Evaluation of Duplicated Code Detection Tools in Cross-Project Context

Johnatan A. de Oliveira[1], **Eduardo M. Fernandes**[1], **Eduardo Figueiredo**[1]

[1]Software Engineering Lab, Department of Computer Science,
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil

{johnatan-si, eduardomorfernandes}@ufmg.br, figueiredo@dcc.ufmg.br

Abstract. *Two or more code segments are considered duplicated when there is a high rate of similarity among them or they are exactly the same. Aiming to detect duplicated code in single software projects, several tools have been proposed. However, in case of cross-project detection, there are few tools. There is little empirical knowledge about the efficacy of these tools to detect duplicated code across different projects. Therefore, our goal is to assess the efficacy of duplicated code detection tools for single projects in cross-project context. It was concluded that the evaluated tools has no sufficient efficacy in the detection of some types of duplicated code beyond exact copy-paste. As a result, this work proposes guidelines for future implementation of tools.*

1. Introduction

Duplicated code is a bad smell defined as replication of code segments through the source code of a software project. One of the problems derived from duplicated code is the high complexity of refactoring a system with high number of duplicated code. The reason is because a refactoring would demand changes in various parts of the code. The dissemination of errors through the system is another issue caused by duplicated code practices, because errors in a replicated segment are spread throughout the system [Fowler 1999].

In case of duplicated code detection in single software systems, there are many proposed tools [Juergens et al. 2009, Hotta et al. 2010] and comparative studies [Bellon et al. 2007]. However, in cross-project detection context, there is no empirical study. This type of detection may point to similar code segments that appears in different projects. Considering systems from a single business domain, it could support code reuse in development of projects [Bruntink et al. 2005], such as in a software product line (SPL).

This study aims to assess the efficacy of some duplicated code detection tools for single projects in cross-project context. For this purpose, it was conducted a literature review for identification of tools in order to assess their efficacy. As a result, this study pointed to a need of effective tools in the detection of more types of duplicated code than copy-paste. Furthermore, this work proposes some guidelines to support the development of duplicated code detection tools, based on demands detected through this study.

The remainder of this paper is organized as follows. Section 2 presents the background, describing the types of duplicated code and SPL, and related work that proposed comparison of duplicated code detection tools. Section 3 discusses the experimental

setup, including the research questions. Section 4 presents the results obtained through this study, including guidelines proposed for future implementation of detection tools. Finally, conclusions and future work are presented in Section 5.

2. Background and Related Work

According to [Fowler 1999], bad smells are symptoms of problems in source code. There are several bad smells defined in literature, but this study focuses on duplicated code, which has four types [Bellon et al. 2007]: *Type I* that is simple copy-paste, *Type II* that consists of copy-paste with simple variations in identifiers, literals, types, etc., *Type III* that is copy-past with variations in operations and code blocks, and *Type IV* that consists of code segments with the same computation, but different implementation.

A SPL is a development paradigm based on software platforms (common features among systems) and customizable features (accordingly to client needs). Systems of a SPL compose a software product family, and their commonalities are reused in the development of new products [Halmans and Pohl 2003]. Therefore, these systems can be useful to assess the efficacy of tools in the detection of duplicated code, and this study took advantage of this property to assess tools.

Previous studies assessed the efficacy of duplicated code detection tool. The work of [Bellon et al. 2007] compares six tools in order to identify their recall and precision, considering the detection techniques used by each tool and using C and systems developed in Java programming languages, they noted that the abstract syntax tree-based approaches tended to have higher precision, however the runtime is greater, token-based approaches had higher recall but were faster. In other direction, [Burd and Bailey 2002] focused on the evaluation of five duplicated code detection tools in order to identify the benefits of detection for preventative maintenance. In his work too have shown that token based techniques have a high recall but suffer from many false positives.

