A Method Based on Naming Similarity to Identify Reuse Opportunities

Johnatan Oliveira, Eduardo Fernandes, Maurício Souza, Eduardo Figueiredo
Software Engineering Laboratory (LabSoft), Department of Computer Science
Federal University of Minas Gerais (UFMG)
Belo Horizonte, Brazil
{johnatan-si, eduardofernandes, mrasouza, figueiredo}@dcc.ufmg.br

ABSTRACT

Software reuse is a development strategy in which existing software components, called reusable assets, are used in the development of new software systems. There are many advantages of reuse in software development, such as minimization of development efforts and improvement of software quality. New methods for reusable asset extraction are essential to achieve these advantages. Extraction methods may be used in different contexts including software product lines derivation. However, few methods have been proposed in literature for reusable asset extraction and recommendation of these reuse opportunities. In this paper, we propose a method for extraction of reuse opportunities based on naming similarity of two types of object-oriented entities: classes and methods. Our method, called JReuse, computes a similarity function to identify similarly named classes and methods from a set of software systems from a domain. These classes and methods compose a repository with reuse opportunities. We also present a prototype tool to support the extraction by applying our method. We evaluate the method with 38 e-commerce information systems mined from GitHub. As a result, we observe that our method is able to identify classes and methods that are relevant in the e-commerce domain.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*Domain engineering, reusable libraries*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*

General Terms

Design, documentation, management.

Keywords

Software reuse, reusable assets, naming similarity, tool.

1. INTRODUCTION

Software reuse is a development strategy in which existing software components, called reusable assets, are used in the development of new software systems [6]. It has been studied and pointed as an alternative to traditional development aiming to increase software quality and decrease development efforts by using previously developed, and sometimes already tested, software components [11, 12, 14, 17].

The extraction of reusable assets is essential to support software reuse activity by building repositories of reuse opportunities [4]. These methods may be used in different contexts related to software reuse, including the support of feature extraction for a software product line (SPL) [8], for instance. Many methods have been proposed in the literature to support the extraction of reuse opportunities from software systems [1, 5, 7, 10, 22].

There are different approaches used by proposed methods to identify reuse opportunities, such as natural-language processing [10], formal specifications [1], machine learning [5], and other Information Retrieval (IR) approaches [7, 22]. However, to the best of our knowledge, we did not find a method for extraction of reuse opportunities and reuse recommendation considering most frequent elements such as classes and methods from systems of a single domain.

In this paper, we propose a method for extraction of reuse opportunities called JReuse. Considering a set of software systems, JReuse aims to identify methods named with the same token in similarly named classes from different systems. Then, we are able to identify methods, and classes eventually, that may be recommended as reuse opportunities. We also present a prototype tool that applies the proposed method.

We conduct an evaluation of our method through an experiment with 38 Java information systems in the e-commerce domain mined from GitHub. As a result, we observe that our method is able to identify reuse opportunities using naming similarity analysis. We also observe that JReuse provides meaningful classes and methods in the analyzed domain they may be indicated as reuse opportunities to developers of new e-commerce systems.

The remainder of this paper is organized as follows. Section 2 presents a background to support the comprehension of our work. Section 3 proposes a method for reuse opportunities extraction and the prototype tool to support this method. Section 4 presents an evaluation of our method. Section 5 describes the results obtained through the preliminary evaluation and discusses some lessons learned. Section 6 presents threats to the validity of our study. Section 7 discusses related work. Finally, Section 8 concludes this paper with a discussion and presents of future work.

2. BACKGROUND

In software reuse, previously implemented software components are used to support the development of new software systems [6]. The main goal of reuse is the improvement of software quality aspects followed by an increasing development efficiency [17]. There are many approaches to support reuse in software development. Krueger [6] presents an extensive study regarding definitions, approaches and application of software reuse.

There are two types of software reuse: *ad hoc* and systematic reuse [11]. In an *ad hoc* approach, software reuse is applied in an opportunistic way, without planning, taking as an example the reuse of random software code snippets from Web [18]. In turn, systematic software reuse follows specific protocols and processes to provide the use of existing software components when developing new systems [11].

Wang et al. [20] conduct an study regarding the identification of business domain components to support software reuse. According to their work, there are two types of component identification: forward identification, in which software reuse is planned before the development of software systems; and reverse identification, in which reuse opportunities are identified from a set of existing software systems.

