A METHOD BASED ON NAMING SIMILARITY TO IDENTIFY REUSE OPPORTUNITIES

JOHNATAN ALVES DE. OLIVEIRA

A METHOD BASED ON NAMING SIMILARITY TO IDENTIFY REUSE OPPORTUNITIES

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Eduardo Magno Lages Figueiredo

Belo Horizonte

Agosto de 2016

JOHNATAN ALVES DE. OLIVEIRA

A METHOD BASED ON NAMING SIMILARITY TO IDENTIFY REUSE OPPORTUNITIES

Dissertation presented to the Graduate Program in Ciência da Computação of the Universidade Federal de Minas Gerais in partial fulfillment of the requirements for the degree of Master in Ciência da Computação.

Advisor: Eduardo Magno Lages Figueiredo

Belo Horizonte
August 2016

Oliveira, Johnatan Alves de.

O48m A Method Based on Naming Similarity to Identify Reuse Opportunities / Johnatan Alves de. Oliveira. — Belo Horizonte, 2016
xxii, 58 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Eduardo Magno Lages Figueiredo

1. Computação –Teses. 2. Software- Reutilização. 3. Software – Desenvolvimento. 4. Sistemas de recomendação I. Orientador. II. Título.

CDU 519.6*32 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

A method based on naming similarity to identify reuse opportunities

JOHNATAN ALVES DE OLIVEIRA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Eduardo M.P. Figueiredo
PROF. EDUARDO MAGNO LAGES FIGUEIREDO - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]
PROF. ELDER JOSÉ REIOLI CIRILO
Departamento de Computação - UFSJ

[illegible]
PROF. MARCO TÚLIO DE OLIVEIRA VALENTE
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 17 de agosto de 2016.

Acknowledgments

A realização desta dissertação marca o fim de uma importante etapa da minha vida. Gostaria de agradecer a todos aqueles que contribuíram de forma decisiva para a sua concretização

A Deus, que todos os dias de minha vida me deu forças para nunca desistir.

Aos meus pais pelos ensinamentos concedidos a mim, trabalhando incansavelmente para me proporcionar ensino de qualidade. Desta forma, a educação que mim disponibilizaram, não se pode comprar nem vender. Além disso, pela coragem e apoio que mim asseguraram em busca dos meus sonhos. Agradeço também a minha irmã pelo apoio e incentivo nas tomadas de decisões, me proporcionando coragem e forças para alcançar os meus sonhos.

Agradeço eternamente ao professor Eduardo Figueiredo, pela confiança aplicada sobre mim, desde o início dessa "empreitada". O professor foi mais do que um orientador nessa jornada. Registro aqui, a profunda admiração e respeito que tenho por ele, pude perceber o grande profissional que é durante a nossa convivência no mestrado.

Agradeço aos colegas de laboratório de pesquisa (Labsoft) pelas dicas e suporte ao longo de todos os seminários e etapas para alcançar essa conquista. Embora seja um trabalho individual, esta dissertação recebeu contribuições diretas e indiretas para sua realização. Cabe aqui agradecer o apoio dos "co-autores" dos artigos publicados e aos alunos de iniciação científica. Estes, me auxiliaram a enfrentar diversas dificuldades e por essa razão, quero lhes expressar meus agradecimentos sinceros.

Agradeço ao Programa de Pós-Graduação em Ciência da Computação (PPGCC) pela oportunidade de realizar um curso de alto nível de excelência. A todos os professores do mestrado que de alguma forma contribuíram para a minha formação.

Agradeço ainda à CAPES pelo período em que financiou minhas pesquisas.

Enfim, agradeço a todos que contribuíram, de forma direta ou indireta para que eu alcançasse este título.

“É muito melhor lançar-se em busca de conquistas grandiosas, mesmo expondo-se ao fracasso, do que alinhar-se com os pobres de espírito, que nem gozam muito nem sofrem muito, porque vivem numa penumbra cinzenta, onde não conhecem nem vitória, nem derrota”

(Theodore Roosevelt)

Resumo

Reutilização de software é uma estratégia de desenvolvimento em que os componentes de software existentes são utilizados no desenvolvimento de novos sistemas de software. Há muitas vantagens da reutilização no desenvolvimento de software, como a minimização dos esforços de desenvolvimento e melhoria da qualidade de software. Nesta dissertação, é proposto um método para a identificação de oportunidades de reutilização baseados na similaridade dos nomes de dois tipos de entidades orientadas a objetos: classes e métodos. O método desenvolvido, chamado JReuse, computa por meio de uma função de similaridade com o objetivo de identificar classes e métodos de nomes semelhantes, a partir de um conjunto de sistemas de software de um mesmo domínio. Essas classes e métodos compõem um repositório com oportunidades de reutilização. Além disso, apresentamos uma ferramenta protótipo para apoiar o método proposto. O método e a ferramenta foram aplicados em 72 sistemas de software minerados do GitHub, em 4 domínios diferentes: contabilidade, hospital, restaurante e e-commerce. No total, esses sistemas possuem 1.567.337 linhas de código, 57.017 métodos e 12.598 classes. Depois da sua aplicação, JReuse foi avaliada através de uma pesquisa com 32 desenvolvedores do GitHub nos domínios avaliados. Como resultado, foi possível obervar que JReuse é capaz de identificar as principais classes e métodos que são mais frequentes em cada domínio selecionado.

Palavras-chave: Reuso de Software, reuso, artefatos reutilizáveis, oportunidades de reuso, estratégia de extração.

Abstract

Software reuse is a development strategy in which existing software components 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. Few methods have been proposed in literature for recommendation of reuse opportunities. In this dissertation, we propose a method for identification 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 proposed method. We applied the method and tool to 72 software systems mined from GitHub, in 4 different domains: accounting, hospital, restaurant, and e-commerce. In total, these systems have $1,567,337$ lines of code, $57,017$ methods, and $12,598$ classes. After its application, we evaluated JReuse through a survey with 32 developers from GitHub in the evaluated domains. As a result, we observe that JReuse is able to identify the main classes and methods that are frequent in each selected domain.

Keywords: Software reuse, ad-hoc software reuse, reusable assets, reuse opportunities, naming similarity, recommendation systems.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The increasing demand for larger and more complex software systems requires the use of existing software artifacts (Pohl et al., 2005). In this context, software reuse is a development technique in which previously implemented software components, are used in the development of new software systems (Krueger, 1992). Reuse has been studied and indicated as an alternative to the traditional software development aiming to increase software quality and decrease development efforts by using existing, and sometimes tested, software components (Mohagheghi and Conradi, 2007; Morisio et al., 2002; Ravichandran and Rothenberger, 2003).

Methods for identification of reuse opportunities are essential to support the building of repositories of reuse opportunities (Guo and Luqi, 2000). These methods may be used in different contexts related to software developement, including the support of feature identification for a software product line (Lee et al., 2004), for instance. Many methods have been proposed in the literature to support the identification of reuse opportunities from software systems (Caldiera and Basili, 1991; Kawaguchi et al., 2004; Kuhn et al., 2007; Maarek et al., 1991; Ye and Fischer, 2005). However, to the best of our knowledge, we did not find a method able to identify reuse opportunities from several systems, considering most frequent entities such as classes and methods from systems of a single domain.

1.1 Motivation

To support software reuse, developers need first to find the relevant source code fragments to be reused. For instance, a code fragment may be relevant as a reuse opportunity because of its efficient in terms of performance. Other reason to reuse a fragment is it quality because, in general, existing code have been submitted to spection and testing. Finally, by reusing existing fragments, developers may decrease development efforts and, then, increase their productivity.

Since software systems have been increasing and evolving along the years, to identify frequent source code fragments with a particular functionality may be difficult. That is, because of significant number of classes, methods, and lines of code in these systems, the identification of reuse opportunities is a hard task. In this context, automated analysis and identification of reuse opportunities are useful to minimize costs (Ye and Fischer, 2002).

In addition, one of the current drawbacks in the software reuse process is the classification of the most frequent reuse opportunities from a set of software systems. Since many opportunities may be identified, it is important to recommend the most appropriate for reuse to developers, given a domain. For this purpose, some studies have proposed supporting techniques (Caldiera and Basili, 1991; Ye and Fischer, 2005).

Previous work in literature proposed methods to support the identification of reuse opportunities in software systems. These methods apply different techniques for source code analysis, such as natural-language processing (Maarek et al., 1991), formal specifications (Caldiera and Basili, 1991), machine learning (Kawaguchi et al., 2004), and other Information Retrieval techniques (Kuhn et al., 2007; Ye and Fischer, 2005). However, to the best of our knowledge, we did not find a method able to: (i) identify reuse opportunities, (ii) recommend the reuse opportunities identified, and (iii) show the name and the location of the main entities identified as reuse opportunities, given a set of systems (design partial).

1.2 Proposed Work

During the software development, the reuse of existing artifacts is an attractive way to reduce development costs and time-to-market and improve the software quality (Mohagheghi and Conradi, 2007). Source code is the artifact most commonly reused in software development (Morisio et al., 2002). However, to identify the reuse opportunities in large systems, and even in small systems, is a far from trivial task.

In this dissertation, we propose a method for identification of reuse opportunities called JReuse. Considering a set of software systems, JReuse aims to identify similarly named classes and methods from the systems based on lexical analysis. From the most frequent classes, JReuse analyzes the methods of these classes to identify similarly named methods. The dissertation also presents a prototype tool that supports the proposed method. JReuse provides a list of classes and methods recommended as reuse opportunities. This list may guide developers in the use of existing entities that are common in systems from a given domain.

JReuse performs the identification of reuse opportunities in two well-defined steps. First, given a set of systems from the same domain, the proposed method analyzes similarly named classes to identity the most frequent classes. Second, from the classes identified in the previous step, JReuse analyzes similarly named methods to identify the most frequent ones. Our method has been designed to analyze object-oriented software systems, independent of size, and applicable to different domains. The proposed method can be used to provide support to identify opportunities of software reuse. In addition, the method was built to guide users through partial design in the development of new software systems, showing the most frequent entities.

We conducted an evaluation of our method in two steps. First, we performed an empirical study conducted in controlled environment with 72 software systems. These systems were collected from GitHub and belong to four different software domains: accounting, restaurant, hospital, and e-commerce. Second, we conducted a survey with experienced developers from two of the four domains, namely e-commerce and hospital. We evaluated only these two domains because of the low percentage of responses for the other domains. With respect to the first evaluation, we observed that JReuse is able to identify reuse opportunities using naming similarity analysis for classes and methods. Regarding the second evaluation, participants from the survey agree that JReuse provides classes for the analyzed domains as reuse opportunities.

1.3 Publications

This dissertation generated the following publications and, therefore, it contains resources from them.

- Oliveira, J., Fernandes, E., Souza, M., and Figueiredo, E. (2016). A method based on naming similarity to identify reuse opportunities. In *Proceedings of the XII Brazilian Symposium on Information Systems (SBSI)*, pages 305–312 (***best paper***)

- Oliveira, J. and Figueiredo, E. (2016). A recommendation system of reuse opportunities based on lexical analysis. In *Proceedings of the IX Workshop Thesis and Dissertations in Information Systems (WTDSI)*, pages 49–51

- Oliveira, J. A., Fernandes, E. M., and Figueiredo, E. (2015). Evaluation of duplicated code detection tools in cross-project context. In *Proceedings of the III*

Workshop on Software Visualization, Evolution and Maintenance (VEM), pages 49–56

1.4 Outline of the Dissertation

While this chapter introduced this dissertation, the remainder of this document is organized as follow.

Chapter 2 presents background information to support the comprehension of this dissertation. It includes the main concepts related to the study, such as software reuse and reuse techniques. We also discuss related work.

Chapter 3 describes JReuse, a method proposed for identification of reuse opportunities using lexical analysis. We presents the similarity analysis computation used by our method, the method steps, and a supporting tool that implements the method.

Chapter 4 provides an evaluation of the proposed method. This evaluation consists of an empirical study conducted in controlled environment. We present the study design and the main results we obtained by analyzing 72 systems from four different domains: accounting, restaurant, hospital, and e-commerce.

Chapter 5 presents a survey conducted with experienced software developers for two of the four domains analyzed in Chapter 4. We discuss the main obtained results.

Chapter 6 concludes the dissertation with a discussion regarding the proposed method and its applications to the identification of reuse opportunities. We summarize the contributions of the study, and suggestions for future work.

Chapter 2

Background and Related Work

In order to speed up software development with quality and low cost, we can apply software reuse techniques. Such techniques consist of using existing software artifacts in the development of new systems, with minimization of efforts. Even in an *ad hoc* approach, the reuse of source code fragments is a recurrent activity in development settings (Ajila et al., 2012; Wang et al., 2012; Xue, 2011). However, by reusing code fragments in many parts of the system, developers may cause code clones (Fowler, 2009; Marcus and Maletic, 2001; Patil et al., 2015). To support reuse, methods and supporting tools for identification of reuse opportunities are required.

This chapter presents background information and work related to this dissertation. Section 2.1 presents an overview on recommendation systems because our method is based on recommendation system principles. Section 2.2 presents the concepts about software reuse and discusses the advantages and drawbacks of each type of reuse. Section 2.3 shows some examples of the use of similarity analysis to identify opportunities of reuse and code clone. Section 2.4 presents previous work in the area of identification of reuse opportunities from source code. Lastly, Section 2.5 concludes this chapter with some final remarks.

2.1 Recommendation Systems in Software Engineering

Along the years, the availability of data regarding software systems has been increasing in terms of volume and complexity (Herlocker et al., 2004). In the context, Recommendation Systems for Software Engineering (RSSEs) are tools to support developers in many tasks, such as decision-making with respect to the data and recommendation of reuse opportunities (Cubranic and Murphy, 2003). These tools aim to analyze the developers' needs and preferences, indicated implicitly or explicitly through the tool, and then to recommend artifacts to the developer (Robillard et al., 2010). Such artifacts may be source code fragments, for instance (Holmes et al., 2006).

In general, software developers spend significant efforts to find software artifacts in source code repositories (Begel et al., 2010). For example, considering large-sized systems, it may be difficult for developers to find software components to reuse (Ko et al., 2006). Therefore, to extract reuse opportunities is a task that impacts negatively on the productivity of developers (Begel et al., 2010). In this context, a RSSE may support developers in the process for identification of reuse opportunity by recommending source code in an automatic fashion (Holmes et al., 2006). Some examples of studies that propose RSSEs for reuse opportunities are CodeBroker (Ye and Fischer, 2005) and Strathcona (Holmes et al., 2006). CodeBroker aims to identify similar class library elements using a text-based analysis, to support developers in the used of APIs. Strathcona is a Recommendation system to assist developers in finding fragments of code, or examples, of an API's use.

