ARTHUR FERREIRA PINTO

EMPIRICALLY SUPPORTED SIMILARITY COEFFICIENTS FOR THE IDENTIFICATION OF REFACTORING OPPORTUNITIES

LAVRAS – MG

2018

ARTHUR FERREIRA PINTO

EMPIRICALLY SUPPORTED SIMILARITY COEFFICIENTS FOR THE IDENTIFICATION OF REFACTORING OPPORTUNITIES

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Engenharia de Software, para a obtenção do título de Mestre.

Prof. Dr. Ricardo Terra Nunes Bueno Villela
Orientador

LAVRAS – MG
2018

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Pinto, Arthur Ferreira
Empirically supported similarity coefficients for the identification of refactoring opportunities / Arthur Ferreira Pinto. – Lavras : UFLA, 2018.
75 p. : il.

Dissertação (mestrado acadêmico)–Universidade Federal de Lavras, 2018.
Orientador: Prof. Dr. Ricardo Terra Nunes Bueno Villela.
Bibliografia.

1. Code Refactoring. 2. Structural Similarity. 3. Software Architecture. I. Terra, Ricardo. II. Título.

ARTHUR FERREIRA PINTO

EMPIRICALLY SUPPORTED SIMILARITY COEFFICIENTS FOR THE IDENTIFICATION OF REFACTORING OPPORTUNITIES

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Engenharia de Software, para a obtenção do título de Mestre.

APROVADA em 7 de maio de 2018.

Prof. Dr. Marco Túlio de Oliveira Valente UFMG
Prof. Dr. Antônio Maria Pereira de Resende UFLA

Prof. Dr. Ricardo Terra Nunes Bueno Villela
Orientador

LAVRAS – MG
2018

This dissertation is dedicated to my parents Geisa and Eraldo, who have always been supporting me.

ACKNOWLEDGMENTS

Firstly, I would like to thank Federal University of Lavras, specially the Department of Computer Science, for the opportunity to obtain my master's degree.

I would also like to express my sincere gratitude to my advisor Dr. Ricardo Terra for the continuous support of my master study and related research, for his patience, motivation, and immense knowledge. His guidance helped me during these two years of researching. I could not have imagined having a better advisor and mentor for my master study.

Besides my advisor, I would like to thank the rest of my dissertation committee—Dr. Marco Túlio de Oliveira Valente and Dr. Antônio Maria Pereira de Resende—for the careful reading of this text and their insightful comments and constructive suggestions.

My sincere thanks also goes to my fellow labmates—specially to my friend Christian Couto—for having always supported me in my academic activities.

I would like to thank my family—specially my parents and brother—for all the help they gave me making this study possible and supporting me spiritually throughout writing this dissertation and in my life in general.

Last but not the least, I would like to thank God, for always guiding me, for the given opportunities and the inspiration to seek knowledge and know more about the world and about myself.

"The scientist is not a person who gives the right answers,
he’s one who asks the right questions."
(Claude Lévi-Strauss)

// RESUMO

Refatoração de código é definido como o processo de alteração de um sistema de software, preservando seu comportamento externo, mas melhorando sua estrutura interna. Por meio da refatoração, torna-se possível tratar os sintomas da arquitetura de código, conhecidos como *Code Smells*, que podem afetar características como portabilidade, reusabilidade, manutenibilidade e escalabilidade. Várias técnicas para identificar oportunidades de refatoração usam coeficientes de similaridade para encontrar entidades mal posicionadas na arquitetura do sistema, assim como determinar onde deveria estar localizada. Como exemplo, espera-se que um método esteja localizado em uma classe cujos outros métodos sejam estruturalmente semelhantes a ele. No entanto, os coeficientes existentes na literatura não foram projetados para a análise estrutural de sistemas de software, o que pode não garantir uma precisão satisfatória.

Portanto, esta dissertação de mestrado propõe três novos coeficientes – $PTMC$, $PTMM$ e $PTEM$ – para melhorar a precisão da identificação de oportunidades de refatoração para as operações de *Move Class*, *Move Method* e *Extract Method*, respectivamente. Os principais objetivos são: (i) propor coeficientes de similaridade mais efetivos para sistemas orientados a objetos, para localizar com mais precisão entidades mal posicionadas em uma arquitetura de sistema e (ii) alavancar a precisão de ferramentas para identificação de oportunidades de refatoração baseadas na similaridade estrutural por meio da aplicação dos coeficientes propostos.

Primeiramente, foi investigada a precisão de 18 coeficientes de similaridade em 10 sistemas da base *Qualitas.class Corpus* (*training set*) e, com base nos resultados, foi selecionado o coeficiente mais adequado para ser adaptado. Em seguida, foi adaptado o coeficiente selecionado por meio de um experimento empírico composto de uma combinação de tratamentos com replicação sobre algoritmos genéticos para gerar os coeficientes propostos. Por fim, foi implementada *AIRP*, uma ferramenta que implementa os coeficientes propostos para identificar oportunidades de refatoração.

Para avaliar os coeficientes propostos, foram comparados tais coeficientes com outros 18 coeficientes em outros 101 sistemas da base *Qualitas.class Corpus* (*test set*). Os resultados indicam, em relação ao melhor coeficiente analisado, uma melhora estatística de 5,23% a 6,81% para a identificação de oportunidades de refatoração *Move Class*, 12,33% a 14,79% para *Move Method* e 0,25 % a 0,40% para *Extract Method*.

Palavras-chave: Arquitetura de Software. Similaridade Estrutural. Refatoração de Código. Move Class. Move Method. Extract Method.

// ABSTRACT

Code refactoring is defined as the process of changing a software system preserving the external behavior of the code, but improving its internal structure. Through refactoring, it becomes possible to treat code architecture symptoms, known as Code Smells, which can affect features such as portability, reusability, maintainability, and scalability. Several techniques to identify refactoring opportunities rely on similarity coefficients to find misplaced entities on the system architecture, as well as to determine where it should be located. As an example, we expect that a method is located in a class whose other methods are structurally similar to it. However, the existing coefficients in literature have not been designed for the structural analysis of software systems, which may not guarantee satisfactory precision.

This master dissertation, therefore, proposes three new coefficients—*PTMC*, *PTMM*, and *PTEM*—to improve the precision of the identification of *Move Class*, *Move Method*, and *Extract Method* refactoring opportunities, respectively. Our main objectives are: (i) to propose more effective similarity coefficients for object-oriented systems, in order to locate more accurately entities improperly positioned on a system architecture and (ii) to leverage the precision of tools for identification of refactoring opportunities based on structural similarity through the application of the proposed coefficients.

Firstly, we investigated the precision of 18 similarity coefficients in 10 systems of *Qualitas.class Corpus* (training set) to select the most appropriate coefficient to be adapted. Then, we adapted the selected coefficient through an empirical experiment based on a treatment combination with replication over genetic algorithms in order to generate the proposed coefficients. Finally, we implemented *AIRP*, a tool that relies on the proposed coefficients to identify refactoring opportunities.

In order to evaluate the proposed coefficients, we compared them with other 18 coefficients in other 101 systems of *Qualitas.class Corpus* (test set). The results indicate, in relation to the best analyzed coefficient, a statistical improvement from 5.23% to 6.81% for the identification of *Move Class* refactoring opportunities, 12.33% to 14.79% for *Move Method*, and 0.25% to 0.40% for *Extract Method*.

Keywords: Software Architecture. Structural Similarity. Code Refactoring. Move Class. Move Method. Extract Method.

LIST OF FIGURES

Figure 1.1 – *Move Class* refactoring example 12
Figure 1.2 – Proposal of new coefficients 14
Figure 2.1 – *Move Class* refactoring 20
Figure 2.2 – *Move Method* refactoring 21
Figure 2.3 – *Extract Method* refactoring 21
Figure 2.4 – Illustrative example 28
Figure 3.1 – Methodology overview 32
Figure 3.2 – Refactorings plots 39
Figure 4.1 – *Move Class* plot 45
Figure 4.2 – *Move Method* plot 46
Figure 4.3 – *Extract Method* plot 46
Figure 5.1 – Refactoring opportunity graph 49

LIST OF TABLES

Table 2.1 – Relationship between the treatment of Code Smells by the code refactorings 19
Table 2.2 – Similarity coefficients 22
Table 2.3 – Genetic algorithms configuration 29
Table 2.4 – Comparison between *Jaccard* and the resulting coefficient 29
Table 3.1 – Target-systems 36
Table 3.2 – Precision of the 18 similarity coefficients (training set) 38
Table 3.3 – Experiment groups of factors' values 40
Table 3.4 – Experiment's objective function results 41
Table 4.1 – Precision of the 19 similarity coefficients (test set) 44
Table 1 – Precision rates of the 18 coefficients in 10 systems (training set) 63
Table 2 – Precision rates of the 19 coefficients in 101 systems (test set) 65

CONTENTS

1 INTRODUCTION . . . 11
1.1 Problem . . . 11
1.2 Objectives . . . 12
1.3 The Proposal of New Coefficients . . . 13
1.4 Outline of the Dissertation . . . 13
1.5 Publications . . . 15
2 BACKGROUND . . . 16
2.1 Source Code Entities . . . 16
2.2 Code Smells . . . 17
2.3 Refactoring . . . 18
2.3.1 ***Move Class*** . . . 19
2.3.2 ***Move Method*** . . . 20
2.3.3 ***Extract Method*** . . . 21
2.4 Structural Similarity . . . 22
2.5 Optimization Algorithms . . . 25
2.5.1 Genetic Algorithms . . . 25
2.5.2 Illustrative Example . . . 28
3 EMPIRICALLY SUPPORTED SIMILARITY COEFFICIENTS . . . 31
3.1 Methodology . . . 31
3.1.1 Calculation of Similarity Coefficient's Precision . . . 31
3.2 Analysis and Comparison of Existing Coefficients . . . 36
3.3 Empirical Experiment . . . 37
4 EVALUATION . . . 43
4.1 Results . . . 43
4.2 Move Class . . . 43
4.3 Move Method . . . 45
4.4 Extract Method . . . 45
4.5 Threats to Validity . . . 46
5 RECOMMENDATION SYSTEM . . . 47
6 RELATED WORK . . . 50
6.1 Similarity Coefficients . . . 50

6.2 Empirical Studies 51
6.3 Systematic Reviews of Literature 53
6.4 Identification of Refactoring Opportunities 53
6.4.1 Approaches Based on Similarity Coefficients 54
6.4.2 Approaches Based on Search-Based Techniques 55
6.4.3 Approaches Based on Other Techniques 56
7 CONCLUSION 58
7.1 Contributions 58
7.2 Limitations 59
7.3 Future Work 59
REFERENCES 61
APPENDIX A – Coefficient Precision in Training Set 63
APPENDIX B – Coefficient Precision in Test Set 65

1 INTRODUCTION

This chapter is organized as it follows. Section 1.1 provides an overview of the problem addressed by this master dissertation. Section 1.2 presents the main objectives of this work. Section 1.3 provides an overview of the proposal of new coefficients. Section 1.4 shows the structure of this master dissertation, including the organization of its chapters. Finally, Section 1.5 presents the publications generated by this work.

1.1 Problem

During software development, code entities might be incorrectly positioned in the system's architecture, which might affect factors such as maintenance effort (BINKLEY; SCHACH, 1998), productivity, and design effort (CHIDAMBER; DARCY; KEMERER, 1998). In this context, code refactoring is essential to achieve a well-defined architecture in the context of object-oriented programming since it allows the correct repositioning of such entities. However, although there are several techniques that can be applied to identify refactoring opportunities, none of them can guarantee a full precision, i.e, the correct identification of where a respective entity should be located.

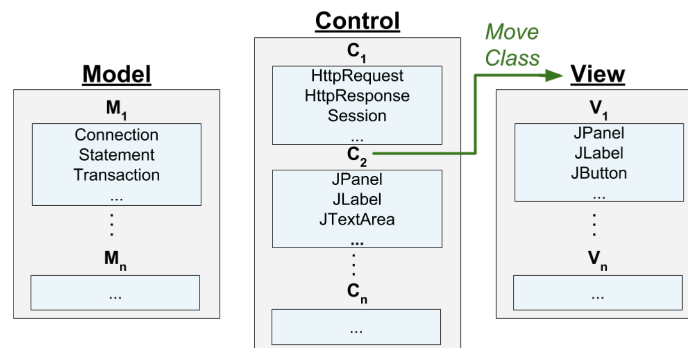
Furthermore, code refactoring is one of the most used practices to treat Code Smells, i.e., any symptoms in the code that might indicate problems in the software architecture (FOWLER et al., 1999). Such Code Smells, frequently imply in the establishment of unnecessary dependencies. However, considering that ensuring architectural design is extremely important for maintainability, reusability, scalability, and portability of software systems (PASSOS et al., 2010), it is expected that the code entities of a software project be refactored in order to reposition them in structurally similar entities.

One of the techniques used to identify refactoring opportunities is the analysis of the structural similarity among code entities, i.e., to verify the similarity of dependencies among its entities (whether in the level of package, class, method, or block), and then performing refactorings when necessary. This implies moving methods and classes, and in the extraction of a code block from a method (generating a new method), through refactorings like *Move Class*, *Move Method*, and *Extract Method* (FOWLER et al., 1999).

Figure 1.1 shows an example of a system that follows the MVC architecture, consisting of three layers: $Model$, $View$, and $Controller$. It is possible to note that C_2 is badly located in the $Controller$ layer since it depends on graphical elements while other classes of the layer have de-

pendencies to manipulation elements of web requests (e.g., `HttpRequest` and `HttpResponse`). Therefore, it is possible to suggest moving such class (*Move Class*) to a structurally similar layer that, in this scenario, would be the *View* layer. Such structural similarity occurs because C_2 and classes from the *View* layer ($V_1, \ldots, V_n$) have dependencies on common graphical elements (e.g., `JPanel` and `JLabel`). Thus, refactoring would not only guarantee an architecture with properly located entities, but would also respect the MVC architecture.

Figure 1.1 – *Move Class* refactoring example



Literature provides several coefficients for the calculation of similarity among code entities. However, the use of such coefficients may not guarantee satisfactory precision since they have not been designed for the structural analysis of a software system. For instance, the *Jaccard* coefficient, one of the most used in Software Engineering, was initially designed to compare the similarity among flower species in different districts (JACCARD, 1912).

1.2 Objectives

This master dissertation aims primarily to propose new similarity coefficients focused in object-oriented systems. More precisely, our intention is to propose coefficients for a more accurate identification of *Move Class*, *Move Method*, and *Extract Method* refactoring opportunities. In this context, our objectives are: (i) to propose more effective similarity coefficients for object-oriented systems, in order to locate more accurately entities improperly positioned on a system architecture and (ii) to leverage the precision of tools for identification of refactoring opportunities based on structural similarity through the application of the proposed coefficients. It is important to emphasize, however, that the new coefficients are specific to these three refactorings since they are the most widely used by developers (SILVA; TSANTALIS; VALENTE, 2016).

In addition, this master dissertation aims to provide an analysis and comparison about the precision of the main coefficients in literature for each of the three covered types of refac-

toring. Moreover, our experiment design might be reused by other researches who could benefit from an empirical experiment based on optimization techniques. Finally, it is intended to present a tool that applies the proposed similarity coefficients to automatically identify refactoring opportunities.

1.3 The Proposal of New Coefficients

In order to propose new similarity coefficients, as mentioned in the previous section, we (i) analyzed the main coefficients in literature and selected one for adaptation, (ii) conducted an empirical experiment for the respective adaptation, (iii) developed a tool to identify refactoring opportunities based on the proposed coefficients, and (iv) evaluated the resulting coefficients in real-world systems. Figure 1.2 illustrates this approach.

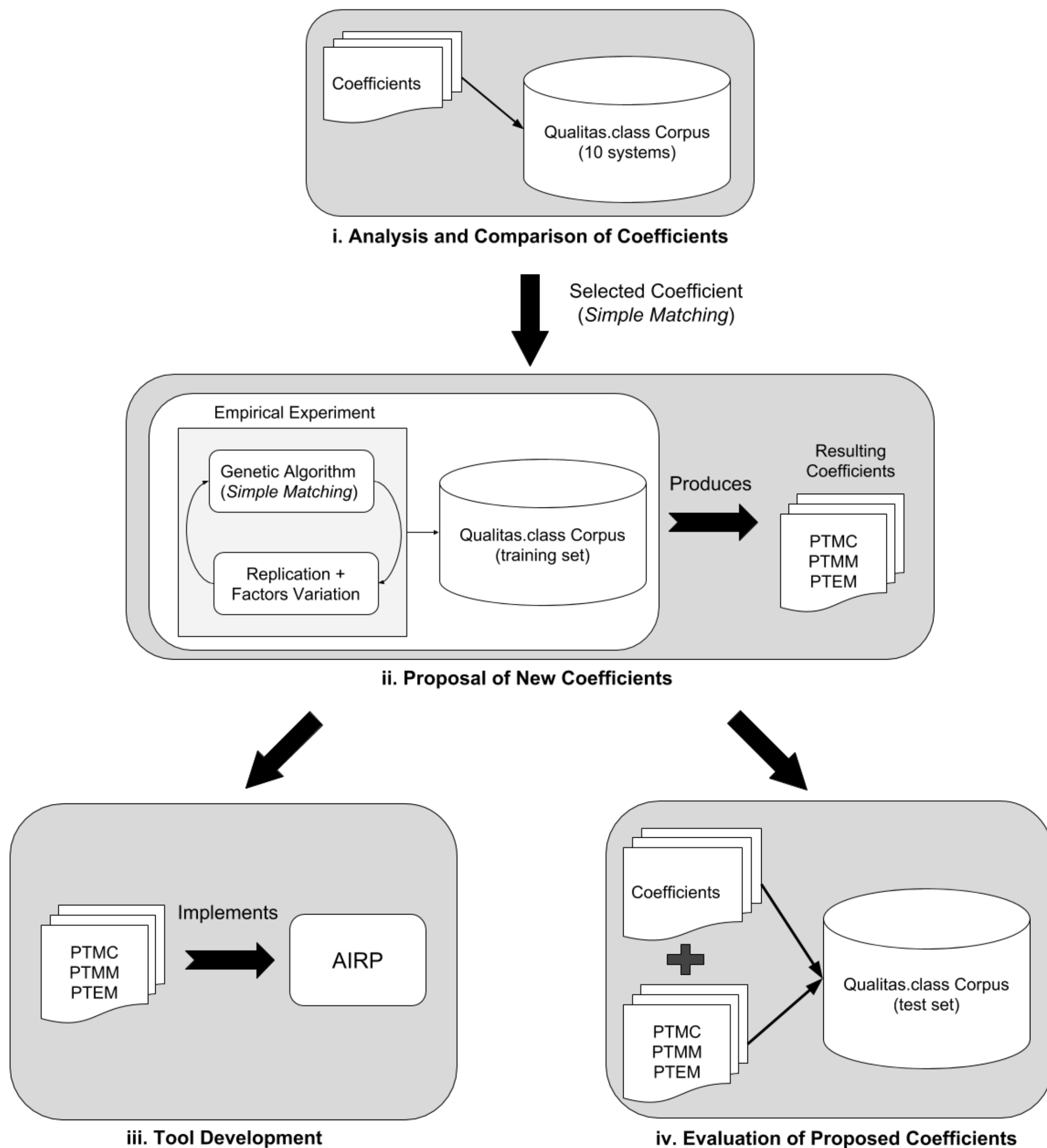
Firstly, we investigated the precision of 18 similarity coefficients in 10 systems (training set) of *Qualitas.class Corpus* (TERRA et al., 2013b). Secondly, we adapted the *Simple Matching* coefficient through an empirical experiment based on a treatment combination with replication over genetic algorithms in order to generate three new coefficients ($PTMC$, $PTMM$, and $PTEM$) with greater precision in the identification of *Move Class*, *Move Method*, and *Extract Method* refactoring opportunities, respectively. Thirdly, we implemented AIRP, a tool that identifies refactoring opportunities based on the proposed coefficients. Finally, we compared the proposed coefficients with those existent in other 101 systems. The results indicate, in relation to the best analyzed coefficient, a statistically significant improvement from 5.23% to 6.81% for identification of *Move Class* opportunities, 12.33% to 14.79% for *Move Method*, and 0.25% to 0.40% for *Extract Method*.

1.4 Outline of the Dissertation

We organized the remainder of this master dissertation as follows:

- Chapter 2 provides a background that introduces fundamental concepts to this dissertation. It presents the source code entities and the Code Smells covered by our methodology and evaluation, as well as an introduction to *Move Class*, *Move Method*, and *Extract Method* refactoring operations. It also presents the concept of structural similarity and the main coefficients in literature. Finally, it provides an explanation of the optimization algorithm used in this dissertation, including an illustrative example of its application.

Figure 1.2 – Proposal of new coefficients



- Chapter 3 proposes new empirically supported similarity coefficients. It describes the used methodology and the applied process of similarity calculation. It also provides an analysis of existing coefficients, with the objective of selecting the coefficient to be adapted. Finally, after selecting the coefficient, we describe the empirical experiment that propose three new coefficients for identifying refactoring opportunities.

- Chapter 4 reports the evaluation of the proposed coefficients in 101 systems. More precisely, this chapter provides the analysis and comparison of the precision of the main coefficients in literature with the new proposed coefficients for *Move Class*, *Move Method*, and *Extract Method* refactorings.

- Chapter 5 describes the implementation and illustrates the usage of *AIRP*, a tool to identify refactoring opportunities based on the proposed coefficients.

- Chapter 6 discusses related studies. It covers different types of studies, such as: (i) proposal of structural similarity coefficients, (ii) empirical studies, (iii) systematic reviews of literature, and (iv) different approaches for identifying refactoring opportunities.

- Chapter 7 presents the final considerations of this master dissertation, including the contributions, limitations, and future work.

1.5 Publications

This dissertation generated the following publications and therefore contains material from them:

- Arthur F. Pinto and Ricardo Terra. Better Similarity Coefficients to Identify Refactoring Opportunities. In *11th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS)*, pages 1-10, 2017.

- Arthur F. Pinto and Ricardo Terra. Empirically supported similarity coefficients for the identification of refactoring opportunities. In *Software Quality Journal (SQJ)*, pages 1-26, 2018. (*Submitted*)

2 BACKGROUND

In order to provide the necessary knowledge for the conception and understanding of this master dissertation, this chapter introduces the fundamental concepts. Section 2.1 describes the definition of the covered source code entities. Section 2.2 introduces the main concepts of Code Smells, in addition to describing the six types covered in this work. Section 2.3 deals with the refactoring process, which involves methods for source code restructuring. Section 2.4 deals with structural similarity, as also presents the current main coefficients in the literature. Section 2.5 introduces optimization algorithms, focusing on genetic algorithms, besides presenting an illustrative example of optimization.

2.1 Source Code Entities

In order to understand the covered structure in this master dissertation, it is essential to understand the concept of each source code entity in object-oriented design. Object-oriented systems are composed of source code entities with different roles and functionalities, representing each structure level of a program and making the system easier to understand. By understanding the behaviors of these entities and the dependencies between them, it is easier to identify and evaluate the ramifications of changes, as well as make such changes without requiring other changes in the rest of the system (KNOERNSCHILD, 2012). This dissertation addresses the four main code entities—blocks, methods, classes, and packages—described below (GOSLING et al., 2015):

- *Code Block*: Sequence of instructions, involving declarations of variables and local classes, delimited by a pair of braces;
- *Method*: Declaration of an executable code that can be invoked, giving a fixed (or variable[1]) number of values as arguments. It is composed of one or more code blocks;
- *Class*: Definition of the data and the behavior of system objects. It can be composed of methods, blocks or even nested classes; and
- *Package*: Namespace that organizes a set of related classes. A program consists of a set of packages.

[1] In certain languages, you can specify methods that take a variable number of arguments, known as *varargs* (SCHILDT, 2015).

Although the structure of the entities in a system can be done arbitrarily, it is of utmost importance to consider and analyze its dependencies in order to guarantee low coupling and high cohesion, resulting in essential improvements for the software development.

Structural dependencies occur when a compilation unit depends on another in relation to compile time or link time (FOWLER, 2004). In general, it can be said that it refers to any type of connection between two source code entities. Among the several types of dependency that exist in an object-oriented programming, it can be mentioned as more relevant: method and attribute access (*access*), variable declaration (*declare*), object creation (*create*), class extension (*extend*), interface implementation (*implement*), exception activation (*throw*) and use of annotations (*useannotation*).

2.2 Code Smells

Also known as Bad Smells, they were denoted by Fowler (FOWLER et al., 1999) as any symptom in the code that might result in maintenance problems, as well as negatively affect software lifecycle properties. Fowler categorized 22 types of Code Smells by associating them with different code refactoring opportunities. Subsequently, new types of Code Smells were categorized by other authors (KERIEVSKY, 2004; ABEBE et al., 2009).

Considering that the identification of refactoring opportunities basically aims to treat Code Smells in the source code of a program, it is essential to present the types covered by this dissertation. This work addresses the following six Code Smells (FOWLER et al., 1999):

- *Long Method:* A method of the system that has taken relatively large proportions. The main point is not the number of lines of the method, but the semantic distance between what the method does and how it does. *Long Methods* are not only difficult to understand, but often indicate the use of unnecessary dependencies, as well as the execution of unwanted subroutines;

- *Large Class:* Like *Long Method*, it concerns a class of the system that has also taken relatively large proportions. A class with variables, methods, or blocks in excess may indicate improper code positioning, resulting in worst performance and more dependencies, leading to an increased maintenance effort;

- *Divergent Change:* It is the inability to make changes in specific classes without having to make other changes in the structure of the class itself. For example, suppose that to

make a change in a method M_1 of class C_1, you must also make changes to methods M_2 and M_3 of the same class C_1. This is the clear situation that shows the lack of cohesion of the class caused by the incorrect application of the dependencies of a project;

- *Shotgun Surgery:* This is the opposite of *Divergent Change*. It occurs when you make changes to specific code entities such as blocks or methods, and you need to change other classes. Suppose a method M_1 of a class C_1. If, to make changes in M_1, it is necessary to change methods of classes C_2, C_3, and C_4, then this is a clear case of *Shotgun Surgery*. This symptom may indicate that the respective method or code block is positioned in an improper class;

- *Feature Envy:* It occurs when you observe that one method or code block overly uses internal properties of another class. This is one of the most common symptoms in Software Engineering since it can be observed several situations where, for example, a method calls several methods from another object to calculate some value. This symptom points to cases where the respective method or block is improperly positioned; and

- *Refused Bequest:* It occurs when subclasses inherit methods from a superclass and each subclass makes use of a different part of the inherited methods. More specifically, the superclass contains methods that are not common to all subclasses. This indicates a problem in the system hierarchy.

As previously mentioned, Code Smells are symptoms that *may* indicate a problem in the architecture. On some occasions, however, Code Smells are not problems. For example, test classes, of course, will depend excessively on internal properties of another class, in this case, the class to be tested. Although this situation fits in a case of *Feature Envy*, it is not a structural problem in fact. Such cases are called *false positives*.

In order to treat such Code Smells, code refactorings may be performed. Table 2.1 shows the relationship between how each Code Smell presented in this section is addressed by *Move Class*, *Move Method*, and *Extract Method*, code refactorings presented in the next section.

2.3 Refactoring

Code refactoring is the process of changing elements or the structure of a software system while preserving the external behavior of the code, but improving its internal struc-

Table 2.1 – Relationship between the treatment of Code Smells by the code refactorings

Code Smell	***Move Class***	***Move Method***	***Extract Method***
Long Method			D
Large Class		D	I
Divergent Change		D	I
Shotgun Surgery		D	I
Feature Envy		D	I
Refused Bequest		D	I

D = Direct treatment, I = Indirect treatment

ture (FOWLER et al., 1999). Through refactoring, it becomes possible to treat code architecture symptoms, known as Code Smells, which can affect features such as portability, reusability, maintainability, and scalability.

Several techniques can be used for code refactoring. Considering that this master dissertation focuses exclusively on classes, methods, and blocks of a system, we relied on *Move Class*, *Move Method*, and *Extract Method* refactorings, used to reposition classes, methods, and code blocks to address the occurrence of Code Smells.

The refactoring process consists of making small changes to the system structure, such as moving a method from one class to another, moving a class to a particular package, extracting some code block from a method to a new method, and so on. More importantly, the cumulative effect of these small changes can improve the performance of a software, as well as avoid the occurrence of bugs (FOWLER et al., 1999).

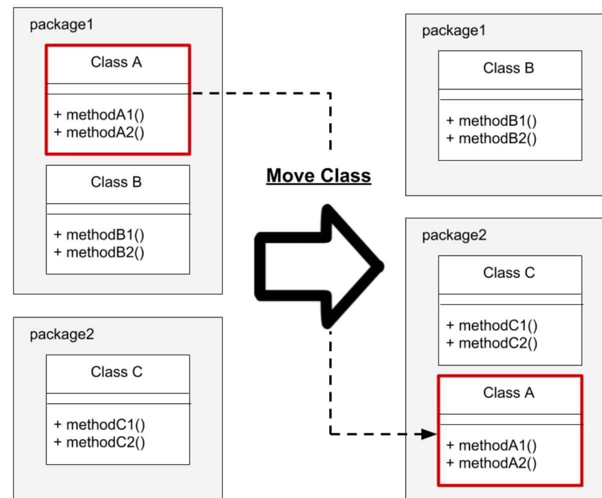
The following Sections 2.3.1, 2.3.2, and 2.3.3 respectively present *Move Class*, *Move Method*, and *Extract Method*, which are the main refactoring techniques applied to avoid the occurrence or the treatment of the six types of Code Smells mentioned in the previous section.

2.3.1 *Move Class*

In certain situations, classes of a system may be located in inappropriate packages, specially when they make a high use of internal properties of entities located in a different package, i.e., they have high amount of dependencies with another specific package.

Such classes can be moved to the most suitable package, through *Move Class* refactoring, in order to achieve a higher degree of cohesion. Figure 2.1 illustrates an example of *Move Class* operation. In this case, class *A* of `package1` is moved to `package2`, i.e., class *A* will no longer exists in `package1`, being located instead in `package2`.

Figure 2.1 – *Move Class* refactoring



It is important to notice that although the application of *Move Class* in order to reposition a class in its proper package does not addresses a specific Code Smell defined by Fowler, it is critical to avoid the occurrence of any Code Smell since the inappropriate location of a class represents an inadequate internal structure, i.e, it results in methods and blocks that are inappropriately positioned, leading to the occurrence of Code Smells previously mentioned in Section 2.2.

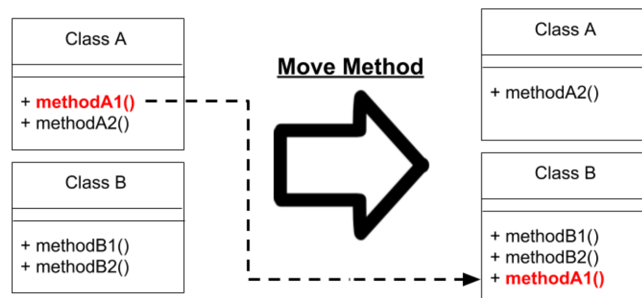
2.3.2 *Move Method*

During the software development, methods and blocks of the source code are usually improperly positioned in the structure of the system. In many cases, a method makes high use of properties from external entities than from its respective class. Thus, it becomes crucial for the system structure that such methods are moved to their proper classes.

Move Method refactoring allows the repositioning of a method to a distinct class. This operation is illustrated in Figure 2.2. Method `methodA1` is moved from class *A* to class *B*, i.e., it will no longer exists in class *A*, being located instead in class *B*.

In this way, it becomes possible to treat the following Code Smells: *Large Class*, in which the repositioning of misplaced methods will reduce the size of the respective class; *Divergent Change* and *Shotgun Surgery*, in which it will be possible to position methods that may need inadequate changes in a specific class that does not cause this need for changes; *Feature Envy*, in which methods that overuse internal properties of another class can be moved to

Figure 2.2 – *Move Method* refactoring



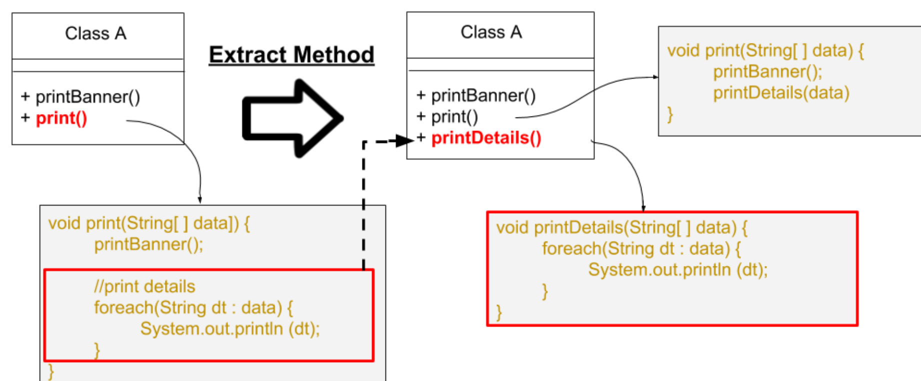
such class; and *Refused Bequest*, in which superclass methods, which do not have properties in common with all subclasses, will be able to be moved to subclasses that actually use them.

2.3.3 *Extract Method*

In certain situations, although the system's methods and classes are properly positioned, its internal structures may have code snippets that do not match the properties used by them. It might mean that certain code blocks are incorrectly positioned and need to be repositioned.

Regarding this problem, *Extract Method* refactoring allows the extraction of a code block from a method. Basically, the desired code block becomes a new method whose name explains its purpose. Figure 2.3 illustrates an example of *Extract Method* application. In this example, the last code block of `print` method is extracted, i.e., the block is removed from method `print` and a new method is generated with the respective block (in this case, `printDetails`). Finally, the block removed from `print` is replaced with a call to the new generated method.

Figure 2.3 – *Extract Method* refactoring



Extract Method refactoring provides means of dealing with Code Smell *Long Method*, in which the extraction of code blocks into new methods reduces the size of the origin method. Although *Extract Method* directly deals only with *Long Method*, it can indirectly be used for

the treatment of other Code Smells (e.g., *Large Class*, *Divergent Change*, *Shotgun Surgery*, *Feature Envy*, and *Refused Bequest)* since after extracting the code block into a new method, this method can be moved to another class through *Move Method*. Consequently, it is possible to deal with Code Smells in situations where they occur in only certain code blocks.

2.4 Structural Similarity

In order to identify the occurrence of Code Smells and hence code refactoring opportunities, it is primordial to analyze the structural similarity of the code entities present in a software design. Similarity, in the context of software architecture, deals with the concept of how much similar two or more code entities are, i.e., the measurement of how similar entities are according to its dependencies.

For the calculation of similarity indices, Table 2.2 reports the main similarity coefficients proposed in the literature (TERRA et al., 2013a).

Table 2.2 – Similarity coefficients

Coefficient	Definition	Range
Baroni-Urbani and Buser	$[a+(ad)^{\frac{1}{2}}]/[a+b+c+(ad)^{\frac{1}{2}}]$	[0, 1*]
Dot-product	$a/(b+c+2a)$	[0, 1*]
Hamann	$[(a+d)-(b+c)]/[(a+d)+(b+c)]$	[-1, 1*]
Jaccard	$a/(a+b+c)$	[0, 1*]
Kulczynski	$\frac{1}{2}[a/(a+b)+a/(a+c)]$	[0, 1*]
Ochiai	$a/[(a+b)(a+c)]^{\frac{1}{2}}$	[0, 1*]
Phi	$(ad-bc)/[(a+b)(a+c)(b+d)(c+d)]^{\frac{1}{2}}$	[-1, 1*]
PSC	$a^2/[(b+a)(c+a)]$	[0, 1*]
Relative Matching	$[a+(ad)^{\frac{1}{2}}]/[a+b+c+d+(ad)^{\frac{1}{2}}]$	[0, 1*]
Rogers and Tanimoto	$(a+d)/[a+2(b+c)+d]$	[0, 1*]
Russell and Rao	$a/(a+b+c+d)$	[0, 1*]
Simple Matching	$(a+d)/(a+b+c+d)$	[0, 1*]
Sokal and Sneath	$2(a+d)/[2(a+d)+b+c]$	[0, 1*]
Sokal and Sneath 2	$a/[a+2(b+c)]$	[0, 1*]
Sokal and Sneath 4	$\frac{1}{4}[a/(a+b)+a/(a+c)+d/(b+d)+d/(c+d)]$	[0, 1*]
Sokal binary distance	$[(b+c)/(a+b+c+d)]^{\frac{1}{2}}$	[0*, 1]
Sorenson	$2a/(2a+b+c)$	[0, 1*]
Yule	$(ad-bc)/(ad+bc)$	[0, 1*]

The * symbol indicates the maximum similarity

For the understanding of each similarity coefficient, consider two code entities *A* and *B*. Taking into consideration that this dissertation analyzes structural dependencies among code entities, it has the following variables:

a = number of dependencies in both entities,

b = number of exclusive dependencies of entity *A*,

c = number of exclusive dependencies of entity *B*, and

d = number of the remainder from the total dependencies considered.

In order to illustrate the calculation of structural similarity between two code entities, we applied the *Jaccard* and *Simple Matching* coefficients in two methods of myAppointments, a simple personal information management system (PASSOS et al., 2010). In this way, we selected methods `loadAppointments`, responsible for loading appointments of the system, and `getAppointmentRowAsDate`, responsible for returning the appointment date of a specific row in the data set. Both methods are present in the same control class. Code 2.1 shows the implementation of both methods.

Code 2.1 – Fragment of class **AgendaController**

```
1  public class AgendaController
2    ...
3
4    public void loadAppointments() throws Exception {
5      List<Appointment> apps = agendaDAO.getAppointments
6        (DateUtils.getCurrentDay(),
7             DateUtils.getCurrentMonth(),
8                    DateUtils.getCurrentYear());
9
10     int i = 0 ;
11     for(Appointment app : apps) {
12       agendaView.insertAppointRow
13         (i, DateUtils.toString(app.getDate(),DateUtils.HOUR_FMT),
             app.getTitle());
14       i++ ;
15     }
16   }
17
18   private Date getAppointmentRowAsDate(int row) {
19     String[] appHour = agendaView.getAppointmentRow(row)[0].split(":");
20     return DateUtils.newDate
21       (DateUtils.getCurrentDay(),
22         DateUtils.getCurrentMonth(),
23         DateUtils.getCurrentYear(),
24         Integer.parseInt(appHour[0]),
25         Integer.parseInt(appHour[1]));
26   }
27 }
```

Based on the analysis of both methods and the structural dependencies present in their structures, it has:

- $a = 2$, since both methods access methods from `AgendaView` and `DateUtils` (highlighted by red color);

- $b = 4$, since method `loadAppointments` has the following exclusive dependencies: the throw of an exception of type `Exception`, the declaration of `List` and `Appointment`, and the access to methods from `AgendaDAO` (highlighted by blue color);

- $c = 1$, since the exclusive dependency of the method `getAppointmentRowAsDate` is the declaration of type `Date` as return (highlighted by green color);

- $d = 32$, since when considering the entire system, another 32 types are used by other system entities, i.e., the number of all other types of the systems. In this example, there is a total of 39 types and $d = 39 - (a) - (b) - (c) = 32$.

We treat types indifferently, e.g., if the analyzed systems have dependencies to types of external libraries they are counted in the same way. On the other hand, dependencies to primitive types (e.g., `int`, `char`, `byte`, etc.), to wrappers classes (e.g., `Integer`, `Long`, `Character`, etc.) and `String` type are disregarded during the analysis since almost all classes establish dependencies with these types, not contributing to the similarity calculation.

Therefore, when applying, as example, the *Jaccard* and *Simple Matching* coefficient, the following similarity indexes are found:

$$Jaccard = \frac{a}{a + b + c} = \frac{2}{2 + 4 + 1} = 0.28 \tag{2.1}$$

$$Simple\ Matching = \frac{a + d}{a + b + c + d} = \frac{2 + 32}{2 + 4 + 1 + 32} = 0.87 \tag{2.2}$$

Thus, it is important to mention that only the gross value does not indicate the similarity level with accuracy since it is necessary to observe and compare the other similarity values of the system altogether. For instance, 0.28 (*Jaccard*) can be considered a high similarity value

if the mean similarity of the system is 0.12. Similarly, 0.87 (*Simple Matching*) can be considered as a low value, depending from the other values. Therefore, it should be emphasized that the similarity index provides only an approximation of reality in order to make comparative analyzes or even find refactoring opportunities due to problems in the code structure.