The present study, in turn, aims to evaluate a set with twenty duplicated code detection tools, in order to assess their efficacy in the context of cross-project detection, as well as to report the main features and issues of each tool. Furthermore, this study aims to conduct tests, for each tool, using as entry a set with twenty-one real software projects from an specific business domain. Its goal is to assess the efficacy of tools in the detection of the four types of duplicated code.

3. Experimental Setup

This section presents the experimental setup for this study, including: the strategy for selecting target systems to be used as input in tool evaluation, the selection of duplicated code detection tools, and the procedures for installing, running and evaluating each tool using different sets of system projects as input. Each step of this study is illustrated in Figure 1 and described in the following subsections. In order to perform this study, the following research questions were conceived concerning duplicated code detection tools:

- *RQ1: What duplicated code detection tools have been reported in literature?*
- *RQ2: Are the identified tools able to detect the four types of duplicated code in cross-project context?*
- *RQ3: Are the evaluated tools able to detect cross-project duplicated code?*

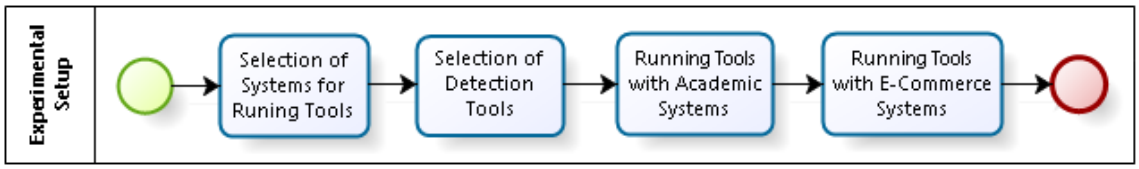

Figure 1. Steps of the conducted study.

3.1. Selection of Systems for Running Tools

In order to assess the efficacy of duplicated code detection tools through the running of each tool, two sets of software systems were selected: a systems academic (SPL) that was chosen because it allows us to evaluate whether the duplicated code tools are able to find reused code and e-commerce systems mined from GitHub[1]. The former was conceived to support the preliminary assessment of efficacy for each available tool, and the latter was conceived to be used in the assessment of tools filtered in the previous assessment.

The systems that compose the academic corpus were developed by graduate students through aspect-oriented programming with code reuse in an academic course between 2013 and 2014, at the Federal University of Minas Gerais (UFMG). These systems were implemented based on a platform system called SimulES, that consists of a software product line. They were chosen because of the conviction that exists duplicated code across all the systems, at least in respect to the used platform.

The e-commerce domain was chosen because of activities and the common functions across systems (e.g., purchases and payments). The process of system selection consisted of three steps: selection of 100 e-commerce bookmarked systems from GitHub, cleaning of selected systems (files other than .java files were removed), and discard of systems not implemented in Java (79 were discarded, remaining 21 systems).

3.2. Selection of Detection Tools

Aiming to select duplicated code detection tools to be evaluated in this study, we conducted an *ad hoc* review in order to identify the most cited tools in the literature. This literature review was based on the protocol for systematic literature reviews [Kitchenham et al. 2009], that suggests the following steps: planning (including the definition of a review protocol), conducting (including the data extraction) and reporting (in this case, we present a table with the collected data).

The literature review returned 19 tools described in Table 1. Columns in Table 1 indicate information about the tools: plug-in (PLG), developed programming language (PL), open-source or freeware (OSF), detection programming languages (DE), other detected bad smells (OTHER), is online (ON), documented (DO), graphical user interface (UI), detection technique (TE), and release year (YR) – N/A indicates that the information is not available in literature or website of the tools.

In order to reduce the amount of tools to be evaluated (given the infeasibility to evaluate all the found tools), the following inclusion criteria were defined for this study: the tool must be available online, it must to detect duplicated code in entire software projects, not in single source files only, it must have been cited in more than one paper,

[1]http://www.github.com/

it must be compatible with Java programming language, and it must be open source or freeware for non-commercial use.