Some studies investigated advantages and drawbacks of systematic software reuse [11, 12]. Mohagheghi et al. [12] studied the impacts of reuse on software quality through an empirical study on large-scale system components. They concluded that reuse provides software components with lower defect-density and higher stability when compared with non-reused components. Mohagheghi and Conradi [11] conducted a literature review to investigate the impact of software reuse in industrial development context. They identified flaw decreasing, reduction of development efforts, and increasing of productivity as the main advantages of reuse.

Systematic software reuse is supported by reusable assets, also known as reuse libraries, that are collections of components to be used in the development of new software systems [4]. Components of a reuse repository are called reusable assets, and their extraction from existing software systems may be supported by automated tools [3].

Many strategies are proposed in literature, based on techniques such as: natural-language processing, in which lexical inspection of source code elements is conducted to identify reuse opportunities [10]; formal specifications, in which reusable components are extracted with support of software models and metrics analysis, for instance [1]; architectural style [13], where software reuse is supported by the analysis of interacting components in a high-level abstraction such as software design and modeling; and machine learning, in which different analyzes are conducted to extract reuse opportunities, such as automated semantic categorization of software components [5].

3. THE PROPOSED APPROACH

In this section, we present our proposed approach for extraction of reuse opportunities, composed of an extraction method and a prototype tool. Section 3.1 describes the adopted strategy to compute naming similarity. Section 3.2 describes each step of the proposed method for reuse opportunities extraction. Section 3.3 presents its supporting tool and how it applies our method to provide reuse opportunities.

3.1 Identifying Similarity

Some studies in the literature investigate the textual similarity identification [19, 24]. There are may application for similarity analysis, such as comparison of dialects, spell check, and plagiarism detection [9]. JReuse relies on static code analysis techniques to extract reuse opportunities. An adaptation of the normalized Levenshtein's algorithm by Yujian et al. [23] is used to compute the lexical similarity of classes and methods by name. In short terms, given two strings A and B, this similarity function computes the number of changes required to match A and B.

To extract similarly named classes, we adopted a threshold of 80%. This threshold was derived empirically by the authors. We considered some well-known naming conventions for classes to define this threshold. For instance, the similarity computed for `Costumer` and `CostumerDAO`, that are common classes in e-commerce systems, is 72%. However, intuitively both classes implements different functions (e.g., DAO classes implement data base persistence).

In turn, to extract similarly named methods, we adopted a threshold of 100%. This threshold was chosen because we observed that some well-known naming conventions for some methods may lead to similarly named entities that clearly represents opposite functions. For instance, in general `getters` and `setters` (e.g., `getProduct` and `setProduct`) are named with similarity between 80% and 100% according to our similarity function. However, we know these methods implement different functions. Therefore, in this case, we should not extract these methods as the same reuse opportunity.

An example of the similarity analysis is shown in Table 1. In this example we present five matches between two software systems, i.e., five classes names from System A that have at leas t 80% similarity rate with class names from System B. The similarity rate alone is not enough for electing a class as a possible reuse opportunities, we also consider which classes are more frequent among the systems (e.g., a name of class with matches in 10 different systems is more frequent than a name of class that matches in only 2 systems).

Table 1: Similarity evaluation

System A	System B	Similarity Rate
ProductController	Product**s**Controller	94%
OrderProduct**Id**	OrderProduct	87,5%
Review**s**	Review	85%
ShoppingCart**dto**	ShoppingCart	80%

3.2 The Extraction Method

A software domain is a set of systems that share a set of common functionalities, requirements or terminology [15]. Thus, we can expect that software systems from the same domain present lexical similarity in names of classes, methods and attributes, as they are chosen in order to be expressive about their goals and the specific features of a business domain [2].

Considering this scenario, our study proposes a method called JReuse for extraction of reuse opportunities from software systems based on naming lexical similarity of two object-oriented code elements: classes and methods. Considering software systems from a specific domain, this method compares names of classes and methods in order to identify common elements among different systems.

Given a data set of software systems from the same domain, JReuse performs the following two steps:

Step 1: Comparison between Classes – In the first step, JReuse compares names of each pair-wise classes from the data set of systems. For this purpose, a similarity function is computed to identify classes named similarly. Step 1 aims to retrieve classes that are possible relevant entities in the context of the analyzed business domain. Therefore, each identified class may be considered a reuse opportunity in this domain.

Step 2: Comparison between Methods – Considering only classes pointed as reuse opportunities in the previous step, JReuse compares pairs of methods by name to identify recurrent methods in the analyzed domain. Step 2 applies the same similarity function used in Step 1, to every pair of methods from two different classes.

We believe that recurring names of classes may point relevant entities in a given domain. Furthermore, frequent names of methods in these classes may indicate common behaviors and requirements in such entities, as stated by Cybulski and Reed [2].