2.2 Software Reuse

In software reuse, previously implemented software components are used to support the development of new software (Krueger, 1992). The main goal of reuse is the improvement of software quality aspects followed by an increasing development efficiency with low cost (Ravichandran and Rothenberger, 2003). There are many approaches to support reuse in software development. Krueger (1992) presents an extensive study regarding definitions and application of software reuse.

There are two types of software reuse: *ad hoc* and systematic reuse (Mohagheghi and Conradi, 2007). *Ad hoc*, reuse is applied in an opportunistic way, without planning, taking as an example the reuse of random software code snippets from the Web (Sojer and Henkel, 2011). In turn, systematic software reuse follows specific protocols and processes to provide the use of existing software components when developing new systems (Mohagheghi and Conradi, 2007).

Wang et al. (2005) conducted a study regarding the identification of business domain components to support reuse. According to their work, there are two types of component identification: forward and reverse. In forward identification, reuse is planned before the development of software systems. On the other hand, in reverse identification, reuse opportunities are identified from a set of existing software systems.

Some studies investigated advantages and drawbacks of systematic software reuse (Mohagheghi and Conradi, 2007; Mohagheghi et al., 2004). Mohagheghi et al. (2004) 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 (2007) 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.

Many strategies are proposed in literature, based on techniques such as: natural-language processing, formal specifications, architecture style, and machine learning. In natural language processing, lexical inspection of source code elements is conducted to identify reuse opportunities (Maarek et al., 1991). In formal specifications, the reusable components are extracted with support of software models and metrics analysis (Caldiera and Basili, 1991). In architectural style (Monroe and Garlan, 1996), where software reuse is supported by the analysis of interacting components in a high-level abstraction, such as software design and modeling, different analyzes are conducted to extract reuse opportunities, such as automated semantic categorization of software components (Kawaguchi et al., 2004).

2.3 Analysis of Similarity

Some studies proposed and discussed in literature (Kukich, 1992; Navarro, 2001; Fluri et al., 2007) investigating the analysis of similarity and their applications. For example, the analysis of similarity utilized in this dissertation is Levenshtein Distance. The Levenshtein Distance denotes the minimum number of operations needed to transform one string into the other (Levenshtein, 1966). The operations are: (i) insert a character, (ii) delete a character, or (iii) substitute a character. Algorithm is based on the problem of the longest common subsequence (Levenshtein, 1966). A larger distance means that the strings are less similar, that is, that more operations are necessary to transform one string into another, whereas a distance of 0 operations denotes that the strings are equal (Fluri et al., 2007). The runtime-complexity is $O(n.m)$, where n is the number of characters in $string_a$ and m in $string_b$.

Many studies have applied similarity analysis in the identification of reuse opportunities and code clone detection (Li et al., 2016; Marcus and Maletic, 2001; Roy and Cordy, 2008; Selim et al., 2010; Yuan and Guo, 2012). As an example, Li et al.

(2016) present a study that applies similarity analysis techniques to support the identification of reuse opportunities. They present a method for identification of similar implementations of Android mobile application. The proposed method aims to provide the identification of families of applications. For this purpose, they compute a similarity function based on the number of similar methods and the total number of methods in two different systems under comparison.

Li et al. (2006) propose a tool called CP-Miner for code clone detection. CP-Miner relies on data mining techniques and targets on copy-paste occurrences of code clone. The method parses source code to compute hash values for sentences. After, the tool analyzes source code to find frequent subsequence that may present clones, using a mining algorithm.

2.4 Related Work

Previous work investigates the identification of reuse opportunities from software systems (Inoue et al., 2005; Koziolek et al., 2013; Li et al., 2005; Mende et al., 2009; Michail and Notkin, 1999; Oliveira et al., 2007; Ye and Fischer, 2005). Inoue et al. (2005) propose a graph-based technique to support the extraction of frequently used components in a given software component repository. The proposed technique relies on ranking components based on their usage by other components from the repository. The authors also present a supporting tool called SPARS-J, for analysis of Java classes.

Koziolek et al. (2013) present a technique for identification of reuse opportunities based on domain analysis. The proposed technique aims to support the assessment of potential SPL by organizations. This technique encompasses feature modeling of the domain, comparison of systems in architectural level, and the extraction of reusable components. Li et al. (2005) present an approach for identification of reusable components from legacy systems. The proposed approach aims to support reengineering tasks; that is, the implementation of new systems based on existing source code. For this purpose, the authors propose the generation of the Abstract Syntax Tree (AST) for analysis and extraction of modules and components as candidate for reuse. Therefore, we may consider this approach as a recommendation system.

Mende et al. (2009) propose a tool to support software evolution and maintenance by identifying similar methods along the source code and recommend merging of these methods to the developer. The proposed tool may be considered as a recommendation system. This tool computes code clones in method-level and uses the Levenshtein' distance for textual comparison of methods. Michail and Notkin (1999) propose a tool, CodeWeb, to support the comparison of software libraries in terms of components (i.e., classes and methods) provided by these libraries. For this purpose, the tool performs naming similarity computations to identify similar classes and methods from a set of libraries. We consider the proposed tool as a recommendation system, since it provides the identification of the appropriate libraries for reuse.

Oliveira et al. (2007) propose a method and a supporting tool for recommendation of reusable software components. The proposed tool applies a technique for software reuse and identification of candidates for reuse called Automatic Identification of Software Components (AISC). The tool, called Digital Assets Discoverer, performs static analysis of code to identify reuse opportunities. The tool also provides an interactive graphic interface and exports feature using a metadata representation model.

Ye and Fischer (2005) present a supporting tool called CodeBroker to support runtime identification of reusable software components. The proposed tool relies on information retrieval techniques for identification of reuse candidates. Since the tool executes in runtime, it provides recommendation of source code components in production environment. For this purpose, CodeBroker is based on search engines and Javadoc artifacts for code analysis.

In turn, our reuse opportunities identification method and supporting tool aim to identify candidates for reuse in software systems from an specific domain, using lexical analysis. Unlike other approaches presented, our method can be used for two purposes. First, provide support to identify reuse opportunities in software. Second, guide users through partial design in developing new software systems, showing the most frequent entities. Our method also ranks software entities identified as reuse opportunities by frequency in which the appear in different systems from the same domain. We expect this approach to be helpful in reuse recommendation by suggesting methods and classes that are the most used in software systems given a specific domain.

2.5 Final Remarks

This chapter provided the background information necessary to fully understand the approach proposed in this dissertation. It discussed the concepts of recommendation systems, software reuse, and techniques for identify reuse opportunities. These concepts are essentials since our approach is a recommendation system that aims to achieve a better software design by recommending reuse opportunities.

In the next chapter, we detail our method for identifying reuse opportunities from source code. The method called JReuse is divided in two steps. First, JReuse analyzes all classes of a domain. Second, JReuse analyzes all methods from classes identified in the previous step. Then, the method provides a list of entities identified as reuse opportunities of the domain analyzed.

Chapter 3

Proposed Method

Strategies for identifying reuse opportunities are essential to support the building of reuse opportunity repositories. Software systems have been increasing and evolve. Therefore, to identify reuse opportunities is difficult. For this purpose, developers need supporting methods and automated tools. In this context, we propose a method for identification of reuse opportunities. This chapter explains in detail the proposed method for identification of reuse opportunities. Section 3.1 discusses each step of the similarity-based identification process used in our method. Section 3.2 describes the proposed method for reuse opportunity identification. Section 3.3 presents a supporting tool that implements our method. Finally, Section 3.4 concludes the chapter with final remarks.

3.1 Identifying Similarity

Some studies in the literature investigated the textual similarity identification (Tian et al., 2014; Zhen et al., 2008). There are many applications for similarity analysis, such as comparison of dialects, spell check, and plagiarism detection (Liu and Lu, 2008). The proposed method called JReuse relies on static code analysis techniques to identify reuse opportunities. We conducted an ad hoc review in order to select algorithms that compute the similarity between strings to be utilized in this work. Furthermore, this study aimed at identifying the most utilized algorithms in the literature, appropriate for this study purposes. In order to achieve these purposes, we selected the Levenshtein's algorithm. Levenshtein's algorithm (Yujian and Bo, 2007) 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 turn **A** into **B**.

To identify similarly named classes, we adopted a threshold of, at least, 75% similarity between two entities. In addition, to identify similarly named methods, we adopted the same threshold used to identify classes. These thresholds were derived empirically by the author of this dissertation. This threshold was chosen because we observed that some well-known naming conventions for classes and methods may lead to similarly named entities that clearly represents different purposes . We also considered some well-known naming conventions for classes and methods to define this threshold. For instance, the similarity computed for `Costumer` and `CostumerDAO`, that are commonly named classes in e-commerce systems, is 72%. However, intuitively these classes implement different functions (i.e., DAO classes implement data base persistence).

An example of similarity analysis is shown in Table 3.1. In this example, we present eight matches between names of classes from two software systems. In this example, classes from System **A** and System **B** has at least 75% similarity rate with class names. The adopted thresholds covers, for instance, the similarity between names that vary from singular to plural (e.g., `Client` and `Clients`).

Table 3.1. Similarity Evaluation

System A	System B	Similarity Rate
Shopp**ing**Cart	ShoppCart	75%
OrderProduc**tId**	OrderProduc	78%
Orderservi**ce**	Orderservi	83%
Reviwe**s**	Reviwe	85%
Client**s**	Client	85%
CartControll**er**	CartControll	85%
Product**s**	Product	87%
Product**s**Controller	ProductController	94%

3.2 The Method Steps

A software domain is a set of systems that shares a common set of functionalities, requirements, or terminology (Neighbors, 1992; Pressman, 2005). Thus, we expect that software systems within the same domain present lexical similarity in names of elements, such as classes and methods. In this context, similarly named elements may contribute to the comprehension of the characteristics in a given business domain (Cybulski and Reed, 2000).

Considering this scenario, our study proposes a method called JReuse for identification 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, the method compares names of classes and methods in order to identify common elements among different systems. We believe that recurring names of classes may indicate reuse opportunities in a given domain. Furthermore, frequent names of methods in these classes may indicate common behaviors and requirements in such entities (Cybulski and Reed, 2000). Regarding the identification of methods, we adopted two rules. First, JReuse excludes all **get** and **set** methods. Second, JReuse analyzes the return type of each method. For instance, if the pair of methods has the same return type, they are considered as reuse opportunities. Otherwise, the methods are considered as different functions.

In general, only the similarity rate is not enough for electing a class as a possible reuse opportunities. We also consider the classes that 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). We believe that recurring names of classes may indicate reuse opportunities in a given domain. Furthermore, frequent names of methods in these classes may indicate common behaviors and requirements in such entities (Cybulski and Reed, 2000).

Our method compares different elements as illustrated in Figure 3.1 and described as follows. First, JReuse compares all classes from the set of systems to identify the similarly named classes, JReuse has not synonymous analysis. Therefore, entities such as Client and Customer are considered as different entities. Second, considering only classes pointed as reuse opportunities in the previous step, JReuse compares pairs of methods by name to identify similarly named methods.

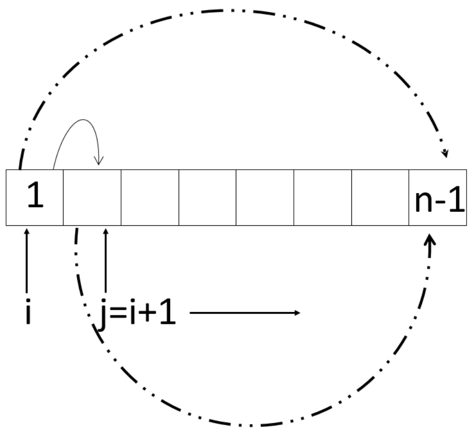


Figure 3.1. Steps to Identify Common Entities

For both steps, the same process is performed as illustrated in Figure 3.1. Consider `array[1..n]` an array of names of elements (classes or methods) and two pointers $i = \{1, .., n-1\}$ and $j = \{2, .., n-1\}$. For each i, we compare `array[i]` with `array[j]` for $j = \{i+1, .., n-1\}$. If `array[`i`]` is similar to `array[`j`]` with a minimum similarity rate of 75%, then the method registers a reuse opportunity.

Seven steps are required by the JReuse to identify reuse opportunities as presented in Figure 3.2. The steps are described as follows.

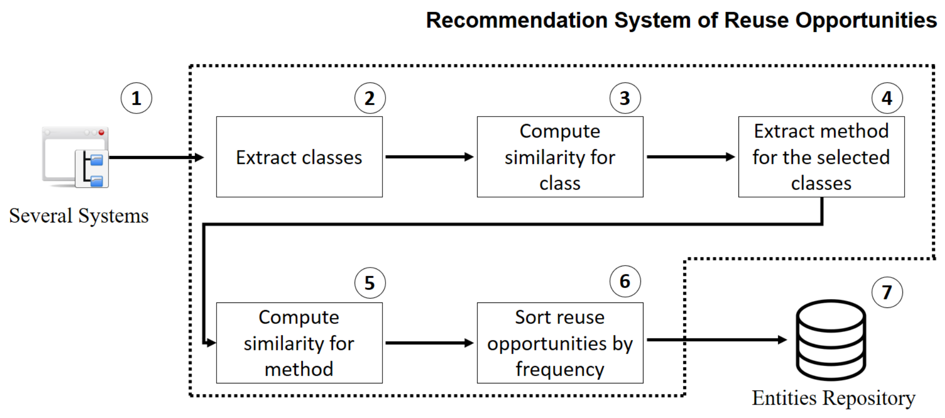

Recommendation System of Reuse Opportunities
1
Several Systems
2
Extract classes
3
Compute similarity for class
4
Extract method for the selected classes
5
Compute similarity for method
6
Sort reuse opportunities by frequency
7
Entities Repository

Figure 3.2. Steps of the Method called JReuse

1. In this step, the method receives as input software systems from the data set provided by the user. These systems are supposed to belong to the same domain. Then, the method collects all classes, filters out files that are not Java and eliminates all system projects that are Android.

2. In this step, the method identifies the class to compute the similarity between the classes entities of other systems. The method does not compare classes of the same system.

3. In this step, the JReuse compares names of classes in pairs to identify class names with at least 75% of similarity. Classes with similar names (matches) are gathered and each class name receives a score that is the number of systems in which the class occurs. The higher the score, the higher the class seems to be relevant for the analyzed domain.

4. For each class identified as reuse opportunities, their methods are identified to compute the similarity between these entities. In this step, all methods are identify from the selected classes, with exception the methods `set` and `get`.

5. From of each class group identified as reuse opportunities, their methods are analyzed with the same threshold (with at least 75%) adopted for classes. However, for a method to be considered a candidate for reuse opportunity, it is necessary that the return types of both methods are the same. The similarity calculation is performed between different systems; the method does not evaluate the similarity between the entities of a single software system.

