Source Code Reuse Evaluation by Using Real/Potential Copy and Paste

Takafumi Ohta, Hiroaki Murakami, Hiroshi Igaki, Yoshiki Higo, and Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University, Japan
1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan
{t-ohta, h-murakm, igaki, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—Developers often reuse existing software by copy and paste. Source code reuse improves productivity and software quality. On the other hand, source code reuse requires several professional skills to developers. In source code reuse, developers must locate reusable code fragments, and judge whether such reusable code is adequate to copy and paste into the source file under development. This paper presents extraction and analysis methods for developers' source code reuse behavior (copy and paste). Our method extracts developers' actual source code reuse (real copy and paste). Then, by using a code clone detection tool, the method extracts code fragments for (potential reuse). Our study of real and potential copy and paste provides a quantitative assessment for source code reuse by developers.**

Keywords—Reuse Based Software Engineering Teaching, Code Clone, Source Code Reuse

I. Introduction

As the size and complexity of software systems increases, adopting effective software development methodologies in software projects is crucial. RBSE is one of the software development paradigms that promise to increase software development quality and productivity [1]. In RBSE, developers can reuse existing software on various granularities, code fragments [2], components [3], and services [4]. Implementing software by using source code reuse techniques might reduce development cost. Such techniques also improve software quality compared to software implemented from scratch. On the other hand, source code reuse requires several professional skills to developers. Reusing existing software requires deep understanding of current reuse behavior. Developers have to know how to locate reusable code fragments. Even if developers gained the reusable code fragments, they have to merge the code fragments to their developing software adequately. As a result, nurturing RBSE technique is required crucially.

One of the difficult points for nurturing RBSE is to prepare the evaluation criteria for the source code reuse by developers. If it becomes possible to evaluate source code reuse activities, the chance for successful RBSE increases. Therefore, in our research, we propose an extraction method of source code reuse for each developer, and an evaluation criteria to judge whether each reuse behavior is good or not.

During implementing code, developers sometimes copy code fragments and paste them on the source file. In order to evaluate the source code reuse behaviors, we obtain such copy and paste (in short, C&P) by each developer (we call this "real C&P ") precisely. In addition, we make a comparison between the real C&P and the potential C&P. If there exist two code fragments that are similar to each other in a software project, a developer could have reused one of two code fragments to implement the other code fragment. In conventional research, code clone detectors were used for presuming source code reuse [5]. We call such source code reuse presumed by code clone detectors "potential C&P". Ideally, it is desirable that the real C&P does not require any edits after pasting. On the other hand, since the ideal code fragment does not necessarily exist, we cannot always perform such ideal source code reuse. Therefore, by comparing the real C&P to the potential C&P, we evaluate whether the real C&P requires too many edits after pasting.

In this paper, we propose extraction and comparison methods for real/potential C&P by developers to evaluate developers' behaviors on source code reuse. The result of our case study ensured that our methods were able to assess developers' reuse behaviors quantitatively. The next section describes the reuse based software engineering in our study.

II. Reuse Based Software Engineering

Reuse Based Software Engineering (RBSE) represents a software development paradigm that promises to shorten development cycles based on software reuse [1]. Quality and productivity of software could be increased by reusing existing software [1]. In software development, developers reuse various units of software, code fragments [2], components [3], and services [4].

A success of RBSE depends on the skilled developers [6]. If a developer does not sufficiently understand the existing software, the developer cannot adequately detect and adopt reusable units. In addition, ad-hoc reuse of the code fragments sometimes introduce faults [7]. For this reason, some researchers consider education of RBSE as one of the most important topic in the software engineering [6]. Actually, a large amount of research on education for RBSE have been performed [8][9][10][11].

Our research motivation is to construct a criteria to evaluate source code reuse for each developer. In order to nurture developers based on the viewpoint of source code reuse, assessments of the actual reuse behaviors based on the practical criteria are needed . Constructed criteria enable developers to judge whether each C&P performed by them is good or not.

