2013 20th Asia-Pacific Software Engineering Conference

How Much Do Code Repositories Include Peripheral Modifications?

Noa Kusunoki, Keisuke Hotta, Yoshiki Higo, Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University, Japan
1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan
{k-noa, k-hotta, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—In the last decade, a variety of studies on mining software repositories has been conducted. Mining repositories has a potential to obtain useful knowledge for the future development and maintenance. When software repositories are mined, large commits in them are often excluded from mining targets because large commits include merging and we believe that large commits include peripheral modifications, which may affect negative impacts on mining code repositories. However, if large commits include code modifications, excluding large commits loses such modifications unintentionally. Moreover, such data cleansing assumes that there are no peripheral modifications in small commits. In this paper, we investigate how much peripheral modifications are included in commits in code repositories. As a result, we found that excluding large commits is insufficient to remove hindrances in commits for mining code repositories.**

***Keywords*-Mining software repositories, Large commit, Source code analysis**

I. Introduction

Recently, the research areas related to mining software repositories are very active and attract much attention [1], [2]. A software repository includes a variety of historical information on the past activities related to its software. Mining software repositories means actions and techniques for extracting and deriving useful knowledge for the future development and maintenance.

Historical code repository is one of the well-mined repositories. They have rich information to gain knowledge or principles that are useful for software engineering [3], [4]. Therefore, great efforts have been spent on mining historical code repositories. Such research efforts have a wide variety of interests, including defects [3]–[7], changes [8]–[13], or clones [14].

When we mine a software repository, large commits in it are often excluded from the analysis targets. This is because large commits often include merging [19] and we believe that large commits consist of peripheral modifications such as reformatting source code. However, if a large commit includes modifications on program instruction code itself, excluding the large commit may lose important code changes unintentionally. Consequently, in this research, we investigate the following research question.

RQ1 Do large commits consist of only peripheral modifications?

Excluding large commits provides us one more important thing to worry about. Such exclusion assumes that all the small commits includes code changes and there are no peripheral modifications in the commits. However, none of research efforts have investigated characteristics of small commits yet. Consequently, we investigate the following research question.

RQ2 Do small commits consist of only modifications on program instruction code?

In order to investigate the research questions, we propose a new technique to distinguish the following types of modifications from each other at a fine-grained level:

- modifications on program instruction code,
- modifications for reformatting source files,
- modifications on code comments, and
- modifications on other files than source files.

The remainder of this paper is organized as follows: in Section II, we define some terms used in this paper and explain the proposed technique; Section III shows an evaluation on students' projects and an experiment on open source software; related works are described in Section IV; lastly, Section V concludes this paper.

II. Overview of the Investigation Technique

In order to investigate our two research questions, it is necessary to discriminate changes affecting program instructions from those not affecting them. For this objective, the investigation technique in this research classifies every modification in a commit into the following four categories.

- **Type-A:** modifications for white spaces, tabs, and new line characters.
- **Type-B:** modifications for comments.
- **Type-C:** modifications for program instruction code.
- **Type-D:** modifications in other files than source files, or additions/deletions of such files.

The investigation in this study desires a fine-grained analysis on modifications to achieve a high accuracy. Hence, it analyzes every modification at the token-level. Herein, "a modification at the token-level" means "addition/change/deletion of a token". In other words, every token modified in a commit is classified into the above categories. Note that Type-D modifications cannot be segmentalized into the token-level because the proposed technique focuses

