Identification and Prioritization of Reuse Opportunities with JReuse

Johnatan Oliveira[1], Eduardo Fernandes[1], Gustavo Vale[2], Eduardo Figueiredo[1]

[1]Department of Computer Science, Federal University of Minas Gerais, Brazil
[2]Department of Informatics and Mathematics, University of Passau, Germany

{johnatan.si, eduardofernandes, figueiredo}@dcc.ufmg.br
vale@fim.uni-passau.de

Abstract. Software reuse aims to decrease the development efforts by using existing software components in the development of new systems. Previous work propose tools to support the identification of reuse opportunities. Such tools apply different techniques, such as software design and source code analyses. However, none of them combines lexical analysis with prioritization and identification of reuse opportunities in several systems of a single domain. To fill this gap, this paper proposes JReuse, a tool that computes naming similarity for classes and methods of Java systems. Based on naming similarity, JReuse identifies reuse opportunities and prioritizes them by their frequency among systems. We evaluate JReuse with 35 e-commerce systems collected from GitHub by assessing the agreement among the JReuse recommendations and the opinion of a group of experts. We observe agreements of 89% and 72% for classes and methods, respectively. Therefore, our data suggest that JReuse is able to recommend reusable classes and methods in a given domain.

Keywords: Software Reuse, Reuse Opportunity, Supporting Tool

1 Introduction

Software reuse is a development strategy in which existing software components support the development of new systems [4]. The reuse of existing components potentially increases the software quality and the developers' productivity, since reused components are recurrently evaluated and improved, instead of developed from scratch [6, 7]. Previous works propose tools with different techniques to support the identification of reuse opportunities [5, 9, 14]. For instance, the Guru tool [5] relies on source code analysis to identify reuse opportunities based on the similarity of attributes from different classes. Another tool, called Aesop [9], relies on the analysis of architectural diagrams. Finally, CodeBroker [14] performs run-time identification and prioritization of opportunities based on semantic code analysis and documentation. However, none of them combines lexical analysis with the prioritization of reuse opportunities. In addition, they do not aim to analyze several systems of a single domain.

This paper proposes and evaluates JReuse, a tool for identification and prioritization of reuse opportunities in Java systems. JReuse computes the naming similarity (i.e., lexical analysis) of classes and methods among systems. Therefore, the tool aims to analyze systems from a domain with similar naming conventions. In addition, the tool prioritizes the identified reuse opportunities by their frequency among different systems. We evaluate JReuse with 35 e-commerce systems collected from GitHub[1]. In this evaluation, we compare the reuse opportunities identified by JReuse with the opinion of a group of e-commerce experts. We observe an average agreement of 89% between the tool and experts with respect to classes. Similarly, the average agreement was 72% between JReuse and the experts' opinion for methods.

Our main contribution with this paper is the JReuse tool. The potential users of the tool are software developers and engineers concerned about the identification of reuse opportunities at class and method levels in systems of a specific domain. The recommendations provided by JReuse have different practical applications. We provide some examples as follows. First, methods and partial classes can guide developers in the implementation of a new system. Second, the results provided by JReuse may support recovering a partial design for systems of a domain. Third, the recommendations provided by JReuse can support the evolution of existing systems.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents JReuse. Section 4 provides the tool's evaluation. Section 5 discusses threats to validity. Section 6 concludes the paper and suggests future work.

2 Related Work

Different techniques may support the identification of reuse opportunities. Maarek et al. [5] proposes the Guru tool that relies on the source code analysis. Guru groups code elements that implement similar functionalities based on the extraction of attributes of classes. However, Guru does not provide a general view of the reuse opportunities, via prioritization of opportunities, and requires that developers search for specific functionalities of interest instead. Monroe and Garlan [9] present Aesop that relies on architectural analysis. The authors assume that code analysis does not suffice to identify reuse opportunities, due to the limitation of representing concerns via code. Hence, Aesop analyzes the architectural diagram of a system to identify the main components and its interactions. Unlike Guru and Aesop, JReuse identifies and prioritizes reuse opportunities without requiring the developer to provide a search key.