A process on C&P consists of the following four steps. First, a developer opens a source file that contains reusable

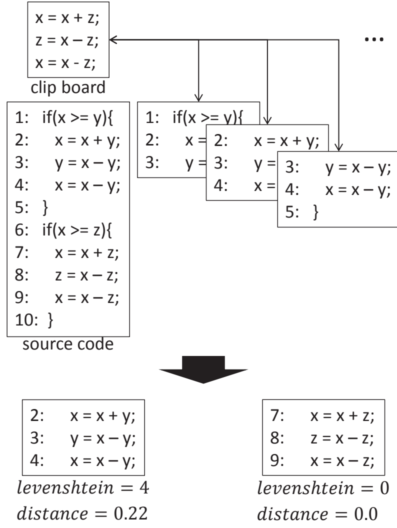


Fig. 1: An extraction process of a real copied code

code fragments. Second, he/she copies the specific code fragments. Third, he/she pastes the copied code fragments into the developing source file. Finally, he/she edits the pasted code as needed.

In the final step, it is desirable that the C&P does not require too many edits of the pasted code. Hence, we focus on the size of edits after pasting as the evaluation criteria of source code reuse. Namely, our criteria assess C&P with few number of edits after pasting is good. Here, the criteria require an adequate threshold for the amounts of edits. It is difficult to find an adequate value as the threshold because the threshold might depend on each real C&P. In order to configure the threshold for each real C&P, we focus on potential C&P. If there exist any code fragments that are similar to the code fragments implemented by a developer, the developer might have reused the code fragments potentially. A Comparison between real/potential C&P enables us to identify ideal C&P that includes the fewest amounts of edits in the real C&P.

Hereafter, we explain our methodology to extract the real/potential C&P.

III. Real and Potential C&P

In this section, we explain definitions and extraction methods of real/potential C&P. Our extraction method assumes that developers use a VCS (Version Control System), and both real/potential C&P are extracted from each commit in the software repository.

A. Real C&P

Developers often copy some code fragments and paste them into a developing source file. In our definition, "real C&P" consists of "a real copied code fragment (in short, a real copied code)" and "a real pasted code fragment (in short, a real pasted code)". We define a set of the file path, the start line and end line of the copied code fragment as the "real copied code". Moreover, we define a set of the file path, the start line and end line of the pasted code fragment as the "real pasted code". Developers often edit the copied code fragment after pasting. So, in our definition, "real pasted code" means the code fragment that is edited by developer after pasting until the next commit.

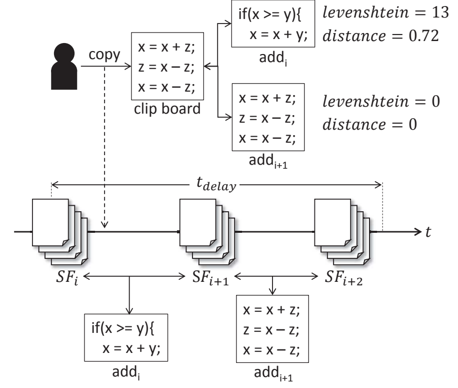


Fig. 2: An extraction process of a real pasted code

In order to extract real C&P, we utilize a clipboard history, an application history and an edit history of a set of source files in a software project. The clipboard history records a copied document and time when the document is copied. The application history records name of the application that developers have executed (e.g. Eclipse, text editor), the file path that such application has opened and open/close time of each file. The edit history of a set of source files records snapshots of each source file at a short-time interval. Many editors and IDEs have the capability to record the edit history automatically.

Next, in order to extract a real copied code, our process specifies a snapshot of a source file just before a developer has copied a code fragment to the clipboard using the clipboard history, the application history, and the edit history. The specified snapshot of the file contains a real copied code. Fig. 1 shows the extraction method of the start and end line of the real copied code. In order to specify the real copied code, our method calculates similarities between the code fragment in the clipboard and each code fragment in the source file including the real copied code from the head of the source file based on the Eq. 1, successively.

$$distance(clip, f_i) = \frac{levenshtein(clip, f)}{length(clip)} \quad (1)$$

The variables $clip$ and f_i represent the copied code in the clipboard history and each extracted code fragment in the source file including the real copied code, respectively. Here, we regard $clip$ and f_i as character strings. The function $levenshtein$ calculates the Levenshtein distance between $clip$ and f_i. The function $length$ calculates the length of $clip$. Hence, $distance$ represents the ratio of the modification between the $clip$ and the f_i as the similarity. Here, we use the threshold T_s to decide whether the copied code is pasted into the source files.