3.3 Tool Support

To allow a preliminary evaluation of our method, we developed a prototype tool that implements the proposed method for Java software systems. We selected Java because this is one of the most popular programming languages[1]. First, our tool receives a set of software systems from the same domain. Then, the tool processes classes and methods from the input set to identify similarly named entities according to the proposed method. Finally, the tool provides a list of classes and methods identified as reuse opportunities.

Four steps are performed by our tool to extract reuse opportunities as follows:

1. The tool retrieves the name of all classes and methods from system data set. To support the identification of similarly named entities (classes and methods), we used the Eclipse Java Development Tools (JDT) parser. This tool is used to retrieve the entities of source code for analysis.

2. The tool compares names of classes in pairs to identify class names with at least 80% of similarity. Classes with similar name (matches) are gathered and each class name receives a score that is the number of systems in which the class occurs. The higher a score, the higher a class seems to be relevant for the analyzed domain.

3. For each group of classes identified in the previous step, our tool compares method names with 100% similarity rate.

4. Finally, the tool persists in data base the identified entities (classes and methods) extracted as reuse opportunities.

[1]http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015

The architectural design of our tool is presented in Figure 1. In Step 1, the tool receives Java software systems to analysis. Then, in a processing step, we have two modules: Step 2 for analysis of similarly named classes, and Step 3 to identify similarly named methods. Finally, the Step 4 is composed by a database were the results of analysis are stored.

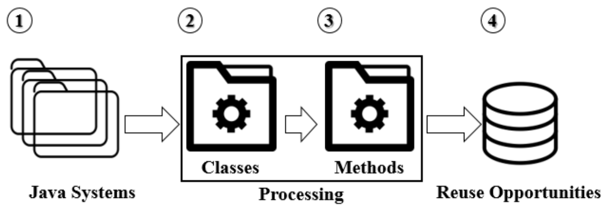

Figure 1: Architecture of the Tool Support

4. EVALUATION SETUP

This section describes our evaluation of the method for extraction of reuse opportunities through an experiment. Section 4.1 presents the goals of our study and the research questions we designed to guide our experiment. Section 4.2 describes the data set used to run our prototype tool. Section 4.3 presents the experiment steps.

4.1 Goals and Research Questions

The main goal of this study is to evaluate our method developed to extract reusable assets from systems of a single domain. We are interested in whether JReuse is able to identify recurrent classes and methods in an specific software domain. We are also interested in assessing the relevance of the results provided by our method. For this purpose, we chose the e-commerce domain for evaluation. We also conceived the following research questions (RQs) to guide our study.

RQ1 *What are the most frequent classes in e-commerce information systems? And how are they distributed through systems?*

RQ2 *What are the most frequent methods considering the similarly named classes identified by the method? And how are they distributed through these classes?*

Through RQ1 we are interest on investigating: the efficiency of the prototype tool to identify classes that are similarly name in different systems; and verify if the most frequent identified classes are relevant for information systems from the e-commerce domain. In turn, with RQ2 we aim to assess the same aspects of our tool in case of similarly named methods from the classes pointed as similar. We expect that JReuse is able to provide a list of classes and methods whose recommendations for reuse makes sense considering the evaluated domain.

4.2 Data Set

To evaluate our method, we chose only systems from the e-commerce domain for several reasons: **First**, these systems, in general, are intuitive information systems and easy to evaluate; **Second**, there is a large amount of e-commerce systems available for download in GitHub[2]. **Third**, since e-commerce is a well-defined domain, the authors of this paper believed that it would be easy to find reuse opportunities among systems from this domain.

The systems that compose our data set were retrieved from GitHub repositories in January 2015. We searched for information systems sorted by stars. In GitHub, stars are a meaningful measure for repository popularity among the platform users, and may be used to support the selection of well-evaluated by developers. To retrieve information systems, we used the following keywords related to the e-commerce domain: *e-commerce*, *ecommerce*, *electronic commerce*, *e-business*, *business*, and *electronic commerce*.

The strategy for defining our data set is illustrated in Figure 2. First, we collected 100 Java information systems from GitHub. Then, we used the following exclusion criteria to filter these systems: non-Java software systems, once GitHub does not verify the main programmings language of a systems automatically, and systems with less than 500 lines of code (LOC) (56 were discarded, remaining 44); systems written in other languages than English were removed, since our method is based on lexical similarity, and language may impact our analysis (6 systems were discarded). Finally, we removed all files with extensions others than `.java` from the 38 resulting e-commerce systems (for evaluation convenience).