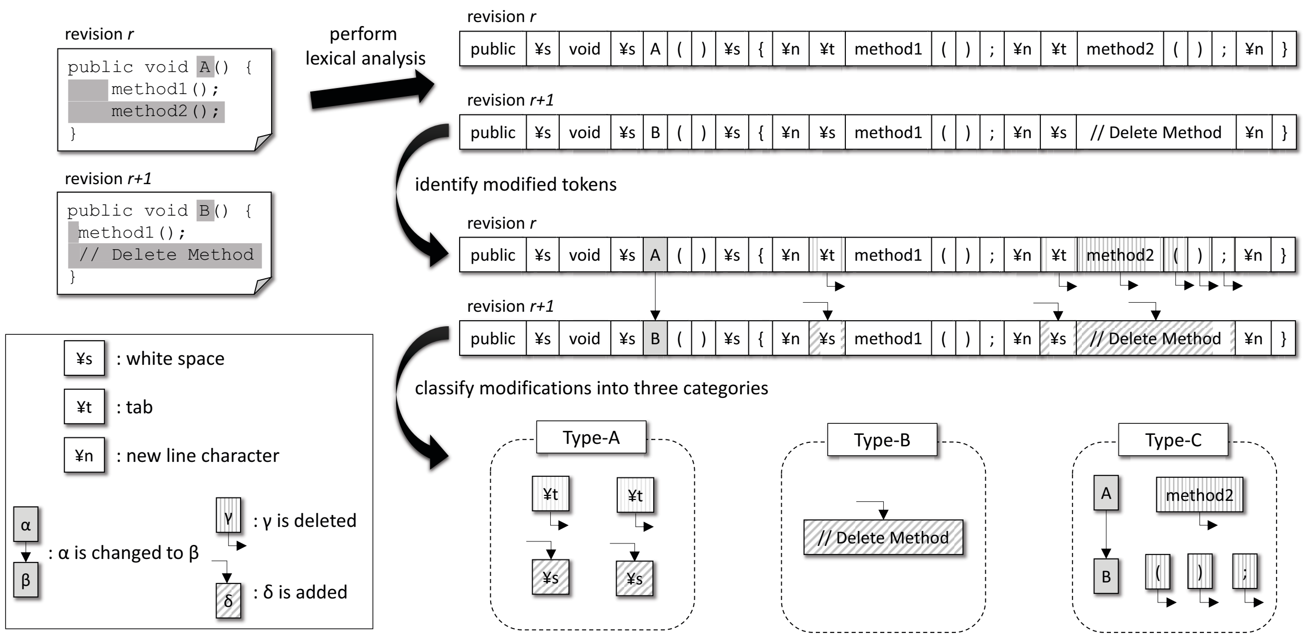


Figure 1. An Overview of the Investigation Technique

on modifications on source files and there is no need to split modifications in other files.

The investigation technique takes a historical code repository as its input, and classifies every modification on every commit into the above three categories.

The investigation technique on a commit consists of the following three steps, and it repeats the steps for all the commits included in the given repository.

- **STEP-1:** Performing lexical analysis.
- **STEP-2:** Detecting modified tokens.
- **STEP-3:** Classifying modified tokens.

Figure 1 shows how the investigation technique works on a commit. This example assumes that only a single source file is modified in the commit for the ease of explanations, but the investigation technique works in the same way in the case of commits in which two or more source files are modified. The followings briefly describe each step with the example.

STEP-1: Lexical Analysis

The first step performs lexical analysis on both of two files before and after modified. Hence, this step provides two token sequences for every file modified in the commit. A remarkable point of the lexical analysis in this study is that it is interested not only in usual tokens (such as identifier or comma) but also in comments, white spaces, tabs, and new line characters. This means that comments, white spaces, tabs, and new line characters are regarded as tokens as you can see in Figure 1.

STEP-2: Detection of Modified Tokens

This step detects added/changed/deleted tokens in the commit. It detects mappings of tokens by comparing two token sequences created from source files before and after modified. The comparison is based on textual representations and types of tokens. This step identifies what tokens are added/changed/deleted in this commit with the token mappings. It identifies a pair of changed tokens, six deleted tokens, and three added tokens from the example in Figure 1.