More importantly, this master dissertation emphasizes the fact that the existing similarity coefficients in literature were not created to measure similarity between source-code entities. For example, the *Jaccard* coefficient, one of the most used in Software Engineering, was initially designed to compare the similarity between floral species in different districts (JACCARD, 1912).

2.5 Optimization Algorithms

Optimization refers to the process of finding and comparing solutions to a given problem, maximizing and/or minimizing the values of the variables that compose the objective function until the best possible solution is found (CHIS, 2010). However, in many cases, a problem does not have an optimal solution, which results in the search for solutions that meet the desired objective satisfactorily. This concept, in turn, can be applied to adapt and improve the similarity coefficients discussed in the previous section.

When searching a solution of optimization problems, several approaches can be applied, involving different types of algorithms. Among the several approaches, one of the most used by researchers is the application of a simple genetic algorithm, which has shown to be able to find satisfactory solutions effectively (HARMAN, 2007; MKAOUER et al., 2016). Therefore, it has become the chosen approach for application in this master dissertation.

2.5.1 Genetic Algorithms

A genetic algorithm defines a set of candidates for the proposed solution and, at each iteration (each generation of the genetic algorithm), selects and combines the most suitable candidates, which may undergo slight changes. Thus, at the end of all execution, the solution set will consist of candidates with relative great potential to optimize the objective function (SIVANANDAM; DEEPA, 2007).

Considering the problem of optimizing similarity coefficients, the algorithm defines weights and exponents for each variable of the chosen coefficient formula and each iteration seeks to adapt each weight and exponent in order to optimize the result of the objective func-

tion, finally selecting the set of best results. For instance, considering the *Jaccard* coefficient (presented in Table 2.2), and the weights $P_{a'}$, $P_{a''}$, P_b and P_c, as also the exponents $E_{a'}$, $E_{a''}$, E_b, and E_c, corresponding, respectively, to variables a of the numerator and a, b and c of the denominator. Thus, it has as a result:

$$\text{Resulting Coefficient} = \frac{P_{a'} * a'^{E_{a'}}}{P_{a''} * a''^{E_{a''}} + P_b * b^{E_b} + P_c * c^{E_c}} \tag{2.3}$$

During the execution of a genetic algorithm, we must setup the following parameters:

- *Objective function:* Function representing the data to be optimized through variables that need to be minimized or maximized. In this dissertation, for example, our goal is to maximize the number of similarity indexes that have better values (high values for the similarities that should be maximized and low values for similarities that should be minimized), i.e., the objective function is the total sum of optimized similarities;
- *Representation of the population:* Describes the universe of every individual in population, i.e., the universe for the candidates to optimize the objective function. In this master dissertation, the population is randomly generated, focusing on it diversity;
- *Population size:* The initial number of possible candidates for the desired variables applied to optimize the objective function;
- *Number of generations:* Number of times the genetic algorithm will be repeated, i.e., iterate adapting the desired values;
- *Crossover operator:* Function that combines candidates in order to possibly generate better candidates to the solution set;
- *Selection operator:* Function that chooses candidates that are more likely to cross-check between them, i.e, a function to select the best candidates from the population to perform a crossover;
- *Mutation operator:* Function to perform a slight change (mutation) in the value of the generated candidate(s) from a crossover, which prevents its stagnation, besides allowing it arriving at any point of the search space;
- *Probability of crossover:* The probability in percentage of a crossover be performed;

- *Probability of mutation:* The probability in percentage of a mutation be performed;
- *Crossover distribution index:* A non-negative real number used as an index to determine the diversity of the generated candidates, i.e., a large value gives a higher probability for creating near-parent solution candidates and small value allows distant candidates to be created from the crossover;
- *Mutation distribution index:* A non-negative real number used as an index to determine the diversity of the mutated candidates, i.e., a large value gives a higher probability for creating near-parent solution candidates and small value allows distant candidates to be created from the mutation.

Among the different operators of selection, crossing and mutation, this dissertation uses, respectively, *Binary Tournament*, *Simulated Binary Crossover*, and *Polynomial Mutation*. The *Binary Tournament* method selects, among all the possible generated candidates until then, two individuals that present higher results in relation to the objective function so that they are crossed. The *Simulated Binary Crossover* occurs through the crossing of two individuals, combining their binary representations, in order to generate two new individuals. Such a combination considers a defined probability, analyzing whether each binary index should be combined or not. Finally, the *Polynomial Mutation* also considers a defined probability in order to change one or more binary indexes of the individuals resulting from the crossover. In both crossover and mutation operators, a distribution index must be established in order to evaluate the diversity of the selected solutions in search space, which guarantees the selection of more heterogeneous individuals.

After defining its parameters, the genetic algorithm will randomly generate the stipulated quantity of candidates as their initial generation (each candidate refers to a set of all variables from the similarity coefficient). Then, the algorithm will generate the same number of new candidates (the *population size*) by crossing over the current best-performing candidates selected by the *selection operator*. Such crossover will be performed by the *crossover operator* considering the *probability* and the *distribution index* parameters. Each generated candidate will replace the least suitable candidate, in order to keep the population always with the same size. In addition, the generated candidate may have a small change in their values through the *mutation operator* and its parameters of *probability* and *distribution index*. This process will

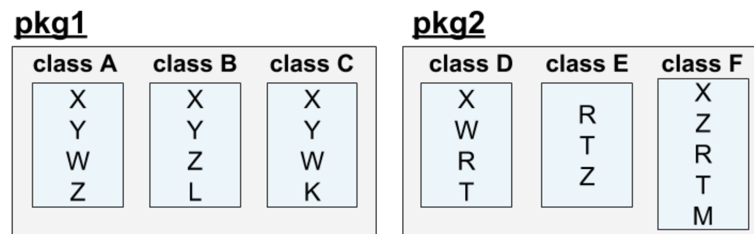
be repeated according to the defined *number of generations* and, finally, the candidate with the best result will be presented as the output.

2.5.2 Illustrative Example

For a better understanding of the approach of optimization algorithms and concepts of genetic algorithms, this section presents the application of a genetic algorithm to optimize the *Jaccard* similarity coefficient in a little example of *Move Class* refactoring.

For instance, take as assumption a system with two packages *pkg*1 and *pkg*2 with a set of classes and their dependencies, respectively, as *pkg*1={*A*={X,Y,W,Z}, *B*={X,Y,Z,L}, *C*={X,Y,W,K}} and *pkg*2={*D*={X,W,R,T}, *E*={R,T,Z}, *F*={X,Z,R,T,M}}, as shown in Figure 2.4.

Figure 2.4 – Illustrative example



Assume it is known that the current system architecture is the ideal one and hence we intend to use a similarity coefficient in which does not result in suggestions of *Move Class* refactoring, i.e., maintain the current architecture and guarantee good similarity indexes among classes from the same package. Thus, we intend to adapt a coefficient in order to maximize the similarity of $sim(A,B)$, $sim(A,C)$, $sim(B,C)$, $sim(D,E)$, $sim(D,F)$, and $sim(E,F)$. Simultaneously, we expect to minimize the similarity of $sim(A,D)$, $sim(A,E)$, $sim(A,F)$, $sim(B,D)$, $sim(B,E)$, $sim(B,F)$, $sim(C,D)$, $sim(C,E)$, and $sim(C,F)$. However, only the gross value does not indicate the similarity level with accuracy since it is necessary to observe and compare the other similarity values of the system altogether. In this way, we aim to obtain the largest possible difference between the arithmetic mean of similarities aimed to maximize and to minimize.

Considering the *Jaccard* coefficient as an example, we decided to apply the genetic algorithm defining weights and exponents for each coefficient variable, as previously described in Equation 2.3. In this way, we used *jMetal*[2], a tool specialized for optimization problems. Thus, we relied on the default tool configuration presented in Table 2.3, changing only the

[2] <https://jmetal.github.io/jMetal/>

objective function, population size and representation, and number of generations, which were obtained after a series of attempts aimed to improve the coefficients, considering the computational resource disposition, execution time, and data set.

Table 2.3 – Genetic algorithms configuration

Objective function:	The total sum of optimized similarities	**Crossover operator:**	Simulated Binary Crossover
Population size:	1200	**Probability of mutation:**	0.06
Representation of the population:	$\{x \in \mathbb{R} \mid -3 \leq x \leq 3\}$	**Probability of crossover:**	0.09
Number of generations:	150	**Mutation distribution index:**	20.0
Selection operator:	Binary Tournament	**Crossover distribution index:**	20.0
Mutation operator:	Polynomial Mutation		

In this scenario, we found as a possible solution set, the weights 1.34 ($P_{a'}$), 0.33 ($P_{a''}$), 1.0 (P_b), and 0.34 (P_c), as also 0.5 ($E_{a'}$), 1.0 ($E_{a''}$), 2.0 (E_b), and 2.0 (E_c), respectively to variables a of the numerator and a, b, and c of the denominator. The comparison of similarity between *Jaccard* and the resulting coefficient is shown in Table 2.4.

Table 2.4 – Comparison between *Jaccard* and the resulting coefficient

Maximize				
Objective	***Jaccard***	**Similarity**	**Resulting Coefficient**	**Similarity**
sim(A,B) =	$\frac{3}{3+1+1} =$	0.6	$\frac{1.34*\sqrt{3}}{0.33*3^1 + 1.0*1^2 + 0.34*1^2} =$	0.9961
sim(A,C) =	$\frac{3}{3+1+1} =$	0.6	$\frac{1.34*\sqrt{3}}{0.33*3^1 + 1.0*1^2 + 0.34*1^2} =$	0.9961
sim(B,C) =	$\frac{2}{2+2+2} =$	0.3333	$\frac{1.34*\sqrt{2}}{0.33*2^1 + 1.0*2^2 + 0.34*2^2} =$	0.3148
sim(D,E) =	$\frac{2}{2+2+1} =$	0.4	$\frac{1.34*\sqrt{2}}{0.33*2^1 + 1.0*2^2 + 0.34*1^2} =$	0.3790
sim(D,F) =	$\frac{3}{3+1+2} =$	0.5	$\frac{1.34*\sqrt{3}}{0.33*3^1 + 1.0*1^2 + 0.34*2^2} =$	0.6928
sim(E,F) =	$\frac{3}{3+0+2} =$	0.6	$\frac{1.34*\sqrt{3}}{0.33*3^1 + 1.0*0^2 + 0.34*2^2} =$	0.9876
Mean		**0.5055**	**Mean**	**0.7277**

Minimize				
Objective	***Jaccard***	**Similarity**	**Resulting Coefficient**	**Similarity**
sim(A,D) =	$\frac{2}{2+2+2} =$	0.3333	$\frac{1.34*\sqrt{2}}{0.33*2^1 + 1.0*2^2 + 0.34*2^2} =$	0.3148
sim(A,E) =	$\frac{1}{1+3+2} =$	0.1667	$\frac{1.34*\sqrt{1}}{0.33*1^1 + 1.0*3^2 + 0.34*2^2} =$	0.1253
sim(A,F) =	$\frac{2}{2+2+3} =$	0.2857	$\frac{1.34*\sqrt{2}}{0.33*2^1 + 1.0*2^2 + 0.34*3^2} =$	0.2455
sim(B,D) =	$\frac{2}{2+2+2} =$	0.3333	$\frac{1.34*\sqrt{2}}{0.33*2^1 + 1.0*2^2 + 0.34*2^2} =$	0.3148
sim(B,E) =	$\frac{0}{0+4+3} =$	0.0	$\frac{1.34*\sqrt{0}}{0.33*0^1 + 1.0*4^2 + 0.34*3^2} =$	0.0
sim(B,F) =	$\frac{1}{1+3+4} =$	0.125	$\frac{1.34*\sqrt{1}}{0.33*1^1 + 1.0*3^2 + 0.34*4^2} =$	0.0907
sim(C,D) =	$\frac{1}{1+3+3} =$	0.1429	$\frac{1.34*\sqrt{1}}{0.33*1^1 + 1.0*3^2 + 0.34*3^2} =$	0.1081
sim(C,E) =	$\frac{1}{1+3+2} =$	0.1667	$\frac{1.34*\sqrt{1}}{0.33*1^1 + 1.0*3^2 + .0.34*2^2} =$	0.1253
sim(C,F) =	$\frac{2}{2+2+3} =$	0.2857	$\frac{1.34*\sqrt{2}}{0.33*2^1 + 1.0*2^2 + 0.34*3^2} =$	0.2455
Mean		**0.2044**	**Mean**	**0.1744**

It is possible to observe that the coefficient resulting from the optimization showed to be much more efficient than *Jaccard* since it obtained a difference between the mean of similarities that should be maximized and the mean of the similarities that should be minimized of 0.55, in contrast to *Jaccard* which obtained a difference of only 0.30. Therefore, the resulting coefficient proved to be the more adequate and accurate for the similarity analysis in the presented example.

3 EMPIRICALLY SUPPORTED SIMILARITY COEFFICIENTS

In order to create three new similarity coefficients that are more efficient in identifying respectively *Move Class*, *Move Method*, and *Extract Method* refactoring opportunities, this master dissertation adopts a specific methodology (Section 3.1) towards the selection of a coefficient to be adapted through the analysis of the main existing similarity coefficients (Section 3.2) and the empirical experiment for the proposal of three new coefficients after adaptation of the respective selected coefficient (Section 3.3).

3.1 Methodology

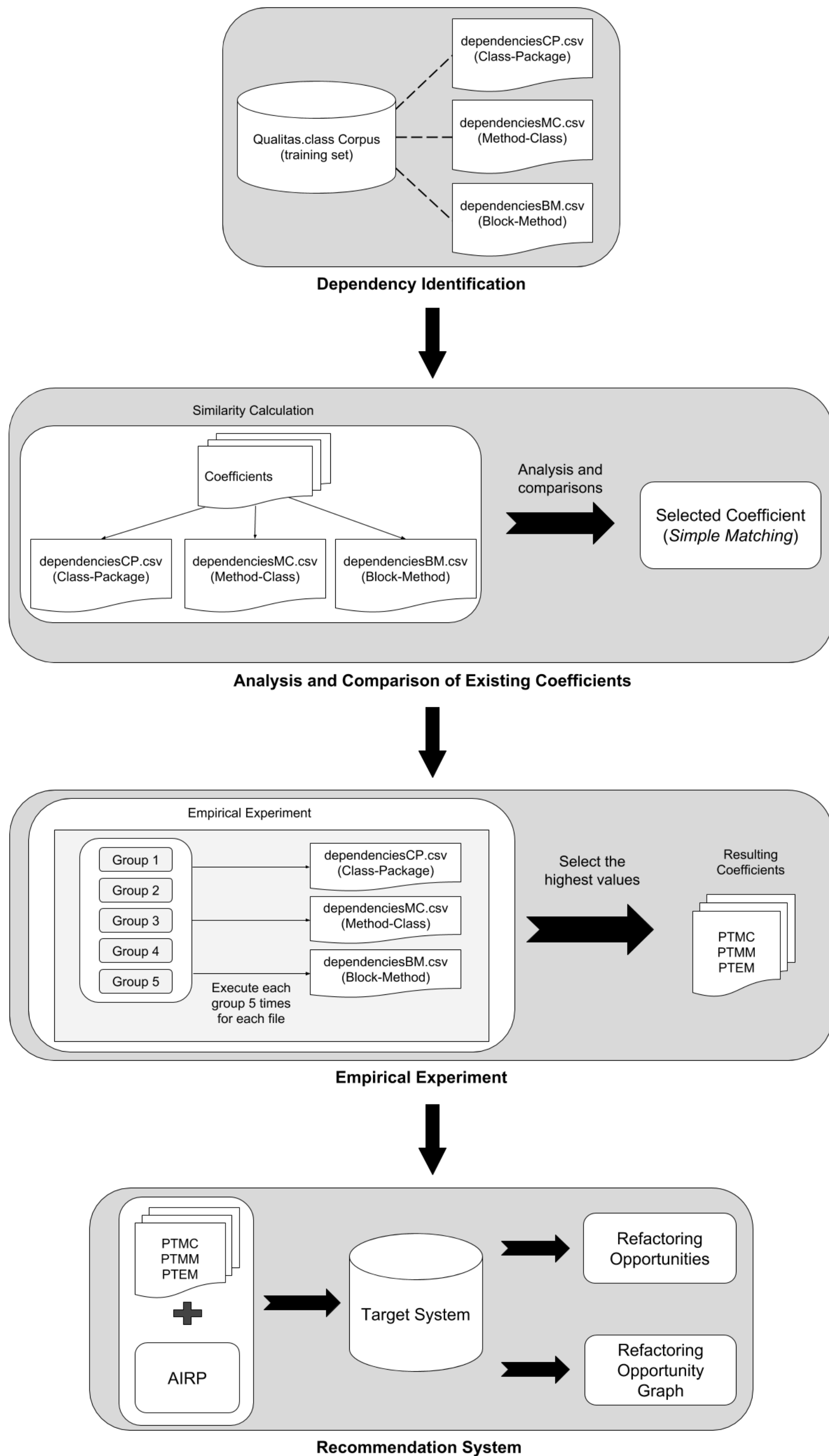
This dissertation defines a specific methodology that must be followed in order to achieve the intended results. Figure 3.1 illustrates the process of this respective methodology used to propose new similarity coefficients and identify refactoring opportunities.

Firstly, we identify all dependencies of the 10 systems used as training set and separate them into files related to the level to be analyzed, i.e., dependencies between classes and packages for *Move Class*, between methods and classes for *Move Method*, and between blocks and methods for *Extract Method*. Secondly, we calculate the precision of each similarity coefficient for the entities of the systems using such files. Then, after the analysis and comparison, we selected *Simple Matching* coefficient to be adapted. Thirdly, we designed an empirical experiment that has five groups of configurations for genetic algorithms to be executed in each dependency file and replicated five times each. After, we selected the highest values to propose our three new similarity coefficients ($PTMC$, $PTMM$, and $PTEM$).

3.1.1 Calculation of Similarity Coefficient's Precision

The precision of a similarity coefficient—i.e., the measurement of how correctly a coefficient identifies, for refactoring suggestions, where a respective entity should be located—can be calculated by measuring if the highest similarity index found between source code entities indicates the same position of a predefined ideal structure. For example, consider a predefined ideal system's architecture containing a class A located inside a package $pkg1$. To calculate the precision of a similarity coefficient for the identification of *Move Class* refactoring opportunities, we measure the similarity between class A with its respective package $pkg1$. If it is

Figure 3.1 – Methodology overview



the highest similarity, i.e., if the similarity between class *A* and any other package has a lower value, then we can say that the coefficient *precisely* identified the correct location of class *A*.

The calculation of similarity coefficient's precision, used to identify the refactoring opportunities in each methodology step, has a different approach according to the refactoring operation as detailed below:

Move Class: In order to calculate the coefficient precision to identify *Move Class* refactoring opportunities, we verify if the ideal package has the highest similarity with the analyzed class. The similarity of a class with a package is given through the arithmetic mean of the similarity between the respective class and the other classes in the package. This decision is based on the fact that measuring similarity of a simple class and the entire package itself could result in a high similarity even though the classes were not similar. Consequently, it would be possible to suggest *Move Class* refactorings for packages that actually have similar classes. It must be noticed that this approach also analyzes inner packages, i.e., packages inside packages.

***Move Method*:** A similar approach to the one used for *Move Class* is applied to calculate the coefficient precision to identify *Move Method* refactoring opportunities, in which we verify if the ideal class has the highest similarity with the analyzed method. In this case, the similarity analysis between methods and classes considered the similarity mean between the respective method and the other methods in the class.

***Extract Method*:** In order to calculate the coefficient precision to identify *Extract Method* refactoring opportunities, we compare, in an ideal class structure, the similarity between methods before and after the possible extraction. First, we calculate the similarity mean between the method with the respective block that might be extracted with the other methods in the class. After, we simulate an *Extract Method* refactoring to the respective block and then, we recalculate the similarity mean between the method where the block was extracted with the other methods in the class (including the new method generated). Finally, the arithmetic means of both sets of values are compared in order to verify whether or not the *Extract Method* refactoring should be suggested. If the similarity index was higher before the extraction, then we consider as the right precision.

In this way, we analyzed and evaluated the coefficients using as the ideal architecture, systems from *Qualitas.class Corpus* (TERRA et al., 2013b), database with 111 object-oriented systems, more than 18 million LOC (*Lines Of Code*), 200 thousand compiled classes, and 1.5 million compiled methods. Although the systems may have misplaced entities, *Qualitas.class Corpus* is composed of more mature and stable systems and hence we assume that the current structure of systems has a reasonable similarity index in relation to possible code refactorings. More important, the heterogeneity of the systems leads to more solid analyses and evaluations toward the generalization of the results.

To summarize, it is expected that most of the structure is maintained and few refactoring should be performed. Thus, it is possible to evaluate the precision of each similarity coefficient in the structure of each analyzed system, i.e., we analyze the coefficient precision verifying if the highest similarity indexes indicate the respective system structure.

Experimental Rules: To avoid the occurrence of false positives in the resulting refactoring recommendations, after a series of experimental tests and executions, we defined nine experimental rules for the implementation of the proposed solution:

1. *We disregard the entity under analysis when the system searches for refactoring opportunities*. When the similarity between a class *A* and its respective package *Pkg* is calculated, the system considers *Pkg* to be *Pkg* - $\{A\}$. In this case, individually located entities are totally discarded;

2. *We did not consider packages and test classes*, since most systems organize their test classes into a single package. Therefore, this package contains classes related to different parts of the system, i.e., they are not structurally related. For this purpose, an approach that disregards packages and classes containing the text "`test`" (uppercase or lowercase) anywhere in its name is used. Although neither all test package or class containing the text "`test`" and not every package or class that contains it is in fact a test package or class, the gain in precision is greater than the loss when test packages or classes are erroneously detected;

3. *We ignored trivial dependencies*. Dependencies—such as those established with primitive types and wrappers (e.g., `int` and `java.lang.Integer`), dependencies of

`java.lang.String` and `Java.lang.Object`—are filtered. This rule was inspired by the concept of *stop words* in relation to information retrieval (BAEZA-YATES; RIBEIRO-NETO, 2011). Since the vast majority of code elements establish dependencies with these types, they do not actually contribute to the similarity calculation. Although these dependencies may affect the results in few systems, they are mostly irrelevant for the majority types of systems and hence we applied this rule focusing on a more generalized approach;

4. *We did not analyze class, method, or block entities that establish dependencies with fewer than three types*. Although there may be entities with few dependencies that should be refactored, these entities contain little information for calculating similarity or for making any inference based on their structural dependencies. This rule was defined based on a study that came up with this configuration after a preliminary empirical assessment concluded that it provides a small number of recommendations but at the same time the number of methods analyzed is relatively large (TERRA et al., 2018). However, it should be noted that although these entities are not analyzed whether they should be refactored, they are still considered as a possible refactoring destination, whether they have at least some dependency;

5. *We did not analyze entities that are not co-located with at least two other considered entities*. For instance, a package with only two classes does not provide enough structural information to recommend whether or not to move its classes. Also, it is necessary at least two other entities to establish a mean between their similarities. Again, this criterion disregards only the analysis of such entities, but can still be considered as a refactoring destination;

6. *We did not extract the first block of a method*. The extraction of the first block of a method will not change similarity. Since the extraction of a block will result in the extraction of its internal blocks, this operation only recreates the respective method. In this situation, the ideal would be to move the method;

7. *We assigned any dependency present on an internal entity class, method or block to its external entities*. Given that an internal code entity (e.g., nested classes, internal

methods and blocks, etc.) might be within the scope of the external entity(ies), any established dependency must be assigned to external entities;

8. *We disregard methods and blocks belonging to an* `Interface` *type*. Due to the very nature of interfaces being composed of abstract members, they should not be moved or extracted since this would lead to a series of conflicts in the project structure; and

9. *We disregard constructor methods and their respective blocks*. Considering the requirement of constructors in a class, the movement of this method type is disregarded, although it is intended to evaluate the possible extraction of its internal blocks.

3.2 Analysis and Comparison of Existing Coefficients

This section analyzes and compares the existing similarity coefficients in order to find the most adequate coefficient to be adapted. Thus, we propose new similarity coefficients in order to obtain higher precision rates. Thus, we investigate 18 similarity coefficients (Table 2.2) in 10 systems (training set) of *Qualitas.class Corpus*, selected after a manual investigation considering the most analyzed systems in relation to code refactoring (DALLAL, 2015). The selected systems are presented in Table 3.1.

Table 3.1 – Target-systems

System	Version	System	Version
Ant	1.8.2	JFreeChart	1.0.13
ArgoUML	0.34	JHotDraw	7.5.1
Collection	3.2.1	JMeter	2.5.1
Hibernate	4.2.0	JUnit	4.1
JEdit	4.3.2	Tomcat	7.0.2

We applied all analyzed similarity coefficients to each entity of the selected target systems, analyzing whether an entity has the highest similarity with its enclosing entity or if a method should be extracted (as described in Section 3.1.1). For instance, for a class *A*, each coefficient is applied by comparing *A* with each package in the system, seeking to verify whether the highest found similarity rate (*Top #1*) corresponds to the package in which *A* is actually positioned. This evaluation also analyzes the second (*Top #2*) and the third (*Top #3*) higher rate

in order to verify whether the applied coefficient has at least results close to the desired one. Finally, the arithmetic mean is calculated on each system of training set considering *Top #1*, *#2*, and *#3*. Considering that some data do not follow a normal distribution, we performed the analysis over the median of these sets of means since median represents a better estimate of central tendency than the overall mean for a system.

We evaluated each coefficient separately in relation to the types of code refactoring *Move Class*, *Move Method*, and *Extract Method*. Table 3.2 presents the precision of the addressed coefficients in relation to the identification of the *Move Class*, *Move Method*, and *Extract Method* refactoring opportunities. *Extract Method* refactorings consider only a single success rate since for *Extract Method* refactorings it is only analyzed whether the method should be extracted or not, therefore, the possibility of success in the second or third attempt is discarded (*Top #2* or *#3*). For space constraints, the detailed table with the results of each system is presented at Appendix A.

For a complementary analysis between the data, Figure 3.2 presents a violin plot regarding *Top #1* of each refactoring, where it is illustrated the density of each coefficient (i.e., probability of a variable assume a certain value), as well as its quartiles, including its median. The dashed line indicate the highest coefficient median.

After analyzing and comparing the main existing coefficients, we selected *Simple Matching* to be adapted in order to generate a new coefficient. The *Simple Matching* coefficient has an easy structure to be adapted, defining weights for the variables, thus contributing to obtain good results in the proposal of new coefficients. In contrast, a large part of the analyzed coefficients had already defined weights. Although it was not the best-performing coefficient, the possible weight definition of *Simple Matching* allows simulating coefficients as *Sokal and Sneath 2*, which was the best coefficient in *Move Method* and second best in *Move Class*[1], as well as *Russell and Rao*, which obtained greater precision in *Extract Method*. Additionally, the coefficient considers the universe of dependencies, in contrast to coefficient *PSC*, for instance.

3.3 Empirical Experiment

Given the fact that genetic algorithms have nondeterministic execution and results, it becomes a problem to know if their results are satisfactory or even adequate for the expected

[1] *PSC* was the best coefficient for *Move Class*, however, it was not selected because it englobes elementary arithmetic operations among variables a, b, and c. Such mathematical relation is out of the scope of this master dissertation.

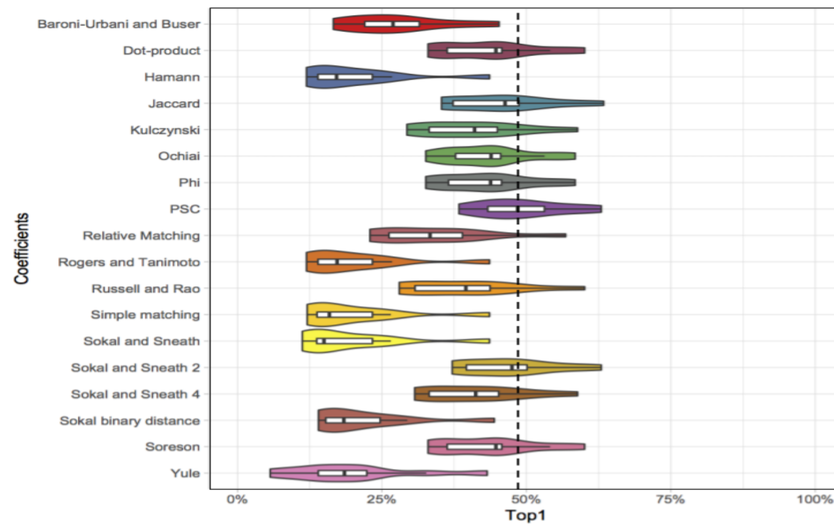
Table 3.2 – Precision of the 18 similarity coefficients (training set)

MOVE CLASS			
Coefficient	**Top1**	**Top2**	**Top3**
Baroni-Urbani and Buser	26.91%	40.39%	47.10%
Dot-product	44.73%	55.69%	62.34%
Hamann	17.14%	22.94%	25.15%
Jaccard	46.30%	59.24%	64.25%
Kulczynski	41.03%	55.41%	63.02%
Ochiai	43.89%	56.72%	63.55%
Phi	43.80%	56.84%	63.00%
PSC	48.49%	62.33%	71.32%
Relative Matching	33.33%	45.95%	52.75%
Rogers and Tanimoto	17.23%	23.17%	25.35%
Russell and Rao	39.52%	53.14%	61.12%
Simple matching	15.90%	22.94%	25.20%
Sokal and Sneath	14.96%	22.61%	24.80%
Sokal and Sneath 2	47.50%	60.10%	66.08%
Sokal and Sneath 4	41.23%	55.52%	63.12%
Sokal binary distance	18.40%	24.57%	27.06%
Sorenson	44.73%	55.69%	62.34%
Yule	18.51%	29.42%	34.62%

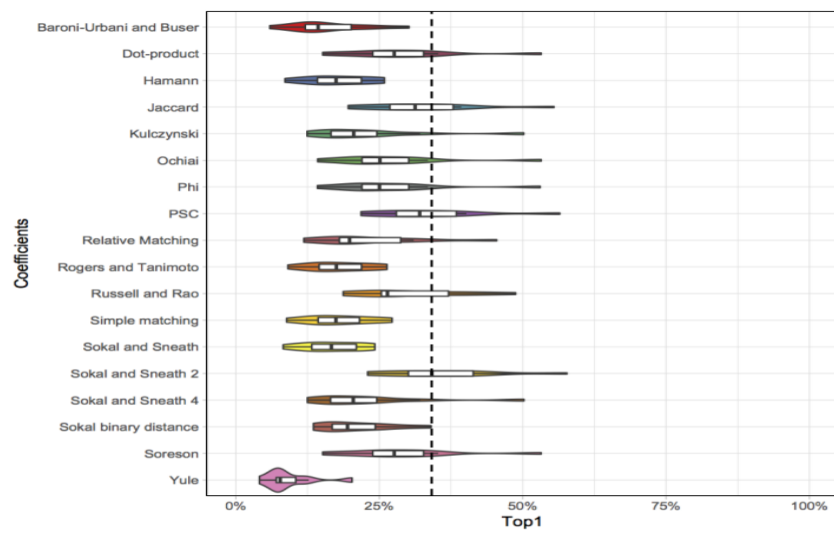
MOVE METHOD			
Coefficient	**Top1**	**Top2**	**Top3**
Baroni-Urbani and Buser	14.34%	19.03%	23.66%
Dot-product	27.61%	37.43%	44.09%
Hamann	17.47%	23.78%	28.48%
Jaccard	31.25%	41.54%	48.59%
Kulczynski	20.54%	28.38%	35.27%
Ochiai	25.11%	33.88%	40.41%
Phi	25.03%	33.88%	40.41%
PSC	32.05%	43.75%	51.06%
Relative Matching	19.82%	27.90%	33.41%
Rogers and Tanimoto	17.52%	23.95%	28.84%
Russell and Rao	26.43%	36.13%	43.52%
Simple matching	17.45%	23.78%	28.32%
Sokal and Sneath	16.65%	23.06%	27.58%
Sokal and Sneath 2	34.17%	45.26%	52.27%
Sokal and Sneath 4	20.46%	28.33%	35.27%
Sokal binary distance	19.46%	26.73%	33.12%
Sorenson	27.61%	37.43%	44.09%
Yule	7.72%	12.81%	17.28%

EXTRACT METHOD	
Coefficient	**Hits**
Baroni-Urbani and Buser	54.80%
Dot-product	59.23%
Hamann	10.37%
Jaccard	62.92%
Kulczynski	69.65%
Ochiai	63.71%
Phi	63.69%
PSC	69.42%
Relative Matching	72.27%
Rogers and Tanimoto	10.29%
Russell and Rao	77.55%
Simple matching	10.31%
Sokal and Sneath	10.34%
Sokal and Sneath 2	68.85%
Sokal and Sneath 4	69.48%
Sokal binary distance	18.24%
Sorenson	59.23%
Yule	56.45%

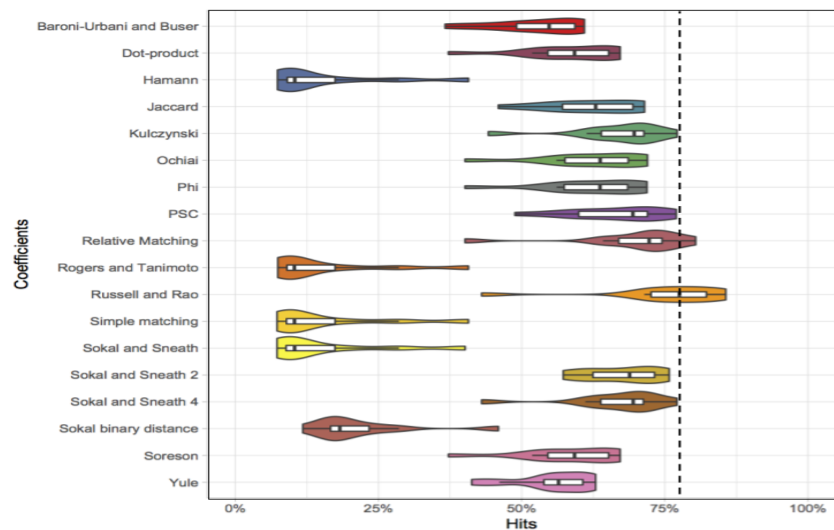
Figure 3.2 – Refactorings plots



(a) *Move Class*



(b) *Move Method*



(c) *Extract Method*

purpose. In order to address this problem, we designed an experiment for the proposal of the new coefficients. The experiment consists in a treatment combination with replication, i.e., the repetition of multiple executions in different groups of factors' values. This design is composed by two factors, the initial population size and the number of generations for the genetic algorithm.

After selecting the *Simple Matching* coefficient to be adapted, several executions of a genetic algorithm is applied to the referred coefficient, having 10 systems of the *Qualitas.class Corpus* as training set.

The conducted experiment aims to find which candidates represent a better value for the objective function. For this purpose, the genetic algorithm relies on the configuration used in the illustrative example of Section 2.5.2 from Chapter 2 (see Table 2.3). However, in order to find better results and lower the degree of uncertainty, we defined—after a series of attempts and tests according to the size of the training set and our computational resources—five configuration groups, with the combination of different factors' values, and replicated five times each, totaling 25 executions. More precisely, each group maintains the original genetic algorithm configuration, changing only the initial population size and the number of generations (the experiment factors), as shown in Table 3.3.

Table 3.3 – Experiment groups of factors' values

	Population Size	**No of Generations**
Group 1	2000	1000
Group 2	1500	1500
Group 3	500	2000
Group 4	300	2500
Group 5	100	3000

It is important to clarify that each new proposed coefficient is resulting from a different set of executions of the genetic algorithm (25 executions for each coefficient). Since a single iteration of the genetic algorithm performs thousands of comparisons between code entities to improve results, we argue that this number of executions is satisfactory.

After generating the set of solutions, we obtained several possibilities that result in the same highest precision rate. Therefore, we selected a single solution that shows the highest distance between the mean of similarity indexes that it seeks to maximize with the mean of indexes that it seeks to minimize, i.e., we selected the solution with the largest *gap* between the similarities of correctly positioned entities and those improperly positioned, since a greater

distance indicates that the coefficient will tend to maximize and to minimize the similarities correctly in other systems.

Thus, considering the possibility of defining weights to the *Simple Matching* coefficient variables, it was initially simulated as genetic algorithm candidates (included in the initial population) the weights of *Sokal and Sneath 2* coefficient for *Move Class* and *Move Method* refactorings, as well as *Russell and Rao* weights for *Extract Method* refactorings, which facilitates the selection of new candidates since they were the best results in their respective refactorings, i.e., the resulting coefficient started the genetic algorithm with *Sokal and Sneath 2* and *Russell and Rao* precision and was optimized from then on. Subsequently, we performed the executions of the genetic algorithm on the 10 systems (training set) in order to obtain more adequate weights for each coefficient variable and to create the new coefficients to be evaluated in the other 101 systems (test set).

Finally, each execution of the experiment was carried out in order to find a higher value for the objective function. Table 3.4 shows the values of each execution.

Table 3.4 – Experiment's objective function results

MOVE CLASS					
	Execution #1	Execution #2	Execution #3	Execution #4	Execution #5
Group 1	3268	3176	3282	3112	3198
Group 2	3288	3061	3241	3281	3186
Group 3	3273	3044	3273	3171	3098
Group 4	3211	2989	3065	3163	3177
Group 5	3138	3250	3151	**3382**	3281

MOVE METHOD					
	Execution #1	Execution #2	Execution #3	Execution #4	Execution #5
Group 1	9794	9736	10142	9256	9915
Group 2	9376	10058	9794	9736	10129
Group 3	9060	9126	9051	**10189**	10075
Group 4	9944	9370	9995	9611	9922
Group 5	9890	9446	9164	9892	10091

EXTRACT METHOD					
	Execution #1	Execution #2	Execution #3	Execution #4	Execution #5
Group 1	21128	21165	21217	21192	21110
Group 2	21173	21235	21141	21229	21261
Group 3	**21291**	21207	21135	21112	21175
Group 4	21285	21156	21262	21287	21262
Group 5	21145	21234	21235	21290	21270

In this way, we proposed the following coefficients, where each one corresponds to a type of refactoring code: *PTMC* for *Move Class* operations, *PTMM* for *Move Method* operations and *PTEM* for *Extract Method* operations. The implementation of the genetic algorithm on *Simple Matching* in the defined training set resulted in the first version of the three sought coefficients. The resulting weights $P_{a'}$, $P_{d'}$, $P_{a''}$, P_b, P_c and $P_{d''}$, as also the resulting exponents $E_{a'}$, $E_{d'}$, $E_{a''}$, E_b, E_c and and $E_{d''}$, corresponding to variables a and d of the numerator and a, b, c and d of the denominator, are reported in the following equations:

$$PTMC = \frac{2a^3 + 0.1\sqrt{d}}{1.71a^2 + 1.98b^2 + 1.78c^2 + 0.1d} \tag{3.1}$$

$$PTMM = \frac{2a^3 + 0.85d}{1.64a + 1.95\sqrt{b} + 0.1c + 0.9d} \tag{3.2}$$

$$PTEM = \frac{0.48\sqrt{a} + 1.56d^2}{1.82a + 1.89b + 1.87c^2 + 0.47d^2} \tag{3.3}$$

4 EVALUATION

In order to evaluate the efficiency of the proposed coefficients, this chapter compares the precision of the respective coefficients with other 18 coefficients in the literature (Table 2.2), involving the other 101 systems of the *Qualitas.class Corpus* (test set). In this way, we performed an evaluation with a huge data set that includes very distinct systems, which is an outstanding and unusual approach from other researches. Therefore, we analyzed and compared each proposed coefficient according to its respective code refactoring, i.e., this chapter presents a different comparison for *Move Class*, *Move Method*, and *Extract Method*.