Ye and Fischer [14] present the closest technique to ours, called CodeBroker, for identification of reuse opportunities at run-time using information retrieval. Their tool relies on semantic analysis and performs both source code and documentation analysis. CodeBroker provides a list of methods as reuse opportunities prioritized by their relevance. In turn, JReuse applies lexical analysis, i.e. a lighter technique for analysis of multiple systems at the same time. JReuse also aims to support reuse by identifying and prioritizing not only methods as reuse opportunities, but also classes.

[1] https://github.com/

3 JReuse

This section presents the JReuse tool. Section 3.1 describes the tool's architecture and the algorithm adopted for similarity computation. Section 3.2 presents the main implementation technologies and the user interface of JReuse. An illustrative video of JReuse, as the source code of the tool, are both available in our research website [11].

3.1 Architecture

JReuse is an Eclipse plug-in that computes similarity among names of classes and methods to identify the most frequent terms in systems of a single domain. Figure 1 presents the architecture of JReuse. Four modules compose the tool, namely *Collector*, *Extractor*, *Similarity Calculator*, and *Ranking Calculator*.

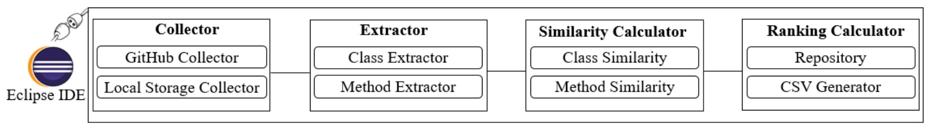


Fig. 1. Architecture of JReuse

Collector collects a set of systems for identification of reuse opportunities. Two sub-modules compose this module: GitHub Collector and Local Storage Collector. GitHub Collector is responsible for collecting systems from GitHub given a search string provided by the user. Local Storage Collector collects systems from a local directory informed by the user. The user may choose between both means to collect systems. *Extractor* extracts the code elements necessary for computation of naming similarity. This module has two sub-modules. Class Extractor is responsible to retrieve both names and types of classes. Method Extractor does the same for methods.

Similarity Calculator calculates naming similarity among classes, through the Class Similarity sub-module, and Method Similarity does the same for methods. Both sub-modules use the Levenshtein's algorithm for naming similarity computation (details in the paragraph Similarity Algorithm). Finally, *Ranking Calculator* generates the sorted list of reuse opportunities. Such list prioritizes the opportunities based on a similarity score, i.e., higher scores come first. CSV Generator and Repository sub-modules export the JReuse results as CSV and persists the results on the repository.

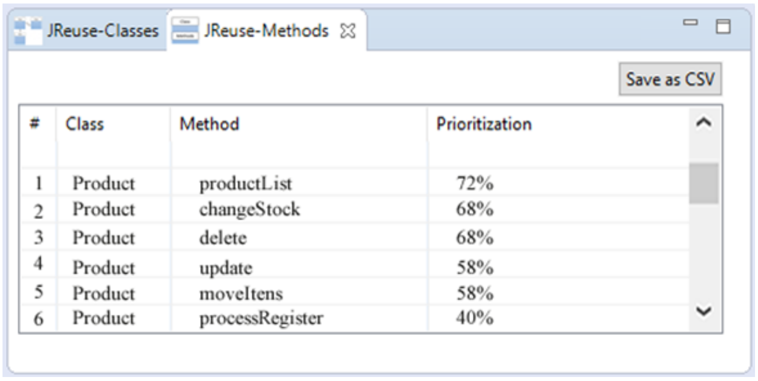
Similarity Algorithm. Previous work [1, 3, 15] propose algorithms for naming similarity computation. We chose the Levenshtein's algorithm [15] because of its simplicity. This algorithm relies on lexical analysis to computes a similarity score between two strings. Such score ranges from 0% to 100%. The higher the score, the more similar the strings are. We consider two names as similar if their score is at least 75%. We derived this value empirically, given naming conventions for classes and methods. Thus, we prevent elements with similar names that address different concerns. For instance, two classes named `Customer` and `CustomerDAO` have score equals 72%. The first one is an entity abstraction and the second one is a data base entity.