STEP-3: Classification of Modified Tokens

The final step classifies every added/changed/deleted token detected in the previous step into the above three categories. The classification is performed based on types of modified tokens. For instance, the example in Figure 1 has deletion of two tabs and addition of two white spaces, and so the classification regards these four modifications as Type-A. The example also has addition of a comment, which is classified into Type-B. The other modifications, which include the change on identifies from "`A`" to "`B`", and the deletion of "`method2`", "`(`", "`)`", and "`;`", are classified into Type-C in a similar fashion.

III. Experiment

We conducted an experiment on two open source systems, Columba and SQuirreL SQL Client in order to answer the RQs. Herein, we show the RQs again:

RQ1 Do large commits consist of only peripheral modifications?

RQ2 Do small commits consist of only modifications on program instruction code?

The numbers of revisions in the repositories were 463 and 6,737, respectively.

Figure 2 shows each type of modifications included in every of the commits in the Columba and SQuirreL repositories. Every commit forms a vertical bar, and commits are sorted in the descending order of their size. Each bar consists of four colors, blue, read, gray, and water-clear, each of which corresponds to *format*, *comment*, *other files*, and *code* modifications, respectively. Their lengths reflect the rate of their LOCs for all the lines modified in the commit. If there is no water-clear part in a given bar, there is no *code* modification in the commit. Also, the figure includes the LOC of each commits[1]. Note that the figure has the logarithmic axis for the commit LOC.

We divided them into five groups based on their size. The first group includes top 20% of the large commits, and the last group includes top 20% of the small commits. Then, we measured following rates $F(G)$, $C(G)$, and $O(G)$ for each group of the commits.

$$F(G) = \frac{\sum_{c \in G} format(c)}{\sum_{c \in G} loc(c)} \times 100 \quad (1)$$

$$C(G) = \frac{\sum_{c \in G} comment(c)}{\sum_{c \in G} loc(c)} \times 100 \quad (2)$$

$$O(G) = \frac{\sum_{c \in G} other(c)}{\sum_{c \in G} loc(c)} \times 100 \quad (3)$$

$$\begin{aligned} loc(c) &= format(c) + comment(c) \\ &\quad + other(c) + code(c) \quad (4) \end{aligned}$$

where,

- G is a given group of commits,
- $format(c)$ is the LOC of $format$ modifications in given commit c.
- $comment(c)$ is the LOC of $comment$ modifications in given commit c.
- $other(c)$ is the LOC of $otherfiles$ modifications in given commit c.
- $code(c)$ is the LOC of $code$ modifications in given commit c.

Table I shows the measurement result. The two systems have different trends. In Columba, the *other files* modifications occupied more than 40% on the top 20% commits, which was the highest rate in all the five groups. On the other hand, in SQuirreL, only 13.5% LOCs were included in the *other files* modifications, which was the lowest rate. As a result, there are different trends between the total rate of the three type modifications of the two systems. In Columba, all the file groups have approximately the same value. Meanwhile in SQuirreL, the smaller commits are, the higher value their total rate becomes.

Next, we ignored the *other files* modifications. In this experiment, we distinguished the other files from source files by checking file's extension. That is because when we conduct repository mining experiments, we often ignore commits not related to source files. Figure 3 shows the result. We can see that *format* and *comment* modifications often appear in commits regardless of their size. Table II show more quantitative data. Each cell of this table calculated with the following formulae.

$$F'(G) = \frac{\sum_{c \in G} format(c)}{\sum_{c \in G} loc'(c)} \times 100 \quad (5)$$

$$C'(G) = \frac{\sum_{c \in G} comment(c)}{\sum_{c \in G} loc'(c)} \times 100 \quad (6)$$

$$loc'(c) = format(c) + comment(c) + code(c) \quad (7)$$

The table shows that around 30% modifications are either of the *format* or *comment* modifications in the middle commits (20-80%). On the other hand, in the top 20% commits, they occupy only around 10%.

Our answer to **RQ1** is **NO**. In the experiment, the top 20% large commits included *format*, *comment* and *other files* modifications, however the total rate of them for all the modifications is less than half. Thus, we conclude that excluding large commits loses many *code* modifications unintentionally.