The remainder of this chapter is organized as follows. Section 4.1 presents the overall results of the comparison between the proposed coefficients and the 18 similarity coefficients covered in this master dissertation. Sections 4.2, 4.3, and 4.4 analyse the precision rates and the improvements for the coefficients in *Move Class*, *Move Method*, and *Extract Method* refactorings, respectively. Finally, Section 4.5 discusses the threads to validity.

4.1 Results

For the evaluation of the results, we adopted the same analysis approach described in Section 3.2, but now to the 101 systems of test set. Again, we applied all analyzed similarity coefficients to each entity of the target systems, analyzing whether an entity has the highest similarity with its enclosing entity or if a method should be extracted. Then, we calculate the arithmetic mean for each system of test set, and finally, we performed the analysis over the median of these sets of means. Therefore, Table 4.1 reports the results of the precision rates of coefficients analyzed for *Move Class* (including *PTMC*), *Move Method* (including *PTMM*), and *Extract Method* (including *PTEM*) refactorings. Again, the analysis considers the median of the sets of means since some data do not follow a normal distribution. For space constraints, the detailed table comprehending the results of each of the 101 systems is presented in Appendix B.

4.2 Move Class

The proposed coefficient reached higher values compared to the others, having an approximate median of 62.20%, 71.52%, and 79.45% in relation to the similarity rates *Top #1*, *#2*, and *#3*. *PSC* presented the second best median among the coefficients analyzed. For *Top #1*, *#2*, and *#3*, *PSC* reached, respectively, the precision values of 55.98%, 71.26%, and 77.31%.

Table 4.1 – Precision of the 19 similarity coefficients (test set)

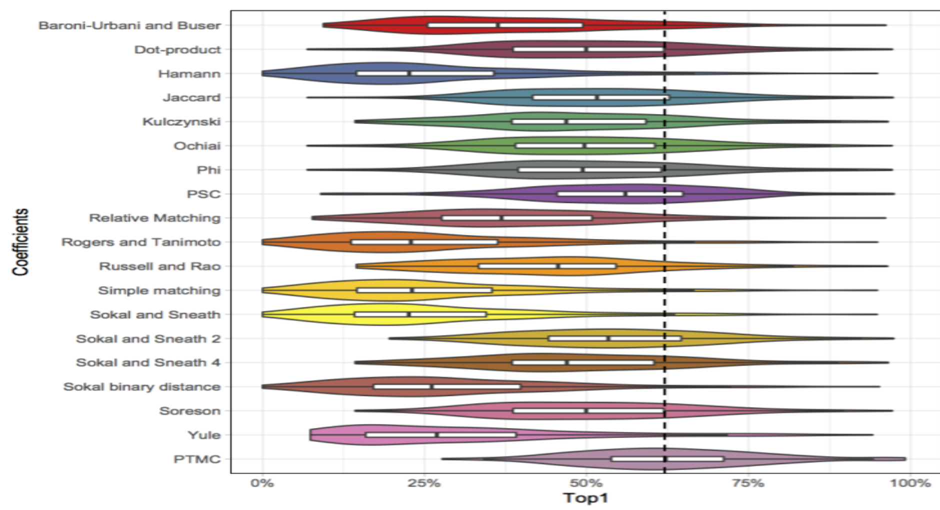
MOVE CLASS			
Coefficient	**Top1**	**Top2**	**Top3**
Baroni-Urbani and Buser	36.31%	48.65%	57.17%
Dot-product	49.93%	65.22%	72.87%
Hamann	22.62%	32.18%	38.77%
Jaccard	51.60%	67.52%	75.48%
Kulczynski	46.86%	64.15%	72.61%
Ochiai	49.66%	64.94%	72.47%
Phi	49.39%	65.22%	72.35%
PSC	*55.98%*	*71.26%*	*77.31%*
Relative Matching	36.86%	54.35%	63.71%
Rogers and Tanimoto	22.88%	32.18%	38.88%
Russell and Rao	45.59%	61.90%	71.31%
Simple matching	23.05%	33.18%	39.23%
Sokal and Sneath	22.56%	32.25%	38.64%
Sokal and Sneath 2	53.35%	69.05%	76.59%
Sokal and Sneath 4	46.95%	63.99%	72.54%
Sokal binary distance	26.09%	35.69%	44.36%
Sorenson	49.93%	65.22%	72.87%
Yule	26.88%	38.64%	47.79%
PTMC	62.20%	71.52%	79.45%

MOVE METHOD			
Coefficient	**Top1**	**Top2**	**Top3**
Baroni-Urbani and Buser	17.09%	25.08%	29.97%
Dot-product	31.25%	41.45%	49.23%
Hamann	20.21%	27.18%	31.61%
Jaccard	35.43%	46.60%	53.09%
Kulczynski	25.33%	36.09%	42.79%
Ochiai	29.30%	41.45%	47.49%
Phi	29.46%	41.29%	47.41%
PSC	36.71%	48.45%	55.98%
Relative Matching	23.23%	34.41%	40.39%
Rogers and Tanimoto	20.36%	27.47%	31.81%
Russell and Rao	30.70%	43.51%	49.86%
Simple matching	20.25%	27.04%	31.59%
Sokal and Sneath	19.56%	26.37%	30.46%
Sokal and Sneath 2	*38.02%*	*49.35%*	*56.04%*
Sokal and Sneath 4	24.35%	35.54%	42.94%
Sokal binary distance	23.75%	30.91%	35.49%
Sorenson	31.25%	41.45%	49.23%
Yule	9.45%	17.38%	22.25%
PTMM	52.89%	61.09%	64.66%

EXTRACT METHOD	
Coefficient	**Hits**
Baroni-Urbani and Buser	61.94%
Dot-product	69.62%
Hamann	10.48%
Jaccard	71.73%
Kulczynski	78,36%
Ochiai	72.95%
Phi	72.68%
PSC	76.11%
Relative Matching	82.46%
Rogers and Tanimoto	10.52%
Russell and Rao	*87.93%*
Simple matching	10.48%
Sokal and Sneath	10.32%
Sokal and Sneath 2	74.67%
Sokal and Sneath 4	77.27%
Sokal binary distance	16.52%
Sorenson	69.62%
Yule	65.90%
PTEM	88.32%

More importantly, when comparing the *Top #1* of the coefficients, *PTMC* presented a statistically significant improvement from 5.23% to 6.81%, according to the paired Wilcoxon Test with a 95% confidence (*p-value* = 2.2^{-16}). Similarly as illustrated in Section 3.2, Figure 4.1 presents a violin plot regarding Top #1 of each *Move Class* refactoring, in order to provide a complementary analysis between the data.

Figure 4.1 – *Move Class* plot



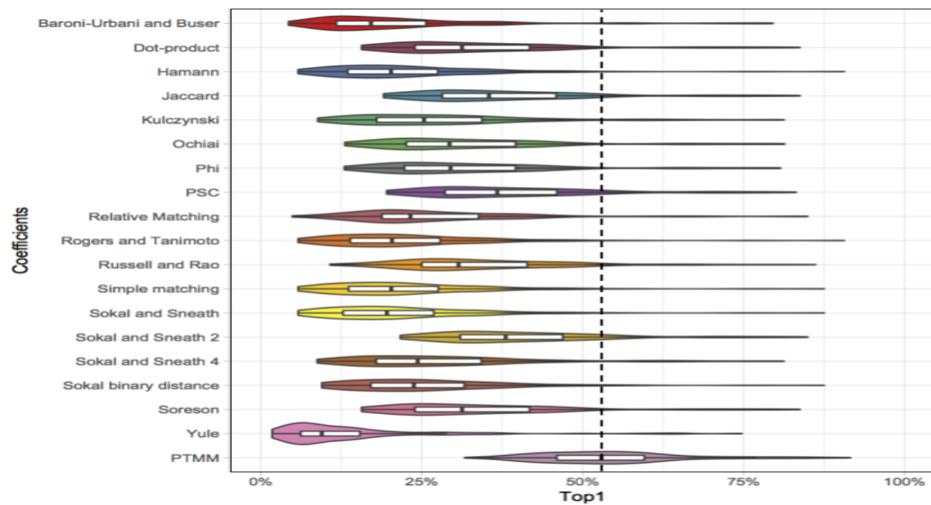
4.3 Move Method

PTMM surpassed the other coefficients data in the three cases of similarity rate (*Top #1*, *#2*, and *#3*), reaching, respectively, a median of approximately 52.89%, 61.09%, and 64.66%. *Sokal and Sneath 2* had the second highest rate where, considering the three cases of similarity rate (*Top #1*, *#2* and *#3*), the coefficient reached, respectively, the median values of 38.02%, 49.35%, and 56.04%. Moreover, *PTMM* presented a statistically significant improvement from 12.33% to 14.74%, when comparing *Top #1* of the coefficients, according to the paired Wilcoxon Test with a 95% confidence (*p-value* = 2.2^{-16}). Again, Figure 4.2 presents a violin plot regarding Top #1 of each *Move Method* refactoring.

4.4 Extract Method

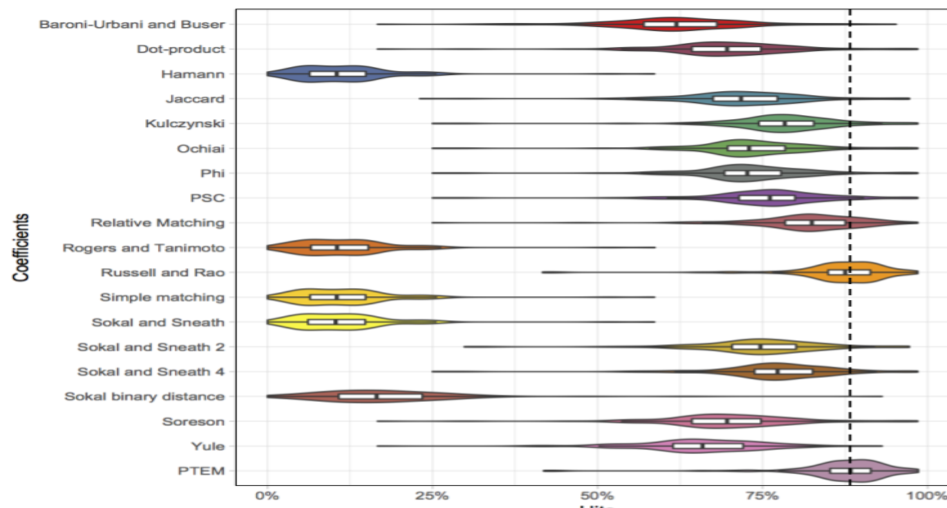
PTEM presented a median of approximately 88.32%, surpassing the second best coefficient (*Russell and Rao*), which had a median of 87.93%. In addition, *PTEM* presented a statistically significant improvement from 0.24% to 0.40% over *Russell and Rao*, according

Figure 4.2 – *Move Method* plot



to the paired Wilcoxon Test with a 95% confidence (*p-value* = 2.2^{-16}). Finally, Figure 4.3 presents a violin plot regarding each *Extract Method* refactoring.

Figure 4.3 – *Extract Method* plot



4.5 Threats to Validity

Since the experimental rules play an important role on the results, one could question their underlying design (construct validity). However, each rule has its rationale properly explained and justified in Section 3.1.1.

Moreover, as it is common in empirical studies in Software Engineering, the results cannot be extrapolated (external validity). Although the test set has a reasonable number of 101 systems, we acknowledge it is important to evaluate the new coefficients in real development scenarios, which we describe as future work.

5 RECOMMENDATION SYSTEM

In order to use the new coefficients proposed in this master dissertation to identify refactoring opportunities, we developed *AIRP*[1], a plug-in prototype for IDE Eclipse. Regarding its design and implementation, *AIRP* is composed of an architecture based on the following four main modules:

- *Dependency Extraction Module*: Responsible for identifying and storing all dependencies of a class, method, or block. In other words, it identifies the set of types at which a code entity establishes structural dependency. This includes method calling, attribute access, instantiation, variable declaration, annotation, etc. In order to perform such extraction, we used a syntactic analysis through the AST (*Abstract Syntax Tree*) that checks each element in the source code and, if it refers to a dependency to other entities, stores it according to its package, class, method, and/or block;

- *Similarity Calculation Module*: It calculates the structural similarity of a given code entity in the target system. This calculation is performed using the formula of the chosen similarity coefficient, making a comparison between the previously stored dependencies of a given class, method, or block with its respective entity (package, class, or method), as described in the methodology of this dissertation (See Section 3.1.1). By default, it uses the $PTMC$, $PTMM$ and $PTEM$ coefficients, although it is possible to select any of the other 18 coefficients presentes in Table 2.2;

- *Recommendation Module*: This module calls the previous one to calculate the resulting similarity of every possible *Move Class*, *Move Method*, and *Extract Method* refactorings in the code. After calculating similarity, its results are compared with a minimum acceptance index of improvement specified by the user. If the result is higher than this user-defined *threshold*, a list of suggestions is presented with possible refactoring opportunities. Afterwards, it reports the recommendations for the user analysis, being ordered by their similarity index improvement. If the user does not specify a minimum acceptance index, all refactorings that leads to an improvement are displayed; and

[1] *AIRP* is publicly available at: <https://github.com/pqes/AIRP>

- *Visualization Module*: In order to provide more details on the structural organization of the target system, this module generates a refactoring opportunity graph using the Zest[2] library, as illustrated in Figure 5.1. This graph contains the implemented architecture and reports all refactorings in the list of suggestions from previous module. The module also highlights the selected refactoring by the user, showing more details about it, and its respective resulting similarity upgrade. In this way, it makes possible a greater understanding by the user regarding the process performed by the tool, being able to observe the repositioning of involved entities and the architecture resulting from each recommendation.

In order to identify refactoring opportunities, *AIRP* receives one project as input and then every instruction in the code is analyzed and parsed by an AST to identify and map every dependency of the project (*Dependency Extraction Module*). Next, *AIRP* calculates the similarity of every entity in the code after each possible refactoring, as described in Section 3.1.1 (*Similarity Calculation Module*). Then, if a resulting similarity from a refactoring is higher than the previous similarity, the respective refactoring, as well as the improved similarity value, is added to a sorted recommendation list (*Recommendation Module*). Finally, the respective list is presented to the user together with a graph that illustrates the implemented architecture and the possible refactorings in the given list (*Visualization Module*).

It is worth noting that *AIRP* only makes refactoring suggestions to the developers analyze them before actually performing them. According to Szöke et al. (SZŐKE et al., 2015), applying code refactorings automatically—without previous analyses by humans—may impact negatively on the system's maintainability.

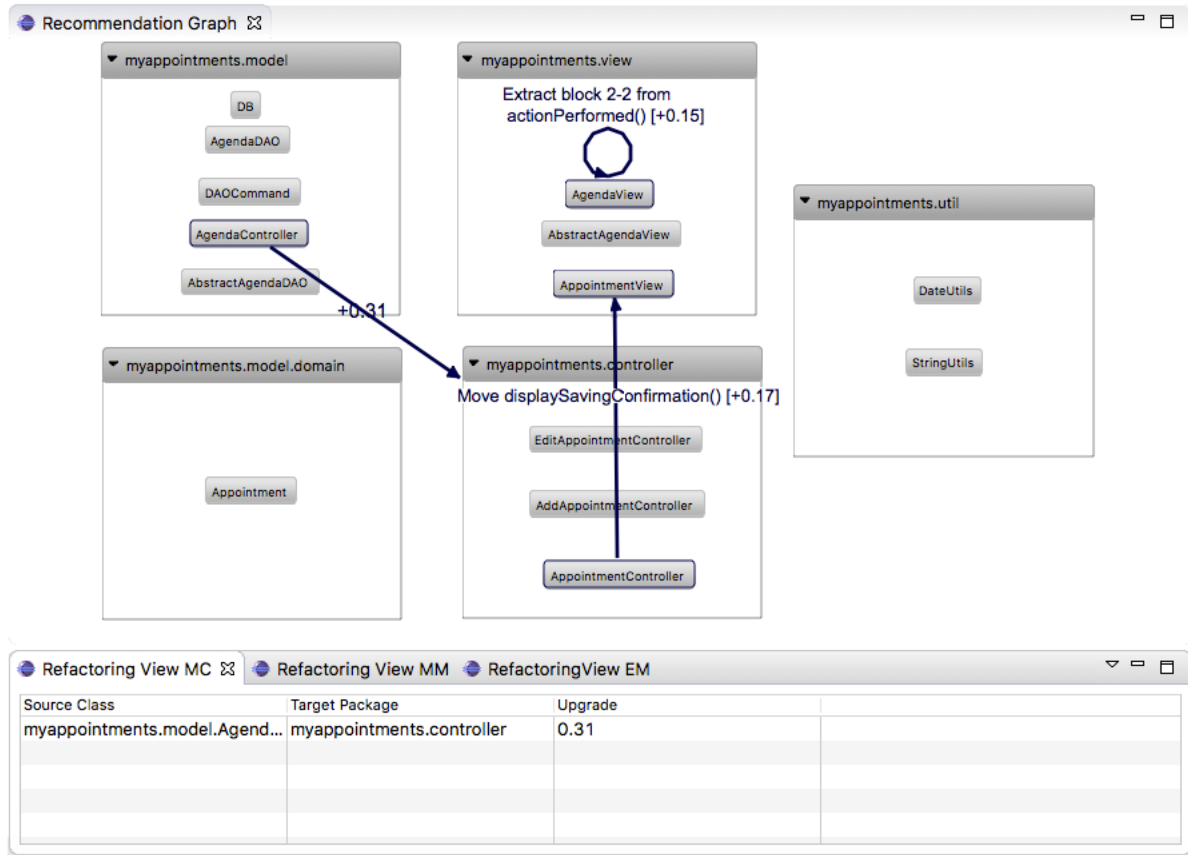
For a better visualization of the proposed tool, we executed *AIRP* in an example involving a modified version (in order to create the refactoring suggestions) of myAppointments (PASSOS et al., 2010), a personal information management MVC-based system with modules *Model*, *View*, *Controller*, and *Util*.

Figure 5.1 illustrates three examples of suggestions for refactoring identified by *AIRP*. For the modified version of myAppointments, class *AgendaController* has a similarity mean value 0.31 higher with the classes of package *myappointments.controller* (*Move Class*), method *displaySavingConfirmation*() from class *AppointmentController* has a similarity mean value 0.17 higher with methods from *AppointmentView* (*Move Method*),

2 <https://www.eclipse.org/gef/zest/>

and method *actionPerformed* results in a similarity mean value 0.15 higher with methods from class *AgendaView* if one of its inner blocks is extracted to a new method (*Extract Method*).

Figure 5.1 – Refactoring opportunity graph



6 RELATED WORK

This chapter discusses the most relevant studies related to this master dissertation. Although we found only one study regarding the proposal of new similarity coefficients in the last decades, some studies present methodologies, techniques, and tools for the identification of refactoring opportunities, or empirical studies regarding the concepts discussed in this work. Considering that the objective of this dissertation is to improve the identification of refactoring opportunities through similarity coefficients, we focused on studies that use such coefficients.

The remainder of this chapter is organized as follows. Section 6.1 presents an adaptation of *Jaccard* coefficient. Section 6.2 describes empirical studies about the evaluation of similarity coefficients and automatic code refactorings. Section 6.3 introduces robust systematic reviews of the literature about refactoring operations. Finally, Section 6.4 discusses studies about well-known tools used to treat Code Smells through code refactoring.

6.1 Similarity Coefficients

Naseem et al. (NASEEM; MAQBOOL; MUHAMMAD, 2010) propose a new similarity coefficient aimed at its application in clustering algorithms. In turn, the new proposed coefficient is an adaptation of the *Jaccard* coefficient, called *Jaccard-NM*, whose adaptation was done by the simple addition of a new variable (n) that considers the universe of analyzed factors, i.e., all the existing factors in the set in which analyzed entities are present. The proposed coefficient is defined as follows.

$$\textit{Jaccard-NM} = \frac{a}{a + b + c + n} \tag{6.1}$$

$$n = a + b + c + d \tag{6.2}$$

$$\begin{aligned} \textit{Jaccard-NM} &= \frac{a}{a + b + c + (a + b + c + d)} \\ &= \frac{a}{2(a + b + c) + d} \end{aligned} \tag{6.3}$$

It is important to emphasize that the proposed new similarity coefficient is compared only to the original *Jaccard* coefficient from which it was adapted and in only three systems. The study selected seven pre-defined types of relationships between classes of a system and

applied both coefficients to cluster classes from the following systems: Printer Language Converter, Fact Extractor System, and Statistical Analysis Visualization Tool. The results showed that *Jaccard-NM* had a better performance on two of the three analyzed systems. It was 30% better in the first case and 8% in the second. For the third case, however, *Jaccard-NM* was 5% worse than *Jaccard*.

On the other hand, this master dissertation has an in-depth analysis of the proposed coefficients for a more precise identification in relation to other existing coefficients. For structural similarity calculation, we considered all the main types of dependencies between code blocks, methods, classes, and packages of a system. We also provided a comparison of our proposed coefficients with other 18 of the main coefficients in literature, as well as an evaluation in 101 systems.

6.2 Empirical Studies

Terra et al. (TERRA et al., 2013a) perform a robust evaluation in 111 systems of the base *Qualitas.class Corpus*, involving 18 of the main similarity coefficients. Considering the study purpose, it is fundamental to the accomplishment of this dissertation since the used coefficients and their analyses, besides the approach involving structural dependencies, acted as the main basis for the proposal of new coefficients, as well as their evaluations.

In a nutshell, the study came up with four main findings: (i) structural dependencies are precise enough to indicate whether a class is located in the correct package, e.g., *Relative Matching*, *Kulczynski*, *Russell and Rao*, and *Sokal and Sneath 4* indicate, in the worst scenario, over than 70% of precision; (ii) consider the multiplicity of dependencies does *not* improve the overall precision, i.e., strategies that use the traditional set performs better than multiset-based for all coefficients; (iii) consider the dependency type does not improve the overall precision, i.e., treating the dependencies as a single type provides better results for all coefficieents, except for *Russell and Rao* and *Sokal and Sneath 4* that presented results slightly better using the dependency type; and (iv) *Relative Matching* and *Russell and Rao* were the most suitable coefficients to measure the similarity among classes of object-oriented systems, reaching a precision of 60.83% and 60.27%, respectively.

It can be said that, not only their results, but also their methodology played a major role for our methodology design. For example, we considered the same 18 coefficients in literature,

as well as we applied a similar structural similarity calculation. Also, some, but not all, of our experimental rules was based in their experimental setup.

Szöke et al. (SZŐKE et al., 2015) present a case study where it is discussed whether automatic code refactorings actually improves the overall maintainability of software systems. They analyze the impacts of different types of automatic refactoring on four software projects in real development scenarios. They also investigate the impacts of these refactorings in several quality metrics that are used in the maintainability model, e.g., LLCOM (Logical Lines Of Code), NOA (Number Of Ancestors), and CC (Clone Coverage). Finally, they evaluate the acceptance by the developers. Although automatic refactoring presented an overall system maintainability improvement in three of the four analyzed systems, manual refactoring presented an overall improvement in all systems. According to the authors, this occurs because despite the automatic refactoring tool provides complementary options, developers usually rely on default refactoring settings, which is easeful but may not be the more appropriate.

Although the study reports that refactorings usually have a positive impact on maintainability, their execution without proper analysis can negatively impact the system. This finding was essential to understand that automatic refactorings might not always be the best approach to be considered. Although it is extremely important to investigate the actual use of automatic refactorings, this master dissertation proposes coefficients that identify more precisely refactoring opportunities that may or *may not* be amenable to automatic refactoring.

Mark Harman (HARMAN, 2007) presents a review about widely used search-based techniques and its application in software engineering. The study describes three main search-based algorithms: Hill Climbing, Simulated Annealing, and Genetic Algorithms. The three main types of Hill Climbing are: (i) First-Ascent Hill Climbing, which iteratively searches for neighbouring solutions until a higher quality solution is discovered; (ii) Steepest-Ascent Hill Climbing, which examines all neighbouring solutions and moves to the solution of highest quality; and (iii) Multiple-Restart Hill Climbing, which repeat multiple times the search for the optimal solution with different initial condition. On the other hand, Simulated Annealing makes a series of tentative changes to find the best solution, but also accept changes that reduce the quality of the solution in order to escape from local minima. Finally, Genetic Algorithms searches for a solution to a problem through the crossover, selection and mutation of an initial population of possible candidates, as describes by this master dissertation in Section 2.5.1. Although this dissertation does not focus on the use of search-based algorithms, this work be-

comes relevant since such algorithms can be used in experiments that propose new coefficients or as a different way of identifying refactoring opportunities. Moreover, these algorithms can be eventually used for a comparison or cross-validation with the techniques applied by this dissertation.

6.3 Systematic Reviews of Literature

Dallal (DALLAL, 2015) presents a systematic review of the literature regarding code refactoring, which addresses 47 studies on the types of refactoring activities, the different approaches to identify refactoring opportunities, as well as the data sets and the means used to evaluate them.

In a nutshell, the study came up with four main findings: (i) *Move Method*, *Extract Class*, and *Extract Method* were the most considered refactoring activities (more than 20% of the primary studies); (ii) the most used approaches to identify refactoring opportunities were quality metrics-oriented approach (31.91%), precondition-oriented approach (23.40%), clustering-oriented approach (23.40%), graph-oriented approach (14.89%), code slicing-oriented approach (4.25%), and dynamic analysis-oriented approach (2.13%); (iii) the most used approaches to empirically evaluate the refactoring opportunities techniques were intuition-based evaluation (48.94%), quality-based evaluation (29.79%), mutation-based evaluation (25.53%), comparison-based evaluation (27.66%), behavior-based evaluation (6.38%), and 12.77% of the primary studies does not evaluate the refactoring candidates, only their applicability to one or more systems; and (iv) JHotDraw were the most used data set to evaluate the identification techniques, followed by Apache Ant and ArgoUML.

Although their results present a huge variety of approaches to be followed, such review was highly relevant in this dissertation since the performed analyses allowed us to compare several approaches and brought to us the most appropriate one to be considered, as well as allowed us to better understand the process of identifying code refactoring opportunities, and also their implementation and evaluation.

6.4 Identification of Refactoring Opportunities

Although this dissertation relies on a similarity coefficient approach to identify refactoring opportunities, there are several papers that present approaches and tools that use search-

based algorithms or other techniques to identify such opportunities. Section 6.4.1 presents related works that rely on similarity coefficients. Section 6.4.2 presents the main studies in literature that rely on search-based techniques. Section 6.4.3 presents related works that rely on other techniques.

6.4.1 Approaches Based on Similarity Coefficients

Terra et al. (TERRA et al., 2018) proposed JMove, an Eclipse IDE plug-in indicating *Move Method* refactoring opportunities based on the structural similarity coefficient *Sokal and Sneath 2*. It is important to note that the tool follows a structural similarity calculation similar (but more sophisticated to avoid false positives) to the one used in this master dissertation. The tool, however, involves analysis only of *Move Method*. More important, the use of the $PTMM$ coefficient proposed in this dissertation could improve the precision of JMove up to 14.74%.

Silva et al. (SILVA; TERRA; VALENTE, 2014) proposed JExtract, an Eclipse IDE plug-in to identify *Extract Method* refactoring opportunities based on *Kulczynski* similarity coefficient. This tool also follows a similar structural similarity calculation, having also changes in its rules to avoid false positives. JExtract, in contrast to JMove, focuses only on the *Extract Method* refactoring. Again, we argue that the use of the $PTEM$ coefficient proposed in this dissertation could improve precision of JExtract around 9.96%.

Despite the fact that this dissertation has the main objective to propose new coefficients, we also developed a tool, named *AIRP*, to identify opportunities of the both previously mentioned refactorings, as also *Move Class*. *AIRP* applies our proposed coefficients, which can lead to the previously mentioned improved precision. Futhermore, *AIRP* allows the user to change the selected similarity coefficient to one of the 18 addressed in this dissertation, which includes those used by JMove and JExtract.

Tsantalis and Chatzigeorgiou (TSANTALIS; CHATZIGEORGIOU, 2009a; TSANTALIS; CHATZIGEORGIOU, 2009b) present the JDeodorant tool, which identifies Code Smells and applies different code refactoring techniques in order to treat them. JDeodorant does not use the similarity comparison between the dependencies of a project. On the other hand, it uses the *Jaccard* coefficient to calculate only the similarity between attributes and methods of the analyzed classes, which can affect its effectiveness since *Jaccard*—although it is one of the most used coefficients in Software Engineering—does not present good precision in relation to other existing coefficients.

6.4.2 Approaches Based on Search-Based Techniques

O'Keeffe and Cinnéide (O'KEEFFE; CINNÉIDE, 2008) present a study to demonstrate that object-oriented programs can be automatically refactored using a search-based approach to conform more closely to a given quality model. For this purpose, they developed CODe-Imp, a prototype tool that can be configured to perform several automated refactorings, and can be operated using different search-based techniques, as well as different evaluation functions based on combinations of established metrics.

In order to find and demonstrate the best approach to automatically refactor code entities, they performed an experimental investigation in relation to the search-based techniques in two Java programs. The main idea was to use different search-based algorithms to identify and apply a set of defined refactorings, and then evaluate and compare their quality improvement through functions that use QMOOD metrics. Their experiment analyzed the following set of algorithms: First-Ascent Hill Climbing, Steepest-Ascent Hill Climbing, Multiple-Restart Hill Climbing, and Low-Temperature Simulated Annealing. Although they applied a set of 14 code refactorings, none of them was the refactorings addressed in this dissertation (*Move Class*, *Move Method*, and *Extract Method*).

The results and findings are presented as two case studies, one for the comparison of search-based techniques and the other for the comparison of the evaluation functions, i.e., the comparison between each QMOOD metric analyzed. The results showed that First-Ascent Hill Climbing consistently produced quality improvements at a relatively low cost, Steepest-Ascent Hill Climbing produced the greatest mean quality improvements in two of the six cases, Multiple-Ascent Hill Climbing produced individual solutions of highest quality in two cases, and Simulated Annealing produced the greatest mean quality improvement in one case. In respect to the QMOOD evaluation functions, results provided some evidence in favour of use of the QMOOD Flexibility function, and strong evidence in favour of use of the Understandability function. The QMOOD Reusability function was not found to be suitable to the requirements of search-based software maintenance because it resulted in solutions including a large number of featureless classes.

Mkaouer et al. (MKAOUER et al., 2016) proposes an approach to identify and perform code refactorings based on the many-objective genetic algorithm NSGA-III and the QMOOD model. Their approach relies on a genetic algorithm that takes as the population all legal refactoring sequences that can be applied in a system, i.e., each possible candidate to the solution

represents a sequence of code refactorings. The genetic algorithm defines eight objective functions; more specifically, six objectives for the each QMOOD model quality measure along with two other objectives to reduce the number of refactorings to apply and maximize the design coherence after refactoring. Finally, the output solution is the one that best satisfies all the objectives at the same time.

An evaluation on seven large open-source systems and one industrial project compares the findings to other many-objective techniques (IBEA, MOEA/D, GrEA, and DBEA-Eps), a mono-objective genetic algorithm, and also the JDeodorant tool. The results reported the best performance for their approach considering each of the QMOOD model quality measures in most of the analyzed executions.

6.4.3 Approaches Based on Other Techniques

Czibula and Czibula (CZIBULA; CZIBULA, 2008) propose two agglomerative hierarchical clustering algorithms for identifying refactorings in order to recondition the class structure of software systems. The study relies on a clustering-oriented approach to identify refactoring opportunities for *Move Method*, *Move Attribute*, *Inline Class*, and *Extract Class*. Both proposed algorithms seek to group the analyzed entities based on the distance between their internal properties. The first algorithm groups the entities while a minimum distance value (threshold) is satisfied, while the second algorithm groups the entities in a predefined number of clusters that is calculated based on the most distant entities.

In order to validate their clustering approach, they analyzed three cases: (i) a controlled scenario evaluation with code examples; (ii) a evaluation in JHotDraw system; and (iii) a real scenario evaluation in a distributed system that consists of several subsystems in form of standalone and web-based applications. While the first two cases was only manually analyzed by the authors where they claimed that the results was satisfactory, the real scenario evaluation was analyzed by the system's developers where 17 identified refactorings were accepted by the developers as useful in order to improve the system, 13 refactorings were acceptable for the developers, but they concluded that these refactorings are not necessary in the current stage of the project, and 26 refactorings were strongly rejected by the developers. They concluded that although the approach had some rejected refactorings, it was able to identify refactoring opportunities to improve an architecture quality. Moreover, their fast execution—i.e., the overall running time—is an important advantage in comparison with other techniques.

Kimura et al. (KIMURA et al., 2012) propose a technique to find refactoring opportunities relying on a dynamic analysis-oriented approach. The authors analyzed method traces (invocations of methods during program execution) to identify *Move Method* refactoring opportunities. The proposed approach divides all method traces into multiple sets representing different phases, and then it counts the invocations of each method in the same phase. Next, it creates a graph showing colored patterns of each method in each phase and finally, the refactoring opportunities are detected according to the investigation of which methods have different patterns, i.e., methods with a different pattern should be moved to another class.

The proposed technique was evaluated on two Java systems (FRISC and MASU). They identified seven *Move Method* opportunities candidates for FRISC, being all approved by the developers. On the other hand, they identified 58 *Move Method* opportunities candidates for MASU, where 39 were approved by the developers.

Although some of the analyzed studies focus on the identification of code refactoring opportunities or on approaches involving structural similarity coefficients, we were unable to find other studies that propose new coefficients aiming at greater precision in the identification of such opportunities. Therefore, the proposal of new coefficients—based on statistics, robust analyses, and comparisons between the main existing similarity coefficients focused on three different types of refactoring—emphasizes the originality of this master dissertation.

7 CONCLUSION

Identifying code refactoring opportunities is essential for a developer to maintain a well-defined software architecture, as well as high cohesion and low coupling. Among the existing techniques, some rely on the structural similarity calculation. However, the main similarity coefficients proposed in literature are not designed to deal with object-oriented code structures, which may achieve low precision and non-realistic rates.

To address such problem, this article proposes three new structural similarity coefficients to provide more precise ways in the identification of code refactoring opportunities. We analyzed the main similarity coefficients proposed in literature in a training set of 10 well-designed systems, and we improved and adapted the most adequate coefficient through an experiment with a genetic algorithm and multiple comparisons. We therefore come up with *PTMC*, *PTMM*, and *PTEM* for identification of *Move Class*, *Move Method*, and *Extract Method* refactoring opportunities, respectively.

In an evaluation on another 101 well-designed systems, the proposed coefficients are more accurate than the state-of-the-art coefficients used to identify refactoring opportunities. In numerical terms, *PTMC*, *PTMM*, and *PTEM* showed a statistically significant improvement, respectively, from 5.23% to 6.81%, 12.33% to 14.79%, and 0.25% to 0.40%, in relation to the second best coefficients. These results can impact in many techniques that relies on structural similarity calculation, for example, the precision of the JMove and JExtract tools could be improved up to 14.74% and 9.96%, respectively. We also presented *AIRP*, a tool to identify refactoring opportunities based on proposed coefficients.

We organized this chapter as follows. First, Section 7.1 reviews the contributions of our master dissertation. Next, Section 7.2 points the limitations of this work. Finally, Section 7.3 presents the further work.

7.1 Contributions

This dissertation makes the following contributions:

- Three new coefficients proposed to identify refactoring opportunities for *Move Class*, *Move Method*, and *Extract Method* with a statistically significant improvement, respectively, from 5.23% to 6.81%, 12.33% to 14.79%, and 0.25% to 0.40% w.r.t. the best existing state-of-the-art coefficients. (Section 3.3 and Chapter 4);

- A set of nine experimental rules to avoid the occurrence of false positives in the identification of refactoring opportunities based on structural similarity calculation of object-oriented systems (Section 3.1.1);
- An analysis and comparison about the precision related to *Move Class*, *Move Method*, and *Extract Method* refactoring of 18 of the main similarity coefficients in literature, as well as the proposed coefficients in 101 open-source systems (Chapter 4); and
- A tool called *AIRP* that identify refactoring opportunities for *Move Class*, *Move Method*, and *Extract Method*, as also implements the proposed coefficients and presents a refactoring opportunity graph for a better user visualization (Chapter 5).

7.2 Limitations

This master dissertation has the following limitations:

- Although our empirical experiment has a reasonable number of executions, i.e., number of replications times groups of factors, this number was limited due to computational resources, which might affect its performance;
- The evaluation of the similarity coefficients was made in a controlled scenario of 101 systems. And although the test set has a reasonable number of well-designed systems, it is important to evaluate the coefficients in real development scenarios; and
- We have not evaluated whether our proposed coefficients provides better precision rates in comparison with techniques to identify refactoring opportunities that are not related to similarity calculation.

7.3 Future Work

In particular, we plan to complement this study with the following future work:

- Use more coefficients in the analysis stage. Even though we considered 18 of the main coefficients in literature, there are plenty more coefficients that could be used in our methodology, leading to an improved analysis and even the possibility of better results;

- Apply cross-validation techniques. In order to provide a more efficient methodology for the training and test stages, cross-validation techniques can be used to verify the performance of the used approach or even to find a better one. This might also improve the selection of the target systems, leading to a better training set;

- Perform a comparative study among techniques in order to identify refactoring opportunities that use or not similarity coefficients. In order to find the best approach to identify refactoring opportunities, we plan to conduct a study comparing the main techniques, including a comparison between our proposed coefficients and other studies;

- Analyze and compare the improvement of each proposed coefficient in practical terms, i.e., an empirical investigation about the precision leverage of our coefficient in existing tools considering how many opportunities or how much percent of improvement the coefficients provide indeed; and

- Measure the acceptance degree of new coefficients in real scenarios. A case study evaluating the identification of refactoring opportunities by our proposed coefficients in real development scenarios can lead to new findings, e.g., issues, changes to be done, efficiency, as well as new contributions.

REFERENCES

ABEBE, S. L. et al. Lexicon bad smells in software. In: **16th Working Conference on Reverse Engineering (WCRE)**. [S.l.: s.n.], 2009. p. 95–99.

BAEZA-YATES, R. A.; RIBEIRO-NETO, B. A. **Modern Information Retrieval: the concepts and technology behind search**. 2. ed. [S.l.]: Pearson, 2011.

BINKLEY, A. B.; SCHACH, S. R. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. In: **20th International Conference on Software Engineering (ICSE)**. [S.l.: s.n.], 1998. p. 452–455.

CHIDAMBER, S. R.; DARCY, D. P.; KEMERER, C. F. Managerial use of metrics for object-oriented software: An exploratory analysis. **IEEE Transactions on Software Engineering**, v. 24, n. 8, p. 629–639, 1998.

CHIS, M. **Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques**. [S.l.]: Information Science Reference, 2010.

CZIBULA, I. G.; CZIBULA, G. Hierarchical clustering based automatic refactorings detection. **WSEAS Transactions on Electronics**, v. 5, n. 7, p. 291–302, 2008.

DALLAL, J. A. Identifying refactoring opportunities in object-oriented code: A Systematic Literature Review. **Information and Software Technology**, v. 58, p. 231–249, 2015.

FOWLER, M. **UML Distilled: a Brief Guide to the Standard Object Modeling Language**. [S.l.]: Addison-Wesley, 2004.

FOWLER, M. et al. **Refactoring: Improving the Design of Existing Code**. [S.l.]: Addison-Wesley, 1999.

GOSLING, J. et al. **The Java Language Specification: Java SE 8 Edition**. [S.l.]: Oracle America, 2015.

HARMAN, M. The current state and future of search based software engineering. In: **Future of Software Engineering**. [S.l.: s.n.], 2007. p. 342–357.

JACCARD, P. The distribution of the flora in the alpine zone. **New Phytologist**, Wiley Online Library, v. 11, n. 2, p. 37–50, 1912.

KERIEVSKY, J. **Refactoring to Patterns**. [S.l.]: Addison-Wesley, 2004.

KIMURA, S. et al. Move code refactoring with dynamic analysis. In: **28th International Conference on Software Maintenance (ICSM)**. [S.l.: s.n.], 2012. p. 575–578.

KNOERNSCHILD, K. **Introduction to Java Application Architecture: Modularity Patterns with Examples Using OSGi**. [S.l.]: Prentice Hall, 2012.

