prone Than the Other Methods? Are Methods Containing Comments Other Than SATD Comments More Fix-Prone Than Methods Without Any Comments?*

Figure 3 shows boxplots of the percentage of defect fixing changes in SATD and non-SATD files for the four projects. This comparison shows a detailed relationship between SATD and code quality at the method level. Figure 3 shows that the fix-proneness was higher if methods had SATD comments for PMD and SQuirreL. This result was similar to the comparison at the file-level shown in Fig. 1. Contrastingly, the differences of the fix-proneness were less clear for FreeMind and Hibernate.

Figure 4 shows boxplots of the percentage of defect fixing changes in commented and non-commented methods for the four projects. As same as the results for SATD methods, commented files have a higher percentage of defect fixing changes for PMD and SQuirreL; the differences of the fix-proneness were less clear for Hibernate. For FreeMind, the difference could not be observed from Fig. 4c.

Those differences were tested by Man-Whitney test and quantified by the effect size with Cliff's delta. Regarding SATD methods, all differences except for Hibernate were statistically significant at $\alpha = 0.05$. Regarding commented methods, all differences including Hibernate were statistically significant at $\alpha = 0.05$.

Table 3 shows the effect-size for the four projects. The criteria indicate that the differences between SATD and NSATD were medium or large for PMD and SQuirreL. The difference for FreeMind was small, and that for Hibernate was negligible. The effect sizes at the method-level were smaller than those at the file-level.



(a) PMD

(b) SQuirreL

(c) FreeMind

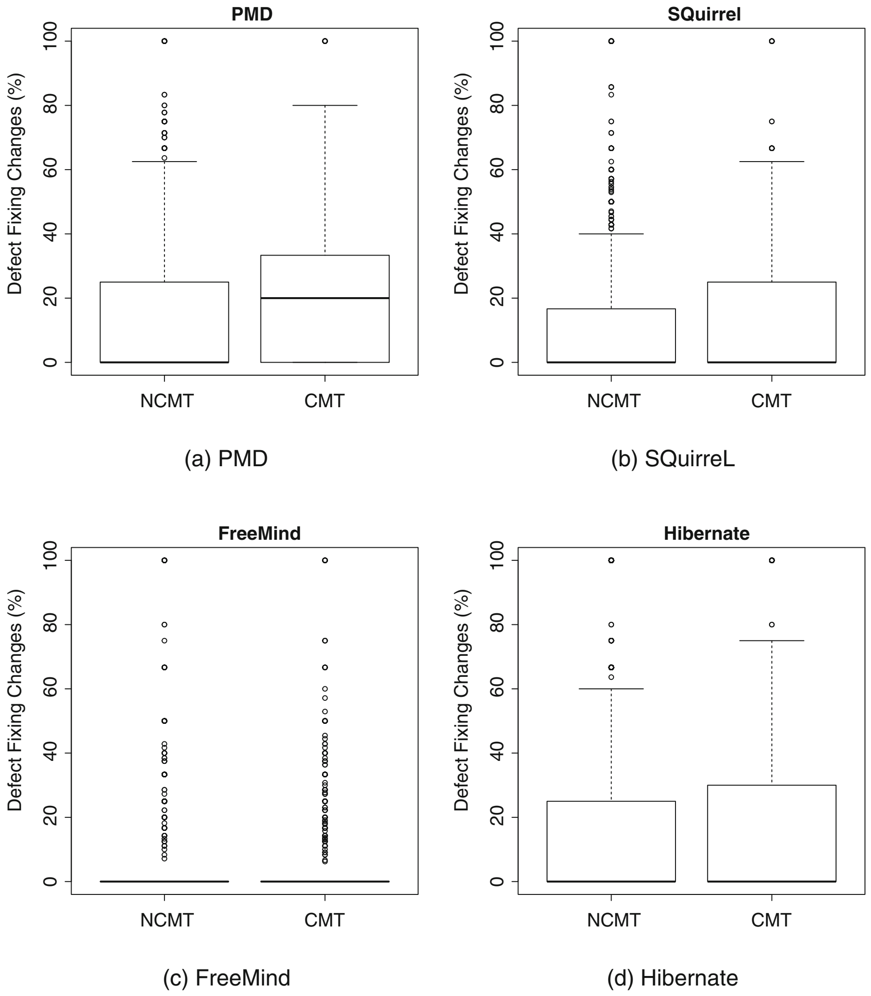
(d) Hibernate

Fig. 3 Percentage of defect fixing changes for SATD and NSATD methods

The differences between commented and non-commented files were small for PMD and SQuirreL. The differences for FreeMind and Hibernate were negligible. These effect sizes were smaller than those on SATD methods. The effect sizes at the method-level were smaller than those at the file-level.