The code fragment that has the smallest similarity is considered as a real copied code. Fig. 1 shows the results

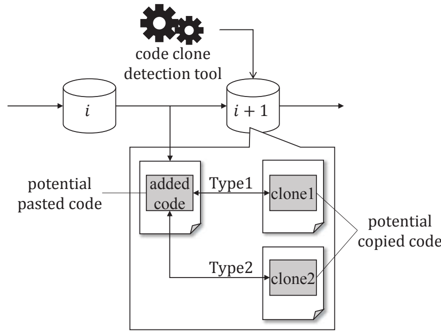


Fig. 3: An overview of the extraction method for the potential C&P

of the comparisons. The code fragments whose similarity is less than 0.3 are the code fragments of the 2nd line to the 4th line and the 7th line to the 9th line. In this example, the code fragment in line 7 to 9 is regarded as the start and end lines of the real copied code.

Fig. 2 shows an extraction process of the real pasted code. In the figure, SF_i indicates a snapshot of source files included in the edit history. First, our process extracts the added code between the consecutive snapshots that are recorded within t_{delay} time from the time when SF_i is recorded. In Fig. 2, there exist two snapshots named SF_{i+1} and SF_{i+2}. The code fragment $\mathrm{add_i}$ in Fig. 2 represents the added code between the consecutive snapshots SF_i and SF_{i+1}.

Next, similarities between the copied code and each extracted added code are calculated by the Eq. 1, successively. Here, the variable f_i represents the extracted added code between consecutive snapshots of the source files. Hence, Eq. 1 represents a ratio of a modification between the $clip$ and the f_i. Here, we use the threshold T_h to decide whether the copied code is pasted into the source files. In this paper, we set T_h as 0.3. Fig. 2 shows $distance(clip, addi)$ is 0.72, and $distance(clip, addi + 1)$ is 0. In this case, we regard $\mathrm{add_{i+1}}$ as the "temporal" real pasted code. A developer may edit the pasted code before the next commit. Then, based on the "temporal" real pasted code, our method extracts real pasted code from the next commit in the VCS.

B. Potential C&P

If there exist some code fragments that are similar to the implemented code by a developer, the developer could have implemented the code by C&P, potentially. Potential C&P consists of potential copied/pasted code fragment (in short, copied/pasted code) and the similarity type.

Fig. 3 shows an overview of our extraction method for a potential C&P. In Fig. 3 , first, a code clone detecter finds clone pairs from source files in the revision r_{i+1}. Next, our method specifies every potential C&P from the clone pairs. In the potential C&P, only one of the two cloned fragments constituting a clone pair is contained in the added code.

In each clone pair, we call the cloned fragment in the added code as "potential pasted code", and another code fragment as "potential copied code". Here, based on the Bellon's definition [12], we classify each clone pair into type1 to 3 as the similarity type of the potential C&P.

If the potential pasted code has multiple potential copied code, we select a potential copied code that resembles the pasted code most based on the similarity types (type1, type2 and type3).

IV. Evaluation Criteria for Real/Potential C&P

In order to evaluate each real C&P by developers, first, we extract every real C&P and every potential C&P for each commit. Then, we evaluate each real C&P according to the following steps.

Step1: We compare the real copied code with the real pasted code. We regard a combination of the copied code and the pasted code as a clone pair, and classify the real C&P that contains the copied/pasted code based on the types of clones [12]. Here, if the real C&P does not match any types of clones, the real C&P is classified into type0.

Step2: We investigate whether the real pasted code has the corresponding potential pasted code based on the following p-OK value. We define the p-OK value using the definition of the OK value defined in [12]. The p-OK value is calculated from the function $contained$ denoted in Eq. 3.

$$p - OK = max(contained(f_1, f_2), contained(f_2, f_1)) \quad (2)$$

$$contained(f_1, f_2) = \frac{|lines(f_1) \cap lines(f_2)|}{|lines(f_1)|} \quad (3)$$

Here, f_1 and f_2 mean a code fragment respectively, and $lines(f)$ means a set of lines in a code fragment f. If an evaluated p-OK value is more than a threshold, we identify f_1 and f_2 are correspondent. In this evaluation method, we set the threshold as 0.7, same as [12]. If the real pasted code does not have the corresponding potential pasted code, the real C&P is evaluated as "Bad". "Bad" means that the developer has edited most of the real copied code after pasting, and the real pasted code does not have any similar code fragments. That is, it might have been difficult to implement the real pasted code by source code reuse. On the contrary, if the real pasted code has the corresponding potential pasted code, this evaluation process goes to Step3.