6. After the identification of entities (classes and methods) and calculation of similarity, JReuse sortes in decreasing order by frequency of the identified reuse opportunities when necessary.

7. Finally, in this step the method JReuse builds a repository of entities as reuse opportunities. Thus, it is possible, to use the entities previously identified.

3.3 Tool Support

To automate the proposed method, we developed a prototype tool that implements JReuse for Java software systems. We selected Java because (i) it is one of the most popular programming languages, (ii) there is an available Java parser to support source code analysis by the generation of an Abstract Syntax Tree (AST), (iii) many studies have been investigating software reuse in Java systems. Through the Java parser, it is possible to access the structure of the source code, JavaDoc, and comments. It is also possible to change the AST nodes or create new ones to modify the source code, and (iv) it is one of the most popular programming languages[1]. In addition, to support the identification of similarly named entities (classes and methods), we used the Eclipse Java Development Tools (JDT) parser.

As input, our tool receives a set of software systems from the same domain. Second, the tool ignores all files that are not Java and all files present in test packages. Third, we discarded methods named `main`, since this type of method is essential in any `Java` system. Note that, given a system, all classes may contain a `main` method. In general, this method is responsible for starting the system (Rountev, 2004). Then, JReuse processes classes and methods to identify similarly named entities according to the proposed method.

JReuse provides an abstraction for the design organization of a system given a domain. In other words, the reuse opportunities identified by JReuse may be used to compose a partial design for any system that belongs to the analyzed domain in terms

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

of frequent classes and methods. For this purpose, the tool outputs one `CSV` file. Each line of the file has (i) the name of a class identified as reuse opportunity, (ii) a method from the respective class also identified as reuse opportunity, and (iii) the absolute path of the class. The output file is sorted in decreasing order by frequency of the identified reuse opportunities.

Tool Architecture: The architectural design of our tool is presented in Figure 3.3. JReuse is a plug-in for Eclipse IDE. Therefore, it shares common features from the Eclipse source code. In this figure, we present each layer of the JReuse code. The *View Layer* encompasses the implementation of the graphical user interface. The *Controller Layer* is responsible to manage the tool and provide the similarity computation for classes and methods. The *Model Layer* provides the modeling of the data base. Finally, the DAO Layer implements data persistence.

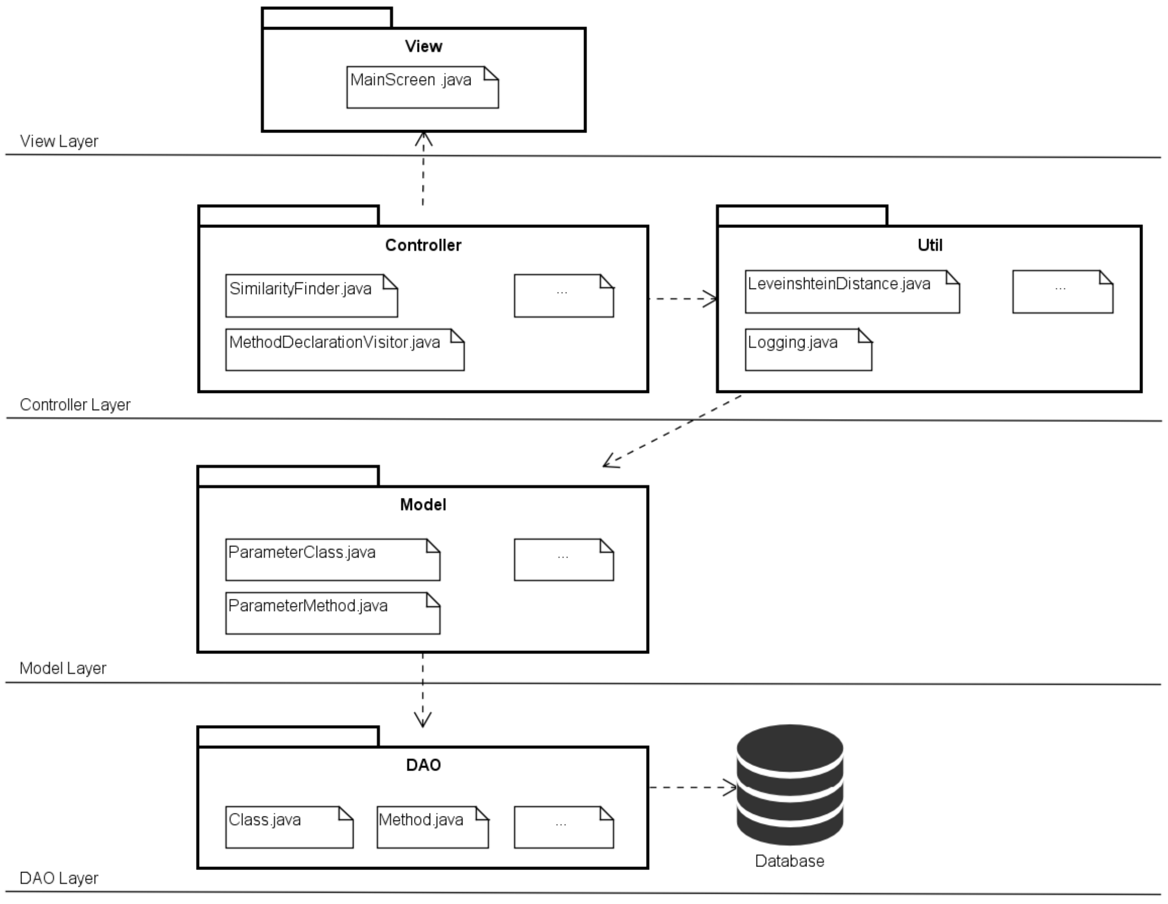
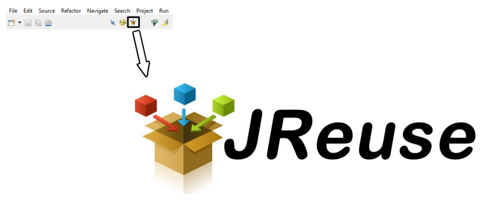
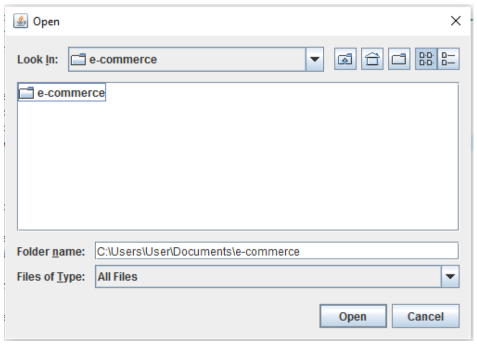


Figure 3.3. Architecture of the tool



(a) JReuse button in the Eclipse menu



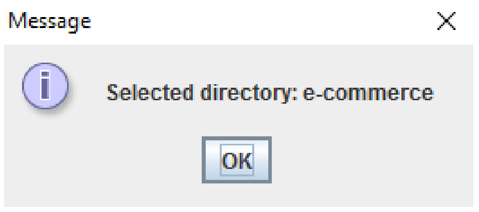
Look In: e-commerce

e-commerce

Folder name: C:\Users\User\Documents\e-commerce

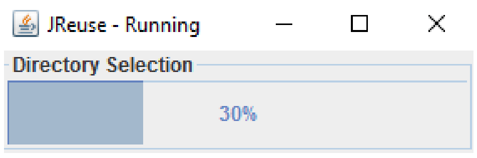
Files of Type: All Files

(b) Screen to choose directory for analysis



Selected directory: e-commerce

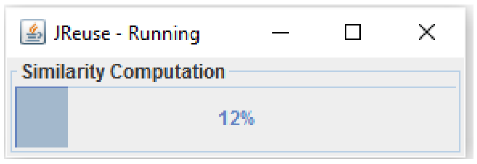
(c) Dialog to inform the selected directory



Directory Selection

30%

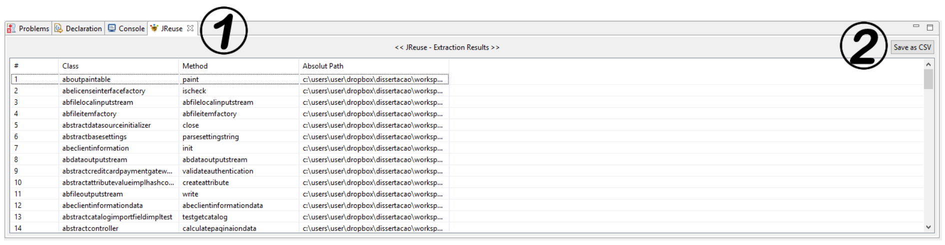
(d) Progress bar for preprocessing Java files



Similarity Computation

12%

(e) Progress bar for similarity computation



<< JReuse - Extraction Results >>

#	Class	Method	Absolut Path
1	aboutpaintable	paint	c:\users\user\dropbox\dissertacao\worksp...
2	abelicenseinterfacefactory	ischeck	c:\users\user\dropbox\dissertacao\worksp...
3	abfilelocalinputstream	abfilelocalinputstream	c:\users\user\dropbox\dissertacao\worksp...
4	abfileitemfactory	abfileitemfactory	c:\users\user\dropbox\dissertacao\worksp...
5	abstractdatasourceinitializer	close	c:\users\user\dropbox\dissertacao\worksp...
6	abstractbasesettings	parsesettingstring	c:\users\user\dropbox\dissertacao\worksp...
7	abeclientinformation	init	c:\users\user\dropbox\dissertacao\worksp...
8	abdataoutputstream	abdataoutputstream	c:\users\user\dropbox\dissertacao\worksp...
9	abstractcreditcardpaymentgatew...	validateauthentication	c:\users\user\dropbox\dissertacao\worksp...
10	abstractattributevalueimplhashco...	createattribute	c:\users\user\dropbox\dissertacao\worksp...
11	abfileoutputstream	write	c:\users\user\dropbox\dissertacao\worksp...
12	abeclientinformationdata	abeclientinformationdata	c:\users\user\dropbox\dissertacao\worksp...
13	abstractcatalogimportfieldimpltest	testgetcatalog	c:\users\user\dropbox\dissertacao\worksp...
14	abstractcontroller	calculatepaginaiondata	c:\users\user\dropbox\dissertacao\worksp...

(f) (1) Data grid view for JReuse results; (2) "Save as CSV" button

Figure 3.4. User interface of JReuse

User Interface: Figure 3.4 presents the user interface of JReuse. Figure 3.4(a) illustrates the JReuse button in the Eclipse menu. Figure 3.4(b) presents the screen to choose the directory with Java projects for analysis. Figure 3.4(c) shows a dialog to inform the chosen directory. Figure 3.4(d) is the progress bar for preprocessing, in which JReuse mines all projects to find Java files for identification of reuse opportunities. Figure 3.4(e) is the progress bar for similarity analysis, in which the tool computes similarity among names of classes and, after, among names of methods for the classes pointed as candidates to reuse opportunities. Finally, Figure 3.4(f) presents the data grid view to display the JReuse results of analysis.

3.4 Final Remarks

This chapter presented our method to identify and extract reuse opportunities, through the similarity analysis. The proposed method consists of two phases. First, it computes similarity between classes and methods in different software projects. Second, the method sorts the extracted reuse opportunities (classes and methods) in decreasing order of occurrence. In addition, we described the thresholds derived empirically to support the identification of similar classes and methods.

We also present, an Eclipse plug-in that implements our identification method, by describing the design and main functionalities of the method. Our tool allows developers to obtain recommendations of reuse opportunities, which can be applied in the development of new software systems.

In the next chapter, we present the evaluation of our method. For this purpose, we conduct an empirical study in controlled environment to evaluate JReuse with a total of 72 software systems from four domains: accounting, restaurant, hospital, and e-commerce. The data set is composed by systems collected from GitHub.

Chapter 4

Method Evaluation

This chapter describes an evaluation adopted for analyzed the method proposed and presented in Chapter 3. We conducted an empirical evaluation through study exploratory conducted in environment controlled. To perform the evaluation, we adopted the guidelines based on Wohlin et al. (2012), for this type of evaluation in Software Engineering. The focus of our method is identify the main reuse opportunities in systems of software. Therefore, evaluation consists in analyze the reuse opportunities identified by our method JReuse.

We organized the remainder of this chapter as follows. Section 4.1 presents the study goal and the research questions we designed to guide our study. Section 4.2 describes the data set used to evaluate our method and the prototype tool, both called JReuse. Section 4.3 presents the steps adopted in the evaluation. Section 4.4 presents the results of the evaluation of the classes identified as reuse opportunities. Section 4.5 presents the results of the evaluation of the methods identified as reuse opportunities from the classs identified previously. Section 4.6 provides an overview and discusses the main lessons learned. Section 4.7 presents threats to the validity of our study. Finally, Section 4.8 concludes this chapter with final remarks.

4.1 Goal and Research Questions

In this study, we aim to assess whether JReuse is able to identify frequent classes and methods in a specific software domain. We are also interested in assessing the relevance of the results provided by our method. For this purpose, we chose four domains to be evaluated: accounting, hospital, restaurant, and e-commerce. We also formulated the two research questions (RQs) to guide our study.

RQ1 *What are the most frequent classes in software systems for each selected domain? And how are they distributed through systems?*

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

Through *RQ1* and *RQ2* , we are interested in investigating whether the most frequent identified classes and methods are indicated for software systems for the respective domain. We expect that JReuse is able to provide a list of classes and methods whose recommendations for reuse are relevant for the respective domains.

4.2 Data Set

To evaluate our method, we chose only systems from the domain of accounting, hospital, restaurant, and e-commerce, for several reasons. First, software systems from these domains encompass several business features, such as user personnel, financial, product, and service management. Second, there is a significant number of domain systems available for download in GitHub[1]. Third, from the viewpoint of the authors, the four domains we chose are well-defined in terms of requirements and we believe that it would be possible to find reuse opportunities among systems of these domains.

The systems that compose our data set were extracted from GitHub repositories. We performed the selection of systems for the e-commerce domain in January 2015 and in May 2016 for the other domains. We selected software systems are based on the ranking of starred systems and system length in terms of storage space. In GitHub, stars are a meaningful measure for repository popularity among the platform users, and may be used to support the selection of systems.

There is a diverse terminology to represent a same software domain. For instance, we may refer to the *e-commerce* domain as *ecommerce*, without hyphenation. In order to support the collection of software systems to compose our data set, we developed an algorithm to clone GitHub repositories individually, with the respective systems, based on a well-defined search string for each domain under analysis in this study. Since the goal of our study is to identify reuse opportunities from different software systems, given large system sets per domain, we defined the following search strings.