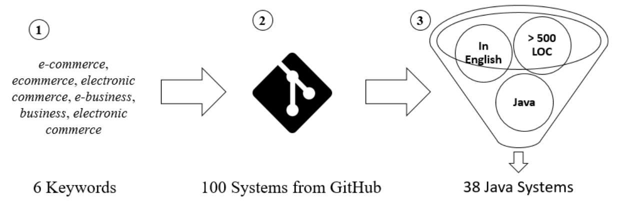
1
e-commerce, ecommerce, electronic commerce, e-business, business, electronic commerce
6 Keywords
2
100 Systems from GitHub
3
In English
> 500 LOC
Java
38 Java Systems

Figure 2: Steps for Collecting Systems from GitHub

For each e-commerce system collected, we considered only the last release of the system. This process was necessary to discard different versions of the same system, that probably contain lots of similarly named classes and methods.

4.3 Experiment Execution

To evaluate the JReuse prototype tool for extraction of reusable assets through the research questions presented in Section 4.1, we designed an experiment composed by the three steps presented in Figure 3. Each step is discussed as follows.

Step 1: Data Set Extraction from GitHub – We cloned information systems from various domains from GitHub, in order to identify a domain that fits to our experiment, i.e., that potentially provides a considerable number of similarly named classes and methods. This *ad hoc* investigation lead us to adopt e-commerce in this exploratory study. Then, we searched for e-commerce information systems in GitHub and

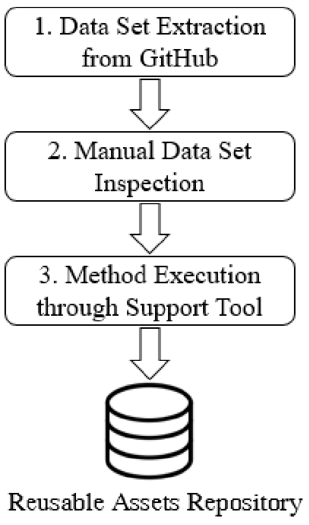
1. Data Set Extraction from GitHub
2. Manual Data Set Inspection
3. Method Execution through Support Tool
Reusable Assets Repository

Figure 3: Experiment steps

extracted these systems as discussed in Section 4.2. Finally, we obtained 38 different systems.

Step 2: Manual Data Set Inspection – We performed a preliminary evaluation of the data set in order to manually identify similarity among the systems. We conducted a manual inspection of similar classes and methods in a sample from the 38 e-commerce systems collected in Step 1. We selected 10 of the 38 systems with lower number of Source Lines of Code (SLOC) to ease the manual process. After this inspection, we were able to find similarly named classes and methods in at least 80% of the sample, thus JReuse should find relevant results.

Step 3: Method Execution through Support Tool – We ran JReuse prototype tool using the 38 e-commerce systems to extract methods and classes similarly name among systems. After the automated analysis, the tool provided a list of methods and classes to compose our reuse repository.

5. RESULTS AND DISCUSSION

In this section, we present the results of the experiment with the JReuse extraction tool and lessons learned through our study. Section 5.1 discusses each proposed research questions. Section 5.2 provides an overview and a some discussion regarding the main lessons learned.

5.1 Research Questions and Answers

RQ1 *What are the most frequent classes in e-commerce information systems? And how are they distributed through systems?*

We analyzed the frequency of similarly named classes in the 38 e-commerce software from our data set. In Table 2, we present the most frequent classes identified by JReuse supporting tool. The classes are sorted by decreasing number of occurrences in systems per class (Count).

Initially, JReuse identified 38 classes as reuse opportunities. To summarize data in Table 2, we adopted the following exclusion criteria for classes to be listed: the class should

[2]https://github.com/

occur in at least 3 different systems, because our method compares classes in pairs and, then, two occurrences may not be significant to a reuse recommendation. We discarded 22 classes based on this minimum threshold, the 16 remaining are presented in Table 2. Consequently, the number of systems to compose the table was reduced to 23.