Step3: We compare the type between the real C&P and the potential C&P. Here, if the type of the real C&P does not match the type of the corresponding potential C&P, the real C&P is evaluated as "Better". "Better" mean that the developer could have reused better code fragment that is more similar to the real pasted code. On the contrary, if the type of the real C&P matches the type of the corresponding potential C&P, the real C&P is evaluated as "Best". "Best" means the developer performed the ideal C&P.

We conducted a case study based on our evaluation criteria.

V. Case Study

In this section, we explain the results that we have applied our extraction method to students' software development project. Table I shows that the development scale and the

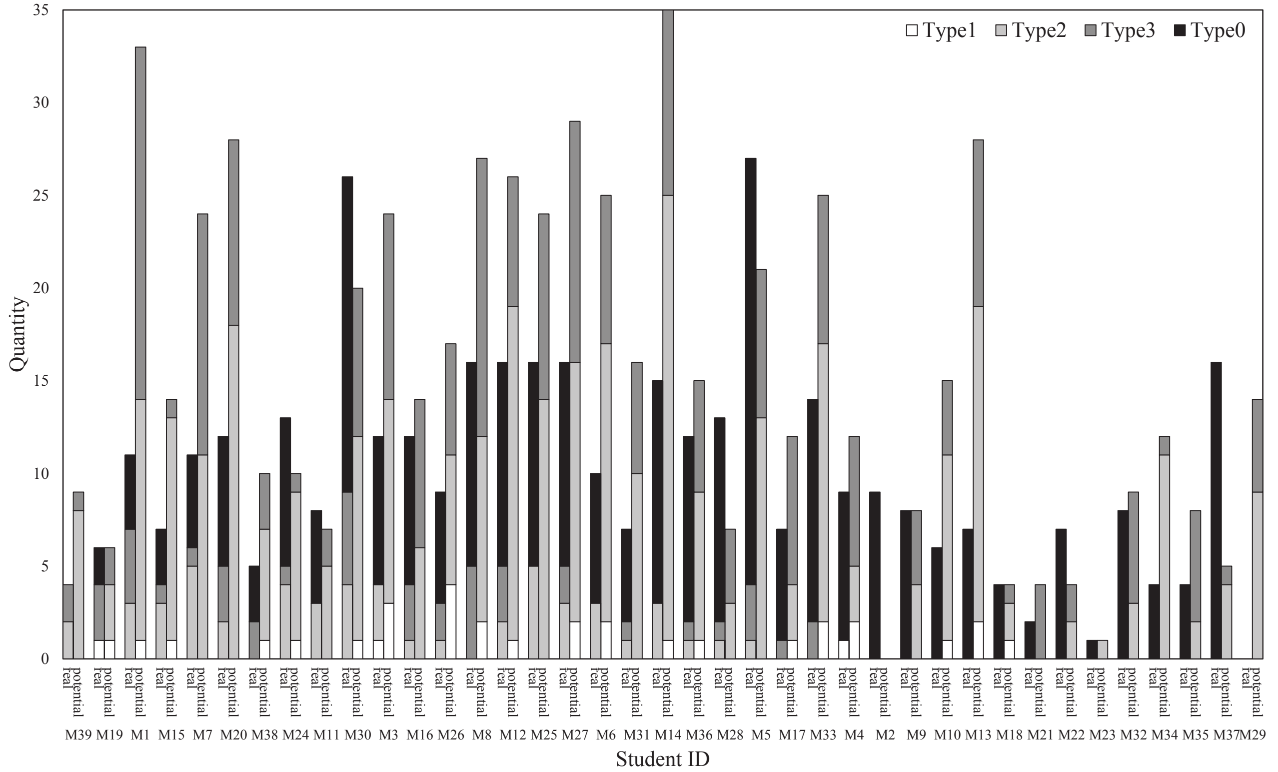


Fig. 4: Extracted real and potential C&P

number of corrected logs for each student team. In the project, each team has 4-5 students, and the duration of the project is 4 days (students developed software for 8 hours per day). The project objective is to develop a web application based on the specifications of the ticket selling system using HTML, JavaScript and Java. The development environment is same for all students (they used Eclipse on the laptop PC).