Limit for Recommendation. We defined two limits to the amount of classes and methods recommended as reuse opportunities. For classes, the limit is 15% of systems in the analyzed domain. For instance, considering the data set presented in Section 4.1, the limit of 15% means that a class should appear in at least 6 out of 35 systems. In our empirical analysis, several classes appear in a small amount of systems, eventually in just one system of our data set. Therefore, this limit aims to prevent the recommendation of non-frequent classes. For methods, the limit is 3, i.e. a method has to appear in at least 3 classes from different systems to be recommended. The limits of classes and methods can be adjusted depending on characteristics of the data set.

3.2 Implementation Technologies and User Interface

We implemented JReuse using the Java programming language with support of the Java Development Tools (JDT). JDT provides libraries for accessing and manipulating the Java source code and supports the source code parsing, as the retrieval of names of classes and methods of Java systems. We analyze the source code structure of target systems via Abstract Syntax Tree (AST). JReuse works only for Java systems, although the technique behind the tool applies to other object-oriented programming languages. The main reasons for selecting Java are: (i) it has a large support for code analysis, including libraries for AST generation, (ii) it is one of the most popular languages [2], and (iii) software reuse have been investigating in the context of Java software systems [8, 12].

Figure 2 illustrates the data grid view of JReuse with results for methods recommended as reuse opportunities. The grid for classes is similar. There are four columns in the view discusses as follows. "#" presents is an identifier per recommendation sorted by the prioritization score. *Class* provides the name of the source class for each method. *Method* shows the method name. *Prioritization* presents the frequency of each method among the systems of the analyzed data set (more details in Section 4). In addition, the button *Save as CSV* exports the results as a CSV file.



#	Class	Method	Prioritization
1	Product	productList	72%
2	Product	changeStock	68%
3	Product	delete	68%
4	Product	update	58%
5	Product	moveItens	58%
6	Product	processRegister	40%

Fig. 2. Data Grid View with a List of Methods Identified by JReuse

4 Evaluation

This section evaluates JReuse. Section 4.1 presents the study settings. Sections 4.2 and 4.3 discuss the evaluation results at class and method levels, respectively.

4.1 Evaluation Settings

This evaluation aims to assess both the JReuse correctness and whether it is able to identify reuse opportunities in Java e-commerce systems. Regarding the tool correctness, we created many test cases and we manually searched for reusable classes and methods in the e-commerce systems. Then, we compared them with the results of JReuse. The results were equals. Thereby, we conclude that JReuse works correctly. In turn, regarding the ability of JReuse to identify reuse opportunities, we conducted a two-level evaluation. First, we investigate if the highly prioritized classes provided by JReuse are specific of a domain. Second, we investigate if the highly prioritized methods are meaningful to the class they were implemented. To support both evaluation steps, we rely on a group of experts composed of four software engineers with experience in software development and reuse, including e-commerce systems. We considered classes and methods as reuse opportunities just if all members of the group of experts agreed with the tool.

We designed three steps to the evaluation of JReuse. First, we run the tool with a data set composed by 35 e-commerce systems collected from GitHub. Second, we analyzed all classes recommended by JReuse. Third, from the top-ten highly prioritized classes, we verified the recommended methods. We chose the e-commerce domain for two main reasons. First, there is a large amount of systems of this domain available for download in GitHub. Second, since e-commerce is a well-defined domain in terms of requirements, e-commerce systems potentially have several reuse opportunities for identification. We collected the systems in November 2016. They have in total around 500 KLOC (mean of 46 KLOC per system), 3 K classes (mean of 368 classes), and 18 K methods (mean of 1 K methods).