MKAOUER, M. W. et al. On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. **Empirical Software Engineering**, v. 21, n. 6, p. 2503–2545, 2016.

NASEEM, R.; MAQBOOL, O.; MUHAMMAD, S. An improved similarity measure for binary features in software clustering. In: **2nd International Conference on Computational Intelligence, Modelling and Simulation (CIMSiM)**. [S.l.: s.n.], 2010. p. 111–116.

O'KEEFFE, M.; CINNÉIDE, M. Ó. Search-based refactoring for software maintenance. **Journal of Systems and Software**, Elsevier, v. 81, n. 4, p. 502–516, 2008.

PASSOS, L. et al. Static architecture conformance checking: An illustrative overview. **IEEE Software**, v. 27, n. 5, p. 132–151, 2010.

SCHILDT, H. **Java para Iniciantes**. [S.l.]: Bookman, 2015. 209–215 p.

SILVA, D.; TERRA, R.; VALENTE, M. T. Recommending automated Extract Method refactorings. In: **22nd International Conference on Program Comprehension (ICPC)**. [S.l.: s.n.], 2014. p. 146–156.

SILVA, D.; TSANTALIS, N.; VALENTE, M. T. Why we refactor? confessions of GitHub contributors. In: **24th International Symposium on Foundations of Software Engineering (FSE)**. [S.l.: s.n.], 2016. p. 858–870.

SIVANANDAM, S.; DEEPA, S. N. **Introduction to Genetic Algorithms**. [S.l.]: Springer Science & Business Media, 2007.

SZŐKE, G. et al. Do automatic refactorings improve maintainability? an industrial case study. In: **31st International Conference on Software Maintenance and Evolution (ICSME)**. [S.l.: s.n.], 2015. p. 429–438.

TERRA, R. et al. Measuring the structural similarity between source code entities. In: **25th International Conference on Software Engineering and Knowledge Engineering (SEKE)**. [S.l.: s.n.], 2013. p. 753–758.

TERRA, R. et al. Qualitas.class Corpus: A compiled version of the Qualitas Corpus. **Software Engineering Notes**, v. 38, n. 5, p. 1–4, 2013.

TERRA, R. et al. JMove: A novel heuristic and tool to detect move method refactoring opportunities. **Journal of Systems and Software**, v. 138, p. 19–36, 2018.

TSANTALIS, N.; CHATZIGEORGIOU, A. Identification of extract method refactoring opportunities. In: **13th European Conference on Software Maintenance and Reengineering (CSMR)**. [S.l.: s.n.], 2009. p. 119–128.

TSANTALIS, N.; CHATZIGEORGIOU, A. Identification of move method refactoring opportunities. **IEEE Transactions on Software Engineering**, v. 35, n. 3, p. 347–367, 2009.

APPENDIX A – Coefficient Precision in Training Set

Table 1 reports the results of the precision rates of the 18 analyzed coefficients w.r.t. the identification of *Move Class*, *Move Method*, and *Extract Method* refactoring opportunities in 10 systems of *Qualitas.class Corpus* (training set).

Table 1 – Precision rates of the 18 coefficients in 10 systems (training set)

MOVE CLASS															
	Ant			ArgoUML			Collections			Hibernate			JEdit		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	29.24%	33.73%	37.33%	16.59%	28.20%	31.51%	45.27%	72.43%	81.07%	21.89%	30.53%	35.38%	26.81%	47.10%	56.28%
Dot-product	37.18%	45.88%	50.52%	32.98%	40.39%	45.37%	60.08%	80.25%	86.83%	45.68%	58.37%	63.46%	54.11%	65.22%	72.22%
Hamann	23.99%	30.28%	31.63%	21.46%	25.76%	27.12%	43.62%	51.03%	55.14%	14.41%	17.54%	18.52%	13.29%	14.73%	17.15%
Jaccard	41.23%	48.13%	52.47%	35.32%	42.44%	47.51%	63.37%	80.25%	89.30%	48.99%	60.94%	65.85%	55.80%	66.43%	72.95%
Kulczynski	39.43%	45.28%	51.12%	32.59%	40.88%	45.85%	58.85%	76.54%	85.60%	43.78%	58.55%	63.09%	49.52%	63.77%	71.50%
Ochiai	38.98%	47.68%	51.12%	32.59%	40.98%	45.85%	58.44%	79.01%	86.83%	45.74%	59.47%	64.56%	53.14%	65.46%	71.98%
Phi	39.43%	47.68%	51.12%	32.59%	41.07%	46.15%	58.44%	79.01%	85.60%	45.74%	59.53%	64.56%	53.14%	64.98%	71.74%
PSC	45.28%	52.47%	59.82%	38.34%	48.20%	52.39%	62.96%	81.48%	90.12%	54.02%	66.03%	71.92%	58.70%	67.87%	79.23%
Relative Matching	26.24%	33.58%	39.13%	22.93%	35.51%	39.90%	56.79%	76.54%	83.95%	29.98%	46.11%	53.28%	41.06%	54.11%	63.53%
Rogers and Tanimoto	23.99%	30.43%	31.63%	21.46%	26.05%	27.32%	43.62%	51.44%	55.14%	14.47%	17.54%	18.58%	13.29%	14.73%	17.39%
Russell and Rao	30.43%	39.73%	45.73%	29.27%	41.46%	45.37%	60.08%	81.89%	87.65%	37.16%	54.32%	61.31%	47.34%	62.08%	70.77%
Simple matching	23.99%	30.28%	31.48%	21.46%	25.76%	27.02%	43.62%	51.03%	55.14%	14.35%	17.54%	18.52%	13.29%	14.73%	17.15%
Sokal and Sneath	23.99%	29.39%	31.33%	21.46%	25.76%	27.02%	43.62%	50.62%	55.14%	14.16%	17.35%	18.27%	12.56%	14.49%	16.91%
Sokal and Sneath 2	42.43%	49.48%	53.52%	37.17%	45.37%	49.17%	62.96%	80.66%	88.89%	50.89%	61.93%	67.32%	57.00%	66.67%	73.19%
Sokal and Sneath 4	39.43%	45.28%	50.97%	32.68%	40.98%	45.85%	58.85%	75.72%	84.77%	43.96%	58.55%	63.09%	50.24%	63.53%	71.50%
Sokal binary distance	25.34%	31.33%	33.13%	22.73%	26.34%	27.80%	44.44%	52.67%	57.61%	16.31%	20.48%	21.46%	14.01%	17.15%	20.53%
Sorenson	37.18%	45.88%	50.52%	32.98%	40.39%	45.37%	60.08%	80.25%	86.83%	45.68%	58.37%	63.46%	54.11%	65.22%	72.22%
Yule	22.64%	26.84%	29.24%	5.66%	8.98%	11.80%	43.21%	62.55%	74.49%	14.65%	21.89%	26.36%	21.01%	35.02%	43.48%

MOVE METHOD															
	Ant			ArgoUML			Collections			Hibernate			JEdit		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	11.71%	15.62%	17.28%	5.93%	9.31%	11.86%	21.71%	27.13%	29.92%	12.18%	17.19%	20.17%	13.70%	17.57%	22.57%
Dot-product	18.63%	26.35%	30.16%	15.15%	22.29%	26.92%	32.87%	40.16%	45.74%	24.62%	33.29%	39.36%	26.17%	35.29%	42.45%
Hamann	12.07%	19.70%	21.59%	8.57%	12.70%	16.82%	22.33%	27.75%	32.87%	14.64%	19.23%	21.59%	16.23%	21.23%	26.84%
Jaccard	21.27%	28.99%	33.26%	19.60%	27.66%	31.79%	38.29%	46.82%	51.16%	27.32%	36.48%	43.09%	29.83%	39.52%	33.43%
Kulczynski	15.08%	23.61%	27.33%	12.42%	19.74%	25.39%	22.79%	29.30%	34.26%	16.09%	22.74%	27.82%	18.29%	27.46%	36.27%
Ochiai	18.36%	26.08%	29.67%	14.27%	21.59%	26.27%	27.91%	35.19%	40.93%	22.26%	30.69%	36.28%	22.31%	32.56%	39.88%
Phi	18.40%	26.17%	29.67%	14.23%	21.50%	26.32%	27.75%	35.19%	40.93%	22.22%	30.61%	36.18%	22.31%	32.56%	39.88%
PSC	22.80%	31.69%	36.31%	21.83%	30.72%	35.40%	35.66%	46.05%	51.32%	27.66%	37.34%	44.57%	29.68%	41.78%	50.80%
Relative Matching	15.13%	20.78%	22.98%	11.86%	17.42%	20.85%	31.01%	38.45%	42.95%	19.75%	27.68%	32.83%	17.98%	25.45%	31.27%
Rogers and Tanimoto	12.39%	19.93%	21.81%	9.08%	12.79%	17.10%	22.33%	28.06%	33.02%	14.74%	19.47%	21.83%	16.33%	21.43%	27.25%
Russell and Rao	18.94%	26.53%	29.76%	18.72%	25.81%	29.80%	41.40%	49.77%	53.95%	26.00%	35.62%	41.55%	25.14%	36.22%	44.20%
Simple matching	12.21%	19.66%	21.54%	8.85%	12.65%	16.68%	22.02%	27.91%	32.71%	14.40%	19.15%	21.43%	16.28%	21.28%	26.79%
Sokal and Sneath	11.27%	18.99%	20.92%	8.25%	12.33%	16.36%	21.40%	27.44%	32.09%	13.86%	18.63%	20.93%	15.82%	20.71%	26.12%
Sokal and Sneath 2	23.43%	31.78%	36.36%	22.98%	31.23%	34.80%	40.78%	49.92%	53.64%	30.13%	39.28%	46.25%	32.20%	43.64%	50.95%
Sokal and Sneath 4	15.08%	23.65%	27.29%	12.47%	19.83%	25.35%	22.64%	29.15%	34.26%	16.01%	22.64%	27.68%	18.29%	27.51%	36.27%
Sokal binary distance	14.90%	21.68%	24.64%	13.53%	18.07%	21.69%	23.72%	30.85%	37.05%	16.65%	21.67%	24.12%	17.21%	23.60%	31.22%
Sorenson	18.63%	26.35%	30.16%	15.15%	22.29%	26.92%	32.87%	40.16%	45.74%	24.62%	33.29%	39.36%	26.17%	35.29%	42.45%
Yule	7.09%	11.98%	13.73%	4.12%	6.16%	8.57%	12.71%	17.98%	22.64%	6.05%	9.70%	12.58%	7.88%	14.12%	18.13%

EXTRACT METHOD					
	Ant	ArgoUML	Collections	Hibernate	JEdit
Coefficient	Hits	Hits	Hits	Hits	Hits
Baroni-Urbani and Buser	48.95%	52.46%	36.63%	59.75%	49.62%
Dot-product	51.83%	54.36%	37.21%	67.20%	55.22%
Hamann	12.65%	11.45%	40.70%	8.68%	9.29%
Jaccard	53.95%	57.54%	45.93%	71.45%	56.99%
Kulczynski	63.65%	61.36%	44.19%	77.15%	64.67%
Ochiai	56.07%	57.15%	40.12%	71.97%	58.83%
Phi	56.12%	56.97%	40.12%	71.88%	58.76%
PSC	59.54%	59.41%	48.84%	76.99%	61.21%
Relative Matching	65.48%	71.39%	40.12%	77.09%	64.21%
Rogers and Tanimoto	12.65%	11.28%	40.70%	8.65%	8.99%
Russell and Rao	72.69%	77.86%	43.02%	84.45%	72.58%
Simple matching	12.65%	11.34%	40.70%	8.75%	9.29%
Sokal and Sneath	12.56%	11.39%	40.12%	8.65%	9.29%
Sokal and Sneath 2	57.26%	59.53%	63.37%	75.76%	62.14%
Sokal and Sneath 4	63.61%	61.13%	43.02%	77.12%	64.44%
Sokal binary distance	21.37%	17.86%	45.93%	15.86%	16.51%
Sorenson	51.83%	54.36%	37.21%	67.20%	55.22%
Yule	55.75%	54.30%	41.28%	60.89%	53.76%

To be continued in the next page.

MOVE CLASS																
	JFreeChart			JHotDraw			JMeter			JUnit			TomCat			Mean		
Coefficient	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**
Baroni-Urbani and Buser	38.16%	46.71%	50.66%	32.22%	48.33%	54.03%	22.44%	30.88%	34.47%	18.67%	44.00%	52.00%	27.02%	36.79%	43.55%	**27.83%**	**41.87%**	**47.63%**
Dot-product	45.18%	53.07%	61.84%	45.78%	60.31%	67.39%	34.65%	44.17%	50.27%	36.00%	57.33%	62.67%	44.27%	54.04%	62.00%	**43.59%**	**55.90%**	**62.26%**
Hamann	26.75%	32.46%	38.82%	13.75%	19.45%	23.18%	15.62%	20.11%	21.18%	18.67%	34.67%	45.33%	11.94%	15.32%	16.04%	**20.35%**	**26.13%**	**29.41%**
Jaccard	46.05%	55.48%	64.69%	47.94%	62.87%	69.16%	35.37%	47.58%	54.22%	36.00%	61.33%	62.67%	46.56%	57.54%	63.81%	**45.66%**	**58.30%**	**64.26%**
Kulczynski	45.39%	54.82%	62.94%	42.63%	60.31%	65.62%	32.85%	44.17%	50.81%	29.33%	56.00%	64.00%	34.14%	50.66%	61.04%	**40.85%**	**55.10%**	**62.16%**
Ochiai	44.96%	54.17%	61.62%	44.99%	60.71%	66.99%	35.73%	44.88%	52.24%	37.33%	58.67%	64.00%	42.82%	54.76%	63.09%	**43.47%**	**56.58%**	**62.83%**
Phi	45.61%	54.39%	61.62%	45.38%	60.71%	67.39%	35.55%	45.42%	52.24%	34.67%	58.67%	62.67%	42.22%	55.01%	63.33%	**43.28%**	**56.65%**	**62.64%**
PSC	50.44%	60.75%	71.71%	48.13%	67.19%	74.26%	42.55%	53.14%	62.48%	42.67%	61.33%	65.33%	48.85%	63.33%	70.93%	**49.19%**	**62.18%**	**69.82%**
Relative Matching	39.47%	45.83%	52.19%	37.33%	52.06%	59.72%	26.21%	32.14%	37.88%	25.33%	45.33%	53.33%	36.67%	46.08%	52.23%	**34.20%**	**46.73%**	**53.52%**
Rogers and Tanimoto	26.75%	32.46%	39.04%	13.75%	19.84%	23.38%	15.80%	20.29%	21.36%	18.67%	34.67%	45.33%	11.94%	15.68%	16.41%	**20.37%**	**26.31%**	**29.56%**
Russell and Rao	41.89%	49.34%	57.46%	43.81%	56.78%	63.46%	31.60%	40.93%	46.68%	28.00%	52.00%	61.33%	43.43%	54.28%	60.92%	**39.30%**	**53.28%**	**60.07%**
Simple matching	26.54%	32.24%	38.60%	13.56%	19.84%	23.38%	15.80%	20.11%	21.01%	16.00%	33.33%	45.33%	12.06%	15.44%	16.28%	**20.07%**	**26.03%**	**29.39%**
Sokal and Sneath	26.54%	32.24%	38.38%	13.56%	19.45%	22.59%	15.26%	19.39%	20.29%	14.67%	33.33%	44.00%	11.22%	14.48%	15.44%	**19.70%**	**25.65%**	**28.94%**
Sokal and Sneath 2	47.81%	58.55%	66.89%	47.35%	64.83%	71.12%	38.06%	49.19%	57.09%	38.67%	61.33%	62.67%	47.65%	58.87%	65.26%	**47.00%**	**59.69%**	**65.51%**
Sokal and Sneath 4	45.61%	55.04%	63.16%	43.03%	60.71%	66.40%	32.85%	44.34%	50.99%	30.67%	56.00%	64.00%	34.14%	50.66%	61.04%	**41.15%**	**55.08%**	**62.18%**
Sokal binary distance	29.39%	35.31%	41.01%	14.93%	20.43%	26.33%	18.13%	22.80%	23.88%	18.67%	37.33%	46.67%	14.72%	19.66%	20.87%	**21.87%**	**28.35%**	**31.93%**
Sorenson	45.18%	53.07%	61.84%	45.78%	60.31%	67.39%	34.65%	44.17%	50.27%	36.00%	57.33%	62.67%	44.27%	54.04%	62.00%	**43.59%**	**55.90%**	**62.26%**
Yule	32.68%	42.32%	45.83%	21.61%	33.60%	40.47%	8.80%	14.90%	18.49%	16.00%	32.00%	40.00%	13.75%	20.27%	26.78%	**20.00%**	**29.84%**	**35.69%**

MOVE METHOD																		
	JFreeChart			JHotDraw			JMeter			JUnit			TomCat			Mean		
Coefficient	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**
Baroni-Urbani and Buser	30.18%	47.73%	60.49%	14.98%	21.06%	24.74%	12.07%	17.40%	20.06%	22.77%	30.36%	39.29%	15.34%	20.49%	26.05%	**16.06%**	**22.39%**	**27.24%**
Dot-product	53.21%	65.67%	72.39%	29.04%	39.57%	46.68%	23.61%	31.12%	37.57%	35.27%	44.64%	55.80%	32.32%	43.58%	50.76%	**29.09%**	**38.20%**	**44.78%**
Hamann	25.56%	36.13%	42.85%	18.71%	26.33%	30.11%	14.08%	18.82%	22.43%	25.89%	31.25%	38.84%	20.56%	27.57%	31.48%	**17.86%**	**24.07%**	**28.54%**
Jaccard	55.44%	66.87%	72.77%	32.67%	43.56%	50.05%	26.69%	36.69%	42.13%	39.29%	50.45%	62.50%	36.70%	48.50%	55.04%	**32.71%**	**42.55%**	**48.89%**
Kulczynski	50.17%	66.18%	73.16%	24.90%	35.07%	42.74%	18.05%	25.86%	32.72%	32.14%	42.86%	53.57%	23.66%	33.27%	41.22%	**23.36%**	**32.61%**	**39.45%**
Ochiai	53.21%	67.29%	73.59%	28.12%	39.31%	46.22%	21.89%	30.12%	35.62%	33.48%	41.07%	55.36%	30.81%	41.25%	48.10%	**27.26%**	**36.52%**	**43.19%**
Phi	53.04%	67.42%	73.76%	28.17%	39.31%	46.32%	21.89%	30.18%	35.62%	33.48%	41.07%	54.02%	30.81%	41.19%	48.13%	**27.23%**	**36.52%**	**43.08%**
PSC	56.46%	67.12%	72.73%	34.41%	45.71%	52.81%	28.93%	39.76%	45.56%	40.18%	50.89%	63.39%	39.37%	51.13%	58.68%	**33.70%**	**44.22%**	**51.16%**
Relative Matching	45.46%	60.10%	66.52%	19.89%	28.12%	34.00%	18.28%	25.74%	30.77%	30.36%	40.63%	48.21%	23.86%	32.25%	39.00%	**23.36%**	**31.66%**	**36.94%**
Rogers and Tanimoto	25.64%	36.52%	43.24%	18.71%	26.48%	30.42%	14.44%	18.99%	22.54%	26.34%	31.25%	39.73%	20.66%	27.57%	31.55%	**18.07%**	**24.25%**	**28.85%**
Russell and Rao	48.76%	60.70%	67.34%	26.64%	36.04%	42.84%	26.21%	35.38%	40.77%	38.39%	51.79%	62.05%	33.13%	43.31%	50.66%	**30.33%**	**40.12%**	**46.29%**
Simple matching	25.34%	36.04%	42.77%	18.61%	26.28%	29.86%	14.32%	19.11%	22.60%	27.23%	32.14%	39.29%	20.16%	27.23%	31.11%	**17.94%**	**24.15%**	**28.48%**
Sokal and Sneath	24.23%	35.02%	41.91%	17.48%	25.41%	29.04%	13.02%	17.69%	21.18%	23.66%	30.36%	38.84%	19.82%	26.56%	30.20%	**16.88%**	**23.31%**	**27.76%**
Sokal and Sneath 2	57.71%	68.15%	73.12%	36.15%	46.88%	53.58%	30.06%	40.12%	45.74%	42.41%	53.57%	64.29%	41.59%	51.97%	58.65%	**35.74%**	**45.65%**	**51.74%**
Sokal and Sneath 4	50.21%	66.05%	73.03%	24.90%	35.07%	42.79%	17.99%	25.80%	32.66%	32.14%	41.96%	52.68%	23.59%	33.20%	41.22%	**23.33%**	**32.49%**	**39.32%**
Sokal binary distance	34.03%	47.60%	54.07%	21.11%	29.60%	35.28%	17.81%	23.85%	27.93%	28.57%	32.14%	41.52%	24.50%	30.57%	35.02%	**21.20%**	**27.96%**	**33.25%**
Sorenson	53.21%	65.67%	72.39%	29.04%	39.57%	46.68%	23.61%	31.12%	37.57%	35.27%	44.64%	55.80%	32.32%	43.58%	50.76%	**29.09%**	**38.20%**	**44.78%**
Yule	20.25%	30.61%	39.43%	7.57%	11.91%	15.24%	8.28%	12.96%	16.63%	11.16%	23.21%	36.16%	7.04%	12.67%	17.93%	**9.22%**	**15.13%**	**20.10%**

EXTRACT METHOD						
	JFreeChart	JHotDraw	JMeter	JUnit	TomCat	Mean
Coefficient	**Hits**	**Hits**	**Hits**	**Hits**	**Hits**	**Hits**
Baroni-Urbani and Buser	42.18%	60.95%	60.09%	57.14%	57.93%	**52.57%**
Dot-product	55.54%	65.49%	62.93%	64.29%	65.89%	**58.00%**
Hamann	7.35%	19.01%	9.05%	28.57%	9.13%	**15.59%**
Jaccard	61.87%	67.68%	63.97%	71.43%	70.03%	**62.09%**
Kulczynski	71.12%	68.70%	70.60%	71.43%	72.36%	**66.52%**
Ochiai	61.05%	68.23%	66.38%	71.43%	68.74%	**62.00%**
Phi	60.99%	68.15%	66.38%	71.43%	68.64%	**61.94%**
PSC	71.69%	72.07%	67.41%	71.43%	73.25%	**66.18%**
Relative Matching	74.79%	73.71%	73.10%	71.43%	80.39%	**69.17%**
Rogers and Tanimoto	7.41%	19.01%	9.05%	28.57%	9.31%	**15.56%**
Russell and Rao	83.03%	77.23%	80.00%	71.43%	85.65%	**74.79%**
Simple matching	7.28%	19.01%	8.79%	28.57%	9.28%	**15.57%**
Sokal and Sneath	7.28%	19.01%	8.79%	28.57%	9.03%	**15.47%**
Sokal and Sneath 2	70.11%	73.94%	67.59%	71.43%	73.80%	**67.49%**
Sokal and Sneath 4	70.74%	68.62%	70.34%	71.43%	72.33%	**66.28%**
Sokal binary distance	11.78%	24.02%	18.62%	28.57%	16.97%	**21.75%**
Sorenson	55.54%	65.49%	62.93%	64.29%	65.89%	**58.00%**
Yule	46.11%	62.91%	62.24%	57.14%	60.05%	**55.44%**

APPENDIX B – Coefficient Precision in Test Set

Table 2 reports the results of the precision rates of the 18 analyzed coefficients, as also the proposed coefficients—*PTMC*, *PTMM*, and *PTEM*—w.r.t. the identification of *Move Class*, *Move Method*, and *Extract Method* refactoring opportunities in the 101 systems of *Qualitas.class Corpus* (test set).

Table 2 – Precision rates of the 19 coefficients in 101 systems (test set)

MOVE CLASS															
	Antlr			Aoi			Aspectj			Axion			Azureus		
Coefficient	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**
Baroni-Urbani and Buser	40.63%	60.00%	78.75%	45.35%	58.95%	67.54%	38.14%	53.67%	63.32%	51.08%	61.29%	74.19%	23.46%	31.77%	38.30%
Dot-product	53.75%	76.25%	86.25%	62.77%	74.46%	82.10%	55.70%	71.84%	80.27%	68.82%	76.34%	89.78%	35.69%	46.91%	54.81%
Hamann	21.88%	43.13%	51.88%	24.11%	45.82%	52.98%	12.92%	22.34%	27.26%	43.55%	53.23%	58.60%	16.21%	20.78%	23.69%
Jaccard	61.88%	76.88%	86.88%	65.16%	77.09%	84.01%	57.50%	73.64%	81.40%	72.58%	77.42%	91.94%	37.77%	49.82%	57.36%
Kulczynski	55.00%	76.88%	85.63%	62.77%	71.84%	81.62%	55.84%	69.76%	78.56%	66.13%	74.73%	87.63%	34.38%	46.02%	52.97%
Ochiai	53.75%	76.88%	86.25%	63.25%	73.03%	82.10%	56.41%	71.89%	80.74%	67.20%	76.88%	86.56%	38.18%	48.93%	56.47%
Phi	56.88%	77.50%	86.88%	63.25%	72.79%	82.82%	56.70%	71.79%	80.45%	66.13%	75.81%	87.10%	38.30%	48.99%	56.71%
PSC	68.75%	81.25%	88.13%	68.26%	81.62%	89.50%	62.85%	78.04%	84.81%	75.81%	86.02%	93.01%	43.11%	55.46%	62.65%
Relative Matching	21.25%	51.88%	73.13%	50.84%	66.83%	74.46%	41.17%	57.79%	68.91%	66.67%	76.34%	83.33%	25.18%	33.43%	41.15%
Rogers and Tanimoto	21.88%	43.75%	51.88%	25.78%	46.30%	52.98%	13.11%	22.48%	27.59%	43.55%	53.23%	58.60%	16.27%	20.90%	23.75%
Russell and Rao	32.50%	65.00%	78.75%	57.52%	72.32%	81.38%	47.80%	65.40%	76.90%	72.04%	80.65%	89.25%	29.69%	39.73%	47.39%
Simple matching	21.88%	43.13%	51.88%	24.11%	45.82%	52.74%	12.92%	22.29%	27.31%	43.55%	53.23%	58.60%	16.21%	20.84%	23.75%
Sokal and Sneath	21.88%	43.13%	51.25%	23.63%	45.82%	52.27%	12.87%	22.10%	26.98%	43.55%	52.69%	58.60%	15.97%	20.55%	23.46%
Sokal and Sneath 2	64.38%	77.50%	87.50%	66.11%	78.52%	85.92%	59.96%	75.01%	82.39%	73.12%	80.11%	93.01%	39.96%	51.19%	58.79%
Sokal and Sneath 4	57.50%	78.13%	86.25%	63.25%	71.84%	82.34%	55.84%	69.90%	78.61%	65.05%	75.81%	87.10%	34.38%	46.32%	53.21%
Sokal binary distance	30.00%	47.50%	57.50%	32.46%	47.26%	51.79%	16.71%	25.93%	30.48%	43.01%	55.91%	60.75%	19.83%	25.59%	28.15%
Sorenson	53.75%	76.25%	86.25%	62.77%	74.46%	82.10%	55.70%	71.84%	80.27%	68.82%	76.34%	89.78%	35.69%	46.91%	54.81%
Yule	26.88%	43.13%	68.75%	36.52%	44.87%	56.32%	17.94%	33.36%	43.11%	44.09%	48.92%	65.05%	18.17%	25.95%	31.12%
PTMC	68.84%	79.56%	90.47%	71.36%	83.37%	89.92%	65.08%	75.10%	83.37%	79.80%	86.05%	98.16%	45.84%	53.81%	63.19%

MOVE METHOD															
	Antlr			Aoi			Aspectj			Axion			Azureus		
Coefficient	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**	**Top1**	**Top2**	**Top3**
Baroni-Urbani and Buser	22.78%	28.65%	34.70%	8.76%	14.76%	18.82%	16.70%	21.96%	27.49%	25.64%	34.19%	37.26%	6.83%	9.65%	12.05%
Dot-product	37.54%	48.04%	57.47%	22.02%	31.88%	39.99%	34.88%	44.25%	52.39%	40.68%	51.97%	55.90%	15.67%	23.14%	28.49%
Hamann	25.62%	32.56%	38.61%	7.76%	13.01%	16.57%	19.96%	26.73%	31.93%	31.79%	41.71%	46.32%	7.46%	10.69%	12.73%
Jaccard	39.15%	51.96%	59.96%	25.58%	36.74%	44.24%	37.99%	48.26%	55.37%	45.30%	55.21%	59.66%	19.08%	27.97%	33.43%
Kulczynski	30.25%	40.04%	49.47%	14.46%	25.58%	33.23%	25.33%	34.91%	43.95%	29.57%	41.88%	51.45%	8.86%	14.19%	18.56%
Ochiai	35.23%	46.26%	55.16%	19.92%	30.28%	38.49%	32.08%	41.70%	49.92%	38.46%	49.57%	55.38%	13.07%	19.27%	24.76%
Phi	35.23%	46.44%	54.98%	19.82%	30.18%	38.39%	32.05%	41.63%	49.87%	38.29%	49.57%	55.21%	12.99%	19.21%	24.71%
PSC	41.28%	55.16%	64.41%	26.23%	38.64%	47.15%	38.37%	49.64%	57.09%	45.64%	55.38%	60.17%	19.56%	29.08%	34.98%
Relative Matching	24.56%	34.88%	42.17%	12.46%	21.02%	27.63%	27.13%	35.94%	42.76%	35.04%	41.03%	45.64%	10.94%	17.27%	21.38%
Rogers and Tanimoto	27.05%	33.99%	39.50%	7.91%	13.26%	16.77%	20.05%	26.86%	32.16%	31.62%	41.54%	46.50%	7.64%	10.91%	12.86%
Russell and Rao	30.96%	44.66%	51.25%	20.52%	31.48%	37.54%	33.83%	44.39%	52.28%	41.37%	50.77%	54.53%	18.79%	27.57%	32.86%
Simple matching	25.80%	32.56%	38.43%	7.66%	12.96%	16.62%	19.99%	26.84%	31.97%	31.79%	41.71%	46.32%	7.58%	10.84%	12.76%
Sokal and Sneath	25.44%	32.03%	38.08%	7.46%	12.46%	16.02%	19.53%	26.22%	31.49%	31.28%	41.03%	44.79%	7.24%	10.30%	12.21%
Sokal and Sneath 2	42.88%	55.52%	63.35%	28.03%	39.29%	47.15%	40.41%	50.92%	57.84%	50.43%	57.44%	61.71%	22.69%	32.03%	37.37%
Sokal and Sneath 4	30.25%	39.68%	49.47%	14.46%	25.58%	33.23%	25.26%	34.89%	43.97%	29.57%	41.88%	51.45%	8.76%	14.10%	18.48%
Sokal binary distance	28.47%	38.26%	43.59%	9.76%	16.72%	20.72%	23.21%	30.20%	35.49%	36.41%	44.44%	50.09%	9.52%	13.00%	15.24%
Sorenson	37.54%	48.04%	57.47%	22.02%	31.88%	39.99%	34.88%	44.25%	52.39%	40.68%	51.97%	55.90%	15.67%	23.14%	28.49%
Yule	12.99%	19.75%	27.22%	6.06%	10.31%	13.71%	6.18%	11.53%	17.13%	7.69%	14.53%	21.03%	3.13%	5.74%	8.37%
PTMM	55.20%	67.24%	69.16%	39.40%	46.99%	52.12%	50.82%	57.60%	63.60%	66.11%	71.73%	75.52%	41.86%	52.25%	55.80%

EXTRACT METHOD					
	Antlr	Aoi	Aspectj	Axion	Azureus
Coefficient	**Hits**	**Hits**	**Hits**	**Hits**	**Hits**
Baroni-Urbani and Buser	61.72%	63.80%	56.80%	58.50%	61.94%
Dot-product	70.63%	73.67%	68.75%	71.87%	66.14%
Hamann	9.08%	5.57%	10.38%	11.70%	6.53%
Jaccard	71.95%	76.05%	71.73%	77.16%	69.16%
Kulczynski	74.59%	79.89%	77.85%	80.22%	76.32%
Ochiai	72.77%	76.71%	72.43%	74.09%	70.30%
Phi	71.12%	76.47%	72.30%	73.26%	70.25%
PSC	73.93%	78.77%	75.00%	79.67%	74.24%
Relative Matching	82.51%	86.72%	83.95%	83.01%	78.42%
Rogers and Tanimoto	9.41%	5.94%	10.50%	12.26%	6.57%
Russell and Rao	85.64%	91.44%	87.66%	89.42%	84.32%
Simple matching	9.08%	5.71%	10.38%	11.70%	6.52%
Sokal and Sneath	9.08%	5.47%	10.32%	11.70%	6.50%
Sokal and Sneath 2	72.94%	79.51%	74.61%	80.22%	72.69%
Sokal and Sneath 4	72.94%	79.37%	77.47%	79.11%	76.17%
Sokal binary distance	15.18%	9.87%	16.49%	25.35%	13.50%
Sorenson	70.63%	73.67%	68.75%	71.87%	66.14%
Yule	62.21%	66.42%	63.78%	61.56%	65.23%
PTEM	86.17%	91.60%	88.22%	90.19%	84.59%

To be continued in the next page.

MOVE CLASS																														
	Batik			C_JDBC			Castor			Cayenne			CheckStyle			Cobertura			Colt			Columba			Compiere			Derby		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	25.29%	37.21%	47.71%	43.94%	57.29%	61.19%	46.25%	58.76%	66.35%	28.32%	39.82%	46.20%	25.45%	31.25%	38.39%	30.77%	40.00%	44.62%	46.91%	60.31%	77.84%	44.70%	56.78%	63.36%	18.24%	22.29%	26.00%	24.10%	31.88%	38.22%
Dot-product	47.04%	64.60%	73.09%	54.21%	66.32%	71.66%	57.64%	70.75%	76.27%	47.98%	58.20%	65.29%	33.04%	42.41%	49.11%	36.92%	43.08%	52.31%	62.37%	77.32%	84.02%	53.29%	66.04%	72.21%	63.17%	68.56%	71.48%	45.46%	59.80%	67.89%
Hamann	16.03%	20.61%	22.52%	42.92%	52.57%	57.08%	38.91%	46.33%	50.56%	7.24%	9.65%	13.56%	26.34%	31.70%	33.48%	9.23%	13.85%	30.77%	32.99%	46.39%	51.55%	28.59%	37.58%	44.56%	42.51%	53.63%	55.91%	8.70%	11.90%	12.97%
Jaccard	51.53%	66.98%	75.00%	56.06%	70.02%	75.56%	60.14%	72.65%	77.39%	50.32%	61.11%	67.07%	37.05%	44.20%	48.66%	33.85%	41.54%	56.92%	65.98%	78.35%	86.60%	54.36%	67.52%	73.29%	64.41%	69.55%	73.06%	48.21%	62.40%	70.02%
Kulczynski	43.80%	62.02%	71.28%	51.54%	64.27%	69.40%	56.86%	69.89%	76.10%	46.42%	58.91%	66.08%	33.48%	43.30%	48.66%	40.00%	46.15%	55.38%	59.28%	78.87%	84.02%	55.03%	66.71%	73.15%	63.62%	69.06%	72.66%	39.36%	55.15%	65.45%
Ochiai	45.99%	64.69%	72.90%	53.18%	65.09%	71.05%	57.46%	71.35%	76.70%	48.26%	58.84%	65.93%	32.14%	42.86%	50.45%	36.92%	44.62%	55.38%	63.40%	77.84%	84.54%	54.36%	66.31%	72.21%	63.72%	68.96%	72.37%	45.61%	60.41%	68.80%
Phi	45.99%	64.89%	72.90%	53.59%	65.50%	71.05%	57.55%	71.10%	76.45%	48.05%	58.77%	65.72%	31.25%	42.41%	49.11%	41.54%	46.15%	55.38%	63.92%	78.35%	83.51%	54.50%	66.44%	72.35%	63.72%	69.11%	72.47%	45.61%	60.03%	68.80%
PSC	56.87%	74.24%	80.34%	56.88%	70.84%	78.44%	62.90%	75.06%	80.50%	54.65%	66.29%	72.39%	40.63%	48.21%	52.23%	40.00%	47.69%	58.46%	73.20%	79.90%	87.63%	57.18%	70.20%	75.44%	66.53%	72.17%	77.31%	53.47%	68.50%	75.21%
Relative Matching	34.35%	50.57%	59.45%	45.79%	57.29%	63.04%	47.71%	63.16%	70.06%	43.08%	55.29%	61.32%	25.89%	34.82%	40.63%	32.31%	44.62%	50.77%	51.55%	73.20%	81.44%	42.01%	52.21%	61.61%	54.28%	60.36%	65.55%	37.99%	50.34%	59.50%
Rogers and Tanimoto	16.41%	20.61%	22.61%	42.92%	53.18%	57.29%	39.60%	46.42%	51.16%	7.24%	9.65%	13.70%	25.89%	31.70%	33.48%	9.23%	15.38%	32.31%	32.99%	46.39%	52.06%	28.59%	37.85%	44.56%	42.71%	53.68%	55.96%	8.70%	11.98%	13.12%
Russell and Rao	44.75%	61.55%	69.66%	49.08%	60.16%	66.74%	53.49%	69.46%	76.79%	50.82%	62.10%	68.20%	32.59%	41.96%	50.00%	44.62%	50.77%	63.08%	59.28%	80.41%	84.54%	46.44%	55.84%	66.58%	58.08%	64.90%	68.91%	46.83%	59.73%	68.80%
Simple matching	16.03%	20.52%	22.42%	43.12%	53.18%	57.29%	38.83%	46.33%	50.65%	7.24%	9.65%	13.56%	25.89%	31.70%	33.48%	9.23%	13.85%	30.77%	32.99%	46.39%	52.06%	28.59%	37.72%	44.56%	42.56%	53.58%	55.91%	8.62%	11.98%	13.04%
Sokal and Sneath	16.03%	20.52%	22.33%	42.92%	52.57%	56.67%	38.48%	45.04%	50.13%	7.10%	9.58%	13.56%	26.34%	31.70%	33.48%	9.23%	13.85%	30.77%	31.96%	45.36%	51.55%	27.92%	37.32%	43.89%	42.41%	53.53%	55.86%	8.47%	11.67%	12.59%
Sokal and Sneath 2	53.91%	68.99%	75.38%	58.73%	70.84%	76.59%	61.26%	73.94%	78.43%	51.10%	62.46%	67.64%	38.84%	45.54%	50.00%	33.85%	40.00%	53.85%	68.56%	79.90%	86.08%	56.11%	67.79%	73.42%	64.66%	70.14%	73.75%	51.11%	64.68%	71.32%
Sokal and Sneath 4	44.08%	62.21%	70.99%	51.95%	64.48%	69.40%	57.03%	69.89%	76.10%	46.13%	58.77%	66.08%	31.70%	41.96%	48.66%	40.00%	46.15%	55.38%	62.37%	77.84%	85.05%	55.70%	66.58%	72.75%	63.57%	69.11%	72.61%	39.66%	55.00%	65.45%
Sokal binary distance	17.37%	21.95%	24.43%	45.59%	57.49%	60.99%	41.24%	49.87%	56.51%	9.51%	11.92%	16.75%	26.34%	30.80%	33.48%	10.77%	18.46%	33.85%	33.51%	46.91%	52.58%	34.23%	42.55%	46.98%	50.91%	55.76%	57.93%	10.22%	13.27%	14.95%
Sorenson	47.04%	64.60%	73.09%	54.21%	66.32%	71.66%	57.64%	70.75%	76.27%	47.98%	58.20%	65.29%	33.04%	42.41%	49.11%	36.92%	43.08%	52.31%	62.37%	77.32%	84.02%	53.29%	66.04%	72.21%	63.17%	68.56%	71.48%	45.46%	59.80%	67.89%
Yule	15.74%	24.62%	33.78%	41.68%	53.18%	56.88%	37.10%	50.30%	58.41%	18.95%	26.33%	36.83%	14.73%	19.64%	26.79%	13.85%	32.31%	44.62%	35.57%	47.94%	65.98%	36.24%	48.32%	56.78%	10.78%	12.70%	14.38%	10.83%	16.32%	18.84%
PTMC	59.52%	72.31%	82.50%	63.87%	71.21%	78.55%	67.23%	76.42%	82.78%	55.73%	64.99%	73.70%	41.59%	50.39%	59.18%	47.15%	44.74%	56.42%	78.35%	81.82%	90.58%	61.88%	68.37%	79.11%	71.12%	70.46%	77.68%	62.01%	69.48%	75.03%