Table I shows Team 1 developed the application with 18,420 LOC by 192 commits. Here, 5,019 LOC and 145 commits indicate the values that are involved in Java source files, respectively. The members of Team1 copied the various documents to the clipboard 1,130 times. 582 out of 1,130 documents indicate the code fragments in Java. The number of the application history and the accumulated edit history for every source file of Team1 are 11,879 and 25,039, respectively.

We use the code clone detector called $CDSW$ [13]. $CDSW$ is a statement based code clone detector. $CDSW$ can detect the type1 to type3 code clone. We set the minimal clone length and the number of allowed gapped statements for parameters of $CDSW$, as 20 and 3 respectively. The minimal clone length means the number of tokens in the detected clones. In our case study, we omitted the real C&P including the real copied/pasted code that the number of lines is less than 3.

TABLE I: Development scale and recorded logs

Team	LOC	commits	clipboard history	edit history	application history
1	5,019(18,420)	145(192)	582(1,130)	25,039	11,879
2	3,814(17,512)	79(123)	649(1,414)	539,872	13,763
3	4,722(18,877)	150(180)	989(2,070)	521,884	16,290
4	4,002(17,415)	114(145)	607(1,423)	330,727	12,120
5	3,654(17,005)	95(145)	744(1,433)	295,884	10,126
6	4,684(18,152)	111(147)	813(1,253)	105,032	11,080
7	4,407(17,811)	91(124)	743(1,599)	309,587	13,184
8	4,305(17,948)	56(84)	316(711)	380,682	11,644
9	3,619(16,968)	50(107)	877(1,395)	258,704	10,297

Fig. 4 shows the number of extracted real/potential C&P by all students for every type. The x-axis and y-axis of Fig. 4 represent the Student ID and the number of C&P, respectively. The Student ID is sorted in ascending order by the ratio of the real C&P of which type is 0.

From Fig. 4, a large part of the real C&P was classified into type0. Namely, almost all students edited the real copied code excessively after pasting. On the other hand, the number of the potential C&P is more than the number of the real C&P for each student. In other words, many students miss the chances to utilize reusable code.

VI. Discussion

We evaluated every real C&P for each student based on our evaluation criteria. We discuss the result of the evaluation for each real C&P.

A. *Classification of the Real C&P*

In Fig. 5, every real C&P is classified into the following three categories, “Best”, “Better”, and “Bad”. The x-axis and y-axis represent the Student ID and the number of the real