Table I
RATE OF LOCS INCLUDED IN MODIFICATIONS OF FORMAT, COMMENT, AND OTHER FILES

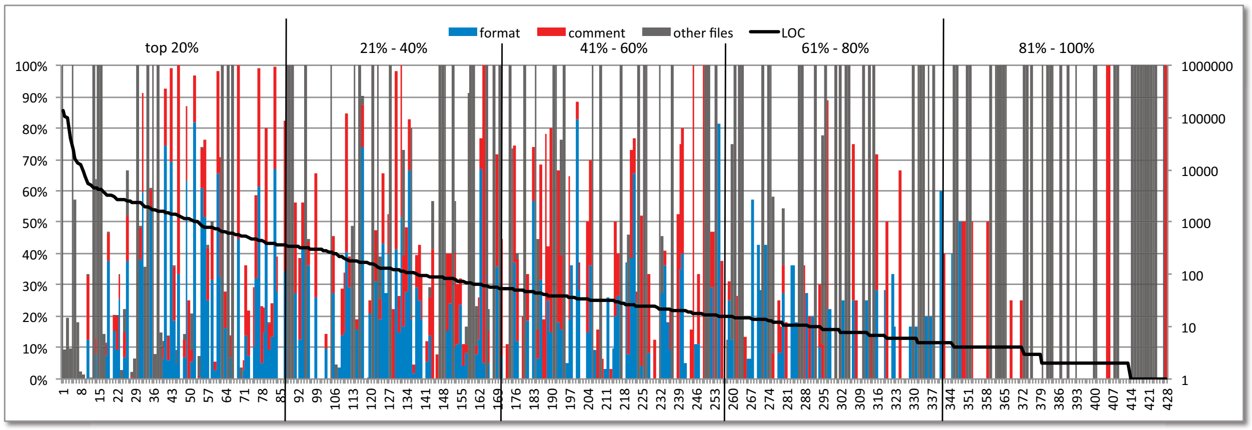
(a) Columba

group	format	comment	other files	total
top 20%	2.9%	1.6%	40.1%	44.7%
21-40%	13.0%	10.2%	24.5%	47.6%
41-60%	12.0%	14.6%	22.3%	48.9%
61-80%	9.0%	6.3%	29.0%	44.4%
81-100%	3.0%	7.2%	34.6%	44.7%

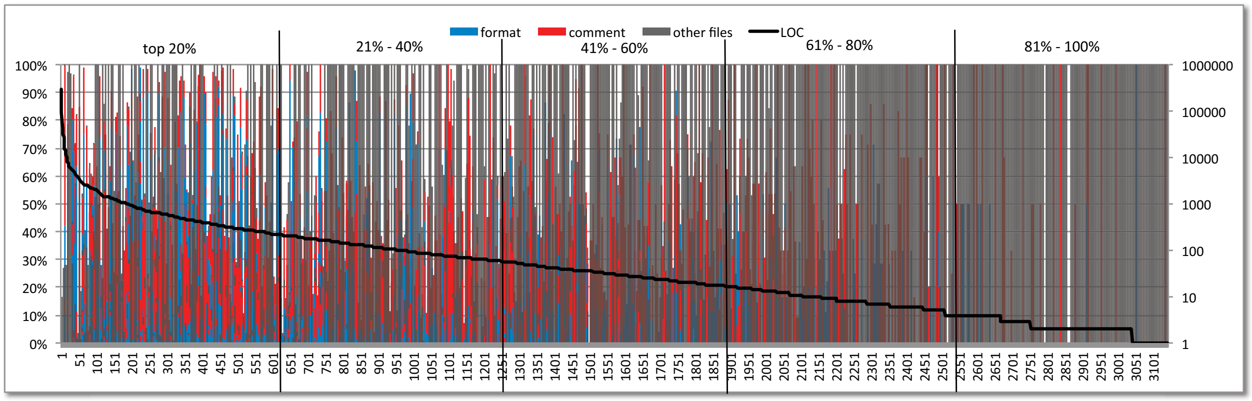
(b) SQuirreL SQL Client

group	format	comment	other files	total
top 20%	5.7%	3.5%	13.5%	22.7%
21-40%	11.0%	9.6%	19.0%	39.6%
41-60%	9.9%	10.9%	29.9%	50.6%
61-80%	6.5%	10.2%	43.7%	60.7%
81-100%	1.4%	3.6%	62.5%	67.5%