MOVE METHOD																														
	Batik			C_JDBC			Castor			Cayenne			CheckStyle			Cobertura			Colt			Columba			Compiere			Derby		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	12.96%	18.30%	22.46%	21.30%	28.63%	31.15%	31.16%	40.21%	45.96%	12.87%	18.63%	22.15%	12.74%	18.43%	21.68%	55.08%	73.44%	77.73%	46.91%	60.31%	77.84%	13.09%	19.72%	25.06%	11.60%	15.28%	17.80%	11.59%	17.30%	21.08%
Dot-product	26.18%	35.71%	42.16%	31.07%	41.45%	48.78%	44.09%	54.67%	60.60%	23.14%	32.62%	39.34%	17.07%	27.37%	34.15%	64.84%	82.03%	87.11%	62.37%	77.32%	84.02%	26.27%	38.16%	44.79%	22.85%	28.95%	32.69%	25.60%	37.63%	42.94%
Hamann	16.31%	21.32%	24.89%	27.48%	32.98%	37.94%	34.13%	43.94%	48.16%	12.00%	17.54%	22.12%	7.59%	12.20%	18.43%	56.25%	76.17%	78.52%	32.99%	46.39%	51.55%	13.35%	19.55%	22.74%	11.17%	14.44%	16.63%	15.78%	25.15%	29.68%
Jaccard	29.60%	40.06%	46.58%	36.41%	45.80%	52.82%	47.53%	59.21%	64.15%	26.09%	36.40%	43.09%	19.78%	30.89%	40.11%	67.58%	84.38%	88.28%	65.98%	78.35%	86.60%	29.89%	42.81%	48.15%	25.20%	31.27%	35.05%	29.25%	41.91%	47.43%
Kulczynski	20.84%	28.50%	35.24%	28.40%	35.27%	39.62%	37.48%	50.12%	56.20%	15.43%	25.10%	31.56%	11.38%	22.22%	31.71%	60.55%	80.86%	84.77%	59.28%	78.87%	84.02%	18.09%	28.77%	36.61%	16.24%	21.58%	25.36%	15.06%	26.73%	33.86%
Ochiai	23.97%	32.25%	39.76%	31.07%	38.47%	44.43%	42.89%	53.61%	59.74%	20.04%	30.35%	36.91%	14.09%	27.10%	32.25%	63.67%	82.81%	85.94%	63.40%	77.84%	84.54%	24.89%	34.54%	43.41%	21.13%	27.13%	30.88%	23.75%	34.85%	40.97%
Phi	23.97%	32.29%	39.69%	30.99%	38.55%	44.12%	42.75%	53.57%	59.69%	20.01%	30.35%	36.88%	13.55%	27.10%	32.25%	64.06%	82.42%	85.55%	63.92%	78.35%	83.51%	24.98%	34.19%	43.07%	21.13%	27.08%	30.82%	23.73%	34.89%	40.96%
PSC	30.63%	42.42%	49.63%	38.78%	48.40%	55.80%	48.88%	60.56%	65.34%	26.06%	36.62%	45.33%	20.60%	31.71%	40.38%	67.97%	86.72%	89.45%	73.20%	79.90%	87.63%	32.30%	43.93%	49.87%	25.75%	32.26%	36.55%	29.31%	42.53%	48.56%
Relative Matching	20.80%	27.54%	31.70%	28.17%	34.96%	39.77%	37.67%	47.06%	51.75%	18.63%	25.51%	30.51%	16.26%	21.95%	26.56%	58.59%	76.56%	80.47%	51.55%	73.20%	81.44%	21.10%	31.18%	37.30%	19.47%	24.89%	27.91%	18.80%	28.52%	34.44%
Rogers and Tanimoto	16.49%	21.87%	25.37%	27.86%	33.51%	38.47%	34.42%	44.57%	48.73%	12.07%	17.57%	22.06%	7.86%	12.74%	17.62%	56.25%	75.78%	78.52%	32.99%	46.39%	52.06%	13.87%	20.33%	23.43%	11.33%	14.68%	16.74%	15.89%	25.28%	29.89%
Russell and Rao	26.77%	36.08%	41.38%	33.44%	42.98%	49.08%	42.51%	52.56%	59.02%	24.52%	33.74%	39.79%	27.37%	34.69%	40.11%	66.41%	80.08%	84.77%	59.28%	80.41%	84.54%	30.58%	42.12%	48.92%	24.72%	31.23%	34.56%	25.20%	36.46%	42.53%
Simple matching	16.35%	21.47%	25.07%	27.56%	33.05%	37.86%	34.32%	43.85%	48.16%	11.88%	17.41%	21.96%	7.86%	13.01%	18.97%	56.25%	76.17%	78.52%	32.99%	46.39%	52.06%	13.61%	19.98%	22.74%	11.03%	14.39%	16.45%	15.72%	25.19%	29.80%
Sokal and Sneath	16.16%	20.58%	24.04%	26.87%	32.29%	37.18%	33.70%	42.84%	47.15%	11.43%	16.81%	21.06%	7.05%	11.11%	15.99%	54.69%	75.00%	77.34%	31.96%	45.36%	51.55%	12.75%	18.78%	21.79%	10.53%	13.70%	15.69%	15.38%	24.62%	29.22%
Sokal and Sneath 2	33.69%	45.66%	51.36%	40.84%	49.31%	55.42%	50.89%	61.70%	66.20%	29.80%	39.63%	47.50%	24.66%	34.15%	42.55%	68.75%	85.16%	89.45%	68.56%	79.90%	86.08%	35.83%	45.91%	51.59%	27.29%	33.59%	37.68%	33.31%	45.02%	50.45%
Sokal and Sneath 4	20.73%	28.42%	35.31%	28.40%	35.27%	39.62%	37.63%	50.02%	56.06%	15.40%	25.06%	31.47%	10.84%	21.41%	30.89%	60.55%	81.25%	84.38%	62.37%	77.84%	85.05%	18.00%	28.51%	36.18%	16.18%	21.54%	25.42%	15.06%	26.64%	33.74%
Sokal binary distance	20.51%	26.51%	30.45%	31.60%	37.10%	40.84%	37.87%	47.06%	51.70%	14.85%	20.65%	25.67%	13.28%	20.33%	26.29%	58.20%	79.69%	81.64%	33.51%	46.91%	52.58%	17.31%	23.08%	27.05%	13.57%	17.25%	19.36%	19.35%	29.04%	33.83%
Sorenson	26.18%	35.71%	42.16%	31.07%	41.45%	48.78%	44.09%	54.67%	60.60%	23.14%	32.62%	39.34%	17.07%	27.37%	34.15%	64.84%	82.03%	87.11%	62.37%	77.32%	84.02%	26.27%	38.16%	44.79%	22.85%	28.95%	32.69%	25.60%	37.63%	42.94%
Yule	4.86%	9.20%	13.73%	13.74%	17.71%	19.39%	15.22%	27.76%	35.18%	6.79%	12.45%	15.88%	9.76%	15.18%	18.97%	36.33%	61.33%	69.53%	35.57%	47.94%	65.98%	7.41%	12.32%	17.31%	5.75%	9.92%	12.40%	3.53%	6.96%	9.22%
PTMM	52.71%	60.92%	62.80%	55.91%	65.22%	68.63%	59.57%	67.98%	72.89%	40.31%	50.49%	53.20%	48.26%	55.92%	59.67%	81.55%	86.28%	88.43%	76.92%	87.29%	88.77%	44.29%	53.62%	57.50%	46.86%	50.00%	53.36%	52.68%	60.34%	63.17%

EXTRACT METHOD										
	Batik	C_JDBC	Castor	Cayenne	CheckStyle	Cobertura	Colt	Columba	Compiere	Derby
Coefficient	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits
Baroni-Urbani and Buser	67.87%	58.02%	62.22%	70.16%	58.80%	70.35%	51.90%	63.59%	70.75%	73.98%
Dot-product	73.95%	73.22%	68.59%	75.29%	54.23%	74.37%	53.89%	64.47%	79.00%	83.42%
Hamann	16.39%	24.39%	13.66%	6.85%	5.99%	25.63%	21.56%	6.50%	9.93%	12.17%
Jaccard	75.12%	77.68%	71.28%	76.66%	54.23%	76.38%	58.48%	66.41%	79.70%	85.45%
Kulczynski	82.47%	75.86%	75.11%	80.92%	62.32%	78.89%	64.07%	69.51%	85.09%	90.23%
Ochiai	77.57%	73.82%	71.36%	77.84%	57.75%	76.38%	58.68%	68.06%	82.13%	86.59%
Phi	77.06%	73.77%	70.71%	77.79%	57.75%	76.38%	58.68%	67.67%	82.00%	86.48%
PSC	79.91%	78.74%	74.13%	77.30%	58.45%	79.40%	60.48%	68.64%	81.42%	88.20%
Relative Matching	82.99%	82.86%	81.34%	86.06%	77.46%	82.41%	73.05%	80.68%	91.59%	92.29%
Rogers and Tanimoto	16.39%	24.39%	14.19%	6.80%	5.99%	25.63%	21.76%	6.70%	9.80%	12.17%
Russell and Rao	87.19%	85.49%	86.84%	90.41%	81.34%	90.45%	80.04%	85.24%	93.42%	95.48%
Simple matching	16.39%	24.29%	13.85%	6.85%	5.28%	25.63%	21.56%	6.70%	9.89%	12.13%
Sokal and Sneath	16.36%	24.19%	13.58%	6.85%	4.58%	25.63%	21.16%	6.21%	9.91%	12.07%
Sokal and Sneath 2	78.16%	80.77%	73.71%	76.61%	56.34%	78.39%	60.88%	68.45%	80.08%	87.11%
Sokal and Sneath 4	82.29%	75.36%	74.58%	80.77%	61.62%	78.89%	63.67%	67.96%	84.94%	90.08%
Sokal binary distance	23.71%	30.30%	20.07%	12.72%	7.75%	35.68%	29.74%	11.17%	14.55%	19.05%
Sorenson	73.95%	73.22%	68.59%	75.29%	54.23%	74.37%	53.89%	64.47%	79.00%	83.42%
Yule	69.23%	50.12%	65.90%	72.06%	62.32%	74.37%	57.88%	69.13%	74.24%	76.37%
PTEM	87.31%	85.81%	87.17%	90.53%	81.87%	90.50%	80.81%	85.60%	93.53%	95.65%

To be continued in the next page.

MOVE CLASS																														
	DisplayTag			DrawSWF			DrJava			Eclipse_SDK			Emma			ExoPortal			FindBugs			FitJava			FitLibraryFF			FreeCol		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	38.94%	56.64%	70.80%	46.67%	67.22%	72.78%	36.31%	48.73%	57.32%	25.52%	40.00%	51.03%	41.67%	56.48%	62.04%	20.32%	29.45%	33.56%	27.02%	39.24%	47.31%	46.51%	69.77%	86.05%	25.72%	35.25%	39.91%	58.59%	68.51%	74.43%
Dot-product	61.95%	76.99%	82.30%	63.33%	77.78%	80.00%	50.00%	65.29%	73.57%	6.90%	12.41%	15.86%	53.70%	64.81%	67.59%	30.14%	41.55%	50.68%	44.62%	54.40%	60.88%	55.81%	76.74%	88.37%	33.04%	45.90%	52.99%	69.47%	79.01%	86.83%
Hamann	36.28%	46.02%	52.21%	37.22%	48.89%	53.89%	19.11%	24.20%	27.07%	14.48%	22.76%	29.66%	25.00%	33.33%	41.67%	15.07%	18.95%	22.83%	6.36%	18.46%	19.68%	37.21%	44.19%	58.14%	14.86%	21.95%	23.28%	35.69%	46.56%	50.00%
Jaccard	62.83%	76.99%	82.30%	62.78%	77.78%	80.00%	50.96%	69.11%	76.43%	6.90%	13.10%	15.86%	57.41%	66.67%	71.30%	33.90%	45.78%	54.34%	45.97%	56.36%	62.47%	55.81%	79.07%	88.37%	35.70%	48.56%	55.65%	71.18%	80.92%	89.12%
Kulczynski	57.52%	74.34%	84.07%	62.22%	80.00%	83.89%	50.32%	65.61%	75.80%	25.52%	35.86%	48.97%	44.44%	58.33%	62.96%	18.15%	23.40%	32.99%	44.38%	54.16%	60.64%	55.81%	76.74%	88.37%	21.73%	37.92%	45.01%	69.47%	80.73%	86.83%
Ochiai	58.41%	79.65%	83.19%	62.78%	78.89%	82.22%	50.00%	66.56%	75.16%	6.90%	13.10%	15.86%	50.00%	62.04%	67.59%	28.31%	38.70%	47.60%	44.74%	55.01%	61.00%	60.47%	76.74%	86.05%	31.26%	44.35%	52.99%	70.42%	80.53%	86.26%
Phi	59.29%	79.65%	83.19%	62.78%	77.22%	81.67%	50.32%	67.20%	75.48%	6.90%	12.41%	15.86%	52.78%	64.81%	70.37%	28.42%	38.47%	47.60%	45.11%	55.01%	60.88%	53.49%	74.42%	86.05%	31.26%	44.35%	52.55%	70.23%	80.92%	87.02%
PSC	64.60%	78.76%	85.84%	63.89%	80.56%	85.00%	61.78%	75.48%	83.76%	14.48%	37.93%	48.28%	56.48%	68.52%	72.22%	33.79%	47.26%	57.08%	50.24%	62.96%	69.44%	53.49%	79.07%	86.05%	36.59%	49.45%	58.09%	73.85%	86.26%	91.79%
Relative Matching	30.97%	50.44%	66.37%	51.11%	76.11%	78.89%	46.50%	58.28%	64.65%	17.93%	26.90%	35.86%	25.93%	45.37%	50.00%	25.11%	37.33%	43.95%	33.50%	48.29%	51.83%	48.84%	72.09%	90.70%	20.62%	35.70%	41.69%	42.37%	68.89%	80.15%
Rogers and Tanimoto	36.28%	46.90%	52.21%	37.22%	48.33%	53.89%	19.11%	24.20%	27.07%	8.28%	13.79%	17.93%	25.00%	33.33%	42.59%	15.07%	19.29%	23.40%	6.36%	18.34%	20.17%	37.21%	44.19%	60.47%	15.74%	22.62%	23.95%	36.07%	46.56%	50.00%
Russell and Rao	48.67%	65.49%	73.45%	55.00%	77.78%	80.00%	49.68%	62.42%	69.11%	24.83%	38.62%	51.03%	25.93%	50.00%	55.56%	31.16%	46.35%	52.51%	36.19%	51.96%	57.09%	60.47%	81.40%	93.02%	25.50%	42.57%	53.22%	44.66%	72.71%	84.54%
Simple matching	35.40%	46.02%	52.21%	37.22%	48.33%	53.89%	18.79%	24.20%	27.07%	27.59%	36.55%	51.03%	25.00%	33.33%	41.67%	15.07%	18.72%	22.60%	6.36%	18.58%	19.80%	37.21%	44.19%	60.47%	15.74%	22.39%	23.73%	35.69%	46.56%	50.00%
Sokal and Sneath	34.51%	44.25%	52.21%	36.67%	48.89%	53.89%	18.79%	23.57%	27.07%	31.03%	43.45%	54.48%	24.07%	33.33%	41.67%	14.50%	17.69%	21.46%	6.23%	18.09%	19.68%	37.21%	44.19%	58.14%	14.63%	21.51%	22.62%	35.69%	46.56%	49.81%
Sokal and Sneath 2	64.60%	77.88%	83.19%	67.22%	77.78%	81.11%	52.55%	70.06%	77.71%	25.52%	35.86%	48.97%	57.41%	67.59%	72.22%	35.27%	48.17%	55.94%	47.19%	57.09%	63.20%	55.81%	79.07%	88.37%	37.03%	50.11%	56.98%	71.56%	83.02%	90.08%
Sokal and Sneath 4	60.18%	76.99%	83.19%	62.78%	79.44%	83.89%	49.36%	65.29%	75.48%	24.83%	37.24%	51.72%	49.07%	62.04%	65.74%	18.15%	23.40%	32.76%	44.74%	54.65%	60.51%	51.16%	74.42%	86.05%	21.73%	37.92%	45.23%	69.66%	81.11%	87.21%
Sokal binary distance	39.82%	46.90%	53.98%	39.44%	48.89%	53.89%	20.38%	25.80%	29.30%	26.90%	41.38%	51.72%	26.85%	39.81%	44.44%	16.32%	20.21%	24.09%	7.46%	19.32%	21.76%	37.21%	44.19%	72.09%	17.74%	24.17%	25.50%	39.89%	47.71%	52.10%
Sorenson	61.95%	76.99%	82.30%	63.33%	77.78%	80.00%	50.00%	65.29%	73.57%	26.21%	38.62%	47.59%	53.70%	64.81%	67.59%	30.14%	41.55%	50.68%	44.62%	54.40%	60.88%	55.81%	76.74%	88.37%	33.04%	45.90%	52.99%	69.47%	79.01%	86.83%
Yule	30.97%	42.48%	56.64%	30.56%	53.89%	63.89%	28.98%	44.59%	51.27%	10.34%	26.90%	40.69%	21.30%	35.19%	44.44%	10.73%	15.30%	19.41%	17.73%	27.14%	33.99%	32.56%	67.44%	81.40%	11.09%	25.94%	34.15%	39.12%	47.90%	53.82%
PTMC	63.16%	77.16%	88.49%	69.91%	78.58%	86.61%	62.21%	70.54%	80.50%	36.32%	48.89%	57.64%	57.81%	68.59%	78.23%	50.65%	57.04%	66.26%	53.73%	57.93%	66.99%	66.97%	81.96%	94.13%	44.35%	56.03%	67.77%	83.38%	86.73%	91.95%

MOVE METHOD																														
	DisplayTag			DrawSWF			DrJava			Eclipse_SDK			Emma			ExoPortal			FindBugs			FitJava			FitLibraryFF			FreeCol		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	26.60%	45.32%	51.23%	25.71%	34.89%	40.57%	11.71%	18.20%	23.88%	36.74%	41.81%	44.42%	17.09%	22.22%	29.06%	21.23%	28.11%	31.43%	10.06%	17.95%	21.44%	37.08%	52.81%	57.30%	12.58%	18.01%	22.29%	9.51%	13.89%	18.79%
Dot-product	45.32%	65.02%	71.43%	41.74%	54.26%	60.43%	26.26%	36.87%	46.09%	45.75%	51.98%	55.89%	32.48%	45.73%	56.41%	31.80%	42.38%	49.11%	22.19%	34.27%	39.48%	47.19%	57.30%	69.66%	26.48%	38.90%	44.49%	20.93%	31.47%	38.51%
Hamann	22.66%	34.98%	39.41%	23.71%	32.55%	41.24%	12.81%	19.48%	22.55%	34.82%	38.95%	40.46%	25.21%	34.19%	35.90%	22.04%	29.29%	32.99%	10.67%	14.91%	18.57%	33.71%	43.82%	46.07%	18.34%	25.99%	30.43%	11.05%	16.55%	20.23%
Jaccard	50.25%	67.00%	72.91%	43.57%	55.59%	62.27%	31.94%	41.86%	49.51%	47.93%	53.64%	57.55%	39.74%	52.99%	62.82%	35.43%	46.60%	53.99%	28.19%	37.72%	42.61%	50.56%	67.42%	78.65%	31.66%	44.90%	51.32%	23.68%	34.73%	42.56%
Kulczynski	43.35%	61.08%	71.43%	31.05%	48.25%	55.59%	17.97%	30.09%	39.07%	44.17%	49.86%	54.54%	29.06%	45.30%	54.70%	24.48%	34.32%	41.27%	14.25%	28.14%	34.89%	40.45%	50.56%	60.67%	18.67%	28.45%	35.44%	16.18%	27.18%	36.41%
Ochiai	45.81%	63.05%	72.41%	39.57%	53.92%	58.76%	23.30%	33.04%	41.91%	45.92%	52.02%	56.11%	32.91%	47.44%	58.12%	30.62%	40.90%	47.49%	19.89%	33.48%	38.33%	41.57%	58.43%	68.54%	23.68%	36.10%	41.53%	20.84%	31.28%	38.60%
Phi	44.83%	62.56%	71.92%	39.57%	53.92%	58.76%	23.25%	32.93%	41.86%	45.92%	52.00%	56.13%	32.05%	47.86%	58.12%	30.62%	40.90%	47.41%	19.94%	33.44%	38.33%	40.45%	58.43%	68.54%	23.68%	35.94%	41.53%	20.79%	31.28%	38.79%
PSC	52.22%	72.41%	78.33%	44.91%	59.43%	68.28%	32.35%	44.23%	52.29%	48.68%	55.23%	59.45%	44.44%	65.81%	70.94%	36.17%	48.45%	55.33%	25.63%	40.23%	46.49%	49.44%	73.03%	77.53%	31.00%	43.75%	51.73%	25.73%	38.00%	46.76%
Relative Matching	38.92%	54.68%	62.56%	33.72%	44.74%	51.25%	20.29%	30.32%	36.52%	42.49%	46.82%	49.18%	27.35%	36.32%	44.02%	28.99%	37.28%	43.57%	20.29%	26.25%	30.83%	38.20%	61.80%	67.42%	24.10%	35.20%	40.46%	11.33%	19.58%	24.48%
Rogers and Tanimoto	22.66%	35.47%	39.41%	23.71%	32.72%	41.57%	12.99%	19.83%	22.84%	34.95%	39.17%	40.63%	25.21%	33.76%	35.47%	22.63%	29.59%	33.36%	10.76%	15.13%	18.70%	33.71%	43.82%	46.07%	18.75%	26.15%	30.84%	11.05%	16.64%	20.28%
Russell and Rao	49.75%	63.05%	67.98%	40.90%	53.76%	60.27%	33.51%	45.97%	54.09%	47.17%	52.22%	54.86%	37.18%	49.57%	59.40%	38.68%	46.97%	52.81%	27.04%	35.29%	40.58%	49.44%	69.66%	80.90%	31.91%	44.98%	51.56%	16.18%	26.48%	33.33%
Simple matching	22.17%	35.47%	39.90%	23.87%	32.55%	41.40%	12.93%	19.65%	22.67%	34.84%	39.04%	40.59%	25.21%	34.62%	35.90%	22.34%	29.29%	32.77%	10.54%	14.87%	18.39%	32.58%	43.82%	46.07%	17.93%	25.74%	30.35%	11.10%	16.64%	20.33%
Sokal and Sneath	22.17%	34.98%	38.92%	23.54%	31.89%	39.90%	12.23%	18.84%	22.09%	34.60%	38.60%	40.07%	24.36%	33.33%	35.04%	21.15%	27.74%	31.58%	10.28%	14.25%	18.00%	32.58%	42.70%	44.94%	17.19%	25.00%	29.69%	11.05%	16.22%	19.67%
Sokal and Sneath 2	52.71%	67.98%	75.86%	46.91%	58.26%	66.94%	34.78%	46.20%	52.81%	49.81%	55.45%	58.62%	43.16%	59.40%	68.80%	38.91%	49.78%	57.69%	30.97%	40.98%	45.61%	51.69%	69.66%	80.90%	35.12%	47.94%	54.61%	26.95%	37.53%	45.27%
Sokal and Sneath 4	42.86%	60.59%	70.94%	31.05%	48.08%	55.43%	17.91%	30.03%	39.01%	44.15%	49.86%	54.47%	29.06%	45.30%	54.27%	24.26%	33.88%	40.90%	14.29%	27.97%	34.89%	39.33%	51.69%	60.67%	18.50%	28.29%	35.28%	16.13%	27.18%	36.50%
Sokal binary distance	27.59%	40.39%	44.83%	29.55%	39.90%	46.58%	16.00%	21.86%	25.80%	36.85%	41.49%	42.86%	28.21%	36.75%	41.03%	26.18%	33.14%	35.80%	12.35%	17.07%	21.17%	32.58%	44.94%	50.56%	21.88%	29.61%	33.96%	12.96%	18.46%	22.28%
Sorenson	45.32%	65.02%	71.43%	41.74%	54.26%	60.43%	26.26%	36.87%	46.09%	45.75%	51.98%	55.89%	32.48%	45.73%	56.41%	31.80%	42.38%	49.11%	22.19%	34.27%	39.48%	47.19%	57.30%	69.66%	26.48%	38.90%	44.49%	20.93%	31.47%	38.51%
Yule	27.09%	32.02%	48.28%	6.18%	19.53%	27.55%	8.64%	14.90%	21.45%	12.70%	15.21%	17.86%	17.52%	20.09%	23.93%	13.39%	17.38%	21.75%	6.18%	14.38%	18.13%	26.97%	42.70%	53.93%	6.41%	9.87%	12.66%	5.13%	9.00%	11.75%
PTMM	60.48%	69.61%	76.41%	60.32%	67.79%	72.17%	56.55%	68.74%	71.28%	60.62%	62.53%	63.35%	52.91%	66.50%	73.99%	51.85%	65.27%	68.44%	45.50%	51.85%	55.93%	66.89%	76.04%	82.15%	52.88%	64.17%	67.91%	38.05%	47.70%	51.49%

EXTRACT METHOD										
	DisplayTag	DrawSWF	DrJava	Eclipse_SDK	Emma	ExoPortal	FindBugs	FitJava	FitLibraryFF	FreeCol
Coefficient	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits
Baroni-Urbani and Buser	58.52%	64.56%	61.89%	62.73%	50.69%	71.25%	63.62%	51.72%	66.23%	68.10%
Dot-product	72.96%	63.51%	63.96%	68.30%	72.02%	78.62%	67.33%	79.31%	73.38%	74.95%
Hamann	9.26%	21.05%	9.26%	6.86%	12.84%	15.11%	3.83%	51.72%	3.57%	4.73%
Jaccard	78.15%	68.07%	67.83%	70.78%	78.67%	78.40%	69.78%	79.31%	73.70%	75.80%
Kulczynski	82.96%	74.39%	72.49%	77.63%	83.72%	85.38%	80.64%	82.76%	77.27%	82.06%
Ochiai	80.37%	70.53%	68.02%	72.58%	75.69%	80.84%	73.64%	79.31%	75.32%	78.04%
Phi	77.41%	70.88%	68.02%	72.58%	75.23%	80.84%	73.49%	79.31%	75.00%	77.92%
PSC	81.48%	68.77%	70.83%	74.97%	85.09%	81.57%	74.87%	89.66%	75.65%	79.54%
Relative Matching	86.30%	82.46%	75.35%	83.81%	65.83%	90.79%	81.45%	62.07%	90.91%	87.53%
Rogers and Tanimoto	9.63%	21.05%	9.45%	6.88%	13.99%	14.99%	3.83%	51.72%	2.92%	4.76%
Russell and Rao	90.74%	82.81%	83.04%	88.64%	76.83%	91.77%	86.95%	65.52%	93.83%	91.09%
Simple matching	9.26%	21.05%	9.22%	6.88%	12.84%	14.86%	3.83%	51.72%	3.57%	4.73%
Sokal and Sneath	9.26%	21.05%	9.22%	6.82%	12.61%	14.74%	3.75%	51.72%	3.57%	4.70%
Sokal and Sneath 2	80.37%	68.07%	70.83%	73.53%	84.17%	77.76%	71.92%	79.31%	74.68%	77.92%
Sokal and Sneath 4	82.59%	74.04%	72.12%	77.59%	83.26%	85.38%	80.41%	82.76%	77.27%	81.53%
Sokal binary distance	12.59%	28.07%	15.81%	11.54%	25.69%	26.66%	8.00%	48.28%	5.19%	8.67%
Sorenson	72.96%	63.51%	63.96%	68.30%	72.02%	78.62%	67.33%	79.31%	73.38%	74.95%
Yule	72.96%	71.23%	66.08%	65.33%	50.46%	77.89%	68.32%	41.38%	79.22%	71.10%
PTEM	90.78%	83.03%	83.13%	88.74%	78.91%	91.81%	87.31%	65.73%	93.93%	91.30%

To be continued in the next page.

MOVE CLASS

Coefficient	FreeCS Top1	FreeCS Top2	FreeCS Top3	FreeMind Top1	FreeMind Top2	FreeMind Top3	Galleon Top1	Galleon Top2	Galleon Top3	GanttProject Top1	GanttProject Top2	GanttProject Top3	GeoTools2 Top1	GeoTools2 Top2	GeoTools2 Top3	Hadoop Top1	Hadoop Top2	Hadoop Top3	Heritrix Top1	Heritrix Top2	Heritrix Top3	HSQLDB Top1	HSQLDB Top2	HSQLDB Top3	HTMLUnit Top1	HTMLUnit Top2	HTMLUnit Top3	Informa Top1	Informa Top2	Informa Top3
Baroni-Urbani and Buser	66.67%	77.78%	82.05%	24.72%	37.50%	47.44%	31.58%	42.11%	47.37%	25.36%	36.02%	48.70%	53.91%	70.91%	79.06%	15.49%	22.72%	27.15%	20.34%	34.96%	40.69%	20.95%	32.40%	46.37%	44.92%	59.04%	66.38%	29.55%	38.64%	46.59%
Dot-product	71.79%	82.91%	87.18%	43.47%	54.55%	64.20%	42.11%	51.20%	58.37%	42.07%	52.74%	63.69%	63.44%	79.82%	86.97%	28.60%	39.06%	45.28%	35.24%	49.57%	58.74%	41.62%	62.01%	74.02%	54.80%	66.67%	75.71%	34.09%	48.86%	59.09%
Hamann	64.96%	71.79%	79.49%	13.07%	19.32%	25.85%	33.01%	37.32%	39.23%	13.54%	17.87%	21.33%	34.25%	52.75%	65.46%	7.74%	10.38%	12.09%	11.75%	19.48%	25.79%	17.32%	27.09%	33.24%	7.63%	23.16%	42.09%	21.59%	31.82%	38.64%
Jaccard	73.50%	84.62%	86.32%	45.17%	57.95%	67.61%	43.54%	52.15%	57.89%	44.38%	55.33%	64.84%	65.34%	80.25%	87.62%	32.60%	42.47%	48.34%	37.25%	53.87%	63.32%	44.13%	64.25%	75.98%	55.93%	68.08%	78.25%	34.09%	51.14%	63.64%
Kulczynski	64.96%	77.78%	82.91%	43.75%	54.83%	64.77%	42.58%	52.63%	56.46%	38.62%	51.30%	59.94%	64.15%	79.11%	86.67%	25.96%	38.55%	45.02%	35.24%	56.16%	64.47%	41.34%	61.17%	75.42%	52.26%	66.95%	78.53%	32.95%	51.14%	57.95%
Ochiai	70.09%	78.63%	82.91%	44.03%	55.40%	64.77%	44.02%	53.59%	56.94%	40.35%	52.16%	61.10%	63.11%	79.70%	87.12%	29.02%	40.00%	45.70%	34.67%	52.15%	63.90%	41.62%	60.89%	75.14%	54.80%	67.80%	77.97%	32.95%	47.73%	59.09%
Phi	74.36%	81.20%	85.47%	44.03%	55.68%	64.49%	44.02%	53.11%	56.46%	40.35%	52.16%	61.10%	63.32%	79.56%	87.09%	29.02%	39.57%	45.87%	34.96%	52.44%	64.18%	41.34%	61.17%	74.30%	55.08%	67.23%	77.68%	32.95%	48.86%	62.50%
PSC	74.36%	82.91%	85.47%	46.88%	60.51%	71.31%	45.45%	53.11%	60.29%	46.11%	60.23%	71.47%	68.67%	82.91%	89.80%	39.74%	50.30%	56.17%	44.70%	61.03%	68.48%	51.40%	72.07%	82.68%	60.17%	74.58%	86.44%	40.91%	61.36%	69.32%
Relative Matching	35.04%	74.36%	78.63%	33.81%	48.58%	56.25%	36.84%	43.54%	47.85%	34.29%	49.28%	59.08%	56.36%	73.16%	81.03%	15.06%	21.62%	26.38%	18.91%	31.52%	38.68%	36.59%	51.96%	69.55%	37.57%	56.78%	65.54%	26.14%	32.95%	44.32%
Rogers and Tanimoto	66.67%	71.79%	81.20%	13.07%	19.60%	25.57%	33.49%	37.80%	39.23%	13.54%	18.16%	21.61%	35.63%	53.70%	66.01%	7.74%	10.38%	12.09%	11.75%	20.06%	26.07%	17.32%	27.09%	33.52%	7.91%	23.45%	42.09%	21.59%	31.82%	38.64%
Russell and Rao	23.93%	76.07%	81.20%	39.49%	54.26%	62.78%	42.58%	47.37%	52.15%	40.92%	59.08%	68.01%	60.66%	77.92%	85.71%	18.81%	27.15%	33.96%	22.64%	37.54%	45.56%	50.84%	63.97%	77.65%	41.53%	58.19%	71.75%	30.68%	37.50%	55.68%
Simple matching	65.81%	71.79%	79.49%	13.07%	19.60%	25.57%	33.01%	37.32%	39.23%	13.54%	17.87%	21.33%	34.49%	52.79%	65.53%	7.83%	10.38%	12.17%	11.75%	19.48%	25.79%	17.32%	27.09%	33.24%	7.63%	23.16%	42.09%	21.59%	31.82%	38.64%
Sokal and Sneath	65.81%	71.79%	79.49%	13.07%	19.03%	25.85%	33.01%	36.84%	38.76%	12.97%	17.58%	21.04%	33.16%	51.68%	65.06%	7.74%	10.21%	11.91%	11.75%	19.20%	25.79%	17.32%	26.82%	32.96%	7.06%	22.32%	27.97%	21.59%	31.82%	38.64%
Sokal and Sneath 2	75.21%	85.47%	86.32%	46.59%	56.53%	68.18%	44.50%	51.67%	57.42%	45.24%	57.35%	67.44%	66.37%	80.77%	87.95%	35.66%	44.60%	50.81%	38.40%	56.73%	65.62%	45.81%	67.04%	78.21%	57.63%	69.21%	79.66%	34.09%	51.14%	62.50%
Sokal and Sneath 4	71.79%	79.49%	87.18%	43.47%	54.55%	65.06%	41.63%	52.63%	56.94%	38.62%	51.30%	59.65%	64.42%	79.37%	86.93%	26.38%	38.72%	45.36%	35.24%	55.59%	65.33%	40.50%	61.17%	75.42%	52.54%	67.23%	78.53%	34.09%	51.14%	62.50%
Sokal binary distance	66.67%	74.36%	81.20%	13.92%	21.02%	27.84%	34.45%	39.23%	42.58%	15.27%	19.60%	21.90%	39.86%	55.12%	66.98%	8.94%	11.74%	13.87%	12.61%	21.49%	29.80%	17.60%	28.21%	35.20%	32.77%	41.81%	49.44%	23.86%	31.82%	38.64%
Sorenson	71.79%	82.91%	87.18%	43.47%	54.55%	64.20%	42.11%	51.20%	58.37%	42.07%	52.74%	63.69%	63.44%	79.82%	86.97%	28.60%	39.06%	45.28%	35.24%	49.57%	58.74%	41.62%	62.01%	74.02%	54.80%	66.67%	75.71%	34.09%	48.86%	59.09%
Yule	64.96%	70.09%	75.21%	11.93%	24.15%	32.10%	25.84%	34.45%	40.19%	17.87%	27.09%	37.46%	49.16%	62.78%	71.67%	12.26%	18.81%	22.38%	17.19%	29.23%	34.10%	13.97%	19.83%	27.93%	11.86%	49.72%	62.43%	26.14%	38.64%	44.32%
PTMC	74.26%	83.91%	92.99%	55.67%	61.92%	73.64%	51.51%	54.16%	62.40%	58.49%	62.15%	70.32%	75.10%	80.88%	92.06%	44.11%	47.83%	54.38%	46.48%	56.35%	69.01%	58.66%	70.46%	85.65%	66.10%	71.52%	80.75%	43.64%	62.86%	76.55%

MOVE METHOD

Coefficient	FreeCS Top1	FreeCS Top2	FreeCS Top3	FreeMind Top1	FreeMind Top2	FreeMind Top3	Galleon Top1	Galleon Top2	Galleon Top3	GanttProject Top1	GanttProject Top2	GanttProject Top3	GeoTools2 Top1	GeoTools2 Top2	GeoTools2 Top3	Hadoop Top1	Hadoop Top2	Hadoop Top3	Heritrix Top1	Heritrix Top2	Heritrix Top3	HSQLDB Top1	HSQLDB Top2	HSQLDB Top3	HTMLUnit Top1	HTMLUnit Top2	HTMLUnit Top3	Informa Top1	Informa Top2	Informa Top3
Baroni-Urbani and Buser	13.40%	18.21%	24.40%	11.53%	18.48%	23.43%	17.14%	23.85%	28.46%	7.58%	13.87%	18.20%	38.97%	49.74%	56.68%	7.74%	11.58%	14.05%	13.43%	20.35%	24.90%	25.49%	34.78%	42.16%	7.45%	11.68%	14.50%	27.70%	36.49%	46.62%
Dot-product	25.77%	32.99%	37.80%	25.50%	37.40%	44.35%	28.13%	38.35%	43.52%	20.37%	34.78%	42.58%	48.17%	60.96%	68.20%	18.11%	25.42%	29.68%	29.07%	40.30%	47.45%	43.49%	56.35%	63.45%	19.74%	27.19%	32.23%	47.30%	64.19%	74.32%
Hamann	12.71%	20.27%	27.49%	14.34%	19.51%	24.24%	17.03%	22.97%	25.05%	9.43%	13.76%	17.66%	35.80%	44.34%	49.02%	10.15%	13.06%	14.88%	20.58%	27.18%	31.81%	27.88%	37.25%	42.59%	11.58%	18.83%	22.05%	29.73%	37.84%	43.92%
Jaccard	28.18%	37.11%	41.58%	29.19%	40.58%	47.52%	30.77%	41.21%	46.48%	25.79%	39.00%	45.07%	50.41%	63.32%	70.15%	21.11%	29.07%	33.77%	32.91%	44.07%	52.00%	47.14%	59.29%	66.04%	22.16%	30.11%	35.25%	50.00%	67.57%	77.03%
Kulczynski	15.46%	27.15%	35.74%	19.88%	33.33%	42.79%	20.00%	27.25%	31.87%	13.98%	26.00%	33.69%	46.05%	59.61%	67.06%	9.88%	15.75%	19.12%	23.88%	36.92%	44.70%	33.76%	50.20%	58.59%	14.40%	21.75%	28.50%	33.11%	50.00%	62.84%
Ochiai	20.96%	31.62%	37.80%	23.87%	36.29%	45.31%	26.04%	35.27%	40.11%	18.96%	31.20%	37.92%	47.89%	61.10%	68.34%	15.94%	23.07%	26.99%	29.30%	40.38%	48.23%	41.88%	55.45%	62.78%	17.72%	24.97%	31.12%	41.89%	60.14%	72.30%
Phi	20.62%	30.58%	37.46%	23.95%	36.36%	45.68%	26.15%	34.95%	40.00%	18.85%	31.31%	38.14%	47.85%	61.12%	68.20%	15.94%	23.02%	26.99%	29.46%	40.53%	48.39%	41.88%	55.41%	62.78%	17.72%	24.97%	30.82%	42.57%	60.14%	72.97%
PSC	28.87%	38.14%	44.33%	30.38%	43.09%	51.37%	30.66%	41.87%	47.47%	28.49%	42.90%	49.62%	52.52%	65.46%	72.28%	22.20%	29.91%	34.51%	35.82%	47.29%	55.70%	47.92%	61.49%	68.43%	22.96%	31.62%	38.57%	51.35%	67.57%	78.38%
Relative Matching	8.25%	13.40%	18.90%	16.78%	24.09%	28.97%	24.51%	35.93%	40.66%	15.60%	26.65%	32.18%	40.61%	54.75%	61.82%	13.84%	19.40%	22.39%	21.84%	31.81%	38.57%	30.98%	44.39%	53.25%	14.00%	20.64%	26.08%	25.68%	37.84%	47.97%
Rogers and Tanimoto	12.71%	21.31%	28.52%	14.41%	19.66%	24.54%	17.03%	22.97%	25.05%	9.53%	14.08%	18.31%	36.02%	44.55%	49.33%	10.15%	12.97%	14.84%	20.66%	27.34%	31.81%	27.96%	37.61%	43.14%	11.78%	19.03%	22.46%	29.73%	37.84%	43.92%
Russell and Rao	14.09%	24.05%	27.84%	21.95%	32.08%	37.03%	33.19%	42.64%	48.90%	24.05%	37.49%	45.61%	46.07%	58.87%	67.75%	19.99%	27.26%	31.46%	30.01%	42.66%	49.80%	37.61%	52.24%	61.37%	19.44%	27.79%	35.35%	39.19%	50.68%	61.49%
Simple matching	12.71%	20.27%	27.84%	14.04%	19.36%	24.17%	16.92%	22.86%	24.84%	9.43%	13.98%	17.66%	35.75%	44.35%	49.08%	10.15%	13.08%	14.92%	20.42%	27.02%	31.66%	27.84%	37.53%	43.02%	11.28%	18.73%	22.26%	29.73%	37.84%	43.92%
Sokal and Sneath	11.68%	19.59%	26.80%	13.38%	18.63%	23.06%	16.59%	22.64%	24.40%	8.78%	12.89%	16.68%	35.14%	43.70%	48.44%	9.43%	12.13%	13.93%	19.56%	26.08%	30.56%	27.69%	37.10%	42.20%	10.88%	18.43%	21.65%	29.05%	37.84%	43.24%
Sokal and Sneath 2	29.21%	39.86%	43.99%	31.56%	43.68%	50.55%	34.18%	42.64%	48.35%	28.17%	41.82%	48.21%	52.25%	63.61%	71.77%	23.70%	32.07%	37.38%	35.98%	48.47%	54.12%	50.78%	63.02%	69.18%	24.27%	33.84%	39.58%	56.08%	68.92%	78.38%
Sokal and Sneath 4	14.09%	27.49%	36.43%	19.96%	33.26%	42.94%	19.89%	27.25%	31.65%	14.08%	26.00%	33.59%	46.02%	59.52%	66.89%	9.75%	15.64%	19.02%	23.80%	36.92%	44.70%	34.04%	50.31%	58.75%	14.40%	22.26%	28.50%	33.78%	48.65%	64.19%
Sokal binary distance	17.18%	26.80%	35.05%	17.07%	22.62%	28.68%	19.89%	25.38%	27.25%	12.24%	19.39%	24.05%	38.71%	46.59%	52.11%	11.53%	14.42%	16.68%	24.12%	30.32%	35.11%	31.84%	41.41%	46.67%	16.41%	22.56%	25.58%	33.11%	39.19%	46.62%
Sorenson	25.77%	32.99%	37.80%	25.50%	37.40%	44.35%	28.13%	38.35%	43.52%	20.37%	34.78%	42.58%	48.17%	60.96%	68.20%	18.11%	25.42%	29.68%	29.07%	40.30%	47.45%	43.49%	56.35%	63.45%	19.74%	27.19%	32.23%	47.30%	64.19%	74.32%
Yule	4.47%	12.71%	15.46%	9.61%	15.67%	20.33%	10.77%	16.81%	22.97%	5.63%	11.27%	14.30%	31.74%	42.72%	49.37%	2.92%	5.57%	8.03%	8.56%	13.83%	19.17%	14.31%	24.63%	30.71%	3.73%	7.25%	9.77%	14.19%	20.27%	27.70%
PTMMM	49.35%	55.51%	56.50%	41.99%	54.48%	56.58%	44.99%	55.62%	59.18%	50.52%	60.05%	63.53%	61.07%	70.61%	75.03%	42.10%	51.24%	53.80%	50.83%	60.97%	64.09%	61.11%	69.30%	74.43%	46.83%	52.86%	57.26%	59.51%	66.45%	72.27%