These observations lead the following answer for RQ2:

- Where the presence of SATD comments increases defect fixing changes at the file-level, defect fixing changes are also increased at the method-level
- Where the presence of SATD comments increases defect fixing changes at the method-level, the presence of comments other than SATD comments also increases defect fixing changes at the method-level



(a) PMD

(b) SQuirreL

(c) FreeMind

(d) Hibernate

Fig. 4 Percentage of defect fixing changes for Commented and non-Commented methods

- The difference of fix-proneness between commented and non-commented methods was statistically significant but smaller than that between SATD and non-SATD methods

3.3 Threats to Validity

This study leaves some threats to validity as well as the past related studies.

Table 3 Cliff's Delta for SATD versus NSATD and Commented versus non-Commented methods

Project	SATD versus NSATD	CMT versus NCMT
PMD	0.381	0.273
SQuirreL SQL Client	0.514	0.156
FreeMind	0.170	0.096
Hibernate ORM	−0.03	−0.102

Regarding internal validity, we classified fix commits and SATD comments by keyword matching. While the keyword matching is simple and scalable, it might overlook the fix commits and SATD comments. However, this approach worked effectively in not a few studies [8, 15, 20]. Manual classification is labor-intensive for large projects in practice, and this is a realistic approach.

Regarding external validity, our target projects were limited in number and programming language. Therefore, the observations in this study might not hold in other projects using other programming language. However, the target projects were collected from various application areas, and most of popular programming languages are based on imperative manner and provide function or method constructs as a basis of development. Therefore, those differences are not critical for the generality of our contribution. Our future work includes experiments on such projects.

4 Related Work

Many studies focused on comments and its relation and effect to aspects of software development.

While comments can be freely written, its main purpose is to help developers comprehending the programs' logic, architecture, and limitations. Tenny [17] and Woodfield et al. [19] conducted experiments about program comprehensions, with students and experienced programmers as their subject, respectively. In their experiments, the subjects evaluated the ease of understanding some variations of a program which were different in modular design and in commenting manner (with or without comments.) Their experimental results showed that comments have a positive impact on program comprehension. The results also implies that the quality of comments also matters for program comprehension.

Steidl et al. [16] proposed a framework to evaluate the quality of comments. They evaluated the coherence between member comments and the name of the corresponding method. Their coherence evaluation presents whether the member comments provide useful information about the method. They discussed the comments written inside method bodies as well, and proposed to use short comments as indicators of parts to be refactored. Moreover, they described that the existence of long comments may infer a lack of external documents, so adding many comments was not recommended. Lawrie et al. [9] also studied the correspondence of comments and code by using a natural language processing technique, for an automated quality assessment.

Recent studies get focused on comments from the perspective of code quality. Aman et al. conducted some empirical studies of the relationships between the amount of comments and the fault-proneness [1, 3]. Their empirical results showed that the presence of comments written inside method bodies (they called "inline comments") is correlated with the fault-proneness in the programs. Therefore, well-written inline comments in a method body could be a sign of its fault-proneness as indicated as code smells by Fowler et al [6]. They also showed that the amount of comments contributed to explain the change proneness of programs [2].

The content of comments was also analyzed and related to the code quality. Potdar and Shihab [12] used a certain type of comments to specify technical debts. They called such commented technical debts as *self-admitted technical debt* (SATD). They manually examined source code comments of OSS projects and showed detailed statistics of SATD. Another study [13] regarding SATD classified SATD into five debts and showed what types of debts were common among their target projects. The relationship between SATD and code quality was examined in [18]. They used typical SATD keywords found in the previous studies to determine SATD files. Then, they related the presence and the introducing of SATD with defect inducing changes, fix inducing changes, and difficulty of code changes.