[1]The number of files modified in commits can be an indicator of commit size. However, in this experiment, we used LOC for seeing how many lines had been modified in every commit.



(a) Columba



(b) SQuirreL SQL Client

Figure 2. Format, comment, and other files modifications

Our answer to **RQ2** is also **NO**. In the experiment, middle and small commits also include such peripheral modifications. Moreover, we found that the rate of peripheral modifications on the middle and small commits was higher than the one on the large commits. Thus, we conclude that excluding large commits remains many peripheral modifications unintentionally.

Our answers to the RQs implies that excluding large commits is insufficient to exclude only peripheral modifications. We need another way to exclude them. For example, constructing a new repository from the original one should be a reasonable way.

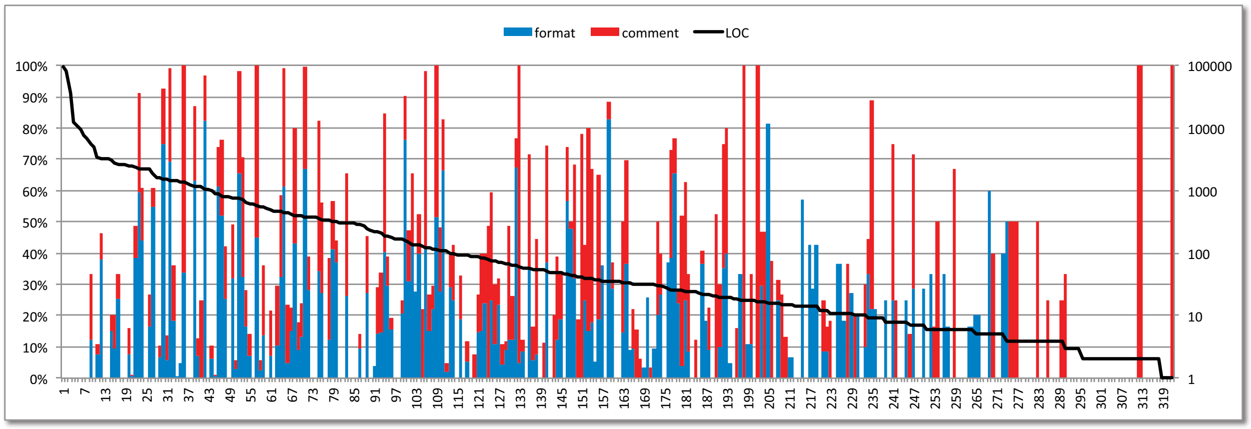
1) creating a new repository,
2) checking out revision r ($r = 1, 2, \cdots$),
3) removing comments from the revision and reformatting it,
4) committing the revision to the new repository,
5) going back to (2) if there are more revisions in the original repository.

Removing commits from source files and reformatting them is not such a complicated task. IDEs such as Eclipse or IntelliJ IDEA have the functions. Reusing the functions is a simple and easy way.