EXTRACT METHOD

Coefficient	FreeCS Hits	FreeMind Hits	Galleon Hits	GanttProject Hits	GeoTools2 Hits	Hadoop Hits	Heritrix Hits	HSQLDB Hits	HTMLUnit Hits	Informa Hits
Baroni-Urbani and Buser	53.79%	49.51%	60.48%	62.77%	70.34%	68.08%	59.60%	73.59%	72.58%	37.86%
Dot-product	67.86%	55.35%	69.90%	64.02%	76.28%	72.91%	69.80%	78.89%	78.45%	60.19%
Hamann	14.06%	5.84%	10.48%	6.38%	6.67%	4.50%	9.55%	15.05%	13.34%	3.88%
Jaccard	70.65%	59.72%	72.24%	66.25%	77.25%	74.06%	71.58%	80.61%	80.94%	59.22%
Kulczynski	77.23%	68.86%	79.71%	72.44%	81.97%	82.08%	80.32%	83.35%	89.59%	66.02%
Ochiai	71.54%	62.49%	72.94%	69.15%	78.81%	76.55%	74.17%	80.91%	83.87%	60.19%
Phi	71.65%	62.38%	72.62%	68.47%	77.80%	76.51%	73.68%	80.79%	82.70%	60.19%
PSC	75.78%	65.41%	79.44%	71.57%	78.86%	77.76%	76.11%	82.84%	83.72%	58.25%
Relative Matching	80.25%	72.32%	76.99%	78.43%	88.71%	87.54%	83.89%	88.31%	90.03%	66.02%
Rogers and Tanimoto	14.29%	5.84%	10.32%	6.58%	6.93%	4.54%	9.96%	15.13%	13.34%	1.94%
Russell and Rao	87.05%	83.57%	84.23%	85.59%	92.33%	91.33%	89.39%	90.83%	92.52%	69.90%
Simple matching	14.06%	5.84%	10.48%	5.32%	6.63%	4.54%	9.80%	15.05%	13.34%	3.88%
Sokal and Sneath	13.84%	5.84%	10.32%	5.22%	6.48%	4.48%	9.47%	15.05%	13.34%	3.88%
Sokal and Sneath 2	74.67%	63.68%	75.19%	69.73%	78.68%	76.67%	73.77%	82.51%	83.14%	60.19%
Sokal and Sneath 4	74.89%	68.43%	78.86%	72.24%	79.97%	81.87%	79.92%	82.87%	89.30%	58.25%
Sokal binary distance	17.52%	9.62%	14.41%	12.57%	10.27%	7.48%	12.63%	19.91%	19.06%	3.88%
Sorenson	67.86%	55.35%	69.90%	64.02%	76.28%	72.91%	69.80%	78.89%	78.45%	60.19%
Yule	56.47%	53.41%	65.34%	67.50%	74.91%	73.28%	62.75%	73.47%	76.25%	60.19%
PTEM	88.32%	83.82%	84.45%	85.70%	93.12%	91.54%	89.45%	91.28%	92.59%	72.88%

To be continued in the next page.

MOVE CLASS

Coefficient	iReport Top1	iReport Top2	iReport Top3	iText Top1	iText Top2	iText Top3	IvataGroupWare Top1	IvataGroupWare Top2	IvataGroupWare Top3	JFin_DateMath Top1	JFin_DateMath Top2	JFin_DateMath Top3	Jag Top1	Jag Top2	Jag Top3	James Top1	James Top2	James Top3	Jasml Top1	Jasml Top2	Jasml Top3	JasperReports Top1	JasperReports Top2	JasperReports Top3	JavaCC Top1	JavaCC Top2	JavaCC Top3	JBoss Top1	JBoss Top2	JBoss Top3
Baroni-Urbani and Buser	25.49%	38.33%	46.60%	17.34%	30.92%	39.60%	70.83%	86.11%	90.28%	47.37%	76.32%	81.58%	28.57%	47.62%	57.14%	53.90%	71.00%	79.93%	61.54%	69.23%	84.62%	29.67%	39.96%	55.79%	68.66%	86.57%	92.54%	21.09%	30.91%	38.91%
Dot-product	39.61%	51.04%	60.66%	30.35%	47.40%	58.38%	70.83%	88.89%	93.06%	60.53%	76.32%	86.84%	39.29%	54.76%	67.86%	62.45%	78.07%	82.53%	61.54%	84.62%	100.00%	42.83%	55.79%	71.32%	74.63%	91.04%	94.03%	37.45%	52.73%	57.45%
Hamann	11.90%	14.12%	15.87%	5.78%	13.87%	17.34%	37.50%	55.56%	65.28%	44.74%	71.05%	81.58%	22.62%	47.62%	52.38%	28.25%	47.58%	61.34%	53.85%	61.54%	76.92%	13.55%	15.33%	16.82%	7.46%	40.30%	62.69%	14.55%	21.82%	25.45%
Jaccard	41.63%	53.19%	62.81%	33.24%	48.55%	63.01%	70.83%	88.89%	93.06%	57.89%	76.32%	86.84%	41.67%	60.71%	69.05%	64.68%	78.44%	82.90%	61.54%	84.62%	100.00%	44.71%	58.95%	72.70%	76.12%	91.04%	97.01%	40.36%	55.64%	60.73%
Kulczynski	28.04%	52.25%	60.93%	23.12%	40.75%	53.76%	66.67%	88.89%	93.06%	55.26%	78.95%	86.84%	42.86%	57.14%	69.05%	52.79%	73.61%	82.53%	38.46%	69.23%	92.31%	43.42%	55.79%	72.80%	77.61%	89.55%	94.03%	40.00%	54.55%	62.18%
Ochiai	36.11%	51.45%	60.52%	27.46%	44.51%	59.54%	70.83%	88.89%	93.06%	60.53%	76.32%	86.84%	41.67%	57.14%	67.86%	59.85%	78.44%	82.53%	53.85%	76.92%	100.00%	42.63%	55.19%	71.81%	76.12%	89.55%	94.03%	37.82%	52.36%	60.36%
Phi	36.11%	51.38%	60.46%	26.59%	43.35%	57.80%	70.83%	87.50%	91.67%	50.00%	78.95%	86.84%	41.67%	55.95%	66.67%	61.34%	79.55%	83.27%	61.54%	76.92%	92.31%	42.73%	55.49%	71.61%	76.12%	89.55%	97.01%	38.55%	53.45%	60.36%
PSC	43.51%	56.69%	67.32%	34.10%	54.05%	68.50%	75.00%	90.28%	94.44%	63.16%	84.21%	89.47%	50.00%	67.86%	76.19%	65.06%	80.67%	84.39%	61.54%	84.62%	100.00%	52.13%	67.75%	77.65%	76.12%	92.54%	97.01%	45.82%	59.64%	65.45%
Relative Matching	26.97%	45.39%	53.13%	24.86%	49.13%	61.56%	59.72%	84.72%	87.50%	60.53%	76.32%	84.21%	30.95%	45.24%	59.52%	53.53%	66.91%	75.46%	7.69%	46.15%	100.00%	36.30%	49.36%	63.30%	67.16%	85.07%	94.03%	27.64%	41.45%	49.45%
Rogers and Tanimoto	11.84%	14.19%	15.87%	5.78%	13.87%	17.34%	44.44%	56.94%	65.28%	44.74%	68.42%	81.58%	22.62%	47.62%	52.38%	29.00%	48.70%	60.97%	53.85%	61.54%	76.92%	13.65%	15.43%	16.82%	8.96%	47.76%	62.69%	14.55%	21.45%	25.09%
Russell and Rao	34.03%	50.24%	58.37%	38.15%	64.74%	76.30%	61.11%	90.28%	91.67%	63.16%	86.84%	89.47%	40.48%	58.33%	70.24%	54.65%	70.63%	77.32%	15.38%	61.54%	100.00%	46.29%	60.83%	71.91%	70.15%	92.54%	98.51%	34.18%	53.82%	60.36%
Simple matching	11.97%	14.19%	15.87%	5.78%	13.58%	16.76%	38.89%	54.17%	65.28%	44.74%	71.05%	81.58%	22.62%	47.62%	52.38%	28.25%	47.21%	61.34%	53.85%	61.54%	76.92%	13.35%	15.23%	16.82%	7.46%	40.30%	62.69%	14.55%	21.82%	25.45%
Sokal and Sneath	11.84%	13.99%	15.80%	5.49%	13.29%	16.76%	37.50%	52.78%	65.28%	42.11%	71.05%	78.95%	22.62%	47.62%	51.19%	27.51%	47.21%	60.97%	53.85%	61.54%	76.92%	13.16%	14.84%	16.72%	5.97%	38.81%	62.69%	14.18%	21.09%	24.73%
Sokal and Sneath 2	43.17%	54.74%	64.36%	35.26%	52.02%	64.45%	72.22%	88.89%	93.06%	57.89%	78.95%	89.47%	46.43%	63.10%	70.24%	65.06%	78.81%	83.27%	61.54%	92.31%	100.00%	46.79%	60.53%	73.89%	76.12%	91.04%	97.01%	41.09%	56.36%	61.09%
Sokal and Sneath 4	28.11%	52.12%	61.00%	21.97%	40.46%	52.60%	69.44%	87.50%	90.28%	47.37%	81.58%	86.84%	41.67%	53.57%	65.48%	53.90%	77.32%	83.64%	38.46%	76.92%	92.31%	43.22%	55.29%	72.90%	73.13%	91.04%	95.52%	40.00%	53.82%	62.18%
Sokal binary distance	12.64%	15.27%	17.15%	7.80%	16.18%	19.36%	45.83%	56.94%	63.89%	42.11%	68.42%	81.58%	27.38%	46.43%	51.19%	31.97%	49.07%	63.20%	46.15%	53.85%	69.23%	15.13%	17.21%	18.20%	14.93%	50.75%	74.63%	17.09%	24.73%	27.64%
Sorenson	39.61%	51.04%	60.66%	30.35%	47.40%	58.38%	70.83%	88.89%	93.06%	60.53%	76.32%	86.84%	39.29%	54.76%	67.86%	62.45%	78.07%	82.53%	61.54%	84.62%	100.00%	42.83%	55.79%	71.32%	74.63%	91.04%	94.03%	37.45%	52.73%	57.45%
Yule	15.27%	18.83%	23.60%	15.90%	28.03%	37.57%	51.39%	76.39%	86.11%	42.11%	71.05%	84.21%	22.62%	40.48%	52.38%	39.41%	55.76%	70.26%	30.77%	69.23%	84.62%	23.74%	35.71%	47.18%	68.66%	85.07%	92.54%	20.73%	29.09%	36.00%
PTMC	53.84%	59.89%	70.03%	48.46%	58.01%	72.53%	84.11%	89.10%	97.31%	71.01%	82.05%	91.92%	59.15%	67.29%	71.15%	74.76%	81.74%	85.97%	73.24%	92.84%	100.00%	59.47%	63.91%	72.62%	81.85%	92.84%	100.00%	49.62%	57.45%	68.60%

MOVE METHOD

Coefficient	iReport Top1	iReport Top2	iReport Top3	iText Top1	iText Top2	iText Top3	IvataGroupWare Top1	IvataGroupWare Top2	IvataGroupWare Top3	JFin_DateMath Top1	JFin_DateMath Top2	JFin_DateMath Top3	Jag Top1	Jag Top2	Jag Top3	James Top1	James Top2	James Top3	Jasml Top1	Jasml Top2	Jasml Top3	JasperReports Top1	JasperReports Top2	JasperReports Top3	JavaCC Top1	JavaCC Top2	JavaCC Top3	JBoss Top1	JBoss Top2	JBoss Top3
Baroni-Urbani and Buser	36.74%	41.81%	44.42%	15.76%	21.89%	26.97%	57.11%	72.05%	78.07%	78.13%	87.50%	93.75%	25.59%	39.81%	51.18%	20.65%	29.22%	34.42%	73.91%	94.20%	98.55%	20.76%	27.44%	30.49%	23.57%	38.85%	46.50%	18.56%	24.57%	27.64%
Dot-product	45.75%	51.98%	55.89%	22.59%	32.92%	38.97%	68.19%	80.96%	86.99%	78.13%	90.63%	96.88%	37.44%	54.50%	66.35%	34.42%	44.16%	48.18%	79.71%	94.20%	98.55%	35.99%	44.25%	49.42%	44.59%	55.41%	68.15%	34.98%	43.39%	50.87%
Hamann	34.82%	38.95%	40.46%	15.76%	21.89%	26.01%	48.19%	59.04%	63.37%	90.63%	93.75%	93.75%	16.11%	26.54%	27.96%	27.40%	34.42%	38.44%	47.83%	63.77%	63.77%	25.42%	29.69%	31.17%	43.95%	54.14%	57.96%	22.96%	28.84%	33.78%
Jaccard	47.93%	53.64%	57.55%	26.09%	35.64%	42.38%	69.88%	81.69%	88.19%	78.13%	90.63%	96.88%	39.81%	57.35%	67.30%	37.27%	45.58%	51.82%	79.71%	94.20%	97.10%	39.59%	48.78%	53.95%	49.04%	65.61%	77.71%	37.38%	48.60%	54.61%
Kulczynski	44.17%	49.86%	54.54%	22.50%	31.17%	37.30%	57.35%	75.66%	84.10%	81.25%	96.88%	100.00%	30.33%	47.87%	56.40%	34.42%	43.12%	49.09%	65.22%	89.86%	95.65%	28.37%	36.09%	41.23%	40.13%	50.32%	61.15%	31.11%	42.06%	49.93%
Ochiai	45.92%	52.02%	56.11%	23.12%	32.05%	38.44%	65.54%	80.00%	86.51%	78.13%	87.50%	100.00%	35.07%	51.18%	63.98%	36.49%	44.55%	49.74%	79.71%	94.20%	98.55%	34.19%	41.45%	46.53%	42.04%	59.24%	66.88%	33.24%	44.06%	51.94%
Phi	45.92%	52.00%	56.13%	23.03%	32.05%	38.62%	64.58%	79.04%	85.54%	78.13%	90.63%	100.00%	34.60%	50.71%	62.56%	36.10%	44.68%	49.48%	79.71%	92.75%	97.10%	34.16%	41.29%	46.43%	41.40%	58.60%	68.79%	33.38%	44.06%	51.94%
PSC	48.68%	55.23%	59.45%	27.41%	37.57%	43.61%	69.64%	82.41%	89.16%	78.13%	96.88%	100.00%	39.81%	57.82%	68.72%	39.87%	49.22%	56.23%	78.26%	94.20%	97.10%	40.07%	49.45%	55.98%	54.78%	66.88%	78.34%	40.05%	52.74%	58.88%
Relative Matching	42.49%	46.82%	49.18%	20.58%	28.20%	31.70%	42.89%	60.00%	78.31%	65.63%	84.38%	93.75%	34.12%	51.18%	60.19%	24.55%	34.68%	40.39%	82.61%	86.96%	98.55%	30.43%	40.58%	44.83%	20.38%	31.21%	40.76%	27.50%	38.05%	42.86%
Rogers and Tanimoto	34.95%	39.17%	40.63%	15.67%	22.07%	26.27%	48.67%	59.04%	65.30%	90.63%	93.75%	93.75%	16.59%	26.07%	27.96%	27.40%	34.55%	38.83%	47.83%	63.77%	63.77%	25.61%	30.04%	31.59%	43.95%	54.78%	58.60%	22.96%	29.11%	33.64%
Russell and Rao	47.17%	52.22%	54.86%	25.13%	34.85%	40.11%	49.64%	67.95%	82.41%	65.63%	87.50%	93.75%	42.18%	56.87%	64.45%	30.52%	43.51%	49.09%	79.71%	85.51%	98.55%	38.53%	48.04%	53.66%	27.39%	40.76%	49.68%	35.51%	45.53%	52.47%
Simple matching	34.84%	39.04%	40.59%	15.67%	21.98%	26.27%	48.43%	59.04%	63.61%	87.50%	93.75%	93.75%	15.64%	25.59%	27.96%	27.14%	34.16%	38.44%	47.83%	57.97%	57.97%	25.35%	29.66%	31.23%	43.95%	54.14%	57.96%	22.83%	28.84%	33.24%
Sokal and Sneath	34.60%	38.60%	40.07%	15.41%	21.02%	25.04%	47.71%	57.83%	62.65%	87.50%	93.75%	93.75%	15.64%	26.07%	27.49%	26.36%	33.25%	37.66%	47.83%	57.97%	57.97%	24.26%	28.60%	30.04%	40.76%	54.14%	57.96%	22.43%	28.17%	32.71%
Sokal and Sneath 2	49.81%	55.45%	58.62%	28.37%	37.22%	44.40%	71.08%	82.89%	89.40%	81.25%	90.63%	96.88%	43.13%	58.77%	69.67%	40.65%	49.35%	54.68%	79.71%	94.20%	97.10%	43.89%	53.15%	58.64%	54.14%	67.52%	78.34%	40.45%	51.94%	58.48%
Sokal and Sneath 4	44.15%	49.86%	54.47%	22.50%	31.17%	37.13%	58.07%	76.14%	84.10%	81.25%	96.88%	100.00%	30.33%	47.87%	56.40%	34.16%	42.73%	48.83%	65.22%	89.86%	95.65%	27.83%	35.54%	40.78%	39.49%	52.87%	63.69%	31.11%	41.92%	49.93%
Sokal binary distance	36.85%	41.49%	42.86%	19.53%	24.34%	29.07%	52.05%	62.41%	67.23%	87.50%	90.63%	90.63%	19.43%	28.91%	32.70%	30.00%	37.40%	41.95%	53.62%	68.12%	68.12%	29.50%	34.06%	36.66%	47.13%	59.87%	68.15%	24.43%	31.51%	35.78%
Sorenson	45.75%	51.98%	55.89%	22.59%	32.92%	38.97%	68.19%	80.96%	86.99%	78.13%	90.63%	96.88%	37.44%	54.50%	66.35%	34.42%	44.16%	48.18%	79.71%	94.20%	98.55%	35.99%	44.25%	49.42%	44.59%	55.41%	68.15%	34.98%	43.39%	50.87%
Yule	12.70%	15.21%	17.86%	13.92%	16.90%	21.37%	28.92%	45.30%	56.39%	50.00%	93.75%	96.88%	17.54%	28.44%	44.08%	12.86%	19.87%	22.99%	63.77%	88.41%	95.65%	9.45%	19.54%	22.33%	12.10%	22.29%	26.11%	14.29%	21.50%	23.90%
PTMM	59.26%	63.07%	63.69%	50.07%	56.11%	59.99%	72.54%	85.43%	90.74%	73.08%	92.22%	97.28%	50.89%	69.21%	77.71%	47.46%	61.09%	63.70%	84.24%	86.67%	98.03%	54.96%	63.98%	67.47%	60.25%	72.37%	74.68%	52.89%	63.17%	65.93%

EXTRACT METHOD

Coefficient	iReport Hits	iText Hits	IvataGroupWare Hits	JFin_DateMath Hits	Jag Hits	James Hits	Jasml Hits	JasperReports Hits	JavaCC Hits	JBoss Hits
Baroni-Urbani and Buser	72.95%	58.31%	86.98%	16.67%	69.76%	54.39%	35.75%	52.89%	59.04%	58.17%
Dot-product	79.26%	65.90%	90.48%	16.67%	75.15%	56.98%	54.40%	65.21%	64.03%	66.00%
Hamann	22.87%	10.36%	25.56%	0.00%	15.27%	6.79%	2.07%	10.48%	14.80%	5.00%
Jaccard	80.87%	67.68%	91.59%	23.08%	76.88%	60.08%	54.92%	66.68%	71.26%	67.50%
Kulczynski	84.04%	76.06%	93.17%	25.00%	87.72%	71.09%	75.65%	75.84%	75.73%	73.83%
Ochiai	81.56%	69.85%	92.06%	25.00%	79.04%	63.62%	60.62%	69.51%	70.05%	69.50%
Phi	81.48%	69.79%	88.89%	25.00%	78.14%	62.70%	56.48%	69.22%	70.05%	69.33%
PSC	82.95%	71.83%	92.86%	25.00%	79.34%	62.93%	56.99%	71.39%	76.94%	72.50%
Relative Matching	87.07%	79.95%	93.97%	25.00%	93.41%	77.27%	72.02%	76.98%	82.44%	80.50%
Rogers and Tanimoto	22.87%	10.88%	26.35%	0.00%	15.27%	7.02%	2.07%	10.52%	15.66%	5.00%
Russell and Rao	89.31%	83.18%	94.44%	41.67%	96.11%	82.46%	87.56%	84.92%	89.16%	85.33%
Simple matching	22.87%	10.55%	25.71%	0.00%	15.27%	6.79%	2.07%	10.41%	14.80%	5.00%
Sokal and Sneath	22.87%	9.96%	25.56%	0.00%	15.27%	6.56%	2.07%	10.33%	14.80%	5.00%
Sokal and Sneath 2	82.38%	71.50%	92.06%	33.33%	78.74%	61.71%	54.40%	69.73%	77.80%	69.50%
Sokal and Sneath 4	83.97%	75.53%	88.73%	25.00%	85.03%	69.64%	56.48%	74.84%	73.84%	73.33%
Sokal binary distance	28.28%	18.73%	29.21%	0.00%	19.46%	11.59%	2.07%	14.49%	19.45%	7.00%
Sorenson	79.26%	65.90%	90.48%	16.67%	75.15%	56.98%	54.40%	65.21%	64.03%	66.00%
Yule	74.15%	63.46%	87.94%	16.67%	75.15%	62.09%	54.92%	54.36%	60.07%	62.17%
PTEM	89.55%	83.87%	95.41%	41.85%	96.54%	83.33%	89.79%	85.17%	89.75%	85.54%

To be continued in the next page.

MOVE CLASS																														
	JChemPaint			Jena			Jext			JGraph			JGraphPad			JGraphT			JGroups			JMoney			JOggPlayer			JParse		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	96.18%	96.86%	97.25%	41.00%	47.75%	56.27%	47.03%	54.52%	59.69%	17.93%	26.90%	35.86%	60.85%	71.89%	82.56%	17.29%	39.85%	54.89%	24.30%	41.43%	52.19%	58.33%	85.42%	97.92%	42.86%	62.50%	73.21%	76.92%	87.69%	100.00%
Dot-product	97.16%	97.91%	98.32%	51.93%	69.13%	77.01%	58.14%	66.15%	72.87%	25.52%	35.86%	48.97%	72.60%	81.14%	87.54%	37.59%	57.89%	67.67%	33.86%	54.98%	64.54%	66.67%	87.50%	97.92%	60.12%	73.81%	80.95%	86.15%	93.85%	100.00%
Hamann	94.86%	95.36%	95.74%	25.24%	32.15%	35.85%	40.83%	54.01%	56.07%	6.90%	13.10%	15.86%	52.67%	59.07%	67.97%	23.31%	30.08%	35.34%	9.16%	12.75%	20.72%	43.75%	58.33%	100.00%	9.52%	13.10%	14.29%	70.77%	76.92%	100.00%
Jaccard	97.27%	98.08%	98.41%	54.82%	72.51%	79.26%	61.76%	67.96%	76.49%	25.52%	40.00%	51.03%	72.95%	82.92%	89.68%	44.36%	63.16%	72.18%	37.45%	56.18%	67.33%	70.83%	87.50%	97.92%	60.71%	75.60%	81.55%	86.15%	93.85%	100.00%
Kulczynski	96.50%	97.64%	98.07%	49.68%	64.15%	73.95%	59.17%	65.63%	72.61%	24.83%	37.24%	51.72%	66.90%	82.21%	88.97%	36.09%	56.39%	67.67%	36.65%	57.37%	66.53%	56.25%	85.42%	95.83%	47.02%	67.86%	73.21%	89.23%	93.85%	100.00%
Ochiai	97.13%	97.98%	98.30%	52.57%	67.85%	77.01%	57.88%	66.41%	72.61%	24.83%	38.62%	51.03%	70.46%	81.85%	89.32%	36.84%	60.15%	71.43%	39.44%	55.78%	65.74%	64.58%	87.50%	97.92%	55.36%	71.43%	75.00%	87.69%	93.85%	100.00%
Phi	97.13%	97.98%	98.32%	52.25%	68.17%	76.69%	57.62%	65.37%	72.35%	27.59%	36.55%	51.03%	69.75%	81.85%	89.68%	37.59%	60.15%	72.18%	39.44%	55.78%	66.14%	66.67%	87.50%	97.92%	55.95%	71.43%	76.19%	84.62%	93.85%	100.00%
PSC	97.42%	98.25%	98.55%	61.74%	76.69%	83.44%	62.02%	71.58%	79.59%	31.03%	43.45%	54.48%	73.31%	83.99%	88.97%	46.62%	71.43%	76.69%	45.42%	62.55%	73.71%	70.83%	85.42%	97.92%	64.88%	78.57%	84.52%	89.23%	93.85%	100.00%
Relative Matching	96.12%	96.95%	97.42%	44.37%	59.16%	66.56%	51.68%	61.50%	64.34%	10.34%	26.90%	40.69%	61.21%	74.38%	83.27%	19.55%	40.60%	47.37%	24.70%	48.21%	56.97%	54.17%	77.08%	93.75%	36.31%	58.33%	72.02%	84.62%	90.77%	100.00%
Rogers and Tanimoto	94.89%	95.38%	95.75%	25.56%	32.32%	35.85%	40.83%	54.01%	56.07%	6.90%	13.10%	15.86%	52.67%	60.50%	67.97%	23.31%	30.08%	38.35%	8.76%	13.15%	21.12%	47.92%	64.58%	100.00%	9.52%	13.10%	14.29%	73.85%	76.92%	100.00%
Russell and Rao	96.42%	97.42%	97.78%	52.09%	68.81%	77.49%	58.14%	67.44%	72.35%	14.48%	37.93%	48.28%	66.55%	75.44%	83.99%	24.81%	51.13%	62.41%	27.89%	53.78%	63.35%	52.08%	75.00%	91.67%	34.52%	61.90%	75.60%	89.23%	93.85%	100.00%
Simple matching	94.86%	95.37%	95.72%	25.24%	32.15%	35.85%	40.57%	53.75%	55.56%	6.90%	12.41%	15.86%	52.67%	59.07%	67.97%	23.31%	30.08%	37.59%	9.16%	12.75%	20.72%	43.75%	58.33%	100.00%	9.52%	13.10%	14.29%	70.77%	76.92%	100.00%
Sokal and Sneath	94.83%	95.33%	95.69%	25.24%	32.15%	35.69%	40.05%	53.75%	55.56%	6.90%	12.41%	15.86%	52.67%	59.07%	67.26%	22.56%	30.08%	35.34%	9.16%	12.35%	20.72%	43.75%	56.25%	100.00%	9.52%	13.10%	14.29%	70.77%	73.85%	100.00%
Sokal and Sneath 2	97.38%	98.15%	98.44%	57.07%	73.63%	79.90%	63.31%	68.99%	77.52%	26.90%	41.38%	51.72%	73.31%	82.92%	90.04%	48.12%	66.92%	75.19%	39.04%	59.76%	68.13%	72.92%	87.50%	97.92%	61.31%	77.38%	82.74%	86.15%	93.85%	100.00%
Sokal and Sneath 4	96.54%	97.64%	98.07%	49.84%	63.99%	73.47%	58.91%	65.37%	72.61%	26.21%	38.62%	47.59%	67.26%	81.85%	89.68%	37.59%	57.14%	69.17%	37.05%	56.97%	66.93%	60.42%	87.50%	97.92%	51.79%	68.45%	75.00%	86.15%	93.85%	100.00%
Sokal binary distance	95.19%	95.77%	96.17%	26.53%	35.69%	38.75%	46.51%	55.30%	58.14%	8.28%	13.79%	17.93%	53.38%	64.77%	70.46%	24.81%	32.33%	44.36%	9.16%	16.33%	24.70%	47.92%	66.67%	97.92%	12.50%	14.88%	16.67%	72.31%	75.38%	100.00%
Sorenson	97.16%	97.91%	98.32%	51.93%	69.13%	77.01%	58.14%	66.15%	72.87%	25.52%	35.86%	48.97%	72.60%	81.14%	87.54%	37.59%	57.89%	67.67%	33.86%	54.98%	64.54%	66.67%	87.50%	97.92%	60.12%	73.81%	80.95%	86.15%	93.85%	100.00%
Yule	94.14%	94.92%	96.15%	34.89%	40.68%	45.66%	36.69%	46.25%	56.85%	14.48%	22.76%	29.66%	56.94%	65.84%	74.02%	11.28%	18.80%	23.31%	14.34%	28.29%	34.66%	56.25%	77.08%	93.75%	33.33%	54.17%	65.48%	75.38%	86.15%	100.00%
PTMC	98.22%	98.44%	100.00%	68.82%	76.26%	82.94%	74.40%	73.79%	85.24%	33.98%	44.99%	56.12%	81.77%	82.89%	90.69%	55.39%	63.43%	77.11%	46.22%	59.22%	76.35%	76.64%	87.91%	100.00%	65.73%	73.79%	82.91%	94.38%	92.57%	100.00%

MOVE METHOD																														
	JChemPaint			Jena			Jext			JGraph			JGraphPad			JGraphT			JGroups			JMoney			JOggPlayer			JParse		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	13.65%	16.26%	19.78%	9.72%	15.59%	19.20%	4.30%	7.62%	10.35%	8.85%	14.53%	19.43%	9.85%	17.73%	20.69%	16.72%	24.63%	29.33%	10.08%	17.44%	22.24%	30.57%	50.32%	63.06%	20.59%	29.41%	37.94%	79.52%	82.53%	84.34%
Dot-product	22.88%	29.67%	35.32%	21.75%	33.83%	41.05%	16.02%	23.05%	26.17%	15.96%	27.96%	38.07%	24.38%	34.48%	39.16%	20.82%	36.07%	44.28%	21.28%	31.47%	39.52%	40.76%	64.97%	71.34%	35.88%	52.35%	65.59%	83.73%	87.35%	89.76%
Hamann	16.93%	20.81%	25.30%	14.67%	19.99%	24.57%	13.48%	17.97%	20.12%	7.58%	12.80%	17.85%	24.14%	28.82%	31.03%	19.94%	32.55%	37.83%	9.81%	15.63%	17.92%	18.47%	24.84%	28.03%	21.18%	32.94%	42.06%	76.51%	77.71%	77.71%
Jaccard	25.42%	34.10%	39.32%	27.81%	38.41%	45.40%	20.31%	28.32%	31.64%	19.43%	34.28%	42.81%	31.03%	37.68%	42.86%	29.62%	42.82%	52.49%	24.48%	36.59%	44.48%	45.86%	64.33%	74.52%	43.53%	56.47%	70.88%	83.73%	86.75%	89.76%
Kulczynski	18.81%	24.88%	28.58%	15.41%	24.66%	31.51%	17.19%	26.56%	29.49%	11.69%	21.64%	29.70%	26.11%	36.21%	42.12%	20.82%	32.84%	42.52%	18.03%	28.43%	34.99%	40.76%	61.15%	73.25%	33.24%	49.41%	62.65%	78.92%	80.72%	81.93%
Ochiai	22.09%	29.25%	33.86%	19.94%	31.79%	38.82%	16.60%	24.41%	28.52%	14.06%	26.70%	35.23%	25.37%	35.47%	41.63%	22.58%	36.95%	44.28%	20.05%	29.97%	37.87%	41.40%	65.61%	71.34%	36.18%	52.35%	64.12%	81.33%	83.13%	84.34%
Phi	22.09%	29.25%	33.86%	19.90%	31.65%	38.73%	16.60%	24.61%	28.71%	14.06%	26.54%	35.07%	25.37%	35.47%	42.12%	22.29%	36.95%	44.57%	20.05%	30.08%	37.60%	41.40%	64.33%	71.34%	36.18%	52.35%	63.82%	80.72%	83.13%	84.34%
PSC	26.09%	35.01%	42.17%	28.60%	38.96%	46.69%	23.83%	32.62%	36.91%	21.33%	35.86%	46.76%	33.00%	42.36%	47.54%	32.55%	46.33%	56.89%	26.40%	39.47%	47.95%	48.41%	66.24%	74.52%	44.12%	59.41%	74.12%	83.13%	84.94%	86.75%
Relative Matching	14.62%	19.84%	23.06%	18.60%	27.16%	33.41%	4.88%	8.59%	11.72%	9.48%	18.01%	25.12%	10.59%	17.00%	22.17%	19.35%	25.81%	29.33%	16.75%	26.03%	32.43%	38.85%	59.87%	68.15%	33.82%	45.59%	55.29%	84.94%	88.55%	90.96%
Rogers and Tanimoto	17.05%	21.12%	25.67%	14.99%	20.18%	24.76%	13.87%	18.75%	20.51%	7.58%	12.80%	18.17%	24.14%	28.82%	31.03%	19.94%	32.84%	38.12%	9.92%	15.79%	18.03%	22.29%	26.75%	29.94%	21.18%	32.65%	42.35%	76.51%	77.71%	77.71%
Russell and Rao	19.78%	26.46%	29.85%	25.64%	38.13%	43.91%	10.74%	19.73%	25.00%	11.53%	22.75%	32.39%	21.18%	29.56%	34.24%	26.39%	31.96%	37.24%	24.91%	35.63%	44.16%	52.23%	67.52%	74.52%	47.65%	58.53%	68.82%	86.14%	91.57%	93.37%
Simple matching	16.87%	20.75%	25.36%	14.72%	19.99%	24.57%	13.28%	18.36%	19.92%	7.42%	12.32%	17.69%	22.66%	27.59%	29.80%	19.65%	32.55%	37.83%	9.76%	15.63%	17.87%	21.02%	26.75%	29.94%	20.59%	32.06%	41.18%	76.51%	77.71%	77.71%
Sokal and Sneath	16.69%	19.90%	24.45%	13.84%	19.44%	23.60%	12.30%	16.41%	18.36%	7.27%	12.16%	17.06%	21.67%	26.60%	28.82%	18.48%	31.67%	36.66%	9.49%	14.93%	17.55%	17.83%	23.57%	26.11%	20.00%	31.76%	40.88%	75.90%	76.51%	76.51%
Sokal and Sneath 2	28.40%	37.80%	43.14%	30.82%	41.65%	48.36%	23.24%	32.62%	36.13%	22.91%	36.33%	46.60%	33.00%	40.39%	44.58%	39.00%	53.37%	60.41%	27.95%	40.80%	47.47%	48.41%	65.61%	74.52%	46.18%	61.76%	72.65%	84.94%	85.54%	87.95%
Sokal and Sneath 4	18.69%	24.88%	28.58%	15.36%	24.62%	31.51%	17.19%	26.56%	29.49%	12.01%	21.48%	29.23%	26.11%	36.21%	41.87%	20.53%	32.84%	42.52%	18.03%	28.37%	35.04%	39.49%	61.15%	72.61%	33.24%	49.41%	62.65%	78.31%	80.72%	81.93%
Sokal binary distance	19.24%	24.70%	29.13%	18.09%	24.34%	28.37%	17.58%	23.05%	25.59%	9.48%	15.64%	22.59%	25.62%	31.28%	33.25%	23.75%	34.60%	42.23%	11.63%	18.51%	20.91%	24.20%	29.30%	33.12%	30.59%	40.59%	50.29%	76.51%	77.71%	77.71%
Sorenson	22.88%	29.67%	35.32%	21.75%	33.83%	41.05%	16.02%	23.05%	26.17%	15.96%	27.96%	38.07%	24.38%	34.48%	39.16%	20.82%	36.07%	44.28%	21.28%	31.47%	39.52%	40.76%	64.97%	71.34%	35.88%	52.35%	65.59%	83.73%	87.35%	89.76%
Yule	4.73%	8.19%	9.34%	4.21%	8.19%	11.20%	1.76%	4.69%	7.03%	6.95%	10.58%	14.38%	4.19%	10.34%	13.55%	9.09%	20.82%	24.34%	7.63%	13.97%	18.83%	28.03%	43.95%	59.24%	16.76%	23.53%	34.41%	74.70%	76.51%	77.71%
PTMM	39.40%	47.54%	52.57%	45.50%	55.40%	61.27%	41.18%	46.56%	48.40%	31.67%	44.66%	52.01%	42.74%	48.95%	51.56%	56.19%	66.24%	69.80%	47.96%	56.93%	62.99%	68.60%	74.83%	81.16%	60.69%	74.11%	78.55%	91.59%	90.21%	91.72%

EXTRACT METHOD										
	JChemPaint	Jena	Jext	JGraph	JGraphPad	JGraphT	JGroups	JMoney	JOggPlayer	JParse
Coefficient	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits
Baroni-Urbani and Buser	54.90%	64.85%	63.70%	45.64%	51.33%	38.10%	62.66%	60.00%	57.71%	95.21%
Dot-product	61.97%	67.38%	66.48%	53.54%	55.56%	46.03%	68.00%	64.29%	72.00%	96.41%
Hamann	10.11%	8.34%	14.44%	7.08%	3.78%	15.87%	5.27%	2.86%	14.86%	17.55%
Jaccard	64.65%	70.29%	68.70%	58.26%	57.56%	50.79%	70.01%	64.29%	72.00%	97.21%
Kulczynski	70.58%	79.01%	77.41%	68.29%	70.89%	53.97%	80.92%	78.57%	77.14%	97.21%
Ochiai	65.49%	72.95%	71.85%	59.47%	63.33%	50.79%	73.05%	70.00%	73.52%	96.41%
Phi	65.30%	73.07%	71.67%	59.35%	62.67%	50.79%	72.68%	70.00%	73.33%	96.14%
PSC	67.50%	76.11%	74.26%	65.51%	63.78%	52.38%	78.40%	71.43%	74.67%	97.34%
Relative Matching	82.98%	81.29%	80.74%	67.36%	71.11%	63.49%	80.92%	77.14%	81.14%	94.28%
Rogers and Tanimoto	10.03%	8.34%	14.44%	7.55%	3.78%	15.87%	5.27%	2.86%	15.62%	19.02%
Russell and Rao	85.95%	88.24%	85.37%	76.42%	81.11%	69.84%	88.34%	85.71%	85.14%	96.81%
Simple matching	10.11%	8.22%	14.44%	7.08%	3.78%	15.87%	5.27%	2.86%	14.86%	17.55%
Sokal and Sneath	10.06%	8.22%	14.26%	7.08%	3.78%	15.87%	5.27%	2.86%	14.86%	16.76%
Sokal and Sneath 2	67.56%	74.08%	73.52%	62.95%	61.78%	50.79%	74.31%	64.29%	72.95%	97.21%
Sokal and Sneath 4	70.41%	78.76%	77.04%	67.36%	70.67%	53.97%	80.48%	77.14%	76.38%	96.94%
Sokal binary distance	15.20%	14.29%	18.89%	14.63%	9.78%	17.46%	7.87%	1.43%	24.19%	93.09%
Sorenson	61.97%	67.38%	66.48%	53.54%	55.56%	46.03%	68.00%	64.29%	72.00%	96.41%
Yule	59.49%	67.26%	71.11%	52.03%	50.89%	42.86%	66.22%	61.43%	60.00%	93.09%
PTEM	86.28%	88.31%	85.74%	77.34%	81.71%	69.94%	88.72%	87.35%	85.23%	97.08%

To be continued in the next page.