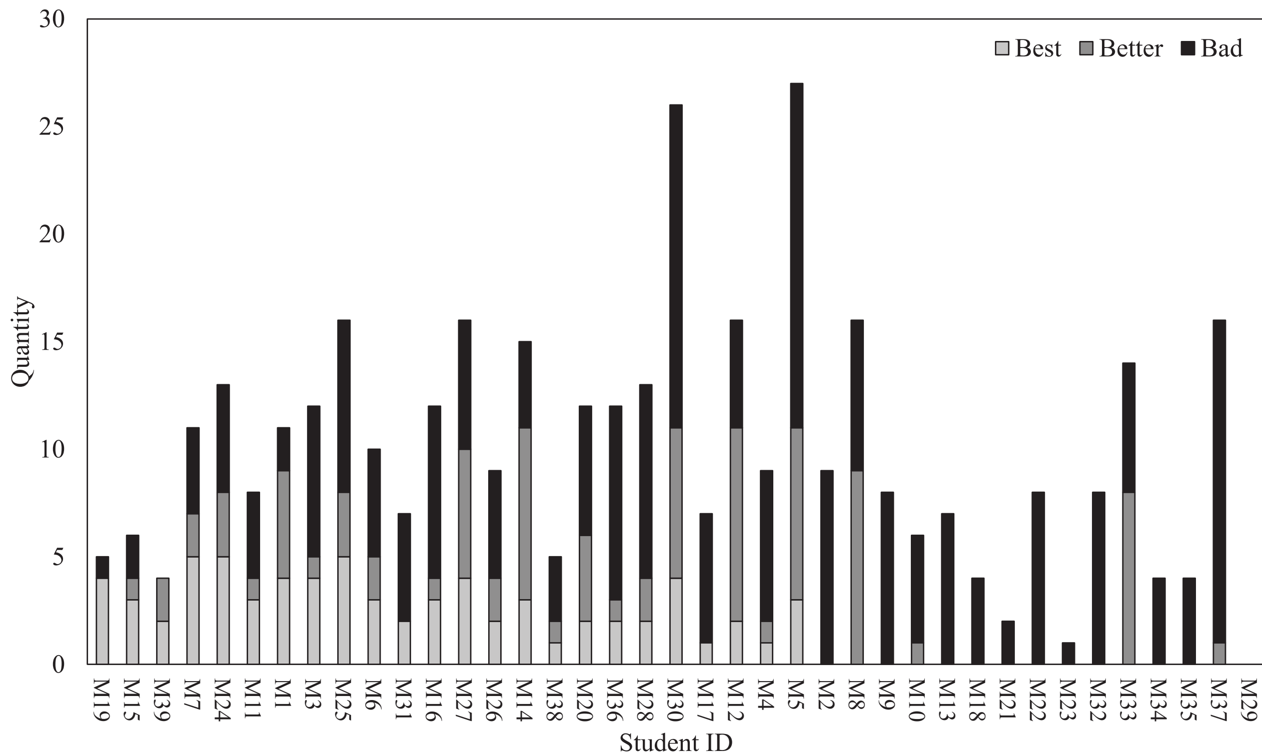


Fig. 5: Classified real C&P for every student

C&P, respectively. The Student IDs are sorted in ascending order by the ratio of the "Best" real C&P to all real C&P.

"Best" C&P
: The real C&P has the corresponding potential C&P, and the types of the real/potential C&P are same.

"Better" C&P
: Though the real C&P has the corresponding potential C&P, the types of the real/potential C&P are different.

"Bad" C&P
: The real C&P does not have the corresponding potential C&P.

On the average, the students performed the real C&P about ten times, and one-third of them are classified as "Better". The category "Better" implies that the students could have reused the better code fragment that requires fewer edits.

A large part of the real C&P is classified into "Bad". The category "Bad" implies that it was difficult to implement the real pasted code by C&P.

These two categories may be able to promote improvement of the source code reuse to developers.

B. The Analysis of the Correlation Between the Authorship and the C&P

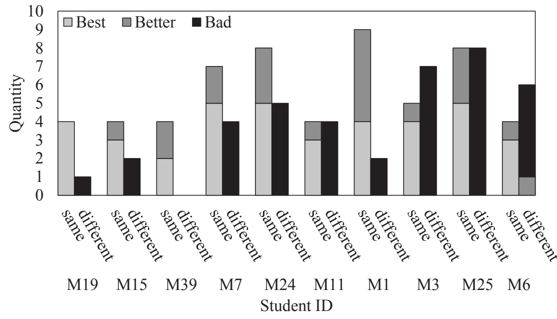
Fig. 6 and Fig. 7 show authorship information of the real/potential C&P of top/worst ten students. In the x-axis of the figures, "same" means that authorship of both real copied/pasted code is same, and "different" means that authorship of both real copied/pasted code is different. The student IDs are sorted in ascending order by the ratio of the "Best" real C&P to all real C&P. As shown in Fig. 6, almost all "Bad" real C&P has the different authorship between the real copied code and the real pasted code. This result implies they have reused the unsuitable code fragment, because they did not understand well the code fragment that anyone else has implemented. On the other hand, Fig. 7 shows the types of the potential C&P are independent of the authorship between the potential copied code and the pasted code, comparatively.

As a result, in order to improve source code reuse, it can be said that the developers should understand the code fragments that anyone else has implemented. Furthermore, the authorship information about the real/potential C&P can serve as a means to obtain efficiently the code fragments that a developer should understand.

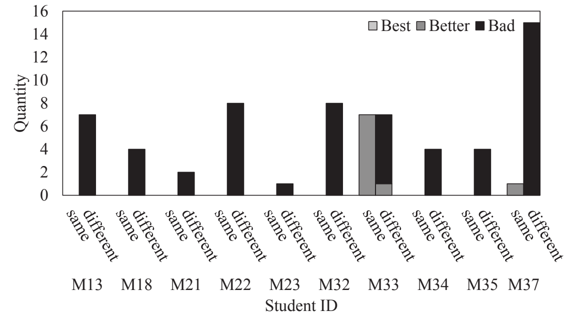
C. Related Work

The research, about extracting the reused code fragments, has been well-studied. Kim et al. conducted an ethnographic study using a logger that records editing behavior like select, copy, paste and undo [14]. Heinemann et al. extract the potentially copied code by utilizing the clone detection tool [5].