Table 2: Frequency of classes per projects

Systems	Classes															
	User	Product	Order	Category	Customer	Address	Item	OrderItem	Payment	Article	Cart	CategoryServiceImpl	CustomerServiceImpl	ProductServiceImpl	Role	ShoppingCartItem
S1		●		●	●						●				●	
S2	●		●	●			●		●							
S3		●	●		●			●	●						●	
S4			●		●	●		●								
S5	●	●		●	●	●										
S6							●									
S7			●			●										
S8	●		●				●									●
S9	●						●									
S10	●	●	●	●	●							●	●	●		
S11		●														
S12	●	●		●												
S13	●														●	
S14												●	●	●		
S15		●			●			●			●					
S16	●		●													
S17		●		●	●											●
S18	●	●		●												
S19	●		●	●			●		●	●	●					
S20	●	●	●		●	●		●		●						
S21	●	●	●	●	●				●			●		●		●
S22						●							●			
S23										●						
COUNT	12	11	10	9	9	5	5	4	4	3	3	3	3	3	3	3
%	31,6	28,9	26,3	23,7	23,7	13,2	13,2	10,5	10,5	7,9	7,9	7,9	7,9	7,9	7,9	7,9

Analyzing Table 2, we can identify the most frequent classes for the purpose of reuse opportunities. In response to RQ1, we conclude that the top-five most frequent classes for e-commerce domain, with more than 5 occurrences, are: `User`, `Product`, `Order`, `Category`, and `Customer`. In fact, according to the viewpoint of authors – Four Software Engineering specialists – these classes are meaningful and relevant to the e-commerce domain.

For instance, classes `Product`, `Order`, `Category`, and `Customer` are elementary entities to be expected in an e-commerce systems. In turn, `User` is the most frequent class identified by the tool and, although it is not specific of e-commerce domain, it is expected in information systems in general.

Therefore, we conclude that our method is able to identify relevant classes for software reuse in 60% of the evaluated e-commerce systems (23 of the 38 systems collected from GitHub). The other less frequent classes (e.g. `Item`, `Payment`, and `Cart`) are also interesting in the e-commerce domain.

Figure 4 shows the number of systems per minimum number of frequent classes identified using our method, considering the 38 e-commerce systems that compose our data set. We observe that: our method extracted at least 1 class as reuse opportunity for 23 of the 38 systems (around 60%); the method also extracted at least 4 classes for 12 of the 38 systems (approximately 31%, or one third of the data set); finally, our method provided at least 5 classes as reuse opportunity for 8 of the 38 (21%, or one fifth of the systems). Considering that classes extracted by our method are, in general, representative and meaningful in the e-commerce domain, we may conclude that the method may support reuse by providing a significant amount of reuse opportunities.

RQ2 *What are the most frequent methods considering the similarly named classes identified by the method? And how are they distributed through these classes?*

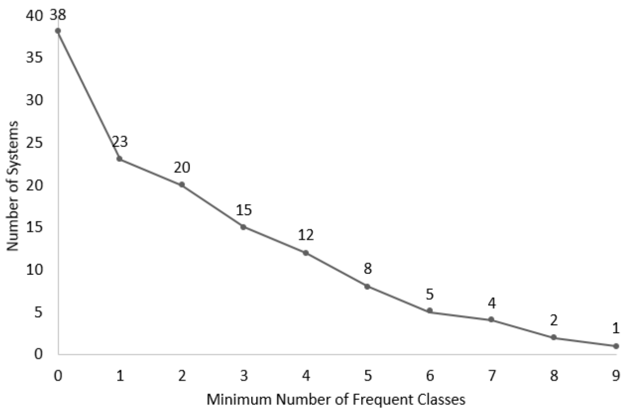

Figure 4: Distribution of frequent classes through systems

Considering the results regarding RQ1, we are able to answer RQ2 with respect to the extraction of similarly named methods from classes identified as similar by JReuse. Table 3 presents the most frequent methods identified only in the classes listed in Table 2. To summarize data, we filtered methods by the following inclusion criterium: it should appear in at least 2 classes from different systems. From 56 similarly named methods extracted, we discarded 34 according to this criteria, remaining 22 methods.

Given a Class name C from the results of RQ1, and M a method name from C, in Table 3 we present the pairs `(C,M)` that were found in at least two systems in our data set. The number in each cell from Table 3 is the number of occurrencies of a given pair `(C,M)` among the analyzed systems.

Table 3: Frequency of methods per class

	Classes									
Methods	Product	Customer	User	Category	Address	Item	Order	Article	ShoppingCartItem	**MAX**
getId	8	5	7	6	2		3	2		8
setId	8	5	6	6	1		3	2		8
getName	5	3		5		3				5
setName	5	3		5		2				5
getPassword		5	8							8
getPrice	9					2				9
setPassword		5	7							7
setPrice	9									9
getEmail		5	4							5
setEmail		5	3							5
getDescription	7									7
setDescription	7									7
toString	4			2						4
getQuantity							3		3	3
getCity		2			4					4
setCity		2			4					4
equals	3		2							3
hashCode	3									3
getCategory	3									3
setCategory	3									3
getPhone		2								2
setPhone		2								2
COUNT	13	12	7	5	4	3	3	2	1	
%	59	55	32	23	18	14	14	9,1	4,5	

For instance, the pairs (Product, getPrice) and (Product, setPrice) are the most frequent among the systems of our data set. Considering the domain under analysis , these results are meaningful, since the product abstraction requires a monetary value for the product. Moreover, the method getPassword occurs in 8 times in User and 5 times in Costumer classes. This scenario is comprehensible considering that these classes intuitively represent entities that have access to the e-commerce system for purchase and payment, for instance. Therefore, in the viewpoint of the authors, the identified methods are relevant and consistent for the domain under analysis.