4.2 Results at Class Level

Table 1 presents the reuse opportunities identified by JReuse at class level in our data set. The columns represent the class name, the class frequency, and if the classes are specific of the e-commerce domain according to the group of experts. We double the three columns due to space constraints, and the most frequent classes come first at the left side. As stated in Section 3.1, JReuse does not recommend classes that appear in less than 15% of the systems (limit). Thus, any class under this condition does not appear in Table 1. The column "Domain Specific" represents the opinion of the group of experts, in which "Y" means that all experts agree that the class is specific of the e-commerce domain, and "N" means that at least one expert does not agree.

Table 1. Results at Class Level

Class Name	Class Frequency	Domain Specific	Class Name	Class Frequency	Domain Specific
Product	80%	Y	Category	35%	Y
PaymentType	69%	Y	ProductService	29%	Y
Client	58%	Y	Order	26%	Y
ProductDao	52%	Y	LoginController	20%	N
ClientDao	52%	Y	UserDao	18%	Y
Item	49%	Y	ProductServiceImpl	18%	Y
ShoppingCart	49%	Y	ShoppingCartController	18%	Y
User	49%	N	OrderedProduct	15%	Y
Customer	40%	Y	ShoppingCartService	15%	Y

The classes presented in Table 1 appear in several systems. For instance, the class *Product* appears in 80% (28 out of 35) of the systems under analysis. Hence, this class probably will be necessary in the development of a new e-commerce system. Regarding the data reported at domain specific column, we observe that the group of experts agrees that most classes are specific of the e-commerce domain. The group of experts disagrees just in the case of *User* and *LoginController*. In summary, the agreement between the classes JReuse recommended and the group of experts is around 89%, a high value for agreement. We conclude that, in general, the reuse opportunities identified by JReuse at class level are actually reusable.

4.3 Results at Method Level

Table 2 summarizes the results for methods identified by JReuse as reuse opportunities. This analysis relies on the top-ten highly prioritized classes from Table 1. The rows list mean, maximum, and standard deviation for the last three columns. The second column presents of methods identified as reuse opportunities for the top-ten classes. The third column presents the percentage of methods in at least a half of similar classes in different systems. With this column, we aim to highlight the methods that occur more frequently in the analyzed classes. The fourth column presents the percentage of agreement from the group of experts with respect to the highly prioritized methods.

For the top-ten classes, we observe an average of 11 methods per class identified as reuse opportunities (maximum of 15). That is, each highly prioritized class has a high amount of similar methods that support the implementation of new e-commerce systems. In addition, we observe a mean of 60% (maximum of 100%) of methods identified as reuse opportunities and that occur in at least 50% of similar classes in the analyzed systems. Finally, regarding the viewpoint of the domain experts, we observe an average agreement of 72% of methods pointed as specific of their respective classes. The agreement reached up to 100% for methods of the class *User*. In summary, as stated in Section 4.2, we conclude that methods identified by JReuse are meaningful and, consequently, reusable in association with the identified classes.

Table 2. Results for Methods from the Top-Ten Classes

Measure	# Identified Methods	% Methods with ≥ 50% Frequency	Domain-Specific Methods
Mean	11	60%	72%
Max.	15	100%	100%
Std. Dev.	3	20%	20%

5 Threats to Validity

We discuss the main threats to the study validity, as the respective treatments, as follows. Our discussion relies on the guidelines proposed by Wohlin et al. [13].

Construct and Internal Validity. We conducted a careful filtering of systems collected from GitHub to evaluate JReuse. However, some threats may affect the correct filtering, as human factors that wrongly lead to the discard of a valid system for evaluation. We minimize this threat by adopting exclusion criteria described in our website [11]. Moreover, some threats may have affected our lexical classification of classes and methods. These threats include error in the nomenclature of classes and methods. To minimize this problem, and to identify bugs in the JReuse implementation, we randomly selected a sample of 10 e-commerce systems from our data set. We then identified the names of classes and methods manually from source code, in order to find synonyms. We compared our manually obtained results with the ones provided by JReuse and observed a low loss of 10% in synonym terms identified by the tool.