In the selection of duplicated code detection tools, the tools CP-Miner [Li et al. 2004], Java CloneDR [Bellon et al. 2007] and Pattern Insight Clone Detection[2] were discarded because they are paid tools. Furthermore, the tools CCFinderX[3], CDSW [Murakami et al. 2013], Covet [Burd and Bailey 2002], Dup [Baker 1993], Duploc [Ducasse et al. 1999], Scorpio [Higo et al. 2013] and Simian[4] had to be discarded because they are not available online.

3.3. Running Tools with Academic Systems

This step consisted of installing and running tools in order to assess their efficacy in detection of duplicated code. These tools were installed in a Microsoft Windows 7 environment. DECKARD [Jiang et al. 2007] and DuDe [Wettel and Marinescu 2005] were discarded because it was not possible to install them. For this purpose, it was used the academic corpus as input, that is a SPL (in case, SimulES) and, therefore, the evaluated tools should be able to identify similar code in the SPL.

Considering that some tools were discarded in the previous step, only Checkstyle [Moha et al. 2010], Clone Digger [Bulychev and Minea 2008], CodePro Analytix[5], Condenser [Mealy et al. 2007], PMD [Juergens et al. 2009], SDD[6], and SonarQube [Campbell and Papapetrou 2013] were evaluated in this step. The process of running the selected tools with the academic corpus consists of SimulES been submitted to clone detection, whose results based the filtering of tools to be used in the next study step.

3.4. Running Tools with e-Commerce Systems

After the evaluation of tools in the previous step, it was verified that only PMD and Atomiq are able to detect at least Type I of cross-projects duplicated code. Then, they were evaluated using the e-commerce corpus, in order to assess their efficacy in cross-project duplicated code detection in real software projects. Both tools display the source directory of each file involved in the duplicated code occurrence, so it was possible to identify cross-project detection.

4. Evaluation and Discussion

This section discusses the results obtained through this study, focusing on answering each research question (Section 4.1). Furthermore, Section 4.2 presents some guidelines to support future implementation of duplicated code detection tools.

4.1. Answering the Research Questions

RQ1: What duplicated code detection tools have been reported in literature? This study identified some tools through an *ad hoc* literature review described in Section 3.2. These tools are presented in Table 1. Among the 20 studied tools, we identified that: 10 are

[2]http://patterninsight.com/products/clone-detection/
[3]http://www.ccfinder.net/ccfinderx.html
[4]http://www.harukizaemon.com/simian/
[5]https://developers.google.com/java-dev-tools/download-codepro
[6]http://sourceforge.net/projects/sddforeclipse/

Table 1. Tools for detection of duplicated code.

Tool	PLG	PL	OSF	DE	OTHER	ON	DO	UI	TE	YR
Atomiq	No	N/A	✓	C, C++, C#, Java, others	None	✓	×	✓	Token	2005
CCFinderX	No	C++	✓	C, C++, Java, C#, COBOL, others	None	×	✓	✓	Token	2005
CDSW	N/A	N/A	N/A	N/A	N/A	×	×	N/A	State-ment	2013
CheckStyle	Yes	Java	✓	Java	Large class, long method, long parameter list	✓	✓	✓	N/A	2001
Clone Digger	N/A	Python	✓	Java, Lua, Python	None	✓	✓	✓	Tree	2008
CodePro Analytix	Yes	Java	×	Java	None	✓	✓	✓	N/A	2001
Condenser	No	Jython	✓	Java, Python	None	✓	✓	×	N/A	2002
Covet	N/A	Java	✓	Java	None	×	×	N/A	Metric	1996
CP-Miner	N/A	Java	✓	Java	None	×	×	N/A	Token	2006
DECKARD	No	C	✓	Java	None	✓	✓	×	AST/ Tree	2007
DuDe	No	Java	✓	Java	None	✓	✓	×	N.F.	2010
Dup	N/A	C, Lex	N/A	C	None	×	×	N/A	Token	1995
Duploc	No	Small-talk	N/A	Inde-pendent	None	×	×	✓	Text	1999
Java CloneDR	N/A	N/A	×	C, C++, Java	None	✓	✓	✓	AST/ Tree	1998
Pattern Insight Clone Detection	No	N/A	×	C, C++, Java, others	None	✓	✓	✓	Data Mining	N/A
PMD	Both	Java	✓	C, C++, C#, PHP, Java, others	God class, duplicated code, others	✓	✓	✓	N.F.	2002
Scorpio	N/A	Java	✓	Java	None	×	×	N/A	PDG/ Graph	2013
SDD	Yes	Java	✓	Java	None	✓	✓	✓	N/A	2005
Simian	Both	Java, .NET	×	C, C++, C#, Java, others	None	×	✓	✓	N/A	2003
SonarQube	No	Java	✓	C, C++, C#, Java, PHP, others	Various (manual metrics analysis)	✓	✓	✓	N/A	2008