An interesting finding is that 4 of the 5 most frequent classes of our data set (Product, Customer, User, and Category) also have the highest number of methods with similar names. These classes presented 59%, 55%, 32%, and 23% of the similar methods with respect to the 22 recurring methods considered. Thus, these classes and respective methods may be relevant reuse opportunities for the domain under analysis.

Another interesting observation is that the frequent method names identified by our method are also recurrent among different classes. Figure 5 represents the top-four most frequent classes identified as similarly named through the 38 e-commerce systems, and the frequent methods identified in these classes except toString, equals, and hashCode. These three methods were discarded for summarize the Figure 5 because they are excessively common in Java implementations. In the left partition we have classes in bold, and in the right partition we have methods. Note that getters and setters were gathered for summarizing the figure because they implement related functionalities. We can observe that the getters and setters for Id, Name, Password and Email are present in different classes. Thus even in the context of a single system, it may be possible to observe reuse opportunities.

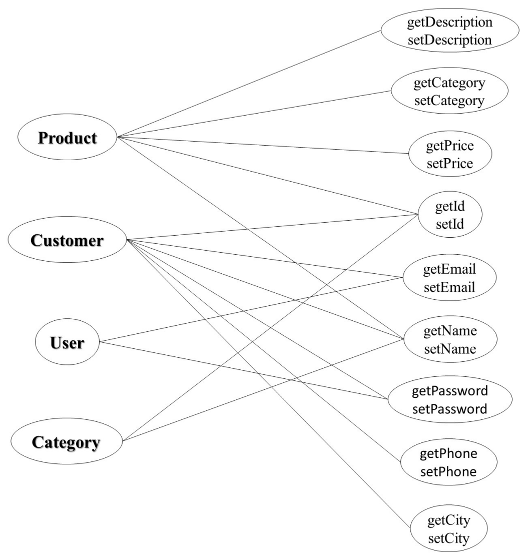

Figure 5: Most frequent methods of the top-four identified classes

5.2 Lessons Learned

In this study, we learned a lot regarding interesting research topics such as software reuse, reusable assets extraction, and recommendation systems. We discuss some of the main lessons learned with support of the following questions.

How much a lexical analysis may support the extraction of reusable assets? As discussed in Section 1, there are many approaches to support software reuse in literature. Lexical analysis is a simple one. However, as pointed by the results of Section 5.1, it may be effective to identify reusable assets in systems from a single domain. Moreover, we initially conceived our method to gather elements with names that are semantically similar. However, through our study we identified some occurrences of similar entities in an intuitive fashion that do not represent the same real-world concept. For instance, in Step 2 of our experiment (see Section 4.3) we found that frequent classes such as Client and Costumer have distinct behaviors although intuitively they represent the same real-world abstraction. Some classes named as Client implement a simplistic system clients which register data basically. In turn, Costumer classes generally implement system clients with more robust features such as data management. Therefore, we conclude that lexical analysis performs satisfactorily to extract reusable assets at least in this domain.

Names of classes and methods are suitable to the entities they represent in a business domain? We discuss in Section 3 that names of classes and methods may be useful for reusable asset extraction. In fact, we observed that naming similarity identification may support reuse opportunities extraction. However, to retrieve similarly named classes and methods may be uninteresting if they are not representative in an specific domain. Section 5.1 highlights extracted classes and methods that fit to e-commerce domain. These entities are the most frequent that our tool detected. Therefore, we believe that names of entities are, in general, sufficiently representative. Moreover, we observed through this study that our method is able to extract relevant reuse opportunities in randomly mined systems from GitHub, provided by different development teams. Therefore, we expect to obtain even more relevant results in the context of an specific organization.