For e-commerce: *e-commerce* OR *ecommerce* OR *electronic commerce*
For restaurant: *restaurant* OR *eatery* OR *restaurants*

[1]https://github.com/

For hospital: *hospital* OR *infirmary* OR *lazaretto*
For accountancy: *accountancy* OR *accounting*

Table 4.1 presents the exclusion criteria applied in the selected systems. First, we collected 400 Java systems from GitHub, 100 for each domain in order descending sorted by stars. Then, we discarded systems according to the following exclusion criteria: (i) non-Java software systems, since GitHub do not verify automatically the main programming languages of the systems, (ii) Java projects developed for Android platform, because Android systems tend to have a different architectural design and code implementation when compared with traditional Java systems, (iii) systems with less than 1,000 lines of code (LOC), and (iv) systems written in other languages rather than English, since our method relies on a lexical similarity technique and, then, natural language may impact significantly the results provided by our method.

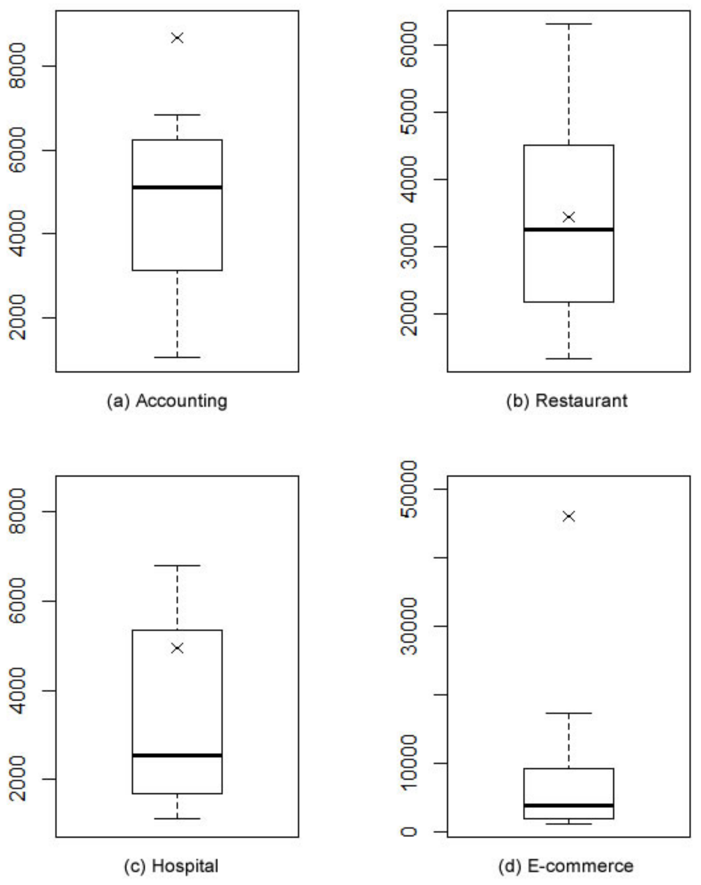
Table 4.1. Filters that were applied to the data set

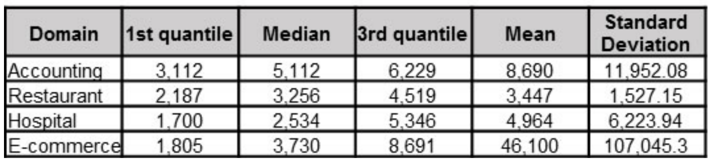
Domains	**Excluded Systems by**			**Selected**
	Not English	**Less than 1,000 LOC**	**Android**	**Systems**
Accounting	9	49	31	11
Restaurant	3	56	28	13
Hospital	16	37	34	13
E-commerce	21	40	4	35

For each selected system, we considered only the last release. This process was necessary to discard different versions of the same system, which probably contain lots of similarly named classes and methods. Finally, we obtained in 72 Java systems for evaluation of the JReuse method.

To better characterize systems in the four domains, Figures 4.1, 4.2, and 4.3 presents software metrics for systems per domain: lines of code (LOC), number of classes (NOC), and number of methods (NOM), respectively. We plotted twelve boxplots, one for each metric. However, because of the heterogeneity of the sample of our data set, we decided to eliminate ”outliers” for each metric. Therefore, all boxplots presented a brief overview of each analyzed domain.

Let's consider Figure 4.1 in the following analysis of LOC. With respect to the accounting domain, we observe that the mean of LOC for the systems is 8,690. Moreover, the median is 5,112, i.e., half of the accounting systems has at least 4 KLOC. That is, a significant number for analysis and identification of reuse opportunities. Regarding the restaurant domain, the mean of LOC is 3,447. In addition, the median is 3,256. Again, we conclude that these systems have a significant LOC for analysis. For the hospital domain, the mean is 4,964 and the median is 2,534 of LOC. Although these values are smaller than the obtained values for the other domains, it remains significant for the study. Finally, with respect to the e-commerce domain, we observe a mean LOC of 46,100 and a median of 3,730. In general, systems from this domain have the highest numbers of LOC and, therefore, they may have several reuse opportunities.



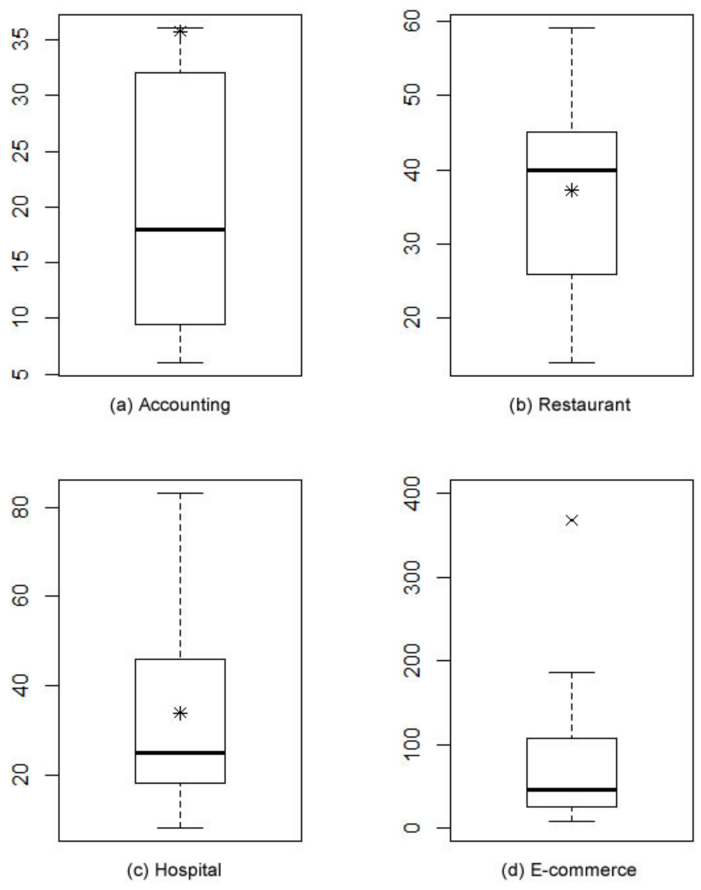


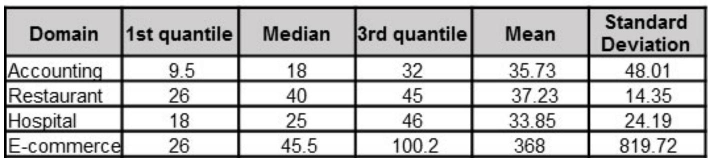
Domain	1st quantile	Median	3rd quantile	Mean	Standard Deviation
Accounting	3,112	5,112	6,229	8,690	11,952.08
Restaurant	2,187	3,256	4,519	3,447	1,527.15
Hospital	1,700	2,534	5,346	4,964	6,223.94
E-commerce	1,805	3,730	8,691	46,100	107,045.3

Figure 4.1. LOC of Sytems per Domain

Let us consider Figure 4.2 in the following analysis of NOC. With respect to the accounting domain, note that the mean of NOC for the systems is 35.73. Furthermore, the median is 18, i.e., half of the accounting systems has at least 18 classes. This number is significant for analysis because we are interested in finding similarly named classes within a pairwise comparison. Therefore, we expect a comparison of $18 * 18 = 324$ pairs that may be reuse opportunities. Regarding the restaurant domain, the mean of

NOC is 37.23. In addition, the median is 40. Again, we conclude that these systems have a significant NOC for analysis. For the hospital domain, the mean is 33.85 and the median is 25 of NOC. Finally, with respect to the e-commerce domain, we observe a mean NOC of 368.9 and a median of 45.5. In general, systems from this domain has the highest numbers of NOC and, therefore, there is a significant possibility of identifying reuse opportunities.

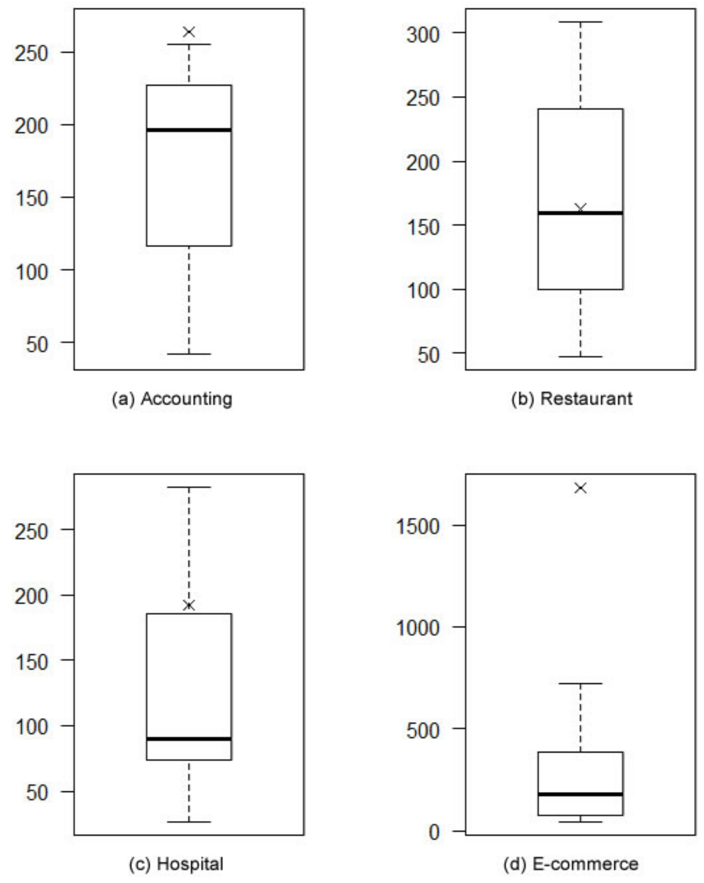




Domain	1st quantile	Median	3rd quantile	Mean	Standard Deviation
Accounting	9.5	18	32	35.73	48.01
Restaurant	26	40	45	37.23	14.35
Hospital	18	25	46	33.85	24.19
E-commerce	26	45.5	100.2	368	819.72

Figure 4.2. NOC of Sytems per Domain

Based in Figure 4.3, we discuss some observations as follows. With respect to systems from the accounting domain, we have a mean NOM of 263.6. In addition, the median is 196, i.e., half of the systems has at least 196 methods. That is, a significant number for analysis and extraction of reuse opportunities. This number is significant because we compute similarly named methods in pairs. Therefore, we expect a comparison of $196 * 196 = 38,416$ pairs that may be identified as reuse opportunities. Considering the restaurant domain, the mean of NOM is 162.4 and the median is 159. Again, we conclude that these system have a significant NOM for analysis. For the hospital domain, the mean is 192.2 and the median is 90 methods. At last, with respect to the e-commerce domain, we observe a mean NOM of 1,683 and a median of 175.5. That is, the highest number of methods for analysis considering the four selected domains.



Domain	1st quantile	Median	3rd quantile	Mean	Standard Deviation
Accounting	116	196	226	263	265,97
Restaurant	100	159	241	162	82,61
Hospital	74	90	186	192	269,78
E-commerce	78	175	370	1,683	3,780,99

Figure 4.3. NOM of Sytems per Domain

4.3 Evaluation Steps

Figure 4.4 presents the three study steps we followed to investigate the research questions described in Section 4.1. Each step is described as below.

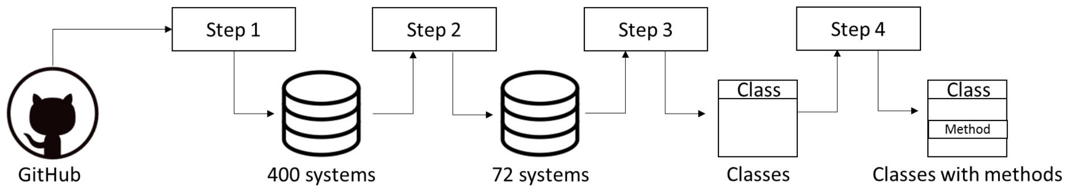


Figure 4.4. Designed an Exploratory Study

Step 1: Automated Search – By using the search strings described in Section 4.2, we cloned from GitHub several software systems, belonging to different domains. We intended to identify appropriate domains to be analyzed in our exploratory study. For this purpose, we considered a domain as appropriate when, from our viewpoint, systems from the given domain contain a significant number of classes and methods for analysis. After performing the search for systems, with support of our algorithm, we obtained 400 software systems from four distinct domains: accounting, restaurant, hospital, and e-commerce.

Step 2: Exclusion Criteria – By applying a set of exclusion criteria defined by the authors, we select the systems according to with the following requirements: (i) software systems written only in English, (ii) software systems with more than 1,000 lines of source code, and (iii) traditional Java software systems, i.e, software that are not exclusive to the Android platform. After applying the exclusion criteria, 72 different software systems remained for analysis.

Step 3: Detection of Similarly Named Classes – We executed the JReuse prototype tool for the 72 collected systems. Per domain, the respective systems were submitted to JReuse for extraction of reuse opportunities. After the automated analysis for each domain, JReuse provided a list with the most frequent classes that occur in the given domain.

Step 4: Detection of Similarly Named Methods – We also executed the JReuse prototype tool to identify similarly named methods. From the classes identified as reuse opportunity in Step 3, JReuse identified similar methods among these classes. In the previous step, JReuse provided most frequent classes per domain, as well as a list of classes, sorted by relevance, with the main classes identified as reuse opportunities. In this step, JReuse complements such list with methods identified as reuse opportunities. That is, the previously obtained list of reuse opportunities is completed with the most frequent methods for the identified classes.

4.4 Results of Frequent Classes

This section presents and discusses the study results aiming to answer our first research question. For this question, we discuss the results obtained with respect to the four domains under analysis: accounting, restaurant, hospital, and e-commerce.