Conclusion and External Validity. We analyzed the data collected from JReuse manually, and two authors contributed to double-check such analysis. In addition, we derived empirically the limits for recommendation of classes and methods (see Section 3.1). Although both the analysis and the limit derivation may have been affected by human factors, we carefully conducted each activity to prevent biases and missing data, for instance. Finally, we evaluated JReuse with a set of 35 e-commerce systems collected from GitHub. Since our evaluation relies on a specific set of systems from a single domain, we may not generalize our study findings to any Java software systems. However, the systems of our data set are the most popular Java e-commerce systems on GitHub given by their number of stars. In addition to e-commerce, we evaluated JReuse in other three domains [10]: hospital, restaurant, and accounting.

6 Conclusion and Future Work

This paper presents JReuse, a tool for identification and prioritization of reuse opportunities in Java systems. Our tool relies on lexical source code analysis. JReuse computes naming similarly, using the Levenshtein's algorithm, for classes and methods extracted from a data set of systems from a specific domain. We evaluate the recommendations provided by JReuse at class and method levels. A group of experts assessed such recommendations with respect to their reusability in the context of e-commerce systems. Agreements of 89% and 72% regarding classes and methods, respectively, suggest that JReuse is able to identify relevant reuse opportunities in both class and method levels for the e-commerce domain. These reuse opportunities can support, for instance, the reuse of parts of code, the recovery of a partial design for a system, or the evolution of existing systems.

As future work, we intend to implement the combination of similarity computation techniques, such as semantic analysis, to extend JReuse. We also aim to identify and prioritize reuse opportunities in systems implemented in languages other than Java.

References

1. Bilenko, M., Mooney, R.: Adaptive Duplicate Detection Using Learnable String Similarity Measures. In: 9th Conference on Knowledge Discovery and Data Mining (KDD), pp. 39–48 (2003)
2. Diakopoulos, N., Cass, S.: The Top Programming Languages 2015. http://spectrum.ieee.org/static/interactive-thetop-programming-languages-2015
3. Gitchell, D., Tran, N.: Sim: A Utility for Detecting Similarity in Computer Programs. In: ACM SIGCSE Bulletin 31, 1, 266–270 (1999)
4. Krueger, C.: Software Reuse. ACM Computing Surveys (CSUR) 24, 2, 131–183 (1992)
5. Maarek, Y., Berry, D., Kaiser, G.: An Information Retrieval Approach for Automatically Constructing Software Libraries. IEEE Transactions on Software Engineering (TSE) 17, 8, 800–813 (1991)
6. Mohagheghi, P., Conradi, R.: Quality, Productivity and Economic Benefits of Software Reuse: A Review of Industrial Studies. Empirical Software Engineering (ESE) 12, 5, 471–516 (2007)
7. Mohagheghi, P., Conradi, R., Killi, O., Schwarz, H.: An Empirical Study of Software Reuse vs. Defect-Density and Stability. In: 26th ICSE, pp. 282–291 (2004)
8. Mojica, I., Adams, B., Nagappan, M., Dienst, S., Berger, T., Hassan, A.: A Large-Scale Empirical Study on Software Reuse in Mobile Apps. IEEE Software 31, 2, 78–86 (2014)
9. Monroe, R., Garlan, D.: Style-Based Reuse for Software Architectures. In: 4th ICSR, pp. 84–93 (1996)
10. Oliveira, J.: A Method Based On Naming Similarity to Identify Reuse Opportunities. MSc dissertation, Federal University of Minas Gerais (UFMG), Belo Horizonte, Brazil (2016)
11. Oliveira, J., Fernandes, E., Vale, G., Figueiredo, E.: JReuse – Data of the Study. http://homepages.dcc.ufmg.br/~johnatan.si/jreuse
12. Roopa, M., Mani, V., Stefan, H.: An Approach for Enabling Effective and Systematic Software Reuse. In: 11th ICGSE, pp. 134–138 (2016)
13. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media (2012)
14. Ye, Y., Fischer, G.: Reuse-Conducive Development Environments. Automated Software Engineering (ASE) 12, 2, 199–235 (2005)
15. Yujian, L., Bo, L.: A Normalized Levenshtein Distance Metric. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 29, 6, 1091–1095 (2007)