How to apply our reusable assets extraction tool in a reuse recommendation system? Methods and classes are elementary entities of object-oriented software systems, and also attributes. Knowing these entities, we are able to describe the architecture of a system. Therefore, with results provided by our tool, we see an opportunity for reuse recommendation through software modeling using class diagrams, for instance. This conclusion lead us to a future work consisting of the improvement of our tool to also identify similarly named attributes and, then, to integrate our JReuse tool into a recommendation system based on class diagram modeling.

To the best of our knowledge, we have not found many recent studies with respect to reusable assets extraction, support tools for this activity, and methods to support the building of reuse repositories with similar approach. Therefore, as an interesting research topic, we lack more quantitative data to measure and compare different techniques that support software reuse.

6. THREATS TO VALIDITY

We based our study on related work to support the method definition, the reusable assets extraction tool development, and the proposal of a recommendation system. Regarding the evaluation of our extraction tool, we conducted a careful empirical study to assess efficiency of the tool with respect to reusable assets extraction that are representative in the e-commerce context. However, some threats to validity may affect our research findings. The main threats and respective treatments are discussed below based on the proposed categories of Wohlin et al. [21].

Construct Validity. Before running our reusable assets extraction method, we conducted a careful filtering of information systems from GitHub repositories. However, some threats may affect the correct filtering of systems, such as human factors that wrongly lead to the discard of a valid system to be evaluated. Considering that the exclusion criteria to system selection were applied in a manual process, we may have discarded interesting systems that we identified as non-Java, for instance.

Internal Validity. We conducted a lexical classification of entities that may be affected by some threats. To treat this possible problem, we selected a sample of 10 e-commerce systems from our data set, with diversified number of code lines. Then, we manually identified the names of entities from source code to find synonyms. We compared our manual results with the results provided by the tool and observed a loss of 10% in synonym terms identified through the automated process.

Conclusion Validity. After running our extraction tool, we gathered manually classes that seemed to represent the same real-world object. For instance, classes named as `Client` and `Costumer` were considered the same type of entity. The same occurred with methods identified by the tool as reuse candidates. However, this is a subjective process that may be affected by human factors. In this first exploratory study, we decided to not unify terms (eg., Customer and Client) in the quantitative analysis.

External Validity. We evaluated our method with a set of 38 information systems, extracted from GitHub. Considering that they may not represent all existing e-commerce systems, our findings may be not be generalized. Furthermore, we evaluated only one system domain, in case, e-commerce information systems. However, the collected systems are the most popular on GitHub that is a largely used platform. Finally, we evaluated systems implemented only in Java programming language, although it is one of the most popular languages worldwide.

7. RELATED WORK

Many approaches have been proposed in literature to extract reusable assets. For instance, Kawaguchi et al. [5] proposes an algorithm to categorize software projects automatically. This method aims to identify similar software systems using duplicated code detection and machine learning techniques.

Information Retrieval (IR) may be used to identify software components and extract reusable assets. For instance, CodeBroker [22] is a tool to support runtime identification of reusable software components using IR techniques. This tool provides source code recommendation in production environment according to the software component under development. CodeBroker is powered by search engines and Javadoc artifacts.

Another related work is presented by Kuhn et al. [7]. This study uses IR to explore linguistic data from source code such as identifiers and comments. This method partitions a software system in different semantic groups to identify source code concerns through the system.

Oliveira et al. [16] proposed a tool for recommendation of reusable assets. Their tool applies a technique that support software reuse and extraction of reusable assets called Automatic Identification of Software Components (AISC). The tool also provides an interactive graphic interface and export feature using a metadata representation model.

In turn, our reusable asset extraction method and supporting tool aim to identify candidates for reuse from software systems from an specific domain, using lexical analysis. Our method also ranks software components identified as reusable assets by frequency in which their appear in different systems from the same domain. We expect that this approach is helpful in reuse recommendation by suggesting methods and classes that are the most used in information systems given an specific domain.

8. CONCLUSION AND FUTURE WORK

In this paper, we propose a method to extract reusable assets from information systems. We also present a prototype tool that implements the proposed method. Finally, we present the preliminary idea of a recommendation tool to suggest software reusable assets in a class diagram-based vision. Through our study, we were able to identify the main issues of the software reuse and opportunity extraction process.

We evaluate our extraction tool through an experiment with 38 information systems from e-commerce domain extracted form GitHub. Our findings point that the proposed method was able to suggest methods and classes that appear in different systems from the set of input systems. We also observed that the most frequent methods and classes pointed by the tool as candidates to software reuse are interesting to most of the information systems from the e-commerce domain.