Our study heavily relies on the studies focused on the relationships between comments and code quality. While the studies by Aman et al. did not care about SATD, this study analyzed the quality of code having non-SATD comments. While Wehaibi et al. [18] considered the quality at the file-level, this study also focused on the quality at the method-level.

5 Conclusion

We replicated a study on the relationship between method comments and quality. The results show that the presence of comments could be a sign of problem even when the method comments did not contain SATD. We also revealed that the presence of SATD comments could also be a sign of problem even at the method-level.

Our future work includes further analyses on the relationship from another perspective. Defect inducing change was one of popular quality measures in literature including the past studies we replicated. The comparison of quality between pre- and post- comment inducing has also remained as future work. With deeper understanding on comments, we can expect a system suggesting risky or smelled code at writing comments.

Acknowledgements The authors would like to thank the anonymous reviewers for their thoughtful comments and helpful suggestions on the first version of this paper. This work was partially supported by JSPS KAKENHI Grant #16K00099.

References

1. Aman, H.: An Empirical Analysis on Fault-Proneness of Well-Commented Modules. In: Fourth International Workshop on Empirical Software Engineering in Practice (IWESEP). IEEE (2012)
2. Aman, H., Amasaki, S., Sasaki, T., Kawahara, M.: Empirical Analysis of Change-Proneness in Methods Having Local Variables with Long Names and Comments. In: ACM IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–4 (2015)
3. Aman, H., Amasaki, S., Sasaki, T., Kawahara, M.: Lines of Comments as a Noteworthy Metric for Analyzing Fault-Proneness in Methods. IEICE - Transactions on Information and Systems **E98.D**(12), 2218–2228 (2015)
4. Buse, R., Weimer, W.: Learning a Metric for Code Readability. IEEE Transactions on Software Engineering **36**(4), 546–558 (2010)
5. Cliff, N.: Ordinal Methods for Behavioral Data Analysis (1996)
6. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring. Improving the Design of Existing Code. Addison-Wesley (2012)
7. Google: Google java style. http://google-styleguide.googlecode.com/svn/trunk/javaguide.html
8. Kim, S., Whitehead Jr., E.J., Zhang, Y.: Classifying Software Changes: Clean or Buggy? IEEE Transactions on Software Engineering **34**(2), 181–196 (2008)
9. Lawrie, D.J., Feild, H., Binkley, D.: Leveraged Quality Assessment using Information Retrieval Techniques. In: the 14th IEEE International Conference on Program Comprehension (ICPC). IEEE (2006)
10. Mann, H.B., Whitney, D.R.: On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. The annals of mathematical statistics (1947)
11. MISRA: MISRA C. http://www.misra-c.com
12. Potdar, A., Shihab, E.: An Exploratory Study on Self-Admitted Technical Debt. In: IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 91–100. IEEE (2014)
13. da S. Maldonado, E., Shihab, E.: Detecting and quantifying different types of self-admitted technical Debt. In: IEEE 7th International Workshop on Managing Technical Debt (MTD), pp. 9–15. IEEE (2015)
14. Scanniello, G., Gravino, C., Risi, M., Tortora, G., Dodero, G.: Documenting Design-Pattern Instances: A Family of Experiments on Source-Code Comprehensibility. ACM Transactions on Software Engineering and Methodology **24**(3) (2015)
15. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? In: International Workshop on Mining Software Repositories (MSR), pp. 1–5. ACM (2005)
16. Steidl, D., Hummel, B., Juergens, E.: Quality analysis of source code comments. In: IEEE 21st International Conference on Program Comprehension (ICPC), pp. 83–92. IEEE (2013)
17. Tenny, T.: Program Readability: Procedures Versus Comments. IEEE Transactions on Software Engineering **14**(9), 1271–1279 (1988)
18. Wehaibi, S., Shihab, E., Guerrouj, L.: Examining the Impact of Self-Admitted Technical Debt on Software Quality. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 179–188. IEEE (2016)
19. Woodfield, S.N., Dunsmore, H.E., Shen, V.Y.: The effect of modularization and comments on program comprehension. In: 5th international conference on Software engineering (ICSE), pp. 215–223. IEEE (1981)
20. Zimmermann, T., Premraj, R., Zeller, A.: Predicting Defects for Eclipse. In: International Workshop on Predictor Models in Software Engineering (PROMISE). IEEE (2007)