More recently, some studies about the reusable source code existing in the developed software is studied. Martinez et al. analyzed whether the committed code fragments already exist in the developed software [15]. The case study shows that 3-17% of the commit is redundant. Barr et al. also studied

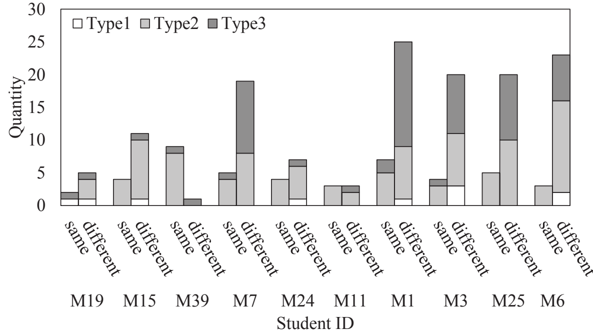


(a) Authorship of the real C&P of the top 10 students

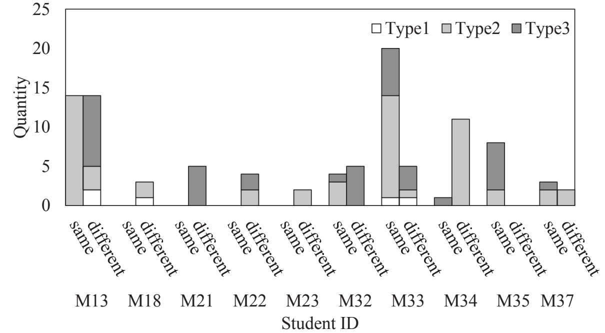


(b) Authorship of the real C&P of the worst 10 students

Fig. 6: Authorship of the real C&P



(a) Authorship of the potential C&P of the top 10 students



(b) Authorship of the potential C&P of the worst 10 students

Fig. 7: Authorship of the potential C&P

the redundancy about the commit [16]. They focused on the graftability of the commit that indicates the ratio of the code fragment could be developed by reusing the source code existing in the developed software and all added code fragment by line granularity. The goal of these studies is validating the hypothesis utilized in the research field of automatic program repair [17][18]. In our research, we aim to identify both real/potential C&P during the software project to assess developer's reuse behaviors.

Kotonya et al. developed the framework of the education of the service-based RBSE [6]. In this research, they mentioned that the difficulty of educating RBSE is how to make the benefits of RBSE visible to students. In their project, the problem scenario is designed to provide a realistic experience of both "development for reuse" and "development with reuse" as well as some element of project management. In our research, we focused on code fragment based software reuse in the viewpoint of the "development with reuse".

VII. Threats to Validity

Our proposed method cannot extract the source code reuse in which developers do not use a clipboard, and the real/potential copied code that exists outside the developing project. In fact, some students copied code fragments from the web pages, in our case study. In our future research, we are planning to evaluate the source code reuse spanning multiple projects and web pages.

Our method using the clipboard history, the application history, and the edit history might extract the wrong real copied code and the wrong real pasted code. If there exist multiple identical code fragments that are similar to the real copied code in the same file, our method may extract the wrong real copied code. In addition, our method may extract the wrong pasted code, when a developer edits a code fragment excessively before the edit history of the file containing the fragment is recorded.

Loggers that extend IDE might improve this kind of problem. Actually, Kim et al. [14] developed the logger by extending the text editor of the Eclipse that could record the range of selected text, and the length and offset of text entries. Although this type of loggers depends on the IDE, the loggers can extract the real C&P with great accuracy. On the other hand, our extract method is independent of IDEs. We will try to combine our method and such loggers to extract the real C&P with great accuracy and robustly in our future research.

We utilized a code clone detection tool "CDSW" that could detect sentence-based code clones for extracting the potential C&P. We compare the real C&P with the potential C&P line by line for evaluating the real C&P. For this reason, the evaluation result of the real C&P might change by using different code clone detection tools.

Source code reuse behaviors may alter according to the development environment such as programming languages, developer's skills, and IDEs. In our case study, the subjects were university students. They developed a web application with

Java, JavaScript, and HTML using Eclipse IDE. Therefore, our case study might not reflect real-world software development environment. On the other hand, in any development environment, developers might reuse source code. We might say that our evaluation method of the real C&P is significant from the perspective of improvement of source code reuse behaviors.