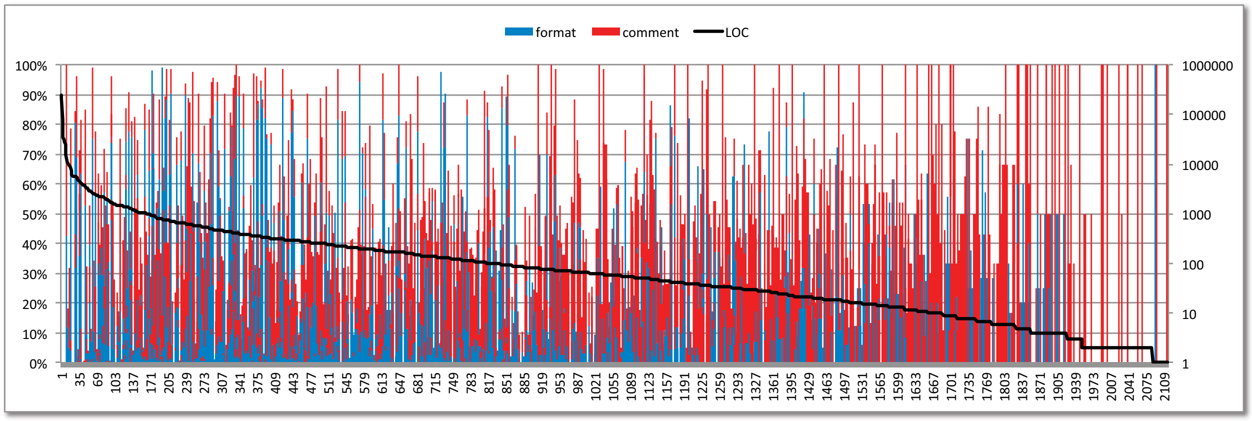
IV. Related Work

Fluri et al. proposed a method classifying modifications based on their semantics [15]. In their method, ASTs are generated from two versions of source code, each of which is before or after the modification. The classification is performed by comparing the two ASTs. On the other hand, our method adopts token sequence comparison because of its rapidity, however, AST comparison will work well if we can define how modifications are separated in AST-level.

Kawrykow and Robillard proposed a method to identify commits that are not significant for repository mining [16]. They defined "*surface*" modification, which does not have any impact on execution behavior. For example, changing variable names is a *surface* modification. They reported that, when we mine software repositories, analyzing such *surface* modification is useless. However, the authors have a different opinion about that. The authors think that any kind of code modifications may be useful for any purpose.



(a) Columba



(b) SQuirreL SQL Client

Figure 3. Format and comment modifications

Their *surface* modification include some code modifications such as renaming variables. On the other hand, our *code* modification include any kind of modification on program instruction code itself.

Hayashi et al. proposed a method to refactor past edit histories [17]. Their method automatically records all the operations that developers performed, then it reorders, merges, or deletes the operations. By using their method, we can obtain well-organized edit histories from the original one including many modifications for different purposes.

Some researchers conducted experiments for investigating large commits in software repositories. Hattori et al. reported that 5 or less source files are modified in most commits [18]. However, they also said that, in a part of commits, 100 or more source files are modified at the same time. Hindle et al. reported that large commits happen when we merge branched source files into it main stream, reformat source files, and delete unneeded source code [19].

As mentioned above, there are many research efforts related to commits in code repositories. However, none of them have investigated the amount of peripheral modifications in code repositories.

V. Conclusion

In this paper, we investigated how much there were modifications of *format*, *comment*, and *other files* in code repositories. We found there is a non-negligible amount of such modifications not only in large commits but also in small commits. Our findings implies excluding large commits is insufficient as a data cleansing before mining code repositories. We need more sophisticated ways to ignore such peripheral modifications.

As a reasonable way to realize that, we suggested that constructing a new code repository for mining code repositories. The construction will be performed as repeated operations of (1) checking out a revision from the original repository, (2) removing comments in the source files and reformatting the source files in the revision, and (3) commit the revision to the new repository. Consequently, built new repositories do not include peripheral modifications.

ACKNOWLEDGMENT

This study has been supported by Grants-in-Aid for Scientific Research (S) (25220003), Grant-in-Aid for Exploratory Research (24650011) from Japan Society for the Promotion of Science, and Grand-in-Aid for Young Scientists (A) (24680002) from the Ministry of Education, Culture, Sports, Science and Technology.

REFERENCES

[1] H. Kadgi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 2, pp. 77–131, 2007.