RQ1 *What are the most frequent classes in software systems for each selected domain? And how are they distributed through systems?*

In this study, we analyzed the frequency of similarly named classes for the systems of each domain. Table 4.2 presents software metrics for systems per domain: lines of code (LOC), number of classes (NOC), and number of methods (NOM). This table categorizes NOC and NOM in two types: (i) *analyzed*, i.e., the number of entities analyzed by the tool and (ii) *recommended*, that is, entities identified by the tool as reuse opportunities.

In general, from Table 4.2 we observe that JReuse identified a smaller number of methods than classes as reuse opportunities. For instance, for domain e-commerce, JReuse identified 75 classes and 28 methods as reuse opportunities. One of the reasons for this results is that the similarity computation of JReuse for methods is more strict than for classes. Thus, the proposed method aims to avoid the recommendation of methods with similar names but different responsibilities. For this purpose, JReuse compares the return type of similarly named methods. As an example, if a pair of similar methods has the same return type, then they are considered as the same reuse opportunities. Otherwise, both methods are considered different.

Table 4.2. Software metrics for systems from each domain

Domains	**Systems**	**LOC**	**NOC**		**NOM**	
			Analyzed	**Recom-mended**	**Analyzed**	**Recom-mended**
Accounting	11	95,588	493	25	2,900	21
Restaurant	13	44,813	484	17	2,111	20
Hospital	13	65,297	446	21	2,516	20
E-commerce	35	1,567.337	12,598	75	57,017	28

In order to present and discuss the most frequent classes extracted as reuse opportunities, we considered the following exclusion criteria of classes. For each domain, we discarded classes that occur in a maximum of two different systems. This decision was taken because our method compares classes in pairs and, then, 3 occurrences may not be significant to a reuse recommendation. We selected the top-ten most frequent classes of each domain, as presented in Figures 4.3, 4.4, 4.5, and 4.6. We submitted the list of most frequent entities to a group of 4 researchers at Software Engineering Laboratory (LabSoft) from Federal University of Minas Gerais (UFMG), for validation of the entities with respect to relevance.

Tables 4.3, 4.4, 4.5, and 4.6 present classes identified as reuse opportunities for e-commerce, accounting, restaurant, and hospital, respectively. We selected only the classes with at least 15% [2] occurrences in the systems of the respective domain. Each table has a “Domain-Specific” field. This field indicates the viewpoint of the focal group regarding a given entity to be specific for the analyzed domain. The focal group's viewpoint is represented by three symbols in table: (i) the (✓) symbol indicates that the focal group agreed that the class is specific for the domain under analysis, (ii) the (✗) symbol indicates that the focal group disagreed that the class is indicated for the domain, and (iii) blank field (Unconfirmed) indicates that the focal group did not converge to a specific opinion on the class. Moreover, each table has a “Labels” filed to inform the level of relevance of the entity identified by JReuse as reuse opportunity.

Scale to Indicate the Level of Relevance of the Entities Identified. To support the identification of the most recommended classes and methods for each domain, Figure 4.5 shows a scale from 0% to 100% that represents the level of relevance to recommend an entity based on frequency of classes and methods identified as reuse opportunity. The thresholds 0% and 50% determine two labels for level of relevance, namely *weak* and *strong*. The *weak* label (from 0% to 50%) indicates that the class is weakly or moderately recommended as reuse given a domain. Finally, the *strong* label (from 50% to 100%) indicates that the class is highly recommended as reuse.

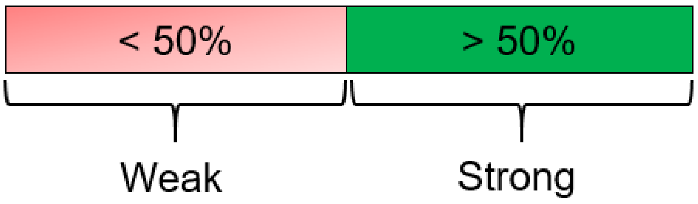


Figure 4.5. Scale of relevance to entity identified as reuse opportunity

[2]The percentage is arbitrary, i.e. can be adapted for domains with more or with less systems for analysis.

With respect to the accounting domain, presented in Table 4.3. For this domain, the classes from `Users` to `TransactionManager` belong to the *strong* label and, therefore, they are the highly recommended classes for accounting systems. Note that, in the other hand, the focal group did not consider the classes `Users`, `DatabaseConnection`, and `Util` as specific classes for the accounting domain. In addition, the classes from `AddFinancialsAction` to `RawMaterial` belong to the *weak* label. The remainder classes have exactly 2 or 3 occurrences in different systems from the accounting domain. Therefore, they are weakly recommended and were omitted from this table.

Table 4.3. Classes with at least 15% occurrences in the accounting domain

Labels	Classes	Frequency	% of systems	Domain Specific
Strong	Users	13	100%	✗
	DatabaseConnection	13	100%	✗
	CashFlow	11	85%	✓
	Util	10	77%	✗
	BalancesAssets	9	69%	✓
	CashBanks	9	69%	✓
	ShareholderEquity	9	69%	✓
	BalancesLiabilities	8	62%	✓
	ChartAccounts	8	62%	✓
	AccountingMovement	8	62%	✓
	AccountsReceivable	8	62%	✓
	AccountsPayable	6	46%	✓
	Transactions	7	54%	✓
	Log	7	54%	✗
	FinancialReportsPoeHelper	7	54%	✗
	InventoryManager	7	54%	✓
	TransactionManager	7	54%	✓
Weak	AddFinancialsAction	6	46%	✓
	Accounts	6	46%	✓
	FeaturesAnalysis	6	46%	✓
	RawMaterial	6	46%	✓

Key: Agree (✓) and Disagree (✗)

Regarding accounting domain, Figure 4.6 presents the top-ten most frequent classes for this domain, with the highest occurrence. Classes are sorted by frequency: `Users`, `DatabaseConnection`, `CashFlow`, `Util`, `BalancesAssets`, `CashBanks`, `ShareholderEquity`, `BalancesLiabilities`, `ChartAccounts`, and `AccountingMovement`. We observe that, although only `CashFlow` is considered specific to the given domain, from the viewpoint of the focal group, all classes from this label are meaningful in accounting systems. In turn, the remainder classes are from the *medium* label. Among these classes, `CashBanks`, `Transaction`, and `Accounts` are considered specific, for instance.

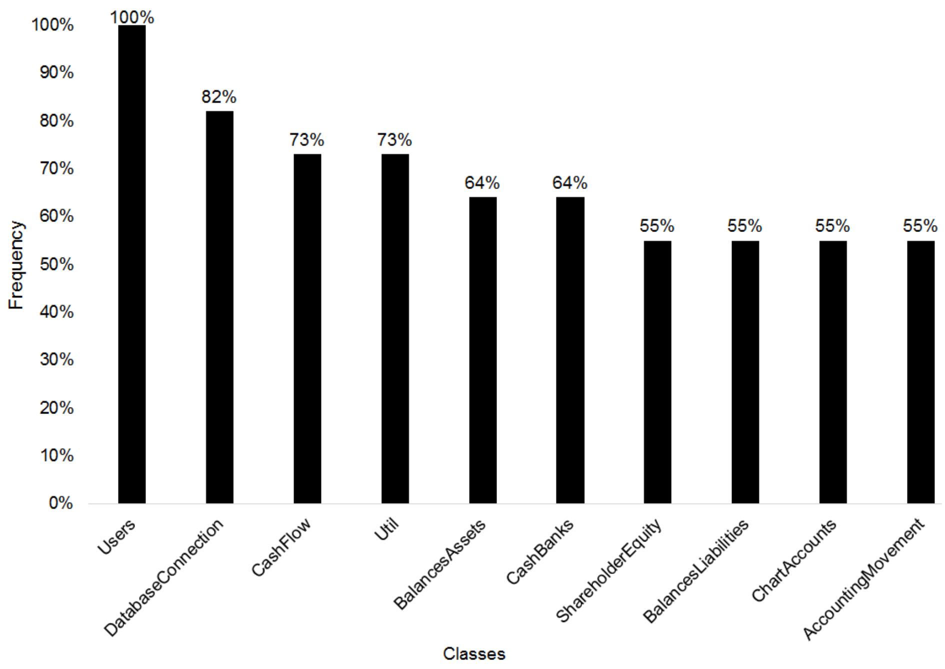


Figure 4.6. Distribution of frequent classes through accounting systems

With respect to the restaurant domain, let us consider Table 4.4. The classes `Login` and `User` belong to the *strong* label. Note that they are not considered as specific classes from the given domain considering the focal group's viewpoint. However, they are relevant in restaurant systems. In addition, the classes from `Client` to `Order` belong to the *strong* label and are relevants for this domain, considering the focal group's viewpoint. Many of them were pointed as reuse opportunities for restaurant systems by the focal group, such as *RestaurantMenu*, `Delivery`, and `Customes`. Moreover, this entities belong to the *weak*

Table 4.4. Classes with at least 15% occurrences in the restaurant domain

Labels	Classes	Frequency	% of systems	Domain Specific
Strong	Login	10	77%	✗
	User	10	77%	✗
	ConnectionManager	9	70%	✗
	Client	9	70%	✓
	Table	8	62%	✓
	PaymentType	8	62%	✓
	Dish	8	62%	✓
	Employee	7	54%	✓
	Order	7	54%	✓
Weak	RestaurantMenu	6	47%	✓
	Delivery	6	47%	✓
	ItemOrdered	6	47%	✓
	Customer	4	31%	✓

Key: Agree (✓) and Disagree (✗)

Regarding the analysis of the restaurant domain, Figure 4.7 presents the top-ten classes with higher occurrence, namely `Login`, `User`, `ConnectionManager`, `Client`, `Table`, `PaymentType`, `Dish`, `Employee`, `Order`, and `RestaurantMenu`. These classes have an high to medium level for recommendation according to the scale from Figure 4.5. The classes with the highest occurrences in this domain are `Login` and `User`, respectively. Both are present in 77% of the analyzed information systems. Nevertheless, they are not specific classes of restaurant systems. However, JReuse identified some frequent classes such as `Client`, `Table`, `PaymentType`, and `Dish`.

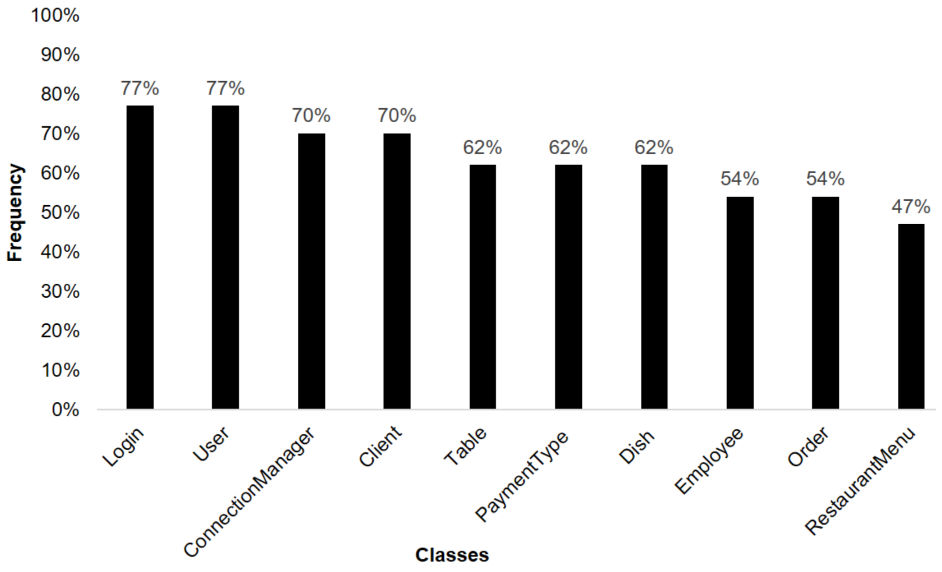


Figure 4.7. Distribution of frequent classes through restaurant systems

Consider Table 4.5 for analysis of the hospital domain. Observe that the classes from `Patient` to `Microbiology` belong to the *strong* label and, therefore, they are highly recommended classes as reuse opportunities. Note that, from the viewpoint of the focal group, the three most frequent classes are considered specific from hospital systems. In fact, classes such as `Patient` and `Doctor` are meaningful in the given domain. In addition, classes from `PatientCondition` to `OperationsWithCards` are from the *weak* label. Finally, the remainder classes have less than 10% of the occurrences

Table 4.5. Classes with at least 15% occurrences in the hospital domain

Label	Classes	Frequency	% of systems	Domain Specific
Strong	Patient	13	100%	✓
	Doctor	13	100%	✓
	Disease	11	85%	✓
	User	10	77%	✗
	Login	9	69%	✗
	Diagnose	9	69%	✓
	Symptoms	9	69%	✓
	PatientDisease	8	62%	✓
	HealthPlan	8	62%	✓
	Immunology	8	62%	✓
	Haematology	8	62%	✓
	Medication	7	54%	✓
	Surgery	7	54%	✓
	MedicalRecords	7	54%	✓
	TypePayment	7	54%	
	Microbiology	7	54%	✓
Weak	PatientCondition	6	46%	✓
	LaboratoryExams	6	46%	✓
	Log	6	46%	✗
	HistoPathology	6	46%	✓
	Connection	6	46%	✗
	Paycash	5	38%	
	Util	5	38%	✗
	OperationsWithCards	3	23%	

Key: Agree (✓), Disagree (✗), and Unconfirmed (field blank)

Figure 4.8 presents the most frequent classes identified for the hospital domain, in decreasing order of frequency. For the 13 systems we collected from this domain, JReuse extracted some relevant entities, such as `Patient`, `Doctor`, and `Disease`, from the focal group's point of view. The classes presented in this figure belong to the *strong* label, as illustrated in Figure 4.5. Note that the classes `Patient` and `Doctor` are present in 100% of the evaluated systems. Similarly to the other domains, JReuse identified some classes that are generic, such as `User` (77%) class, that are expected in systems from other domains.