compatible with Java, 12 are open source or freeware, 15 are able to detect only duplicated code, 11 have graphical user interface and 11 are available online.

RQ2: Are the identified tools able to detect the four types of duplicated code in cross-project context? As mentioned in Section 3.3, among the tools selected though the literature review, only PMD and Atomiq were able to identify at least one of the duplicated code types defined in literature (in case, only Type I).

RQ3: Are the evaluated tools able to detect cross-project duplicated code? It was verified that PMD and Atomiq were able to detect only Type I duplicated code among different systems. In this case, it was detected duplicated code among the systems that shared a single framework JadaSite[7]. Although the tools were able to detect cross-projects duplicated code in the e-commerce systems, it was difficult to identify the source origin of duplicated segments because the tools have no usability support for this kind of detection.

4.2. Guidelines for Development of Duplicate Detection Tools

Though the conducted study with a set of tools for detection of duplicated code, it was conceived five guidelines to support future implementation of tools for this purpose.

G1: To improve the tool usability, in order to minimize the required effort to use a tool and to understand its outputs. The evaluated tools presented some usability issues such as: difficulty to navigate between duplicated code occurrences (in general, results are showed in long lists without result cataloging), difficulty to identify the source file for each duplicated code (they display only directory path and not source project), lack of duplicated code highlighting, lack of click-to-open file for each duplicated code occurrence, and lack of feature for zooming results.

G2: To provide results analysis through statistical methods, graphical visualization or at least numerical indicator such as percentage of lines of duplicated code. It was identified that the evaluated tools do not show statistical numbers related to the duplicated code detection or amount of detected duplicated code for each type. These data could be useful, for instance, in result analysis and comparison of different detection tools.

G3: To combine different techniques (token, tree, etc.) for duplicated code detection, in order to improve the precision of detection results. It could be interesting to combine different techniques that can be more useful to detect an specific type of duplicated code, in order to increase the precision of the tool.

G4: To export result though different file formats, in order to support further analysis and integration with other tools. A feature to result export could be useful to provide an external validation of the result, as well as other processing intended by user.

G5: To provide the selection of projects from different directories to be submitted to duplicated code detection. A practical selection of projects from different data sources. This feature would avoid the cost of creating a single project with all the source code of projects to be analyzed.

4.3. Threats to Validity

The validity of the findings may have been affected by limitations such as: it was conducted an *ad hoc* review though, to minimize the threats, the review was inspired by

[7]https://github.com/IT-University-of-Copenhagen/JadaSite

systematic literature review protocol [Kitchenham et al. 2009]; this study focused only on academic tools, open source or free to use and compatible with Java, although it was collected a large set of tools; the conducted study used projects from e-commerce domain only; and lack of knowledge about the tools may have led to a inappropriate use of tools, though we used default settings.

5. Conclusion and Future Work

Duplicated code is a bad smell that can harm the software development, damaging the software maintenance. In this context, the duplicated code detection can be useful to improve the software quality. However, the identification of cross-projects duplicated code can point to relevant information such as the commonalities among from different projects and can be useful in the feature extraction to compose new software products such as in a SPL [Halmans and Pohl 2003].