As future work, we intend to: increase the JReuse reusable assets extraction tool to compare also attributes from similarly named classes. Then, we will be able to recommend classes as reuse opportunities considering the three basic elements from object-oriented software systems: classes, methods, and attributes; to develop a recommendation tool that uses JReuse for extraction of reusable assets. The main purpose of this tool is to provide reusable assets for developers using a class diagram-based vision. As the developer models its application using UML diagrams, the recommendation searches for reuse opportunities in a reusable assets repository. Then, the tool provides suggestions of architectural components for the systems under modeling. The relationships between classes may be retrieved through an attribute-based analysis, so the tool may support the development of new software systems based on the architecture of existing systems.

We also intend to evaluate the proposed recommendation tool in two steps: **First**, an empirical study with developers to assess the tool's accuracy in suggesting assets that are appropriate to an specific software domain; **Second**, through an empirical study with software systems from an specific software development company, to assess the applicability of our method in the extraction of reuse opportunities from systems developed in the same organization.

9. ACKNOWLEDGMENTS

This work was partially supported by CAPES, CNPq (grant 485907/2013-5), and FAPEMIG (grant PPM-00382-14).

10. REFERENCES

[1] G. Caldiera and V. Basili. Identifying and qualifying reusable software components. *Computer*, 24(2):61–70, 1991.

[2] J. Cybulski and K. Reed. Requirements classification and reuse: Crossing domain boundaries. In *Proceedings of the 6th International Conference on Software Reuse (ICSR)*, pages 190–210, 2000.

[3] W. Frakes and C. Terry. Software reuse: metrics and models. *Computing Surveys (CSUR)*, 28(2):415–435, 1996.

[4] J. Guo and Luqi. A survey of software reuse repositories. In *Proceedings of the 7th International Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*, page 92. IEEE, 2000.

[5] S. Kawaguchi, P. Garg, M. Matsushita, and K. Inoue. Mudablue: an automatic categorization system for open source repositories. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC)*, pages 184–193, 2004.

[6] C. Krueger. Software reuse. *Computing Surveys (CSUR)*, 24(2):131–183, 1992.

[7] A. Kuhn, S. Ducasse, and T. Gírba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, 2007.

[8] J. Lee, K. Kang, and S. Kim. A feature-based approach to product line production planning. In *Software Product Lines*, pages 183–196. Springer, 2004.

[9] H. Liu and R. Lu. Word similarity based on an ensemble model using ranking svms. In *Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 283–286, 2008.

[10] Y. Maarek, D. Berry, and G. Kaiser. An information retrieval approach for automatically constructing software libraries. *Transactions on Software Engineering (TSE)*, 17(8):800–813, 1991.

[11] P. Mohagheghi and R. Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering (ESE)*, 12(5):471–516, 2007.

[12] P. Mohagheghi, R. Conradi, O. Killi, and H. Schwarz. An empirical study of software reuse vs. defect-density and stability. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 282–291, 2004.

[13] R. Monroe and D. Garlan. Style-based reuse for software architectures. In *Proceedings of the 4th International Conference on Software Reuse (ICSR)*, pages 84–93. IEEE, 1996.

[14] M. Morisio, M. Ezran, and C. Tully. Success and failure factors in software reuse. *Transactions on Software Engineering (TSE)*, 28(4):340–357, 2002.

[15] J. Neighbors. The evolution from software components to domain analysis. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 2(03):325–354, 1992.

[16] M. Oliveira, E. Goncalves, and K. Bacili. Automatic identification of reusable software development assets: Methodology and tool. In *Proceedings of the 8th International Conference on Information Reuse and Integration (IRI)*, pages 461–466. IEEE, 2007.

[17] T. Ravichandran and M. Rothenberger. Software reuse strategies and component markets. *Communications of the ACM*, 46(8):109–114, 2003.

[18] M. Sojer and J. Henkel. License risks from ad hoc reuse of code from the internet. *Communications of the ACM*, 54(12):74–81, 2011.

[19] Y. Tian, D. Lo, and J. Lawall. Sewordsim: Software-specific word similarity database. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 568–571, 2014.

[20] Z. Wang, X. Xu, and D. Zhan. A survey of business component identification methods and related techniques. *International Journal of Information Technology*, 2(4):229–238, 2005.

[21] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering.* Springer Science & Business Media, 2012.

[22] Y. Ye and G. Fischer. Reuse-conducive development environments. *Automated Software Engineering (ASE)*, 12(2):199–235, 2005.

[23] L. Yujian and L. Bo. A normalized levenshtein distance metric. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 29(6):1091–1095, 2007.

[24] Z. Zhen, J. Shen, and S. Lu. Wcons: An ontology mapping approach based on word and context similarity. In *Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 334–338, 2008.