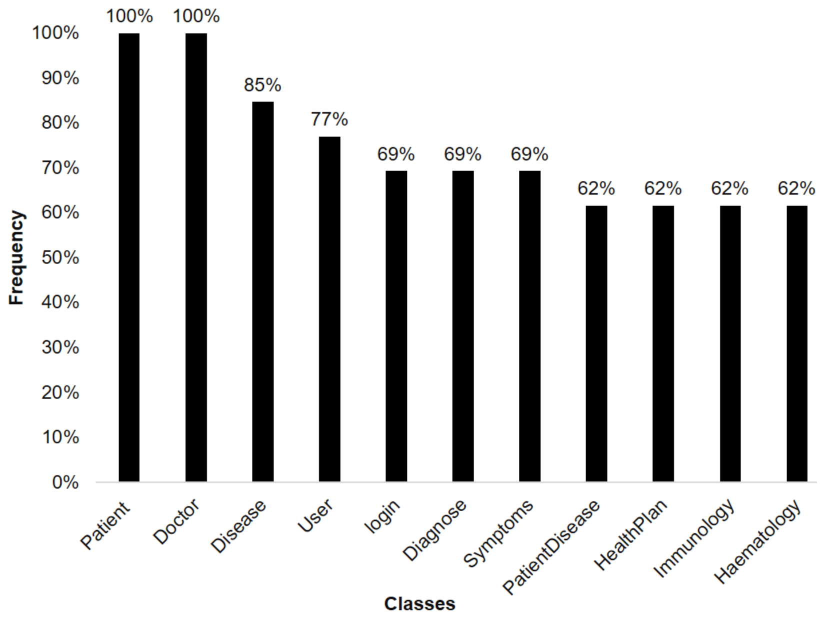


Figure 4.8. Distribution of frequent classes through hospital systems

Let us consider Table 4.6 regarding the e-commerce domain in the following discussion. Note that the classes `Product` to `ClientDao` belong to the *strong* label, according to Figure 4.5. That is, they are highly recommended classes for e-commerce systems, because they are present in more than 50% of the analyzed systems. In addition, the classes `Item` to `ShoppingCartService` are the weakly recommended classes. As aforementioned, classes with less than 15% of the occurrences were omitted.

Table 4.6. Classes with at least 15% occurrences in the e-commerce domain

Labels	Classes	Frequency	% of systems	Domain Specific
Strong	Product	28	80%	✓
	PaymentType	24	69%	✓
	Client	20	58%	✓
	ProductDao	18	52%	✓
	ClientDao	18	52%	✓
Weak	Item	17	49%	✓
	ShoppingCart	17	49%	✓
	User	17	49%	✗
	Customer	14	40%	✓
	Category	12	35%	✓
	ProductService	10	29%	✓
	Order	9	26%	✓
	LoginController	7	20%	✗
	UserDao	6	18%	✓
	ProductServiceImpl	6	18%	✓
	ShoppingCartController	6	18%	✓
	OrderedProduct	5	15%	✓
	ShoppingCartService	5	15%	✓

Key: Agree (✓) and Disagree (✗)

Finally, Figure 4.9 presents the top-ten most frequent classes for e-commerce systems. We sorted the classes in decreasing order of frequency. The most frequent entities are, respectively, `Product`, `PaymentType`, `Client`, `ProductDao`, `ClientDao`, `Item`, `ShoppingCart`, `User`, `Customer`, and `Category`. Note that, according to the focal group the classes `Product`, `Payment`, `ShoppingCart`, `Customer`, and `Client` are elementary entities to be expected in an e-commerce system. In turn, although `User` is one of the most frequent classes identified by JReuse (49% of the systems contain this class), `User` is not specific of the e-commerce domain. However, this entity is meaningful for information systems in general.

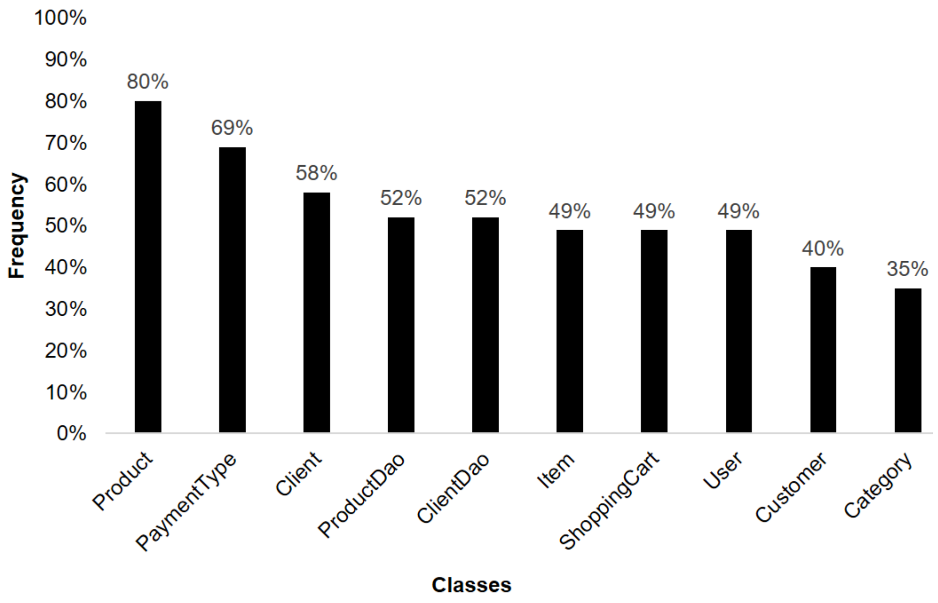


Figure 4.9. Distribution of frequent classes through e-commerce systems

4.5 Results of Frequent Methods

This section presents and discusses the results for of the methods identified from of top-ten most frequent classes, presented in first research question.

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

Research question *RQ2* is related to the extraction of similarly named methods from classes identified as reuse opportunities by JReuse. To summarize the data and present the principal methods of each domain, we adopted the following exclusion criteria. We discarded methods that does not appear in more than 2 classes from different systems. We discarded also methods named `main`, because this type of method is essential in any `Java` system (Rountev, 2004).

To support a discussion of *RQ*2 for each domain under analysis, Tables 4.10, 4.7, 4.8, and 4.9 present the most frequent methods for the top-ten most frequent classes with respect to the accounting, restaurant, hospital, and e-commerce domain, respectively. To support the identification of the most frequent methods for each domain, we use the same labels presented in Section 4.4. Each table is signed with two symbols: (i) the (✓) symbol that indicates the focus group agreed that the method should belong to a given class and (ii) the (✗) symbol that indicates the focus group disagree that a method belongs to a given class.

Considering the accounting domain for analysis, Table 4.7 presents the methods identified as reuse opportunity for the top-ten most frequent classes. The classes with at least 10 methods identified as reuse opportunities are `AccountingMoviment` and `BalancesLiability`. They contain 13 and 12 identified methods, respectively. We observe that both are relevant in the accounting domain, from the viewpoint of the focal group. Through Table 4.7, we may observe methods that may be strongly recommended as reuse opportunities (see Figure 4.5). As as example, `calculatePayment` appears in 20 `Product` classes, against 13 and 11 for `PaymentType`, and `ProductDao`, respectively.

Table 4.7. Methods most often identified from of most common classes among the accounting domain systems.

Methods	Classes									
	AccountingMovement	BalancesLiabilities	BalancesAssets	CashBanks	CashFlow	ChartAccounts	Util	ShareHolderEquity	Users	DatabaseConnection
execute			29%✗					**50%**✗	19%✗	**89%**✓
calculatePayment	**100%**✓					**67%**✗				
addTrans	**100%**✓	**50%**✗	**58%**✓	**58%**✗	**50%**✓				37%✗	23%✗
calcDuplicates	**50%**✓	34%✓	**58%**✓	**58%**✓	**63%**✓	**67%**✗		**50%**✓		
update	**50%**✓	**67%**✓		**72%**✓	**50%**✓					
generateReport	34%✓	34%✓	29%✓				**63%**✓			
deleteById	**67%**✓	**50%**✓	**72%**✓	43%✓	38%✓	50%✗				
convertToCsv							**63%**✓			
findByName	**67%**✓	**67%**✓	43%✓	43%✓	38%✓	50%✗		34%✓	28%✓	
othersValues	**50%**✗	**50%**✗	43%✗	**58%**✗	38%✗	34%✗				
checkPayment	**50%**✓	34%✗			**50%**✓					
generateXls		34%✓	29%✓				**50%**✓	34%✓		
makePay	**67%**✓	34%✗		**58%**✓						
salesValues	**50%**✓	**67%**✓								
validateInput	**50%**✓	**50%**✓	29%✓	43%✓	**50%**✓	50%✗	25%✓	**67%**✗	19%✓	34%✗
inventoryUtil						34%✗	**50%**✓			
printFile						34%✗	**50%**✓			
print							38%✓			
validateCheckPayment	**50%**✓									
parseDateToString				43%✗						
printCustDebt							25%✗			

Key: Agree (✓) and Disagree (✗)

For the restaurant domain, let us consider Table 4.8 with respect to the methods obtained from the top-ten most frequent classes. The classes `Table` and `Order` have at least 10 methods identified as reuse opportunities, with 14 and 13 types of methods, respectively. As an example, note that the method `insertFood` is present in 63% and 100% of the classes `Table` and `Order`, respectively. These values belong to the *strong* label in the scale from Figure 4.5. Methods such as `insertFood`, `generateBills`, and `cancelReservation` are relevant in restaurant systems, from the viewpoint of the focal group.

Table 4.8. Methods most often identified from of most common classes among the restaurant domain systems.

Methods	Classes									
	Table	Order	Dish	PaymentType	RestaurantMenu	User	Client	Login	Employee	ConnectionManager
search	75%✓	43%✓	75%✓	75%✓	100%✓	70%✓	100%✓	70%✗	100%✓	67%✗
insert	38%✓	86%✓	38%✓	100%✓	34%✓	70%✓	100%✓		100%✓	
remove	100%✓	86%✓	75%✓	38%✓	67%✓	60%✓	78%✓	70%✓	100%✓	
login								80%✓		
calculate	63%✗			88%✗						
validationUser						70%✓		50%✓		
insertFood	63%✓	100%✓			100%✓					
disableUser						70%✗				
editReserve	63%✓	86%✓								
deleteTable	75%✓									
generateBills	75%✓	58%✓	63%✗	75%✓						
consultCode	25%✗	72%✗	63%✓		100%✓					
updatePosition	63%✓	72%✗								
cancelReservation	63%✓	29%✗	25%✗							
reserveTable	63%✓	43%✗								
removeObservation	50%✗	72%✗	63%✗	38%✗		40%✗	56%✗			
updateMenu		58%✗			84%✓					
insertRequest		58%✓								
calculateAll	50%✓			50%✓						
lastOrder	25%✓	29%✗								

Key: Agree (✓) and Disagree (✗)

Table 4.9 presents the methods obtained from the top-ten most frequent classes from hospital domain. The classes with at least 10 types methods identified as reuse opportunities are `Patient`, `Diagnose`, `Disease`, and `PatientDisease`. They have 16, 11, 10, and 10 identified methods, respectively. Methods such as `verifyPathology`, `findPatient`, and `validatePatient` are relevant for the given context, from the viewpoint of the focal group. For instance, the method `verifyPathoogy` is present in 73% and 75% of the classes `Disease` and `PatientDisease`, respectively. These values correspond to the *strong* label in the scale of Figure 4.5.

Table 4.9. Methods most often identified from of most common classes among the hospital domain systems.

	Classes									
Methods	Patient	Diagnose	Disease	PatientDisease	HealthPlan	Doctor	Symptoms	Immunology	User	Login
deleteData	47%✓	78%✓	37%✓	88%✓	75%✓	54%✓	78%✓	63%✓	80%✓	34%✗
verifyPathology	39%✗	45%✓	73%✓	75%✓			67%✓			
findPatient	54%✗				63%✗	31%✓				
registerDisease	24%✗	56%✗	64%✓	50%✓	63%✗	31%✓		38%✗		
insertPatient	54%✗									
anamnesis	54%✗				38%✗					
saveData	47%✓	56%✓	37%✓	75%✓	88%✓	39%✓	67%✓	63%✓	50%✓	34%✗
updateData	47%✓	78%✓	37%✓	75%✓	88%✓	39%✓	67%✓	63%✓	70%✓	34%✗
symptoms	16%✗	56%✓	55%✓	38%✓			78%✓			
diagnosisPerformed			55%✗			31%✓				
bloodGroup	47%✓	45%✗		50%✗						
sonography	16%✗	23%✗	28%✗	38%✗	75%✗	24%✓				
patientProfile	39%✓							38%✗		
insertCoagulation	39%✗	34%✗		25%✗			23%✗			
fetchDetail	16%✗	23%✓	19%✓	50%✓	63%✗		23%✗	25%✓		
authorizeUser									40%✗	
scheduling						31%✓				
insertBlood		34%✗	37%✗					25%✗		
sons	24%✓				38%✗					
validatePatient	24%✓									

Key: Agree (✓) and Disagree (✗)

Given the e-commerce domain, let us consider Table 4.10 for analyze the methods obtained from the top-ten most frequent classes. The classes with at least 10 types methods identified as reuse opportunities are `Product`, `Customer`, `Item`, `Client` `PaymentType`, `User`, and `ShoppingCart`, respectively. They have 15, 12, 12, 12, 11, 10, and 10 identified methods, respectively. We observe that all of them are relevant in the e-commerce context, from the viewpoint of the focal group. In Table 4.10, we may observe methods that may be strongly recommended as reuse opportunities for a given class.

Table 4.10. Methods most often identified from of most common classes among the e-commerce domain systems

Methods	Classes									
	Product	Customer	Item	Client	PaymentType	User	ShoppingCart	ProductDao	ClientDao	Category
addCustomerAddress		36%✓								
alter	61%✓	58%✓	100%✓	80%✓	63%✓	30%✓	48%✓	89%✓	89%✓	59%✓
buy				10%✓	67%✓					
calculateSubtotal	8%✗		18%✗		25%✓					
calculateTotal	22%✗		12%✗		50%✗		48%✓	12%✗		
changePassword						59%✓				
changeStock	68%✗				46%✗		65%✓	45%✗		
checkout	15%✗						77%✓			
cities				25%✗						
delete	68%✓	79%✓	95%✓	65%✓	59%✓	83%✓	71%✓	89%✓	78%✓	59%✓
findByCategory	29%✗							62%✓		67%✗
findByEmail		50%✗		15%✗						
login				50%✓		100%✓				
moveitemToCart	11%✗		48%✗				30%✓			
moveListToCart	8%✗		42%✗							
password		15%✓				100%✗			62%✓	
processRegister	40%✗		53%✗		30%✓				39%✗	
processUpdateAccount		29%✗		10%✗		53%✓				
productList	72%✗				55%✗			62%✗		
register	61%✓	86%✓	89%✓	100%✓	71%✓	77%✓	53%✓	95%✓	78%✓	67%✓
removeCustomerAddress		36%✓								
resetPassword						89%✓				
reviewItem			48%✓							
save	29%✓	36%✓	83%✓	25%✓	25%✓	77%✓	65%✓	45%✓	39%✓	75%✓
shoppingCartItem	11%✗		71%✗				89%✓			
update	58%✓	79%✓	71%✓	65%✓	55%✓	95%✓	48%✓	34%✓	17%✓	75%✓
validateAddress		15%✓		20%✓					12%✗	
validatePobox		43%✓		25%✓						

Key: Agree (✓) and Disagree (✗)

The method `productList`, for example, appears in 72% of the `Product` classes (i.e., a strong level of recommendation, according to Figure 4.5), against 55% and 62% for `PaymentType`, and `ProductDao`, respectively. Therefore, this method is a strong candidate to compose a `Product` class, although it may be present in other classes such as `PaymentType` and `ProductDao`. The Count line presents the number of different types of methods that were found for a given type of class. For instance, for the `Customer` class, JReuse identified 12 types of methods as reuse opportunities.