MOVE CLASS

Coefficient	JPF Top1	JPF Top2	JPF Top3	JRat Top1	JRat Top2	JRat Top3	JRE Top1	JRE Top2	JRE Top3	JREFactory Top1	JREFactory Top2	JREFactory Top3	JRuby Top1	JRuby Top2	JRuby Top3
Baroni-Urbani and Buser	64.62%	75.38%	84.62%	38.51%	51.72%	57.47%	21.64%	27.60%	32.71%	40.53%	49.34%	55.95%	33.18%	47.35%	53.00%
Dot-product	64.62%	80.00%	87.69%	45.98%	58.62%	63.22%	40.86%	52.17%	57.71%	54.63%	64.87%	71.37%	50.94%	65.65%	71.30%
Hamann	24.62%	43.08%	55.38%	25.29%	32.18%	35.63%	12.07%	14.41%	15.48%	22.58%	23.57%	28.74%	26.19%	37.58%	40.45%
Jaccard	66.15%	80.00%	89.23%	47.13%	58.62%	63.79%	43.37%	54.51%	60.07%	56.17%	67.29%	73.57%	53.54%	67.44%	73.09%
Kulczynski	69.23%	81.54%	89.23%	45.98%	58.05%	62.64%	33.83%	45.68%	52.36%	51.98%	62.89%	69.82%	51.57%	64.57%	69.96%
Ochiai	67.69%	83.08%	89.23%	46.55%	59.77%	63.22%	39.74%	51.00%	57.06%	54.07%	64.76%	72.03%	52.47%	66.37%	72.47%
Phi	67.69%	80.00%	90.77%	46.55%	59.77%	63.22%	39.72%	50.88%	56.97%	54.74%	65.31%	72.03%	52.29%	66.64%	72.02%
PSC	70.77%	84.62%	89.23%	47.13%	61.49%	65.52%	47.26%	59.83%	65.24%	59.58%	71.26%	76.10%	58.12%	71.39%	76.86%
Relative Matching	64.62%	78.46%	86.15%	41.95%	55.75%	61.49%	34.23%	43.11%	49.33%	39.10%	55.29%	61.89%	36.86%	54.35%	64.48%
Rogers and Tanimoto	24.62%	44.62%	55.38%	24.71%	32.18%	35.63%	12.14%	14.48%	15.55%	22.69%	23.68%	28.85%	26.28%	37.76%	40.45%
Russell and Rao	69.23%	84.62%	87.69%	47.70%	60.34%	65.52%	40.29%	52.72%	58.44%	45.59%	61.34%	67.95%	45.11%	63.68%	71.39%
Simple matching	24.62%	43.08%	55.38%	25.29%	32.18%	35.63%	12.05%	14.36%	15.46%	22.58%	23.57%	28.85%	26.10%	37.67%	40.45%
Sokal and Sneath	24.62%	43.08%	53.85%	24.71%	29.31%	35.63%	12.00%	14.34%	15.29%	22.25%	23.24%	28.30%	25.83%	37.58%	40.27%
Sokal and Sneath 2	67.69%	81.54%	89.23%	47.70%	59.77%	63.79%	45.56%	56.32%	61.55%	58.04%	69.05%	74.67%	54.98%	69.24%	74.53%
Sokal and Sneath 4	67.69%	81.54%	90.77%	45.40%	58.05%	62.64%	33.78%	45.52%	52.29%	51.98%	63.66%	70.37%	51.57%	64.66%	69.87%
Sokal binary distance	27.69%	46.15%	55.38%	25.29%	34.48%	40.23%	13.86%	16.46%	17.63%	25.33%	26.87%	31.94%	28.07%	40.81%	43.05%
Sorenson	64.62%	80.00%	87.69%	45.98%	58.62%	63.22%	40.86%	52.17%	57.71%	54.63%	64.87%	71.37%	50.94%	65.65%	71.30%
Yule	53.85%	73.85%	80.00%	33.91%	45.40%	50.00%	13.14%	17.44%	22.30%	30.73%	37.78%	44.60%	22.96%	28.43%	31.39%
PTMC	75.83%	80.05%	88.58%	54.19%	59.82%	68.97%	54.68%	61.70%	66.37%	66.50%	71.97%	76.36%	68.07%	72.83%	82.54%

Coefficient	jsXe Top1	jsXe Top2	jsXe Top3	JSPWiki Top1	JSPWiki Top2	JSPWiki Top3	JTOpen Top1	JTOpen Top2	JTOpen Top3	Jung Top1	Jung Top2	Jung Top3	Log4j Top1	Log4j Top2	Log4j Top3
Baroni-Urbani and Buser	52.21%	68.14%	72.57%	28.47%	42.71%	55.93%	17.79%	31.00%	36.07%	47.42%	54.84%	65.16%	32.20%	43.64%	51.27%
Dot-product	58.41%	77.88%	83.19%	46.78%	61.36%	69.49%	31.57%	43.50%	52.64%	56.77%	69.35%	74.52%	44.07%	59.75%	66.95%
Hamann	42.48%	46.02%	50.44%	23.39%	29.83%	38.31%	1.21%	17.57%	28.07%	38.71%	45.16%	48.39%	22.46%	27.54%	30.51%
Jaccard	60.18%	77.88%	84.07%	50.51%	62.71%	69.83%	33.07%	48.71%	55.79%	58.06%	70.32%	75.48%	48.73%	61.44%	68.22%
Kulczynski	57.52%	76.99%	79.65%	47.80%	65.42%	72.88%	27.79%	39.86%	49.36%	58.06%	70.65%	75.16%	42.80%	61.02%	67.80%
Ochiai	60.18%	76.99%	81.42%	49.15%	63.05%	72.20%	30.36%	41.86%	49.86%	57.74%	68.71%	75.48%	44.07%	59.32%	68.64%
Phi	58.41%	76.99%	80.53%	48.81%	63.39%	72.20%	30.71%	42.64%	51.14%	57.74%	69.35%	75.48%	44.07%	60.17%	68.64%
PSC	64.60%	79.65%	83.19%	54.92%	67.12%	74.92%	39.21%	53.00%	66.79%	62.90%	73.23%	78.71%	56.78%	70.34%	76.27%
Relative Matching	56.64%	76.99%	80.53%	29.15%	40.34%	49.83%	13.14%	30.00%	42.36%	48.06%	62.58%	69.68%	33.47%	46.19%	58.90%
Rogers and Tanimoto	42.48%	46.02%	50.44%	23.39%	29.83%	38.64%	1.21%	17.64%	28.50%	38.71%	45.16%	48.39%	22.88%	27.97%	31.78%
Russell and Rao	63.72%	81.42%	92.92%	37.29%	53.22%	61.69%	22.14%	41.71%	48.71%	52.90%	67.74%	72.90%	44.49%	54.24%	67.37%
Simple matching	42.48%	46.02%	50.44%	23.05%	29.49%	38.31%	1.21%	17.57%	28.07%	38.71%	44.84%	48.39%	22.88%	27.54%	31.36%
Sokal and Sneath	41.59%	46.02%	50.44%	22.37%	28.81%	37.63%	1.21%	17.57%	27.29%	38.39%	44.52%	48.39%	22.03%	26.69%	30.08%
Sokal and Sneath 2	60.18%	78.76%	84.07%	52.54%	63.73%	70.51%	34.93%	51.07%	59.29%	60.00%	70.65%	76.77%	51.69%	63.98%	69.07%
Sokal and Sneath 4	57.52%	76.11%	78.76%	49.15%	66.10%	72.54%	28.07%	40.64%	49.93%	57.74%	70.32%	75.81%	42.37%	61.02%	67.37%
Sokal binary distance	43.36%	48.67%	50.44%	25.42%	31.86%	40.68%	5.43%	19.07%	45.79%	39.68%	46.45%	51.29%	24.15%	33.05%	36.44%
Sorenson	58.41%	77.88%	83.19%	46.78%	61.36%	69.49%	31.57%	43.50%	52.64%	56.77%	69.35%	74.52%	44.07%	59.75%	66.95%
Yule	43.36%	64.60%	69.91%	17.29%	23.39%	27.80%	11.29%	21.14%	33.57%	38.39%	49.35%	57.74%	16.10%	24.58%	30.08%
PTMC	69.95%	73.83%	78.65%	61.80%	68.49%	72.68%	55.46%	59.97%	72.21%	68.19%	75.13%	83.62%	61.14%	65.32%	72.61%

MOVE METHOD

Coefficient	JPF Top1	JPF Top2	JPF Top3	JRat Top1	JRat Top2	JRat Top3	JRE Top1	JRE Top2	JRE Top3	JREFactory Top1	JREFactory Top2	JREFactory Top3	JRuby Top1	JRuby Top2	JRuby Top3
Baroni-Urbani and Buser	20.00%	29.17%	35.83%	36.05%	41.28%	47.67%	20.76%	27.44%	30.49%	18.72%	22.35%	25.68%	9.18%	12.47%	14.39%
Dot-product	34.58%	49.17%	60.00%	40.70%	55.23%	62.79%	35.99%	44.25%	49.42%	27.46%	34.85%	40.02%	21.85%	28.56%	32.49%
Hamann	22.08%	30.00%	32.50%	28.49%	31.98%	38.95%	25.42%	29.69%	31.17%	20.32%	24.63%	28.08%	12.42%	15.76%	18.41%
Jaccard	37.08%	53.75%	62.92%	44.19%	56.98%	67.44%	39.59%	48.78%	53.95%	31.22%	38.73%	44.70%	25.19%	32.33%	36.74%
Kulczynski	31.25%	46.67%	56.67%	35.47%	44.77%	58.14%	28.37%	36.09%	41.23%	24.20%	29.99%	34.73%	14.82%	20.31%	26.09%
Ochiai	34.58%	47.50%	59.17%	38.95%	54.65%	61.63%	34.19%	41.45%	46.53%	26.54%	32.82%	38.49%	20.47%	26.65%	31.12%
Phi	33.75%	47.50%	57.92%	37.79%	52.91%	59.88%	34.16%	41.29%	46.43%	26.54%	32.76%	38.30%	20.48%	26.61%	31.10%
PSC	42.08%	59.58%	68.75%	44.77%	62.21%	71.51%	40.07%	49.45%	55.98%	31.90%	40.15%	46.92%	25.20%	32.49%	37.80%
Relative Matching	27.92%	40.42%	50.42%	37.21%	49.42%	54.07%	30.43%	40.58%	44.83%	23.03%	28.63%	32.39%	20.10%	25.32%	28.46%
Rogers and Tanimoto	22.08%	30.83%	33.33%	28.49%	32.56%	40.12%	25.61%	30.04%	31.59%	20.44%	24.94%	28.51%	12.62%	15.87%	18.58%
Russell and Rao	37.92%	51.67%	62.50%	45.93%	58.14%	65.12%	38.53%	48.04%	53.66%	29.25%	37.68%	42.36%	24.93%	31.77%	35.64%
Simple matching	21.67%	30.42%	32.92%	28.49%	31.98%	40.12%	25.35%	29.66%	31.23%	20.32%	24.69%	28.33%	12.50%	15.72%	18.45%
Sokal and Sneath	21.25%	30.00%	32.50%	28.49%	30.81%	37.79%	24.26%	28.60%	30.04%	19.83%	23.52%	27.09%	12.13%	15.40%	18.16%
Sokal and Sneath 2	42.92%	57.50%	66.25%	44.19%	61.05%	69.77%	43.89%	53.15%	58.64%	33.68%	42.92%	48.03%	28.73%	36.37%	40.92%
Sokal and Sneath 4	30.42%	46.67%	55.42%	34.88%	43.60%	56.98%	27.83%	35.54%	40.78%	24.26%	30.11%	34.61%	14.69%	20.27%	26.09%
Sokal binary distance	22.92%	31.25%	33.75%	31.40%	37.79%	44.19%	29.50%	34.06%	36.66%	22.48%	28.45%	31.65%	15.62%	20.73%	23.48%
Sorenson	34.58%	49.17%	60.00%	40.70%	55.23%	62.79%	35.99%	44.25%	49.42%	27.46%	34.85%	40.02%	21.85%	28.56%	32.49%
Yule	15.42%	24.17%	28.75%	22.09%	29.07%	38.95%	9.45%	19.54%	22.33%	8.31%	11.58%	15.33%	3.11%	5.05%	7.35%
PTMM	62.92%	69.71%	75.35%	57.78%	68.26%	71.36%	55.16%	62.85%	67.53%	49.75%	56.26%	58.57%	55.17%	65.07%	67.32%

Coefficient	jsXe Top1	jsXe Top2	jsXe Top3	JSPWiki Top1	JSPWiki Top2	JSPWiki Top3	JTOpen Top1	JTOpen Top2	JTOpen Top3	Jung Top1	Jung Top2	Jung Top3	Log4j Top1	Log4j Top2	Log4j Top3
Baroni-Urbani and Buser	14.18%	24.47%	35.11%	10.61%	15.13%	16.69%	12.49%	17.64%	21.21%	19.05%	27.69%	33.33%	12.11%	20.00%	26.76%
Dot-product	32.62%	47.87%	56.38%	23.20%	29.84%	34.37%	25.96%	33.99%	40.17%	25.94%	38.47%	45.24%	26.76%	42.82%	49.58%
Hamann	20.21%	28.37%	30.14%	11.17%	14.99%	19.09%	20.09%	26.74%	30.90%	20.55%	35.59%	41.10%	11.83%	22.25%	29.58%
Jaccard	37.94%	52.13%	61.35%	26.73%	34.65%	39.60%	30.63%	38.19%	44.47%	30.45%	42.23%	50.38%	31.27%	47.04%	53.24%
Kulczynski	30.14%	49.65%	59.22%	17.54%	26.17%	31.82%	20.61%	28.23%	34.70%	23.93%	34.71%	42.11%	20.28%	38.03%	47.89%
Ochiai	32.62%	48.58%	58.87%	22.91%	29.56%	33.80%	24.41%	33.35%	38.77%	25.94%	36.59%	45.61%	24.79%	41.69%	50.70%
Phi	32.27%	48.58%	58.51%	22.91%	29.56%	33.66%	24.39%	33.35%	38.54%	26.19%	36.72%	46.12%	24.51%	41.69%	50.70%
PSC	41.84%	59.93%	70.92%	28.43%	37.91%	44.84%	31.12%	39.51%	46.04%	32.83%	46.37%	53.63%	33.80%	51.55%	59.44%
Relative Matching	23.05%	32.27%	36.88%	15.28%	20.51%	23.62%	21.73%	28.40%	32.53%	12.53%	18.92%	21.05%	19.44%	32.96%	37.46%
Rogers and Tanimoto	20.21%	28.37%	30.50%	11.32%	15.13%	19.38%	20.22%	26.87%	31.06%	20.68%	35.71%	41.48%	12.11%	22.54%	30.70%
Russell and Rao	33.33%	49.65%	56.03%	22.91%	31.97%	35.79%	28.09%	35.42%	40.81%	17.79%	24.44%	28.32%	30.70%	47.04%	52.39%
Simple matching	19.86%	28.01%	30.14%	11.32%	14.99%	18.95%	20.20%	26.89%	30.86%	20.55%	35.46%	41.10%	11.83%	22.54%	30.14%
Sokal and Sneath	19.50%	26.60%	29.08%	10.61%	14.43%	18.39%	19.95%	26.48%	30.46%	20.05%	34.59%	39.47%	11.27%	21.41%	27.89%
Sokal and Sneath 2	43.97%	56.74%	65.60%	29.84%	37.62%	43.42%	33.58%	42.40%	48.45%	34.09%	46.37%	53.13%	34.08%	48.73%	56.06%
Sokal and Sneath 4	30.14%	49.65%	59.22%	17.54%	26.17%	31.82%	20.59%	28.23%	34.66%	23.93%	34.84%	42.48%	20.28%	38.03%	47.89%
Sokal binary distance	23.05%	31.91%	33.69%	14.43%	19.52%	24.05%	24.53%	31.60%	36.47%	24.31%	38.97%	45.61%	13.52%	25.07%	32.96%
Sorenson	32.62%	47.87%	56.38%	23.20%	29.84%	34.37%	25.96%	33.99%	40.17%	25.94%	38.47%	45.24%	26.76%	42.82%	49.58%
Yule	9.22%	18.44%	24.82%	6.93%	10.18%	12.45%	3.86%	8.92%	12.04%	6.52%	20.05%	28.45%	8.45%	15.77%	22.25%
PTMM	61.39%	71.94%	74.72%	41.86%	48.19%	52.96%	51.08%	58.24%	63.04%	42.27%	48.75%	52.91%	48.51%	58.97%	64.66%

EXTRACT METHOD

Coefficient	JPF Hits	JRat Hits	JRE Hits	JREFactory Hits	JRuby Hits	jsXe Hits	JSPWiki Hits	JTOpen Hits	Jung Hits	Log4j Hits
Baroni-Urbani and Buser	53.58%	43.24%	61.14%	63.32%	66.68%	63.20%	57.35%	61.82%	60.09%	48.83%
Dot-product	63.74%	62.16%	67.29%	67.41%	78.99%	69.98%	66.31%	74.46%	58.70%	67.01%
Hamann	1.15%	5.41%	12.36%	15.30%	15.62%	6.54%	4.54%	17.45%	7.89%	4.68%
Jaccard	67.90%	64.86%	70.34%	70.78%	80.98%	71.36%	68.34%	80.52%	61.81%	69.09%
Kulczynski	79.91%	67.57%	75.82%	78.36%	85.50%	79.18%	75.39%	85.05%	69.61%	78.18%
Ochiai	69.75%	70.27%	71.14%	72.23%	82.39%	72.64%	69.89%	78.42%	63.57%	69.87%
Phi	67.90%	67.57%	71.11%	72.03%	82.35%	72.40%	69.89%	78.44%	63.11%	69.87%
PSC	76.91%	72.97%	74.85%	74.27%	83.83%	77.72%	72.16%	84.32%	61.02%	69.61%
Relative Matching	80.37%	75.68%	79.09%	80.01%	90.10%	81.36%	82.92%	82.51%	70.77%	82.08%
Rogers and Tanimoto	1.62%	5.41%	12.40%	15.44%	15.64%	7.75%	4.54%	17.55%	8.35%	4.68%
Russell and Rao	88.68%	86.49%	85.15%	85.29%	93.18%	86.92%	88.17%	87.26%	73.09%	85.19%
Simple matching	1.15%	5.41%	12.37%	15.37%	15.59%	6.54%	4.54%	17.49%	7.89%	4.42%
Sokal and Sneath	1.15%	5.41%	12.31%	15.37%	15.59%	6.54%	3.70%	17.29%	7.89%	4.16%
Sokal and Sneath 2	77.14%	72.97%	73.28%	73.55%	83.17%	76.76%	72.16%	85.25%	64.04%	70.39%
Sokal and Sneath 4	76.44%	62.16%	75.78%	77.77%	85.36%	77.72%	75.15%	84.95%	69.61%	77.14%
Sokal binary distance	6.70%	2.70%	20.08%	19.85%	22.53%	11.86%	6.93%	25.23%	15.31%	21.56%
Sorenson	63.74%	62.16%	67.29%	67.41%	78.99%	69.98%	66.31%	74.46%	58.70%	67.01%
Yule	65.59%	51.35%	63.10%	70.98%	66.38%	66.59%	61.05%	63.82%	61.72%	57.66%
PTEM	89.53%	86.52%	85.27%	86.51%	93.34%	86.94%	88.30%	87.63%	73.20%	85.59%

To be continued in the next page.

MOVE CLASS																														
	Lucene			Marauroa			Maven			MegaMek			MVNForum			MyFaces_Core			NakedObjects			NekoHTML			NetBeans			OpenJMS		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	19.86%	25.61%	29.73%	65.67%	70.15%	71.64%	76.92%	77.69%	78.89%	33.76%	73.80%	76.60%	38.55%	48.30%	53.29%	23.76%	34.23%	42.82%	20.20%	29.88%	36.49%	60.87%	65.22%	86.96%	45.07%	63.94%	75.64%	32.36%	45.19%	56.27%
Dot-product	28.80%	38.97%	44.19%	68.66%	76.87%	81.34%	79.03%	81.00%	82.76%	38.64%	82.49%	86.83%	47.62%	59.18%	66.21%	39.27%	54.49%	62.68%	35.96%	47.82%	54.35%	52.17%	65.22%	82.61%	52.84%	71.57%	81.48%	48.40%	71.43%	80.47%
Hamann	10.69%	15.74%	19.98%	56.72%	70.90%	76.87%	76.92%	77.55%	78.18%	24.00%	31.28%	47.99%	25.85%	31.07%	33.11%	17.92%	23.53%	27.07%	16.29%	20.65%	21.85%	21.74%	34.78%	65.22%	25.00%	40.83%	52.29%	18.08%	22.45%	30.61%
Jaccard	30.60%	41.52%	46.86%	70.90%	78.36%	82.84%	79.52%	81.84%	83.81%	43.05%	84.16%	88.17%	48.98%	61.90%	68.71%	42.01%	57.93%	64.97%	38.96%	51.50%	57.58%	52.17%	65.22%	86.96%	54.55%	72.72%	82.21%	51.60%	71.72%	81.34%
Kulczynski	28.63%	39.84%	44.48%	68.66%	77.61%	84.33%	79.31%	81.35%	83.39%	38.44%	76.47%	85.43%	43.54%	58.28%	65.08%	39.55%	53.23%	62.05%	29.73%	44.89%	52.33%	43.48%	65.22%	86.96%	52.94%	72.13%	82.22%	51.90%	70.85%	79.88%
Ochiai	29.67%	40.19%	45.18%	69.40%	79.10%	82.84%	79.45%	81.21%	82.97%	38.97%	82.95%	85.63%	46.26%	59.18%	65.99%	39.84%	54.32%	63.54%	36.26%	48.72%	55.41%	47.83%	65.22%	86.96%	53.55%	72.21%	81.95%	51.90%	71.72%	81.34%
Phi	29.62%	40.48%	45.35%	71.64%	79.85%	84.33%	79.45%	81.21%	82.83%	39.24%	82.95%	85.63%	46.49%	58.96%	66.21%	39.78%	54.26%	63.54%	36.19%	48.65%	55.48%	43.48%	65.22%	82.61%	53.03%	71.83%	81.93%	51.60%	71.72%	81.05%
PSC	35.37%	46.75%	52.56%	72.39%	82.84%	85.82%	81.70%	83.88%	86.00%	59.56%	89.57%	92.45%	52.15%	65.76%	72.11%	47.45%	62.05%	69.15%	42.49%	56.38%	62.24%	43.48%	65.22%	82.61%	58.30%	75.56%	84.64%	55.98%	74.93%	83.09%
Relative Matching	18.23%	25.03%	30.84%	52.24%	67.91%	70.15%	76.99%	78.61%	80.37%	35.70%	42.78%	60.83%	34.47%	47.85%	54.20%	32.74%	44.13%	51.80%	18.62%	31.61%	41.37%	47.83%	65.22%	82.61%	46.45%	66.54%	77.41%	42.57%	64.14%	74.34%
Rogers and Tanimoto	10.69%	15.91%	19.98%	56.72%	70.90%	76.87%	76.92%	77.55%	78.18%	24.33%	31.42%	47.93%	26.53%	31.07%	33.11%	18.20%	23.64%	27.25%	16.52%	20.80%	21.92%	21.74%	34.78%	65.22%	25.56%	41.38%	52.83%	18.08%	22.74%	31.49%
Russell and Rao	20.96%	29.27%	35.25%	54.48%	75.37%	77.61%	78.68%	81.07%	83.67%	37.90%	45.19%	77.27%	37.64%	51.47%	60.09%	38.52%	51.80%	59.70%	23.87%	38.81%	48.27%	47.83%	65.22%	73.91%	51.63%	71.14%	81.03%	49.56%	68.51%	81.05%
Simple matching	10.69%	15.51%	19.86%	56.72%	70.90%	76.87%	76.92%	77.55%	78.18%	24.33%	31.35%	47.99%	26.08%	30.84%	33.11%	17.92%	23.30%	26.96%	16.44%	20.72%	21.85%	21.74%	34.78%	65.22%	25.19%	40.92%	52.34%	18.08%	22.45%	30.61%
Sokal and Sneath	10.22%	14.81%	19.74%	55.97%	70.90%	76.87%	76.85%	77.48%	78.11%	23.93%	31.15%	47.93%	25.85%	30.84%	32.88%	17.57%	23.13%	26.90%	15.77%	20.20%	21.17%	17.39%	34.78%	65.22%	24.78%	40.48%	52.00%	17.78%	22.45%	30.03%
Sokal and Sneath 2	32.17%	42.92%	48.90%	71.64%	78.36%	82.84%	80.30%	82.27%	84.38%	45.92%	85.90%	90.24%	50.57%	64.17%	69.16%	45.33%	59.76%	65.83%	41.59%	52.85%	58.78%	52.17%	65.22%	86.96%	55.55%	73.23%	82.75%	53.35%	73.18%	81.63%
Sokal and Sneath 4	28.69%	39.95%	44.54%	68.66%	77.61%	85.82%	79.31%	81.35%	83.32%	38.50%	76.74%	85.09%	43.54%	59.64%	64.85%	39.55%	52.95%	61.76%	29.65%	43.02%	52.40%	43.48%	65.22%	82.61%	52.92%	71.57%	82.00%	51.90%	70.55%	79.59%
Sokal binary distance	11.67%	18.35%	21.78%	59.70%	71.64%	78.36%	77.27%	77.90%	78.61%	25.33%	44.65%	52.67%	28.80%	34.47%	36.51%	20.78%	26.56%	30.62%	18.39%	22.67%	24.92%	26.09%	39.13%	69.57%	27.41%	43.25%	54.73%	20.12%	26.82%	34.40%
Sorenson	28.80%	38.97%	44.19%	68.66%	76.87%	81.34%	79.03%	81.00%	82.76%	38.64%	82.49%	86.83%	47.62%	59.18%	66.21%	39.27%	54.49%	62.68%	35.96%	47.82%	54.35%	52.17%	65.22%	82.61%	52.84%	71.57%	81.48%	48.40%	71.43%	80.47%
Yule	16.20%	20.67%	24.22%	63.43%	70.90%	73.88%	76.71%	77.62%	78.11%	26.14%	30.75%	52.67%	27.89%	36.96%	42.18%	16.03%	24.67%	29.71%	10.14%	16.22%	26.58%	56.52%	65.22%	86.96%	38.82%	58.38%	71.28%	27.11%	36.15%	43.73%
PTMC	40.10%	44.70%	53.56%	74.94%	80.14%	92.17%	86.73%	83.82%	89.28%	75.81%	89.56%	96.27%	53.69%	63.52%	74.83%	58.06%	61.82%	70.52%	49.65%	56.13%	63.37%	44.76%	61.07%	75.26%	60.06%	72.81%	84.17%	60.24%	73.50%	84.34%

MOVE METHOD																														
	Lucene			Marauroa			Maven			MegaMek			MVNForum			MyFaces_Core			NakedObjects			NekoHTML			NetBeans			OpenJMS		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	10.05%	13.59%	15.56%	19.83%	30.03%	33.43%	26.81%	31.12%	33.84%	18.82%	21.74%	22.70%	15.91%	23.57%	29.16%	22.97%	25.82%	28.21%	12.66%	16.61%	20.45%	20.10%	39.69%	52.58%	24.62%	38.38%	48.15%	16.83%	25.56%	33.12%
Dot-product	20.08%	26.68%	30.92%	29.46%	45.33%	50.42%	43.88%	51.20%	58.23%	27.02%	31.29%	34.59%	26.10%	37.21%	43.96%	33.15%	38.63%	42.51%	24.25%	35.20%	41.04%	42.78%	64.95%	75.77%	38.21%	53.59%	62.70%	34.61%	46.75%	56.66%
Hamann	13.40%	17.38%	20.19%	26.91%	36.26%	42.21%	29.72%	40.76%	45.48%	20.25%	21.13%	22.54%	21.56%	29.42%	36.04%	26.21%	29.72%	32.12%	14.79%	20.19%	23.99%	26.29%	47.42%	56.19%	19.09%	26.80%	31.61%	21.51%	32.48%	39.72%
Jaccard	23.34%	30.78%	36.25%	34.56%	49.58%	56.09%	46.39%	54.62%	61.85%	29.42%	33.24%	36.87%	29.81%	40.39%	47.60%	35.89%	41.61%	46.96%	29.35%	39.66%	45.62%	52.06%	70.10%	77.84%	41.58%	56.55%	65.22%	38.34%	52.82%	60.92%
Kulczynski	13.90%	18.53%	21.96%	29.18%	42.78%	50.42%	34.84%	44.38%	50.80%	24.30%	28.48%	31.73%	22.73%	33.83%	40.84%	27.40%	32.62%	35.98%	16.16%	23.91%	30.47%	34.02%	65.98%	74.23%	34.35%	50.71%	60.76%	23.75%	39.30%	50.69%
Ochiai	18.25%	24.15%	28.17%	30.59%	43.91%	52.12%	42.07%	50.20%	56.22%	26.94%	30.16%	33.74%	24.55%	36.69%	43.44%	31.34%	36.61%	40.32%	20.19%	29.98%	35.94%	39.18%	65.98%	73.71%	37.10%	52.78%	62.15%	31.95%	44.52%	55.27%
Phi	18.20%	24.15%	28.08%	30.59%	43.34%	52.12%	42.07%	50.20%	56.12%	26.91%	30.16%	33.79%	24.48%	36.56%	43.31%	31.34%	36.54%	40.23%	20.19%	29.91%	35.90%	39.18%	65.46%	74.23%	37.13%	52.79%	62.11%	30.88%	44.41%	55.06%
PSC	23.98%	31.65%	37.24%	33.99%	50.42%	61.19%	45.98%	55.72%	62.55%	30.05%	36.16%	40.07%	30.39%	42.08%	49.29%	36.71%	42.58%	47.84%	27.75%	38.92%	44.62%	54.12%	71.13%	79.90%	44.37%	59.93%	68.58%	38.55%	54.85%	62.51%
Relative Matching	15.70%	20.81%	24.32%	23.23%	32.29%	38.24%	29.42%	43.17%	49.20%	19.29%	25.81%	27.77%	16.36%	25.65%	30.65%	28.97%	34.62%	38.07%	21.60%	30.91%	35.83%	31.44%	54.64%	70.10%	30.53%	45.03%	54.38%	29.39%	43.56%	50.37%
Rogers and Tanimoto	13.73%	17.94%	20.64%	26.91%	36.26%	42.21%	30.22%	41.16%	45.88%	20.36%	21.27%	22.76%	21.88%	29.81%	36.17%	26.57%	30.05%	32.31%	15.08%	20.48%	24.32%	25.77%	50.00%	57.22%	19.36%	27.10%	31.91%	21.83%	32.59%	39.83%
Russell and Rao	22.91%	29.91%	34.60%	28.33%	40.79%	49.86%	34.14%	49.30%	57.13%	23.56%	30.38%	33.19%	22.14%	30.52%	37.73%	33.89%	40.81%	45.08%	29.09%	41.82%	47.08%	51.55%	61.86%	77.84%	38.67%	53.58%	62.42%	38.45%	55.17%	62.51%
Simple matching	13.37%	17.52%	20.08%	26.35%	36.26%	41.64%	30.22%	40.96%	45.88%	20.25%	21.19%	22.62%	21.75%	29.35%	35.84%	26.34%	29.85%	32.10%	15.01%	20.26%	24.13%	26.29%	48.45%	56.19%	19.09%	26.80%	31.59%	21.62%	32.37%	40.04%
Sokal and Sneath	12.66%	16.60%	19.18%	26.06%	35.98%	41.64%	29.02%	39.86%	44.98%	20.03%	21.00%	22.21%	21.30%	28.70%	35.65%	25.78%	29.27%	31.45%	13.85%	19.52%	23.09%	25.77%	46.39%	56.19%	18.52%	26.16%	30.92%	20.98%	31.63%	39.62%
Sokal and Sneath 2	26.79%	34.40%	40.44%	36.54%	51.84%	60.91%	48.49%	56.83%	62.75%	30.93%	35.61%	39.46%	32.34%	44.94%	50.58%	40.10%	45.92%	51.06%	32.44%	43.31%	49.50%	56.70%	70.62%	79.38%	44.33%	58.86%	67.00%	41.96%	55.38%	64.22%
Sokal and Sneath 4	13.87%	18.45%	21.71%	29.18%	42.78%	50.42%	34.84%	44.38%	50.70%	24.35%	28.48%	31.73%	22.73%	33.83%	40.84%	27.37%	32.61%	35.92%	16.16%	23.91%	30.39%	34.02%	65.98%	74.23%	34.34%	50.68%	60.74%	23.86%	38.98%	50.37%
Sokal binary distance	16.29%	20.33%	23.42%	31.16%	38.53%	43.34%	34.14%	43.07%	46.39%	23.67%	25.43%	27.02%	25.84%	34.29%	39.68%	28.41%	32.46%	34.92%	17.58%	22.50%	26.74%	35.57%	54.64%	61.86%	22.71%	30.91%	35.79%	26.30%	36.85%	42.60%
Sorenson	20.08%	26.68%	30.92%	29.46%	45.33%	50.42%	43.88%	51.20%	58.23%	27.02%	31.29%	34.59%	26.10%	37.21%	43.96%	33.15%	38.63%	42.51%	24.25%	35.20%	41.04%	42.78%	64.95%	75.77%	38.21%	53.59%	62.70%	34.61%	46.75%	56.66%
Yule	4.86%	6.82%	8.03%	11.05%	18.98%	23.51%	13.55%	22.19%	26.51%	3.25%	19.79%	21.49%	7.99%	15.06%	19.55%	7.49%	11.45%	14.24%	7.71%	11.36%	15.61%	14.95%	29.90%	37.63%	19.73%	32.92%	42.51%	7.14%	13.31%	21.62%
PTMM	46.34%	49.48%	54.21%	44.42%	57.97%	64.32%	51.52%	63.43%	67.40%	47.68%	53.91%	54.92%	37.43%	46.58%	51.35%	58.29%	63.99%	66.75%	47.91%	59.60%	62.41%	81.23%	90.32%	89.39%	58.13%	65.59%	71.50%	57.22%	67.79%	72.97%

EXTRACT METHOD										
	Lucene	Marauroa	Maven	MegaMek	MVNForum	MyFaces_Core	NakedObjects	NekoHTML	NetBeans	OpenJMS
Coefficient	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits
Baroni-Urbani and Buser	79.08%	65.55%	67.42%	70.94%	84.21%	71.19%	55.80%	62.88%	66.21%	77.71%
Dot-product	83.16%	68.42%	73.41%	77.50%	95.26%	80.26%	63.08%	75.55%	72.09%	79.82%
Hamann	14.90%	15.79%	11.43%	13.26%	58.60%	9.37%	8.02%	13.97%	7.03%	12.35%
Jaccard	84.21%	74.16%	74.32%	79.41%	95.46%	81.94%	65.72%	82.97%	74.41%	79.52%
Kulczynski	89.94%	77.51%	80.13%	83.68%	96.27%	85.20%	74.26%	85.15%	79.81%	86.14%
Ochiai	86.41%	73.21%	75.41%	80.42%	96.00%	82.41%	67.93%	82.10%	75.60%	84.04%
Phi	86.41%	72.73%	74.77%	80.16%	95.87%	82.32%	67.93%	81.66%	74.81%	84.04%
PSC	87.46%	77.03%	78.49%	82.47%	96.14%	84.90%	71.84%	88.21%	77.91%	82.53%
Relative Matching	90.64%	84.69%	88.11%	87.05%	97.15%	88.64%	80.59%	82.97%	83.69%	90.66%
Rogers and Tanimoto	14.84%	15.79%	10.89%	13.40%	58.67%	9.44%	7.91%	16.59%	7.39%	12.35%
Russell and Rao	93.16%	91.39%	92.38%	90.38%	98.04%	92.05%	90.19%	91.27%	88.39%	96.69%
Simple matching	14.84%	15.79%	11.43%	13.23%	58.60%	9.37%	8.33%	13.97%	7.03%	11.45%
Sokal and Sneath	14.58%	15.79%	11.16%	13.15%	58.60%	9.35%	8.02%	11.79%	6.89%	10.54%
Sokal and Sneath 2	85.90%	75.12%	75.05%	81.88%	95.80%	83.77%	71.31%	86.46%	77.13%	81.33%
Sokal and Sneath 4	89.91%	77.03%	79.58%	83.54%	96.27%	84.99%	73.95%	84.72%	78.29%	86.14%
Sokal binary distance	23.46%	20.10%	12.89%	22.82%	63.62%	16.52%	11.08%	27.51%	12.72%	16.57%
Sorenson	83.16%	68.42%	73.41%	77.50%	95.26%	80.26%	63.08%	75.55%	72.09%	79.82%
Yule	81.47%	72.25%	68.87%	74.08%	81.98%	72.42%	62.66%	66.38%	70.71%	76.81%
PTEM	93.60%	92.84%	92.43%	90.38%	98.25%	92.39%	90.09%	91.39%	89.21%	97.31%

To be continued in the next page.