VIII. Conclusion

Reuse of the existing source code by C&P is one of the common software development behaviors. Earlier studies report that the C&P in an ad-hoc manner might cause maintenance problems. Several analyses of the C&P are conducted to clarify why developers perform C&P. On the other hand, there are few methods that evaluate the C&P for each developer.

In this research, we proposed the evaluation criteria for the real C&P. Our criteria have indicated that the real C&P could be classified into three categories ("Bad", "Better", and "Best") based on the comparison of the real C&P and the potential C&P. On average, we confirmed 80 percent of the real C&P (That is, "Bad" or "Better" C&P) in the case study has room for improvement. In addition, through the authorship analysis in the case study, we confirmed that the analysis might support developers' source code understanding.

Acknowledgment

This work was supported by MEXT/JSPS KAKENHI 24700030 and 24680002.

References

[1] E. S. Almeida, A. Alvaro, D. Lucredio, V. C. Garcia, and S. R. L. Meira, "A survey on software reuse processes," in *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pp. 66–71, Aug 2005.

[2] T. Moriwaki, Y. Yamanaka, H. Igaki, N. Yoshida, S. Kusumoto, and K. Inoue, "Towards an analysis of who creates clone and who reuses it," in *Proceedings of the Eighth International Workshop on Software Clones*, Feb 2014.

[3] X. Cai, M. Lyu, K.-F. Wong, and R. Ko, "Component-based software engineering: technologies, development frameworks, and quality assurance schemes," in *Proceedings of the 7th Asia-Pacific Software Engineering Conference*, pp. 372–379, 2000.

[4] M. Huhns and M. Singh, "Service-oriented computing: key concepts and principles," *Internet Computing, IEEE*, vol. 9, pp. 75–81, Jan 2005.

[5] L. Heinemann, F. Deissenboeck, M. Gleirscher, B. Hummel, and M. Irlbeck, "On the extent and nature of software reuse in open source java projects," in *Proceedings of the 12th International Conference on Top Productivity Through Software Reuse*, ICSR'11, (Berlin, Heidelberg), pp. 207–222, Springer-Verlag, 2011.

[6] G. Kotonya and J. Lee, "Teaching reuse-driven software engineering through innovative role playing," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, (New York, NY, USA), pp. 276–282, ACM, 2014.

[7] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "Cp-miner: finding copy-paste and related bugs in large-scale software code," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 176–192, Mar 2006.

[8] A. Parrish, B. Dixon, D. Hale, and J. Hale, "A case study approach to teaching component based software engineering," in *13th Conference on Software Engineering Education and Training*, pp. 140–147, March 2000.

[9] M. Sitaraman, T. Long, B. Weide, E. Harner, and L. Wang, "A formal approach to component-based software engineering: education and evaluation," in *Proceedings of the 23rd International Conference on Software Engineering*, pp. 601–609, 2001.

[10] K. Qian and X. Fu, "Teaching component-based software development," in *Proceedings of the 21st IEEE-CS Conference on Software Engineering Education and Training Workshop*, pp. 13–15, April 2008.

[11] A. Finkelstein, "Software engineering education: a place in the sun?," in *Proceedings of the 16th International Conference on Software Engineering*, pp. 358–359, May 1994.

[12] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, Sept 2007.

[13] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Gapped code clone detection with lightweight source code analysis," in *Proceedings of the 21st IEEE International Conference onProgram Comprehension*, pp. 93–102, May 2013.

[14] M. Kim, L. Bergman, T. Lau, and D. Notkin, "An ethnographic study of copy and paste programming practices in oopl," in *Proceedings of the International Symposium on Empirical Software Engineering*, pp. 83–92, Aug 2004.

[15] M. Martinez, W. Weimer, and M. Monperrus, "Do the fix ingredients already exist? an empirical inquiry into the redundancy assumptions of program repair approaches," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, (New York, NY, USA), pp. 492–495, ACM, 2014.

[16] E. T. Barr, Y. Brun, P. Devanbu, M. Harman, and F. Sarro, "The plastic surgery hypothesis," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, (New York, NY, USA), pp. 306–317, ACM, 2014.

[17] 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*, ICSE '12, (Piscataway, NJ, USA), pp. 3–13, IEEE Press, 2012.

[18] Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang, "The strength of random search on automated program repair," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), pp. 254–265, ACM, 2014.