4.6 Lessons Learned

In this study, we learned a lot regarding interesting research topics such as software reuse, reuse opportunities identification, and recommendation systems. For this propose, we take as an example the e-commerce domain, especially by the popularity and size of these systems on GitHub. We discuss some of the main lessons learned with support of the following questions.

How much a lexical analysis may support the identification of reuse opportunities assets? As discussed in Section 2.2, there are many approaches to support software reuse in literature. Lexical analysis is a simple one. However, as pointed by the results of Section 4.1, it may be effective to identify reuse opportunities 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 our exploratory study which was conducted in a controlled environment (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 identify reuse opportunities at least in this domain.

Names of classes and methods are suitable to the entities they represent in a business domain? We discuss in Chapter 3 that names of classes and methods may be useful for reuse opportunities identification. In fact, we observed that naming similarity identification may support reuse opportunities identification. However, to retrieve similarly named classes and methods may be uninteresting if they are not representative in an specific domain. Chapter 4 highlights identified 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 in this study that our method is able to identify 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 reuse opportunities identification tool in a reuse recommendation system? Methods and classes are elementary entities of object-oriented software systems. 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.

To the best of our knowledge, we have not found many recent studies with respect to reuse opportunities identification, supported by 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.

4.7 Threats to Validity

We based our study on related work to support the method definition, the tool development, and the proposal of a recommendation system. Regarding the evaluation of our method and tool, we conducted a careful empirical study to assess effectiveness of the tool with respect to reuse opportunities identification that are representative in the enterprise software 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. (2012).

Construct Validity. Before running our reuse opportunities identification 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 discard a valid system to be evaluated. Considering the exclusion criteria for selection of systems (see Section 4.2), we implemented an algorithm to automate this process and, then, discard inappropriate systems for analysis. However, we may have discarded relevant software systems by using our algorithm, such as systems misidentified as non-Java systems.

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 entities. 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 identify 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 process is subjective and may be affected by human factors. In this first exploratory study, we decided to not unify terms (e.g., Customer and Client) in the quantitative analysis.

External Validity. We evaluated our method with a set of 72 systems, extracted from GitHub. Considering that they may not represent the 4 domains analyzed, our findings may be not be generalized. Furthermore, we evaluated only four system domains, accounting, restaurant, hospital, and e-commerce. 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, our results may not generalize to other programming languages.

4.8 Final Remarks

This chapter reported a study, conducted to evaluate our method and tool. We report in Section 4.2, the selection process of the domains of systems evaluated. In this context, we investigate how our method performs in a wider range of systems.

Section 4.5 and 4.4, we evaluated our method and tool with 72 open-source systems available on GitHub. Our method called JReuse, reached acceptable results in every domains evaluated, indicating its applicability in different domains and identifying the main opportunities for reuse of each domain. We derive a threshold to measure the level of relevance in recommend particular entity as reuse opportunities. The scale contains two levels, *strong* and *weak*.

The next chapter describes the execution of a survey with GitHub developers for the purpose of assessing the results from JReuse analysis of software systems from 4 distinct software domains (e-commerce, hospital, restaurant and accounting systems). However, because of the low response rate for the domains: accounting and restaurant, we discarded both domains in the analysis in next chapter.

Chapter 5

Survey with Developers

As shown in Chapter 4, our method was applied to identify reuse opportunities for four software domains under analysis: e-commerce, accounting, restaurant, and hospital. However, because of the low response rate for the domains: accounting and restaurant, we discarded both domains in the analysis presented in this chapter. The reuse opportunities identify by our method, need manual inspection in order to evaluate these results. The goal of this chapter is to provide an empirical evaluation of the top-ten most frequent classes in each domain identified by the proposed method. This chapter presents a survey with domain experts in each software domain analyzed. Section 5.1 presents the settings required to design the survey, such as the selection of participants. Section 5.2 presents the background of the survey participants. Section 5.3 presents results of this study. Section 5.4 shows some threats to validity that may affect our findings. Finally, we conclude this chapter with some final remarks in Section 5.5

5.1 Survey Settings

The proposed method is able identify reuse opportunities in systems of different domains and distinct sizes. To assess the relevance of the results of the exploratory study conducted in a controlled environment (presented in Chapter 4), we conducted a preliminary evaluation of JReuse. As a preliminary study, we assess only the identification of classes by the method. For this purpose, we designed a survey with specialists from each of the four software domains under analysis: e-commerce, accounting, restaurant, and hospital. A survey is a research strategy to identify characteristics of a population of individuals (Wohlin et al., 2012). In general, it is conducted with support of associated to the use of questionnaires for data collection (Easterbrook et al., 2008).

The survey is composed by four questionnaires, with the purpose of assessing the JReuse results for each of the four software domains analyzed. We asked domain experts to indicate the level of relevance for a certain class to a given domain. We consider an increasing scale of relevance from 0 to 5; 0 means that the developer disagrees completely that the class is specific of the domain assessed; 5 means that he fully agrees that the class is exclusive of the domain analyzed. The survey was conducted in June and July of 2016.

Each questionnaire contains 17 questions, as shown in Table 5.1. The first four questions, namely Q1 to Q4, are related to background information and aim to provide us to provide us with background information of the participants. The other questions, from Q5 to Q15, are related to classes that may be specific for the given domain, or non-specific, from the viewpoint of the participants. A control class, `Game`, was introduced, to assess the quality of the responses. It is out of the scope of the four domains under analysis. Finally, Q17 is an open question for participants to provide us subjective comments regarding the survey. We do not discuss Q17 in this dissertation, because it represents only feedback for the survey improvement in further replications.

Table 5.1. Survey Settings

Group	Questions	Alternative
Background	(Q1) Do you work with software development ?	"Yes"; "No"; "Partially"
	(Q2)How long do you develop software?	"For less than one year"; "Between one and three years"; "For more than three years"
	(Q3)Choose the highest level in computer science	"PhD"; "Master Degree"; "Complete Graduate"; "Ongoing graduate program"; "High school or below"; "I don't have knowledge about computer science"
	(Q4) Do you develop software products for the <DOMAIN> domain ?	"Yes"; "No"; "Partially"
Domain Specific Questions	(Q5 Q15) Please, consider the following classes. Which classes do you consider that belong to the < DOMAIN > domain? Assume a range from 0 to 5 where 0 means you disagree completely that the class is from the < DOMAIN > domain and 5 means you completely agree that the class is exclusive from the < DOMAIN > domain.	likert scale: 0 to 5
Comments	(Q17) If you have further comments, please use the text area below	Open question

In general, literature recommends the selection of a representative sample participants from the target population to perform a survey (Easterbrook et al., 2008). For this purpose, we based our participant selection on previous work (Salvaneschi et al., 2014; Kalliamvakou et al., 2014). We performed eight steps described as follows, considering the systems we collected from GitHub. First, for each domain, we selected the top-200 most popular repositories sorted by decreasing order of stars. In GitHub, stars are a meaningful measure for repository popularity among the platform users, and they may be used to support the selection of well-evaluated systems by developers. Second, we excluded projects analyzed in Section 4.2. This decision was made to minimize bias with respect to the previous knowledge of participants on the analyzed systems.

Third, we excluded Android projects because the design of these projects may vary when compared with traditional Java projects. We also excluded projects written in other languages rather than English because JReuse performs a lexical analysis of systems and we target only on project written in English. Fourth, we excluded projects with less than 20 classes and 20 methods, because we intend to compose a data set with sufficient number of domain experts for analysis. We also excluded projects with less than an year of life for the same reason. Finally, from the remaining projects, we selected the top-three committers for each project to collect their valid email addresses. We use these emails to invite developers for the survey.

A total of 202 email addresses from domain experts were extracted from the 198 different projects we collected. We sent a specific questionnaire to participants for each system domain. A total of 31 questionnaires, i.e., around 15.34%, were responded. Table 5.2 presents the number of participants who answered the questionnaire per domain. We only got one answer for the Accounting domain and none for Restaurant. Since few participants responded the accounting and restaurant questionnaires, we discarded both domains in the analysis presented in this section. Therefore, our analysis is based on 31 answer, 9 for Hospital and 22 for e-commerce.

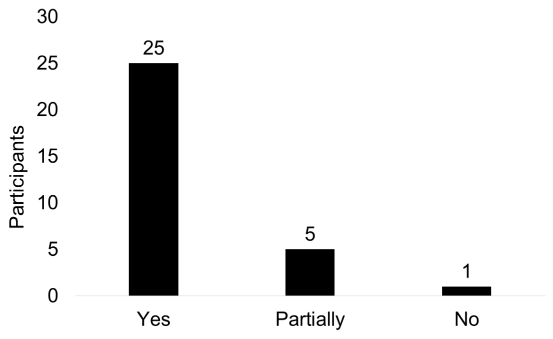
Table 5.2. Population Sampling Distribution

Domain	**Projects**	**Emails Sent**	**Answer**
Accounting	11	11	1
Restaurant	10	13	0
Hospital	56	111	9
E-commerce	121	67	22
Total	198	202	32

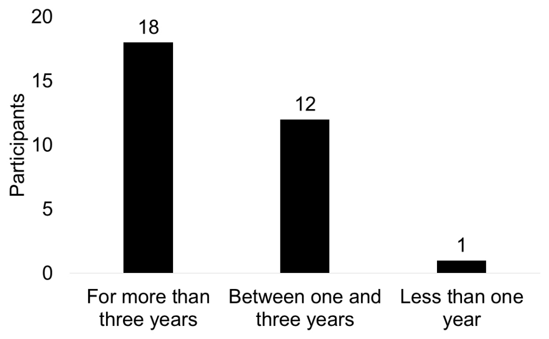
5.2 Participant Background

In this section, we discuss the background information collected from Q1 to Q4. Figure 5.1 presents the background of participants for analysis of the e-commerce and hospital domain (Q1). Figure 5.1(a) shows that 25 of the 31 participants (around 80%) work in the context of software development. Since more than a half of the participants are software developers, we assume that the sample is appropriate to evaluate our method. Figure 5.1(b) shows the results regarding professional experience of participants (Q2). Note that 18 of them (around 58%) have more than three years of professional experience, and 12 (around 38%) have between one and three year of experience. Only one out 31 participants has less than one year of the work experience.

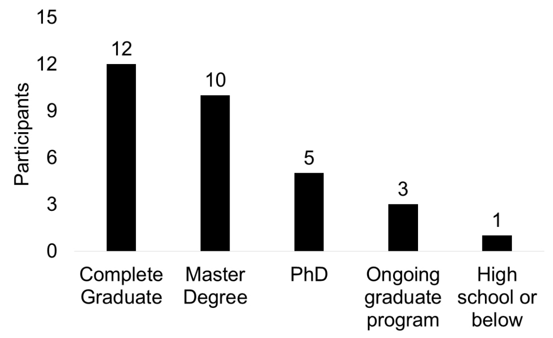
With respect to the education level in Computer Science (Q3), Figure 5.1(c) shows that 27 participants, i.e., around 87%, hold at least a complete graduate degree in the area. Therefore, we conclude that the participants are appropriate to our analysis. Figure 5.1(d) presents the results with respect to the development of product for the respective domain (Q4). In total 26 participants (around 83%) develop software for the domains analyzed and 5 (around 16%) participants do not develop software for the respective domain. However, all 31 participants were selected because they frequently commit in projects of the analyzed domains. Therefore, we assume that the participants are able to evaluate the relevance of entities for the given domain.



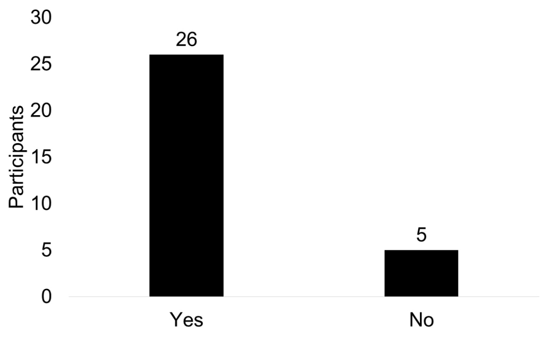
(a) Work in software development



(b) Experience in software development



(c) Knowledge in computer sciences



(d) Development of product for the domain

Figure 5.1. Background of Participants

5.3 Results

This section presents the main results of the survey, with respect to questions Q5 to Q15. These questions are related to the level of relevance for classes, identified by JReuse as reuse opportunities, to a given domain. Note that we considered only the top-ten most frequent classes reported by our method for each domain and a control class (`Game`). Considering all participants, we computed the mean of relevance level for each class from Q5 to Q16. Based on the mean for the classes, we chose 3 as a thresholds to classify a class as relevant ($mean \geq 3$) or irrelevant ($mean < 3$) for the respective domain. After, we computed the number of classes classified as relevant.

Figure 5.2 presents the percentage of classes considered as relevant by the participants per domain. We observe that 90% of the classes identified by JReuse as reuse opportunities are relevant for the e-commerce domain from the participants' viewpoint. On the other hand, 10% of the classes were not indicated as a relevant reuse opportunity. For example, for the e-commerce domain, the only class indicated by the method as reuse opportunity, but not indicated as relevant by domain experts is `User`. In fact, this class is a generic entity for software systems. That is, it may compose systems from several domains.

Figure 5.2 also shows the results for the hospital domain. In this analysis, we observe that the participants agree that 80% of the classes indicated by JReuse are relevant for the respective domain. However, 20% of classes were not indicated as relevant by the participants. The class `Login` presented a mean of 2.34 and `User` presented a men of 2.23 for relevance level. In other words, we conclude that, according to the results presented by the participants, these two classes are generic and may not represent reuse opportunities for the analyzed domain.