[2] A. E. Hassan, "The Road Ahead for Mining Software Repositories," in *Proceedings of the Frontiers of Software Maintenance*, 2008, pp. 48–57.

[3] R. Moser, W. Pedrycz, and G. Succi, "A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction," in *Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 181–190.

[4] H. Hata, O. Mizuno, and T. Kikuno, "Bug Prediction Based on Fine-Grained Module Histories," in *Proceedings of the 34th International Conference on Software Engineering*, 2012, pp. 200–210.

[5] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting Common Bug Prediction Findings Using Effort-Aware Models," in *Proceedings of the 26th International Conference on Software Maintenance*, 2010, pp. 1–10.

[6] H. Zhang, "An Investigation of the Relationships between Lines of Code and Defects," in *Proceedings of the 25th International Conference on Software Maintenance*, 2009, pp. 274–283.

Table II
RATE OF LOCS INCLUDED IN MODIFICATIONS OF FORMAT AND COMMENT

(a) Columba

group	format	comment	total
top 20%	4.7%	2.6%	7.3%
21-40%	18.2%	13.7%	31.9%
41-60%	16.9%	18.7%	35.6%
61-80%	13.4%	14.2%	27.6%
81-100%	5.4%	8.9%	14.1%

(b) SQuirreL SQL Client

group	format	comment	total
top 20%	6.4%	3.8%	10.3%
21-40%	14.3%	11.8%	26.1%
41-60%	12.2%	13.4%	25.6%
61-80%	13.4%	16.5%	29.9%
81-100%	7.2%	16.5%	23.7%

[7] C. Boogerd and L. Moonen, "Evaluating the Relation between Coding Standard Violations and Faults within and across Software Versions," in *Proceedings of the 6th Working Conference on Mining Software Repositories*, 2009, pp. 41–50.

[8] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining Version Histories to Guide Software Changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.

[9] G. Canfora, L. Cerulo, and M. D. Penta, "Identifying Changed Source Code Lines from Version Repositories," in *Proceedings of the 4th International Workshop on Mining Software Repositories*, 2007, pp. 14–22.

[10] M. Askari and R. Holt, "Information theoretic evaluation of change prediction models for large-scale software," in *Proceedings of the 2006 international workshop on Mining software repositories*, 2006, pp. 126–132.

[11] H. Kagdi, J. I. Maletic, and B. Sharif, "Mining software repositories for traceability links," in *Proceedings of the 15th International Conference on Program Comprehension*, 2007, pp. 145–154.

[12] E. Giger, M. Pinzger, and H. C. Gall, "Can We Predict Types of Code Changes? An Empirical Analysis," in *Proceedings of the 9th Working Conference on Mining Software Repositories*, 2012, pp. 217–226.

[13] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting Source Code Changes by Mining Change History," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004.

[14] F. Rahman, C. Bird, and P. Devanbu, "Clones: What is that Smell?" in *Proceedings of the 7th Working Conference on Mining Software Repositories*, 2010, pp. 72–81.

[15] B. Fluri, M. Wuersch, M. Pinzger, and H. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 725–743, 2007.

[16] D. Kawrykow and M. P. Robillard, "Non-essential changes in version histories," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 351–360.

[17] S. Hayashi, T. Omori, T. Zenmyo, K. Maruyama, and M. Saeki, "Refactoring edit history of source code," in *Proceedings of the 28th International Conference on Software Maintenance*, 2012, pp. 617–620.

[18] L. Hattori and M. Lanza, "On the nature of commits," in *Proceedings of the 23rd International Conference on Automated Software Engineering - Workshop Proceedings (ASE Workshops 2008)*, 2008, pp. 63–71.

[19] A. Hindle, D. M. German, and R. Holt, "What do large commits tell us?: a taxonomical study of large commits," in *Proceedings of the 3rd International Workshop on Mining Software Repositories*, 2008, pp. 99–108.