Through this study, it was not possible to identify an efficient tool for cross-project duplicated code detection because the tools were able to find only Type I. Aiming to contribute for the community of duplicated code detection, guidelines were proposed to support future implementations of tools. Among the difficulties and challenges faced during the development of this study, we can cite: lack of tool documentation, inconsistency of documentation, online unavailability of tools, and usability issues of tools.

A suggestion for further work is the development a new cross-projects duplicated code detection tool following, as much as possible, the guidelines proposed in this study. These guidelines include statistical analysis for indication of duplicated code types proposed in the literature, and combination of different detection techniques, usability, result export, and features for selection of systems for duplicate code detection.

References

Baker, B. S. (1993). A Program for Identifying Duplicated Code. *Journal of Computing Science and Statistics*, pages 49–49.

Bellon, S., Koschke, R., Antoniol, G., Krinke, J., and Merlo, E. (2007). Comparison and Evaluation of Clone Detection Tools. *IEEE Transactions on Software Engineering (TSE)*, 33(9):577–591.

Bruntink, M., van Deursen, A., van Engelen, R., and Tourwe, T. (2005). On the Use of Clone Detection for Identifying Crosscutting Concern Code. *IEEE Transactions on Software Engineering (TSE)*, 31(10):804–818.

Bulychev, P. and Minea, M. (2008). Duplicate Code Detection Using Anti-unification. In *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE)*, number 2.

Burd, E. and Bailey, J. (2002). Evaluating Clone Detection Tools for Use During Preventative Maintenance. In *Proceedings of the 2nd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)*, pages 36–43.

Campbell, G. A. and Papapetrou, P. P. (2013). *SonarQube in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition.

Ducasse, S., Rieger, M., and Demeyer, S. (1999). A Language Independent Approach for Detecting Duplicated Code. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, pages 109–118.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Object Technology Series. Addison-Wesley.

Halmans, G. and Pohl, K. (2003). Communicating the Variability of a Software-Product Family to Customers. *Journal of Software and Systems Modeling (SoSyM)*, 2(1):15–36.

Higo, Y., Murakami, H., and Kusumoto, S. (2013). Revisiting Capability of PDG-based Clone Detection. Technical Report, Graduate School of Information Science and Technology, Osaka University.

Hotta, K., Sano, Y., Higo, Y., and Kusumoto, S. (2010). Is Duplicate Code More Frequently Modified than Non-duplicate Code in Software Evolution?: An Empirical Study on Open Source Software. In *Proceedings of the joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 73–82.

Jiang, L., Misherghi, G., Su, Z., and Glondu, S. (2007). DECKARD: Scalable and Accurate Tree-based Detection of Code Clones. In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 96–105.

Juergens, E., Deissenboeck, F., and Hummel, B. (2009). CloneDetective – A Workbench for Clone Detection Research. In *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, pages 603–606.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic Literature Reviews in Software Engineering – A Systematic Literature Review. *Journal of Information and Software Technology*, 51(1):7–15.

Li, Z., Lu, S., Myagmar, S., and Zhou, Y. (2004). CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, volume 4, pages 289–302.

Mealy, E., Carrington, D., Strooper, P., and Wyeth, P. (2007). Improving Usability of Software Refactoring Tools. In *Proceedings of the 18th IEEE Australian Software Engineering Conference (ASWEC)*, pages 307–318.

Moha, N., Gueheneuc, Y.-G., Duchien, L., and Le Meur, A.-F. (2010). DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Transactions on Software Engineering (TSE)*, 36(1):20–36.

Murakami, H., Hotta, K., Higo, Y., Igaki, H., and Kusumoto, S. (2013). Gapped Code Clone Detection with Lightweight Source Code Analysis. In *Proceedings of the 21st IEEE International Conference on Program Comprehension (ICPC)*, pages 93–102.

Wettel, R. and Marinescu, R. (2005). Archeology of Code Duplication: Recovering Duplication Chains from Small Duplication Fragments. In *Proceedings of the 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 8–pp.