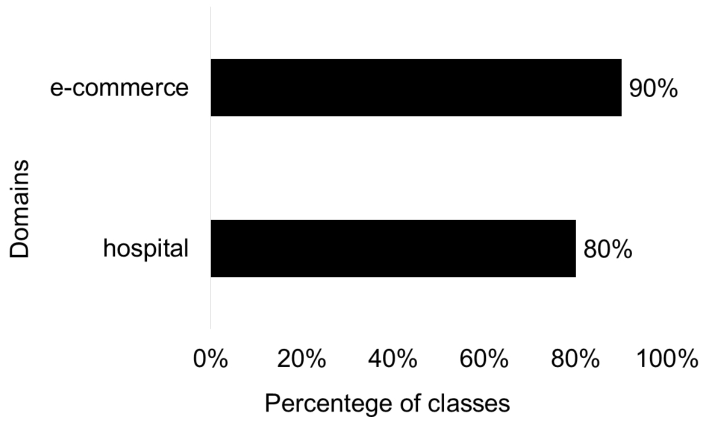


Figure 5.2. Accuracy of the method JReuse compared with participants

In summary, participants of the survey agree that 90% and 80% of the classes from the e-commerce and hospital domains are relevant, respectively. Therefore, our data suggests that JReuse is effective and accurate in the identification of reuse opportunities. Since the variation of percentage for both domains is minimum, i.e., 10% to 20%, we assume that JReuse provides sufficient results regardless the analyzed domain.

5.4 Threats to Validity

With respect to the survey with software developers, we conducted a careful study to assess the relevance of reuse opportunities identified by JReuse. However, there are some threats to validity that may invalidate our findings. We discuss each type of threat to validity of our study, based on Wohlin et al. (2012), as follows.

Construct Validity. In order to compose our participant set, we selected emails of software developers from 404 different Java projects. To provide diversity of the participants of the survey, we selected developers from projects based on the number of stars in GitHub. Although our participant set may not be representative, the 178 selected systems are developed by several contributors, from different domains, and provide distinct functionalities. Furthermore, in this preliminary study, we do not assess the effectiveness of JReuse in terms of methods identified as reuse opportunities. This decision was taken to prevent a high number of questions and the increase of complexity of the survey. Thus, we designed a short survey that participants are motivated to answer. However, we performed a careful analysis of the classes.

Internal Validity. Since our survey was available during a short time (from June to July 2016, specifically), we obtained a small number of participants. The collected results may be, then, insufficient to draw precise conclusion regarding the effectiveness of JReuse. However, we sent invitations for a significant number of participants. In addition, we invited the top-three contributors in terms of commits for each project. Such treatments aim to minimize problems with the lack of availability for developers to participate in the survey. Furthermore, to minimize problems with data collection, we designed an online questionnaire to automatically collect the participants' answers.

Conclusion Validity. With respect to the data analysis, we computed the mean of relevance level reported by participants for each class, per domain. We then computed the percentage of classes considered as relevant by the participants. We defined a thresholds to classify a class as relevant or irrelevant according to the subjective opinion of the author. Therefore, this threshold may not be applied in other contexts or considering more systems. However, to minimize this threat, we analyzed a significant number of participants (i.e., 31 participants) and the top-ten most frequent classes per domain.

External Validity. Regarding the study generalization, we present some relevant issues. First, although most participants have at least one year of professional experience, our participant set may not represent the real context of software development. However, the majority of participants has three or more years of experience. Such participants are developers from different organizations. Second, from the four domains analyzed, we received a significant number of responses only for two domain: e-commerce and hospital. Therefore, our results may not be generalized to other domains.

5.5 Final Remarks

This chapter describes a survey with 31 software developers of two domains: e-commerce and hospital. This survey aims to assess the effectiveness of JReuse in identifying relevant reuse opportunities given a domain. As a preliminary study, we assess only the identification of classes by the proposed method. We present in detail the survey settings, including the process of selection of participants and the questionnaire for participants to respond. A total of 31 participants answered the questionnaire, 22 for the e-commerce domain and 9 for the hospital domain.

Our study provided a positive results regarding the effectiveness of JReuse. We observe that participants agree with 80% to 90% of the classes identified by our method as reuse opportunities for the hospital and e-commerce domain. Therefore, our data suggest that JReuse is able to effectively identify reuse opportunities with respect to classes for different domains.

Finally, the chapter discusses the main threats to the validity of our study. These threats include the the selection of appropriate participants and classes for analysis, and also the generalization of our study findings. Chapter 6 presents the main conclusion of this dissertation. The provided discussion encompasses both the empirical study in controlled environment (Chapter 4) and the survey described in this chapter. The next chapter also suggests future work.

Chapter 6

Conclusion

In this dissertation, we proposed JReuse, a method to identify reuse opportunities in object-oriented software systems given a domain. We also presented a prototype tool that implements the proposed method. JReuse aims to recommend software components for reuse based on the most frequent entities from a set of systems in a given domain.

We evaluated our method in two steps. First, we performed a empirical study conducted in controlled environment. This study was conducted with a total of 72 software systems from four domains: accounting, restaurant, hospital, and e-commerce. We collected all systems from GitHub. Second, we performed a survey with 31 domain experts in two of the four domains: e-commerce and hospital.

With respect to the first evaluation, our findings suggest that JReuse is able to identify several reuse opportunities for the analyzed domains, independent of the analyzed domain and the size of the systems. The number of obtained results is significant. Regarding the second evaluation, we observe based on the results of a survey that the most frequent classes provided by JReuse are relevant for the two domains under analysis.

We organized the remainder of this chapter as follows. Section 6.1 summarizes the contributions of our study. Finally, Section 6.2 suggests future work.

6.1 Contributions

As a result of the work presented in this dissertation, we highlight the following contributions.

- JReuse, a method to support the identification of reuse opportunities in software systems from a given domain. JReuse is based on lexical similarity analysis of names of classes and methods, and relies on the analysis of object-oriented software systems.
- A supporting tool that implements the proposed method for identification of reuse opportunities. This tool is compatible with Java projects. The tool provides an output with the classes and methods identified as reuse opportunities, as well as absolute paths of the classes to be accessed in the source system.
- An evaluation of JReuse in two steps. First, we conducted an empirical study with 72 systems from four different domains (namely, accounting, restaurant, hospital, and e-commerce), collected from GitHub. Second, we conducted a survey with 31 domain experts from GitHub for two domains.

6.2 Future Work

We intend to complement this research with the following future work.

- With respect to the proposed method, we aim to apply other lexical analysis techniques to identify reuse opportunities. We may also implement a hybrid analysis that combines lexical and semantic techniques for identification and recommendation of source code statements as reuse opportunities.
- With respect to the JReuse supporting tool, we suggest an evaluation of the graphical user interface in terms of usability. Moreover, we intend to assess the performance of tool with respect to scalability with larger systems.
- Regarding the evaluation of JReuse, we may extend the survey with domain experts to assess the method recommended by JReuse as reuse opportunities. In addition, we suggest a comparison of the effectiveness of JReuse to the manual identification of reuse opportunities conducted by developers.
- We suggest identify reuse opportunities in other programming languages. In addition, we wish evaluate the feasibility in identify reuse opportunities when there is a well-defined programming pattern in same enterprise systems.

Bibliography

Ajila, S. A., Gakhar, A. S., Lung, C. H., and Zaman, M. (2012). Reusing and converting code clones to aspects - an algorithmic approach. In *Proceedings of the Information Reuse and Integration (IRI)*, pages 9–16.

Begel, A., Khoo, Y. P., and Zimmermann, T. (2010). Codebook: discovering and exploiting relationships in software repositories. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 125–134.

Caldiera, G. and Basili, V. R. (1991). Identifying and qualifying reusable software components. *In Journal Computer - Special Issue on Cryptography*, pages 61–70.

Cubranic, D. and Murphy, G. C. (2003). Hipikat: recommending pertinent software development artifacts. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 408–418.

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

Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. (2008). *Selecting Empirical Methods for Software Engineering Research*, pages 285–311. Springer London.

Fluri, B., Wuersch, M., PInzger, M., and Gall, H. (2007). Change distilling: Tree differencing for fine-grained source code change extraction. *In IEEE Transactions on Software Engineering (TSE)*, pages 725–743.

Fowler, M. (2009). *Refactoring: improving the design of existing code*. Pearson Education India.

Guo, J. and Luqi (2000). A survey of software reuse repositories. In *Proceedings of the International Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*, pages 92–100.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *In ACM Transactions on Information Systems (TOIS)*, pages 5–53.

Holmes, R., Walker, R. J., and Murphy, G. C. (2006). Approximate structural context matching: An approach to recommend relevant examples. *In IEEE Transactions on Software Engineering (TSE)*, pages 952–970.

Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M., and Kusumoto, S. (2005). Ranking significance of software components based on use relations. *In IEEE Transactions on Software Engineering (TSE)*, pages 213–225.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2014). The promises and perils of mining github. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, pages 92–101.

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

Ko, A. J., Myers, B. A., Coblenz, M. J., and Aung, H. H. (2006). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *In IEEE Transactions on Software Engineering (TSE)*, pages 971–987.

Koziolek, H., Goldschmidt, T., de Gooijer, T., Domis, D., and Sehestedt, S. (2013). Experiences from identifying software reuse opportunities by domain analysis. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 208–217.

Krueger, C. (1992). Software reuse. *In Journal of Computing Surveys (CSUR)*, pages 131–183.

Kuhn, A., Ducasse, S., and Gírba, T. (2007). Semantic clustering: Identifying topics in source code. *In Journal of Information and Software Technology*, pages 230–243.

Kukich, K. (1992). Techniques for automatically correcting words in text. *In Journal of Computing Surveys (CSUR)*, pages 377–439.

Lee, J., Kang, K. C., and Kim, S. (2004). A feature-based approach to product line production planning. In *Proceedings of the International Conference on Software Product Lines (SPLC)*, pages 183–196.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, page 707.

Li, J., Zhang, Z., and Yang, H. (2005). A grid oriented approach to reusing legacy code in iceni framework. In *Proceedings of the International Conference on Information Reuse and Integration (IRI)*, pages 464–469.

Li, L., Martinez, J., Ziadi, T., Bissyande, T. F. D. A., Klein, J., and Le Traon, Y. (2016). Mining families of android applications for extractive spl adoption. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 455–460.

Li, Z., Lu, S., Myagmar, S., and Zhou, Y. (2006). Cp-miner: Finding copy-paste and related bugs in large-scale software code. *In IEEE Transactions on software Engineering*, pages 176–192.

Liu, H. and Lu, R. (2008). 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.

Maarek, Y., Berry, D., and Kaiser, G. (1991). An information retrieval approach for automatically constructing software libraries. *In IEEE Transactions on Software Engineering (TSE)*, pages 800–813.

Marcus, A. and Maletic, J. I. (2001). Identification of high-level concept clones in source code. In *Proceedings of the Annual International Conference on Automated Software Engineering, (ASE)*, pages 107–114.

Mende, T., Koschke, R., and Beckwermert, F. (2009). An evaluation of code similarity identification for the grow-and-prune model. *In Journal of Software Maintenance and Evolution: Research and Practice*, pages 143–169.

Michail, A. and Notkin, D. (1999). Assessing software libraries by browsing similar classes, functions and relationships. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 463–472.

Mohagheghi, P. and Conradi, R. (2007). Quality, productivity and economic benefits of software reuse: a review of industrial studies. *In Journal Empirical Software Engineering (ESE)*, pages 471–516.

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

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

Morisio, M., Ezran, M., and Tully, C. (2002). Success and failure factors in software reuse. *In IEEE Transactions on Software Engineering (TSE)*, pages 340–357.

Navarro, G. (2001). A guided tour to approximate string matching. *In Journal Computing Surveys (CSUR)*, pages 31–88.

Neighbors, J. (1992). The evolution from software components to domain analysis. *In International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, pages 325–354.

Oliveira, J., Fernandes, E., Souza, M., and Figueiredo, E. (2016). A method based on naming similarity to identify reuse opportunities. In *Proceedings of the XII Brazilian Symposium on Information Systems (SBSI)*, pages 305–312.

Oliveira, J. and Figueiredo, E. (2016). A recommendation system of reuse opportunities based on lexical analysis. In *Proceedings of the IX Workshop Thesis and Dissertations in Information Systems (WTDSI)*, pages 49–51.

Oliveira, J. A., Fernandes, E. M., and Figueiredo, E. (2015). Evaluation of duplicated code detection tools in cross-project context. In *Proceedings of the III Workshop on Software Visualization, Evolution and Maintenance (VEM)*, pages 49–56.

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

Patil, R. V., Joshi, S. D., Shinde, S. V., Ajagekar, D. A., and Bankar, S. D. (2015). Code clone detection using decentralized architecture and code reduction. In *Proceedings of the International Conference on Pervasive Computing (ICPC)*, pages 1–6.

Pohl, K., Böckle, G., and van Der Linden, F. J. (2005). *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media.

Pressman, R. S. (2005). *Software Engineering: a Practitioner's Approach.* Palgrave Macmillan.

Ravichandran, T. and Rothenberger, M. (2003). Software reuse strategies and component markets. *In Magazine Communications of the ACM*, pages 109–114.

Robillard, M., Walker, R., and Zimmermann, T. (2010). Recommendation systems for software engineering. *In Journal IEEE Software*, pages 80–86.

Rountev, A. (2004). Precise identification of side-effect-free methods in java. In *Proceedings of the International Conference on Software Maintenance (ICSE)*, pages 82–91.

Roy, C. K. and Cordy, J. R. (2008). Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proceeding of the International Conference on Program Comprehension (ICPC)*, pages 172–181.

Salvaneschi, G., Amann, S., Proksch, S., and Mezini, M. (2014). An empirical study on program comprehension with reactive programming. In *Proceedings of the International Symposium on Foundations of Software Engineering (SIGSOFT)*, pages 564–575.

Selim, G. M., Foo, K. C., and Zou, Y. (2010). Enhancing source-based clone detection using intermediate representation. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, pages 227–236.

Sojer, M. and Henkel, J. (2011). License risks from ad hoc reuse of code from the internet. *In Journal Communications of the ACM*, pages 74–81.

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

Wang, X., Dang, Y., Zhang, L., Zhang, D., Lan, E., and Mei, H. (2012). Can i clone this piece of code here? In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 170–179.

Wang, Z., Xu, X., and Zhan, D. (2005). A survey of business component identification methods and related techniques. *In International Journal of Information Technology*, pages 229–238.

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

Xue, Y. (2011). Reengineering legacy software products into software product line based on automatic variability analysis. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 1114–1117.

Ye, Y. and Fischer, G. (2002). Information delivery in support of learning reusable software components on demand. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*, pages 159–166.

Ye, Y. and Fischer, G. (2005). Reuse-conducive development environments. *In Journal Automated Software Engineering (ASE)*, pages 199–235.

Yuan, Y. and Guo, Y. (2012). Boreas: an accurate and scalable token-based approach to code clone detection. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 286–289.

Yujian, L. and Bo, L. (2007). A normalized levenshtein distance metric. *In IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pages 1091–1095.

Zhen, Z., Shen, J., and Lu, S. (2008). 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.