MOVE CLASS																														
	OSCache			PicoContainer			PMD			POI			Pooka			ProGuard			Quartz			QuickServer			Quilt			Roller		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	51.28%	74.36%	84.62%	32.74%	49.56%	69.03%	46.99%	53.90%	59.47%	20.11%	29.41%	36.88%	35.06%	47.62%	55.41%	29.86%	43.81%	53.83%	14.88%	24.79%	28.93%	30.00%	42.50%	51.67%	50.00%	68.42%	89.47%	56.94%	69.44%	75.00%
Dot-product	61.54%	76.92%	79.49%	41.59%	70.80%	83.19%	52.78%	63.25%	67.93%	38.31%	48.95%	58.91%	47.19%	64.94%	76.19%	48.53%	58.55%	67.39%	28.10%	38.02%	49.59%	34.17%	48.33%	58.33%	57.89%	78.95%	97.37%	66.67%	76.39%	84.72%
Hamann	28.21%	33.33%	56.41%	33.63%	41.59%	47.79%	17.15%	42.76%	46.77%	16.67%	19.64%	21.93%	11.69%	15.58%	16.45%	24.56%	34.18%	37.13%	17.36%	26.45%	35.54%	5.83%	34.17%	40.83%	50.00%	50.00%	52.63%	34.72%	45.83%	65.28%
Jaccard	61.54%	76.92%	79.49%	45.13%	75.22%	83.19%	53.90%	65.48%	69.71%	40.04%	52.39%	61.11%	49.78%	67.97%	78.79%	50.29%	60.71%	68.17%	33.88%	46.28%	52.07%	38.33%	48.33%	59.17%	60.53%	81.58%	97.37%	70.83%	79.17%	84.72%
Kulczynski	61.54%	76.92%	84.62%	44.25%	67.26%	80.53%	55.23%	63.03%	69.27%	33.91%	47.80%	57.76%	41.99%	59.31%	69.70%	44.01%	55.80%	65.03%	25.62%	40.50%	55.37%	35.00%	45.83%	50.83%	63.16%	68.42%	97.37%	65.28%	80.56%	81.94%
Ochiai	64.10%	74.36%	82.05%	43.36%	69.91%	83.19%	54.34%	63.03%	68.82%	37.55%	49.81%	59.39%	46.75%	64.94%	72.73%	48.72%	58.15%	66.80%	28.10%	37.19%	52.07%	35.83%	48.33%	56.67%	57.89%	81.58%	92.11%	66.67%	76.39%	83.33%
Phi	64.10%	74.36%	82.05%	43.36%	69.91%	82.30%	54.34%	63.03%	68.37%	37.45%	49.90%	59.48%	46.32%	61.90%	71.43%	48.53%	57.76%	67.78%	28.93%	37.19%	54.55%	37.50%	47.50%	56.67%	60.53%	73.68%	94.74%	65.28%	73.61%	80.56%
PSC	66.67%	74.36%	87.18%	53.10%	77.88%	88.50%	59.24%	69.71%	76.39%	46.93%	59.48%	66.38%	55.84%	73.16%	82.68%	52.85%	63.26%	73.87%	41.32%	60.33%	68.60%	40.83%	50.83%	60.83%	65.79%	84.21%	94.74%	73.61%	80.56%	87.50%
Relative Matching	53.85%	66.67%	84.62%	29.20%	44.25%	74.34%	47.22%	56.79%	62.58%	28.35%	41.09%	57.28%	47.62%	64.50%	71.43%	31.04%	41.26%	55.01%	13.22%	28.93%	41.32%	30.00%	44.17%	50.00%	55.26%	81.58%	100.00%	52.78%	69.44%	75.00%
Rogers and Tanimoto	28.21%	35.90%	56.41%	34.51%	41.59%	47.79%	18.26%	43.65%	47.22%	16.67%	19.73%	22.03%	11.69%	15.58%	16.45%	24.75%	34.18%	37.13%	17.36%	26.45%	38.84%	5.83%	33.33%	40.00%	50.00%	50.00%	52.63%	37.50%	45.83%	66.67%
Russell and Rao	56.41%	66.67%	76.92%	33.63%	55.75%	75.22%	51.22%	62.14%	69.71%	35.25%	47.80%	63.89%	54.55%	74.03%	82.25%	32.61%	44.79%	57.56%	19.01%	42.98%	53.72%	33.33%	45.00%	51.67%	68.42%	97.37%	100.00%	55.56%	75.00%	84.72%
Simple matching	28.21%	33.33%	56.41%	33.63%	41.59%	47.79%	17.15%	42.54%	46.55%	16.67%	19.64%	21.93%	11.69%	15.58%	16.45%	24.56%	34.18%	37.13%	17.36%	26.45%	35.54%	5.83%	34.17%	40.83%	50.00%	50.00%	52.63%	33.33%	45.83%	66.67%
Sokal and Sneath	28.21%	33.33%	56.41%	33.63%	41.59%	47.79%	16.70%	40.76%	44.54%	16.28%	19.25%	21.55%	10.39%	15.58%	16.45%	24.36%	33.79%	36.94%	17.36%	26.45%	34.71%	5.83%	34.17%	40.83%	50.00%	50.00%	52.63%	31.94%	41.67%	63.89%
Sokal and Sneath 2	64.10%	76.92%	79.49%	52.21%	76.11%	84.96%	56.12%	66.37%	70.60%	42.53%	54.79%	62.84%	51.95%	69.70%	80.95%	51.67%	62.87%	71.51%	33.88%	52.07%	56.20%	38.33%	46.67%	57.50%	63.16%	81.58%	97.37%	70.83%	80.56%	84.72%
Sokal and Sneath 4	61.54%	76.92%	82.05%	43.36%	69.91%	80.53%	55.46%	63.03%	69.27%	33.91%	47.89%	57.76%	40.69%	58.01%	67.53%	46.95%	56.58%	66.40%	27.27%	38.84%	55.37%	34.17%	45.00%	51.67%	60.53%	68.42%	94.74%	65.28%	77.78%	80.56%
Sokal binary distance	25.64%	41.03%	58.97%	36.28%	45.13%	53.98%	39.87%	45.43%	47.88%	18.49%	21.55%	22.99%	12.12%	17.32%	17.75%	30.84%	40.47%	44.60%	21.49%	32.23%	47.11%	5.83%	34.17%	40.83%	50.00%	50.00%	55.26%	44.44%	50.00%	66.67%
Sorenson	61.54%	76.92%	79.49%	41.59%	70.80%	83.19%	52.78%	63.25%	67.93%	38.31%	48.95%	58.91%	47.19%	64.94%	76.19%	48.53%	58.55%	67.39%	28.10%	38.02%	49.59%	34.17%	48.33%	58.33%	57.89%	78.95%	97.37%	66.67%	76.39%	84.72%
Yule	48.72%	66.67%	89.74%	19.47%	36.28%	47.79%	29.84%	46.10%	52.12%	9.39%	20.21%	29.50%	27.27%	38.10%	42.86%	22.79%	32.22%	41.45%	14.05%	19.83%	24.79%	25.00%	40.00%	50.83%	47.37%	63.16%	76.32%	41.67%	55.56%	61.11%
PTMC	69.95%	74.60%	79.45%	62.84%	73.59%	86.13%	63.04%	69.87%	77.45%	53.82%	59.42%	74.75%	62.20%	72.71%	81.17%	57.38%	63.55%	75.68%	49.21%	61.59%	70.22%	48.27%	48.02%	57.04%	82.96%	92.34%	100.00%	79.97%	80.77%	85.32%

MOVE METHOD																														
	OSCache			PicoContainer			PMD			POI			Pooka			ProGuard			Quartz			QuickServer			Quilt			Roller		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	32.67%	44.55%	57.43%	16.91%	26.71%	29.97%	35.66%	42.47%	46.43%	21.85%	28.68%	35.37%	35.66%	25.73%	32.02%	14.06%	18.22%	22.22%	28.94%	39.84%	45.53%	20.18%	25.08%	29.36%	21.98%	38.46%	48.35%	36.23%	50.72%	59.42%
Dot-product	51.49%	65.35%	74.26%	27.89%	37.98%	45.10%	43.58%	50.55%	55.31%	38.61%	47.92%	53.88%	43.58%	43.92%	53.23%	28.82%	35.70%	40.86%	43.58%	67.48%	75.45%	33.64%	40.98%	49.85%	34.07%	50.55%	52.75%	44.93%	62.32%	68.12%
Hamann	42.57%	47.52%	53.47%	24.04%	29.67%	36.80%	34.39%	39.46%	43.26%	25.63%	33.04%	36.00%	34.39%	21.54%	25.48%	18.70%	23.92%	26.05%	36.26%	49.43%	54.96%	24.16%	31.50%	36.09%	16.48%	27.47%	39.56%	33.33%	50.72%	53.62%
Jaccard	56.44%	68.32%	79.21%	32.94%	42.43%	48.37%	46.12%	52.93%	58.16%	41.47%	51.65%	56.78%	46.12%	47.28%	55.41%	30.42%	38.63%	43.47%	49.92%	73.66%	79.19%	36.70%	46.79%	54.43%	35.16%	48.35%	54.95%	49.28%	62.32%	68.12%
Kulczynski	47.52%	60.40%	74.26%	27.30%	34.42%	41.54%	36.13%	45.01%	51.51%	31.83%	42.44%	48.11%	36.13%	38.73%	48.70%	24.08%	32.39%	36.92%	35.12%	60.33%	67.80%	34.25%	43.73%	49.54%	26.37%	41.76%	53.85%	39.13%	57.97%	62.32%
Ochiai	46.53%	60.40%	73.27%	26.41%	37.39%	43.32%	40.89%	48.97%	54.20%	36.00%	46.56%	51.94%	40.89%	44.26%	53.14%	27.38%	34.84%	39.37%	42.11%	64.23%	72.85%	32.72%	40.67%	49.54%	31.87%	45.05%	53.85%	43.48%	60.87%	66.67%
Phi	46.53%	60.40%	73.27%	26.71%	36.80%	43.03%	40.89%	48.97%	53.88%	36.00%	46.56%	51.94%	40.89%	43.84%	53.06%	27.33%	34.84%	39.48%	42.11%	64.39%	72.52%	33.03%	41.90%	49.85%	30.77%	45.05%	53.85%	43.48%	60.87%	66.67%
PSC	59.41%	75.25%	83.17%	34.12%	41.25%	49.55%	47.23%	54.99%	60.70%	41.81%	52.18%	57.85%	47.23%	49.71%	59.77%	30.63%	38.68%	42.25%	49.92%	74.80%	80.98%	37.00%	51.07%	59.63%	38.46%	47.25%	56.04%	50.72%	62.32%	71.01%
Relative Matching	40.59%	49.50%	60.40%	19.29%	28.49%	31.16%	40.10%	47.70%	51.19%	27.52%	38.76%	44.67%	40.10%	32.94%	41.32%	23.39%	33.03%	37.77%	38.54%	45.20%	62.76%	20.49%	29.05%	35.47%	34.07%	50.55%	58.24%	46.38%	59.42%	66.67%
Rogers and Tanimoto	42.57%	48.51%	55.45%	24.04%	29.38%	36.80%	34.55%	39.62%	43.58%	26.11%	33.58%	36.48%	34.55%	22.05%	25.57%	18.81%	24.03%	26.27%	36.59%	49.92%	55.12%	24.46%	31.50%	36.39%	16.48%	27.47%	41.76%	33.33%	50.72%	53.62%
Russell and Rao	49.50%	56.44%	70.30%	24.63%	34.72%	40.06%	45.48%	54.36%	58.16%	34.25%	47.04%	52.96%	45.48%	45.26%	53.73%	27.54%	37.51%	41.34%	42.93%	61.79%	75.45%	29.36%	38.53%	46.18%	40.66%	54.95%	61.54%	47.83%	57.97%	68.12%
Simple matching	41.58%	47.52%	53.47%	24.04%	29.67%	36.50%	34.23%	39.46%	43.26%	25.73%	33.14%	36.09%	34.23%	21.79%	25.40%	18.86%	23.92%	26.05%	36.42%	49.92%	55.12%	24.16%	30.89%	35.78%	16.48%	27.47%	40.66%	33.33%	50.72%	53.62%
Sokal and Sneath	41.58%	48.51%	54.46%	23.74%	29.08%	35.61%	33.44%	38.51%	42.31%	25.05%	32.36%	35.17%	33.44%	20.62%	24.06%	18.54%	23.71%	25.63%	35.77%	48.94%	54.31%	23.55%	29.97%	34.56%	16.48%	26.37%	36.26%	33.33%	50.72%	53.62%
Sokal and Sneath 2	58.42%	74.26%	83.17%	35.91%	45.40%	53.12%	46.91%	56.26%	61.33%	43.85%	54.02%	58.58%	46.91%	50.29%	58.42%	33.30%	40.92%	45.66%	54.31%	74.15%	80.16%	40.06%	49.85%	56.27%	35.16%	50.55%	56.04%	52.17%	62.32%	68.12%
Sokal and Sneath 4	47.52%	60.40%	75.25%	27.60%	34.12%	40.95%	35.97%	45.01%	51.35%	31.73%	42.20%	47.97%	35.97%	38.47%	48.53%	23.87%	32.29%	36.76%	35.28%	60.33%	67.97%	34.25%	44.04%	49.24%	24.18%	41.76%	53.85%	37.68%	57.97%	63.77%
Sokal binary distance	42.57%	49.50%	56.44%	24.93%	30.86%	39.17%	36.77%	42.00%	45.96%	28.83%	36.09%	39.68%	36.77%	26.91%	30.93%	21.20%	24.77%	26.96%	39.51%	56.59%	63.74%	27.52%	37.31%	43.43%	16.48%	31.87%	42.86%	39.13%	56.52%	57.97%
Sorenson	51.49%	65.35%	74.26%	27.89%	37.98%	45.10%	43.58%	50.55%	55.31%	38.61%	47.92%	53.88%	43.58%	43.92%	53.23%	28.82%	35.70%	40.86%	43.58%	67.48%	75.45%	33.64%	40.98%	49.85%	34.07%	50.55%	52.75%	44.93%	62.32%	68.12%
Yule	35.64%	45.54%	53.47%	11.28%	16.62%	23.44%	19.65%	31.22%	38.67%	13.37%	21.66%	28.15%	19.65%	18.36%	24.56%	8.47%	11.19%	15.56%	18.54%	21.46%	23.90%	13.15%	20.80%	24.46%	15.38%	27.47%	48.35%	24.64%	44.93%	56.52%
PTMM	65.32%	74.84%	84.86%	51.74%	59.58%	61.60%	58.99%	63.11%	66.51%	53.20%	59.80%	63.68%	56.17%	57.35%	63.93%	46.41%	53.40%	57.88%	76.36%	80.57%	85.24%	53.48%	59.72%	63.67%	57.54%	65.89%	68.61%	54.19%	61.99%	70.44%

EXTRACT METHOD										
	OSCache	PicoContainer	PMD	POI	Pooka	ProGuard	Quartz	QuickServer	Quilt	Roller
Coefficient	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits
Baroni-Urbani and Buser	52.69%	47.15%	61.53%	59.95%	57.90%	63.64%	57.01%	50.87%	72.29%	76.27%
Dot-product	63.44%	58.54%	63.80%	62.99%	63.39%	79.32%	71.34%	65.09%	86.75%	74.58%
Hamann	9.68%	1.63%	12.50%	13.90%	9.03%	22.28%	13.73%	5.21%	12.05%	16.95%
Jaccard	62.37%	64.23%	68.34%	64.73%	66.51%	80.75%	74.66%	69.83%	89.16%	72.88%
Kulczynski	75.27%	65.04%	78.41%	70.11%	74.25%	84.14%	79.64%	77.88%	89.16%	71.19%
Ochiai	67.74%	63.41%	70.45%	66.72%	68.17%	80.57%	76.17%	69.98%	86.75%	74.58%
Phi	66.67%	60.98%	69.64%	66.72%	67.90%	80.21%	75.72%	69.83%	85.54%	74.58%
PSC	69.89%	69.92%	75.49%	69.94%	72.04%	84.31%	81.60%	76.30%	90.36%	69.49%
Relative Matching	78.49%	77.24%	86.04%	78.19%	79.62%	84.31%	82.35%	82.46%	89.16%	91.53%
Rogers and Tanimoto	9.68%	1.63%	12.50%	13.90%	9.14%	22.99%	14.03%	5.37%	13.25%	16.95%
Russell and Rao	83.87%	86.18%	87.82%	82.45%	83.87%	88.24%	87.93%	90.36%	92.77%	91.53%
Simple matching	9.68%	1.63%	12.50%	13.90%	8.98%	22.28%	13.73%	5.21%	12.05%	16.95%
Sokal and Sneath	9.68%	1.63%	12.18%	13.90%	8.98%	22.28%	13.73%	5.21%	12.05%	16.95%
Sokal and Sneath 2	65.59%	65.85%	70.13%	69.24%	70.38%	83.78%	81.90%	74.41%	91.57%	72.88%
Sokal and Sneath 4	74.19%	63.41%	78.25%	70.11%	73.76%	83.78%	79.49%	76.78%	87.95%	71.19%
Sokal binary distance	12.90%	2.44%	17.37%	18.77%	18.39%	32.62%	18.40%	10.58%	18.07%	23.73%
Sorenson	63.44%	58.54%	63.80%	62.99%	63.39%	79.32%	71.34%	65.09%	86.75%	74.58%
Yule	64.52%	52.03%	65.26%	63.25%	62.26%	65.06%	52.34%	59.56%	81.93%	79.66%
PTEM	86.11%	86.40%	88.49%	82.60%	84.53%	88.43%	88.13%	90.55%	92.95%	91.58%

To be continued in the next page.

MOVE CLASS																														
	RSSOwl			SableCC			SandMark			SpringFramework			Squirrel_SQL			Struts			SunFlow			Tapestry			Trove			Velocity		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	41.03%	48.65%	57.17%	89.73%	96.58%	98.63%	27.93%	36.12%	39.80%	35.13%	48.43%	59.12%	64.10%	94.87%	100.00%	25.60%	36.25%	40.34%	62.30%	70.49%	73.77%	9.38%	16.62%	21.58%	66.67%	94.44%	100.00%	58.44%	69.48%	74.68%
Dot-product	53.36%	66.59%	76.01%	89.73%	96.58%	98.63%	40.97%	52.68%	56.52%	48.66%	65.91%	75.15%	71.79%	94.87%	100.00%	40.54%	51.79%	57.17%	68.03%	76.23%	79.51%	17.83%	28.69%	35.92%	83.33%	94.44%	100.00%	77.27%	87.66%	89.61%
Hamann	17.94%	21.08%	27.80%	63.70%	97.95%	97.95%	21.40%	27.76%	32.78%	16.78%	23.34%	28.98%	10.26%	33.33%	100.00%	18.53%	24.10%	24.90%	55.74%	64.75%	65.57%	7.24%	8.04%	9.79%	66.67%	100.00%	100.00%	44.81%	50.65%	52.60%
Jaccard	55.16%	69.06%	76.91%	91.10%	96.58%	98.63%	42.64%	55.02%	59.36%	50.87%	68.12%	76.89%	69.23%	94.87%	100.00%	42.23%	54.28%	60.26%	68.03%	77.05%	80.33%	18.10%	30.03%	37.13%	83.33%	94.44%	100.00%	81.82%	88.96%	92.21%
Kulczynski	46.86%	60.99%	70.18%	80.14%	95.89%	99.32%	39.80%	49.67%	53.34%	48.72%	64.81%	74.22%	69.23%	92.31%	100.00%	38.65%	49.80%	55.18%	60.66%	72.13%	76.23%	16.62%	24.93%	30.70%	88.89%	94.44%	100.00%	78.57%	85.06%	88.31%
Ochiai	52.02%	64.35%	74.44%	84.93%	96.58%	100.00%	41.30%	50.17%	57.02%	49.13%	65.39%	74.74%	71.79%	94.87%	100.00%	40.14%	52.19%	57.97%	66.39%	76.23%	79.51%	17.43%	26.81%	33.65%	83.33%	94.44%	100.00%	77.92%	87.66%	90.26%
Phi	52.24%	64.35%	74.44%	91.78%	96.58%	98.63%	41.64%	50.33%	57.69%	48.84%	65.33%	74.56%	66.67%	94.87%	100.00%	40.14%	52.09%	58.07%	66.39%	74.59%	78.69%	17.16%	26.68%	33.51%	88.89%	94.44%	100.00%	77.92%	87.66%	90.26%
PSC	60.09%	73.09%	79.60%	91.78%	96.58%	99.32%	50.33%	62.54%	69.90%	56.91%	73.46%	81.77%	69.23%	94.87%	100.00%	47.01%	59.26%	63.84%	53.03%	63.69%	67.79%	8.99%	21.60%	30.17%	73.89%	85.00%	85.00%	71.36%	76.56%	77.86%
Relative Matching	43.05%	52.02%	57.40%	45.21%	93.84%	99.32%	21.91%	30.60%	33.95%	38.56%	57.08%	66.43%	74.36%	94.87%	100.00%	34.26%	47.01%	51.79%	47.54%	63.93%	70.49%	14.08%	22.25%	28.69%	66.67%	94.44%	100.00%	64.29%	68.18%	72.73%
Rogers and Tanimoto	17.94%	21.08%	27.80%	63.70%	97.95%	97.95%	21.24%	27.76%	33.11%	16.90%	23.46%	29.04%	10.26%	30.77%	100.00%	18.53%	24.40%	25.20%	56.56%	64.75%	65.57%	7.24%	8.04%	9.79%	72.22%	100.00%	100.00%	44.81%	51.30%	52.60%
Russell and Rao	46.64%	54.71%	60.54%	45.89%	86.99%	98.63%	28.26%	37.63%	40.64%	47.62%	66.26%	75.32%	82.05%	94.87%	100.00%	40.54%	53.39%	59.46%	47.54%	64.75%	71.31%	17.43%	26.41%	35.12%	77.78%	94.44%	100.00%	69.48%	75.32%	81.82%
Simple matching	17.94%	21.08%	27.80%	63.70%	97.95%	97.95%	21.40%	27.59%	32.61%	16.43%	23.23%	29.04%	10.26%	33.33%	100.00%	18.73%	24.30%	25.10%	55.74%	64.75%	65.57%	7.10%	7.91%	9.79%	66.67%	100.00%	100.00%	44.81%	50.65%	52.60%
Sokal and Sneath	17.94%	20.85%	27.80%	63.70%	97.95%	97.95%	20.57%	27.09%	31.94%	16.20%	22.76%	28.80%	10.26%	33.33%	100.00%	18.33%	24.00%	24.60%	55.74%	64.75%	65.57%	6.70%	7.64%	9.65%	66.67%	100.00%	100.00%	44.81%	50.00%	52.60%
Sokal and Sneath 2	57.40%	70.85%	78.03%	92.47%	96.58%	98.63%	46.15%	58.53%	65.05%	52.21%	69.86%	78.22%	69.23%	94.87%	100.00%	44.12%	55.98%	61.45%	68.03%	78.69%	81.15%	19.57%	31.64%	38.61%	83.33%	94.44%	100.00%	83.77%	90.91%	92.86%
Sokal and Sneath 4	46.64%	62.33%	70.63%	88.36%	95.89%	99.32%	39.97%	49.83%	53.51%	48.84%	64.58%	73.93%	69.23%	92.31%	100.00%	38.65%	49.60%	55.08%	62.30%	72.13%	74.59%	16.62%	24.80%	30.83%	88.89%	94.44%	100.00%	77.92%	85.06%	88.96%
Sokal binary distance	20.40%	22.65%	28.03%	63.01%	97.26%	97.26%	28.60%	32.78%	40.64%	18.00%	24.68%	30.66%	12.82%	41.03%	100.00%	23.51%	26.39%	27.49%	58.20%	67.21%	69.67%	8.45%	9.25%	10.86%	77.78%	94.44%	100.00%	51.30%	54.55%	57.14%
Sorenson	53.36%	66.59%	76.01%	89.73%	96.58%	98.63%	40.97%	52.68%	56.52%	48.66%	65.91%	75.15%	71.79%	94.87%	100.00%	40.54%	51.79%	57.17%	68.03%	76.23%	79.51%	17.83%	28.69%	35.92%	83.33%	94.44%	100.00%	77.27%	87.66%	89.61%
Yule	34.08%	43.50%	48.43%	79.45%	93.15%	98.63%	20.40%	25.25%	26.92%	28.11%	42.97%	52.50%	71.79%	94.87%	100.00%	17.43%	24.10%	31.87%	52.46%	68.03%	68.85%	7.37%	10.46%	16.49%	83.33%	94.44%	100.00%	40.91%	54.55%	59.09%
PTMC	62.80%	69.12%	76.87%	97.46%	95.76%	101.34%	55.48%	59.46%	67.36%	62.47%	71.21%	79.44%	63.03%	90.12%	100.00%	54.73%	61.29%	67.34%	71.16%	85.88%	90.32%	27.71%	43.74%	53.15%	99.15%	100.00%	100.00%	93.64%	99.90%	100.00%

MOVE METHOD																														
	RSSOwl			SableCC			SandMark			SpringFramework			Squirrel_SQL			Struts			SunFlow			Tapestry			Trove			Velocity		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	8.70%	13.13%	17.15%	69.26%	77.61%	80.65%	10.26%	17.49%	21.69%	36.80%	49.81%	58.40%	13.22%	35.54%	47.11%	16.16%	22.10%	25.19%	16.87%	30.86%	39.92%	11.28%	17.06%	21.84%	57.69%	75.64%	87.18%	8.70%	13.13%	17.15%
Dot-product	20.51%	29.67%	36.08%	72.30%	78.18%	82.54%	21.40%	33.29%	40.12%	49.85%	64.00%	72.34%	33.88%	51.24%	66.12%	34.44%	44.49%	49.23%	32.10%	42.80%	49.79%	23.92%	33.66%	38.81%	71.79%	78.21%	94.87%	20.51%	29.67%	36.08%
Hamann	8.85%	12.67%	15.17%	46.49%	52.94%	55.98%	12.36%	19.24%	23.62%	36.59%	46.36%	52.22%	5.79%	13.22%	15.70%	17.30%	25.01%	29.01%	17.28%	25.93%	32.51%	11.28%	16.52%	19.40%	37.18%	41.03%	47.44%	8.85%	12.67%	15.17%
Jaccard	22.90%	32.52%	39.49%	72.11%	79.13%	83.68%	26.65%	37.96%	44.49%	52.41%	66.43%	74.51%	35.54%	53.72%	67.77%	38.66%	47.92%	53.00%	37.86%	47.33%	53.09%	27.80%	37.00%	43.05%	74.36%	82.05%	94.87%	22.90%	32.52%	39.49%
Kulczynski	13.54%	23.05%	30.43%	70.59%	76.85%	81.78%	13.53%	22.51%	28.98%	48.75%	64.62%	73.31%	23.97%	44.63%	53.72%	24.16%	35.29%	41.58%	24.69%	38.27%	46.50%	11.46%	21.21%	26.62%	58.97%	73.08%	92.31%	13.54%	23.05%	30.43%
Ochiai	18.17%	27.74%	33.59%	71.73%	78.94%	82.92%	18.89%	30.90%	37.26%	50.15%	64.51%	73.14%	28.10%	47.93%	58.68%	30.78%	41.63%	47.00%	29.22%	42.80%	52.67%	19.58%	29.24%	34.66%	69.23%	76.92%	94.87%	18.17%	27.74%	33.59%
Phi	17.96%	27.79%	33.54%	72.30%	78.94%	82.54%	18.89%	30.73%	37.03%	50.15%	64.42%	73.16%	28.10%	47.11%	58.68%	30.67%	41.69%	47.00%	29.22%	43.21%	52.26%	19.49%	29.24%	34.57%	67.95%	78.21%	94.87%	17.96%	27.79%	33.54%
PSC	24.33%	36.23%	43.16%	67.55%	78.56%	83.68%	27.58%	40.23%	46.53%	55.92%	70.22%	77.24%	38.02%	52.89%	68.60%	38.44%	48.66%	53.91%	38.68%	48.97%	53.91%	26.90%	36.73%	42.69%	71.79%	82.05%	92.31%	24.33%	36.23%	43.16%
Relative Matching	17.05%	24.48%	29.72%	64.71%	75.14%	79.13%	14.58%	23.27%	28.92%	43.44%	56.43%	64.91%	22.31%	43.80%	55.37%	26.61%	38.55%	43.75%	21.40%	35.80%	41.15%	19.77%	28.16%	33.57%	70.51%	84.62%	94.87%	17.05%	24.48%	29.72%
Rogers and Tanimoto	9.01%	12.88%	15.32%	46.30%	52.75%	55.98%	12.36%	19.30%	23.44%	36.95%	46.91%	52.96%	5.79%	13.22%	16.53%	17.65%	25.64%	29.70%	20.16%	27.57%	33.74%	11.37%	16.52%	19.77%	37.18%	41.03%	46.15%	9.01%	12.88%	15.32%
Russell and Rao	24.99%	34.10%	40.51%	64.52%	74.00%	79.70%	21.57%	32.07%	38.48%	49.85%	63.90%	72.08%	30.58%	57.85%	68.60%	35.35%	47.12%	53.51%	29.22%	45.27%	51.44%	29.24%	37.36%	43.50%	74.36%	87.18%	94.87%	24.99%	34.10%	40.51%
Simple matching	8.85%	12.57%	15.17%	46.30%	52.94%	55.98%	12.13%	19.24%	23.62%	36.59%	46.55%	52.50%	5.79%	12.40%	16.53%	17.30%	25.19%	29.18%	20.16%	27.57%	34.16%	11.37%	16.61%	19.40%	37.18%	41.03%	47.44%	8.85%	12.57%	15.17%
Sokal and Sneath	8.55%	12.11%	14.50%	46.11%	52.75%	55.60%	11.55%	18.08%	22.86%	35.30%	45.20%	51.08%	5.79%	10.74%	15.70%	16.62%	24.27%	28.33%	16.46%	24.28%	30.45%	9.66%	15.52%	18.41%	37.18%	41.03%	47.44%	8.55%	12.11%	14.50%
Sokal and Sneath 2	24.83%	35.11%	42.24%	72.11%	79.70%	84.44%	31.02%	42.27%	47.93%	55.44%	68.95%	76.02%	38.02%	56.20%	70.25%	40.72%	49.46%	54.60%	40.74%	49.79%	55.14%	30.96%	40.79%	45.94%	75.64%	84.62%	93.59%	24.83%	35.11%	42.24%
Sokal and Sneath 4	13.49%	23.10%	30.38%	70.97%	77.61%	81.59%	13.59%	22.62%	29.04%	48.67%	64.57%	73.20%	23.97%	44.63%	54.55%	24.10%	35.24%	41.46%	24.69%	38.27%	46.50%	11.19%	21.21%	26.62%	61.54%	71.79%	92.31%	13.49%	23.10%	30.38%
Sokal binary distance	10.48%	14.96%	17.35%	47.06%	52.94%	56.55%	15.51%	22.45%	26.94%	39.74%	50.30%	56.18%	12.40%	16.53%	22.31%	21.59%	29.13%	33.18%	25.51%	35.39%	40.74%	13.72%	18.77%	22.56%	46.15%	52.56%	58.97%	10.48%	14.96%	17.35%
Sorenson	20.51%	29.67%	36.08%	72.30%	78.18%	82.54%	21.40%	33.29%	40.12%	49.85%	64.00%	72.34%	33.88%	51.24%	66.12%	34.44%	44.49%	49.23%	32.10%	42.80%	49.79%	23.92%	33.66%	38.81%	71.79%	78.21%	94.87%	20.51%	29.67%	36.08%
Yule	5.50%	10.03%	13.69%	64.52%	70.21%	76.47%	4.61%	8.92%	13.88%	30.65%	43.46%	51.99%	8.26%	26.45%	38.02%	6.40%	12.79%	18.56%	11.52%	20.99%	28.40%	4.24%	6.59%	10.56%	58.97%	75.64%	87.18%	5.50%	10.03%	13.69%
PTMM	41.68%	50.25%	52.95%	70.98%	79.67%	79.85%	45.43%	53.51%	57.54%	64.18%	72.93%	79.80%	59.65%	74.17%	81.07%	55.05%	59.36%	64.91%	54.84%	65.98%	65.55%	45.86%	54.23%	57.37%	86.09%	92.08%	93.97%	42.37%	50.53%	53.40%

EXTRACT METHOD										
	RSSOwl	SableCC	SandMark	SpringFramework	Squirrel_SQL	Struts	SunFlow	Tapestry	Trove	Velocity
Coefficient	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits	Hits
Baroni-Urbani and Buser	70.34%	55.46%	62.29%	65.09%	64.81%	69.67%	65.76%	59.49%	89.39%	55.97%
Dot-product	74.49%	69.89%	66.62%	72.94%	69.44%	74.76%	68.47%	66.67%	98.48%	70.31%
Hamann	6.89%	16.20%	4.55%	6.36%	12.96%	11.59%	14.04%	0.84%	24.24%	21.50%
Jaccard	75.73%	73.06%	68.70%	75.20%	62.96%	75.36%	69.21%	66.39%	96.97%	72.35%
Kulczynski	80.52%	76.41%	74.95%	80.05%	73.15%	84.01%	75.62%	73.84%	98.48%	79.52%
Ochiai	77.23%	72.54%	69.70%	76.42%	68.52%	79.56%	70.69%	66.67%	98.48%	73.72%
Phi	76.72%	72.36%	69.44%	76.01%	68.52%	79.26%	71.18%	66.67%	98.48%	73.38%
PSC	76.84%	74.65%	70.52%	78.03%	67.59%	78.01%	70.69%	72.15%	98.48%	76.79%
Relative Matching	88.26%	76.58%	85.18%	88.14%	72.22%	90.40%	74.63%	87.76%	98.48%	86.01%
Rogers and Tanimoto	7.04%	16.73%	4.51%	6.60%	12.96%	11.59%	14.04%	0.84%	24.24%	21.50%
Russell and Rao	90.85%	85.04%	89.64%	91.89%	79.63%	92.10%	81.77%	93.67%	98.48%	91.47%
Simple matching	6.92%	16.20%	4.42%	6.44%	12.96%	11.59%	14.04%	0.84%	24.24%	21.50%
Sokal and Sneath	6.77%	14.96%	4.20%	6.12%	12.04%	11.39%	13.30%	0.84%	24.24%	21.50%
Sokal and Sneath 2	77.27%	76.06%	70.65%	77.09%	65.74%	75.91%	68.97%	72.15%	96.97%	76.11%
Sokal and Sneath 4	79.89%	74.65%	73.78%	79.68%	71.30%	83.86%	74.14%	73.42%	98.48%	77.13%
Sokal binary distance	10.68%	26.06%	7.72%	10.78%	17.59%	16.09%	27.59%	5.49%	24.24%	29.69%
Sorenson	74.49%	69.89%	66.62%	72.94%	69.44%	74.76%	68.47%	66.67%	98.48%	70.31%
Yule	73.00%	55.11%	66.28%	69.46%	62.04%	72.01%	70.20%	66.24%	77.27%	61.43%
PTEM	91.21%	86.04%	89.76%	92.34%	79.73%	92.59%	82.08%	93.68%	98.57%	90.91%

To be continued in the next page.

MOVE CLASS																					
	WCT			WebMail			WEKA			Xalan			Xerces			XMojo			Mean		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
Baroni-Urbani and Buser	27.98%	34.52%	39.58%	34.15%	43.90%	46.34%	25.38%	39.70%	47.61%	37.52%	44.20%	54.30%	49.43%	58.71%	64.58%	14.29%	57.14%	71.43%	**39.32%**	**52.24%**	**60.42%**
Dot-product	36.01%	46.73%	55.65%	36.59%	53.66%	63.41%	34.46%	52.74%	61.35%	49.93%	64.67%	69.58%	61.93%	72.54%	76.33%	14.29%	57.14%	85.71%	**50.50%**	**64.50%**	**72.29%**
Hamann	20.54%	23.21%	25.30%	24.39%	34.15%	48.78%	20.84%	32.95%	38.77%	23.47%	28.24%	29.60%	29.73%	31.44%	34.28%	0.00%	42.86%	57.14%	**26.43%**	**36.16%**	**43.43%**
Jaccard	36.90%	47.92%	57.14%	36.59%	58.54%	68.29%	37.95%	56.23%	63.80%	54.02%	65.89%	70.26%	63.83%	73.48%	77.84%	28.57%	71.43%	85.71%	**52.62%**	**66.53%**	**73.98%**
Kulczynski	34.52%	45.24%	52.08%	24.39%	46.34%	48.78%	38.65%	56.11%	65.31%	46.93%	58.94%	65.21%	61.93%	71.59%	75.38%	14.29%	57.14%	71.43%	**48.58%**	**63.33%**	**71.48%**
Ochiai	35.42%	48.21%	55.06%	39.02%	51.22%	63.41%	34.58%	53.78%	62.75%	49.66%	62.35%	67.80%	63.45%	72.92%	76.14%	14.29%	57.14%	71.43%	**50.40%**	**64.48%**	**72.32%**
Phi	35.42%	48.81%	55.65%	39.02%	56.10%	63.41%	35.04%	54.13%	63.21%	49.39%	62.21%	67.67%	63.26%	72.92%	76.52%	14.29%	57.14%	71.43%	**50.56%**	**64.39%**	**72.31%**
PSC	24.29%	37.68%	43.33%	28.90%	53.29%	63.05%	29.35%	46.12%	52.87%	42.98%	53.21%	57.58%	52.42%	62.46%	66.44%	13.57%	56.43%	70.71%	**55.03%**	**69.20%**	**76.36%**
Relative Matching	30.06%	41.37%	48.21%	24.39%	34.15%	41.46%	25.03%	37.02%	47.50%	41.75%	54.57%	63.71%	56.06%	65.15%	70.45%	14.29%	57.14%	85.71%	**39.65%**	**55.29%**	**64.70%**
Rogers and Tanimoto	21.13%	23.51%	25.60%	26.83%	34.15%	48.78%	21.07%	33.18%	38.88%	23.47%	28.51%	30.01%	29.92%	31.82%	34.28%	0.00%	42.86%	57.14%	**26.78%**	**36.37%**	**43.55%**
Russell and Rao	34.52%	45.54%	53.87%	26.83%	34.15%	56.10%	28.99%	42.84%	54.37%	48.29%	62.21%	69.99%	63.07%	74.24%	78.22%	28.57%	85.71%	100.00%	**45.11%**	**61.79%**	**70.81%**
Simple matching	20.83%	23.21%	25.30%	26.83%	34.15%	48.78%	20.61%	33.18%	38.77%	23.47%	28.24%	29.60%	29.55%	31.25%	34.28%	0.00%	42.86%	57.14%	**26.59%**	**36.26%**	**43.70%**
Sokal and Sneath	19.94%	22.62%	24.70%	24.39%	34.15%	48.78%	20.26%	32.25%	38.30%	23.47%	28.24%	29.47%	29.55%	30.30%	33.33%	0.00%	42.86%	42.86%	**26.23%**	**35.92%**	**43.09%**
Sokal and Sneath 2	37.80%	50.00%	58.33%	36.59%	56.10%	68.29%	42.96%	58.67%	66.59%	55.39%	66.98%	70.94%	65.53%	74.24%	78.41%	28.57%	71.43%	85.71%	**54.32%**	**68.02%**	**75.22%**
Sokal and Sneath 4	34.82%	45.83%	52.08%	26.83%	48.78%	53.66%	38.65%	55.88%	65.19%	46.93%	59.07%	64.67%	62.12%	71.59%	75.57%	14.29%	57.14%	71.43%	**48.80%**	**63.53%**	**71.54%**
Sokal binary distance	22.02%	25.00%	26.79%	26.83%	34.15%	46.34%	22.82%	35.62%	40.98%	25.38%	31.38%	32.74%	33.14%	35.80%	37.88%	0.00%	28.57%	42.86%	**29.39%**	**38.72%**	**46.09%**
Sorenson	36.01%	46.73%	55.65%	36.59%	53.66%	63.41%	34.46%	52.74%	61.35%	49.93%	64.67%	69.58%	61.93%	72.54%	76.33%	14.29%	57.14%	85.71%	**50.69%**	**64.76%**	**72.61%**
Yule	15.77%	20.83%	27.98%	12.20%	36.59%	41.46%	21.07%	34.58%	42.03%	29.88%	34.79%	39.56%	33.33%	47.16%	56.25%	14.29%	42.86%	71.43%	**30.79%**	**43.01%**	**51.62%**
PTMC	50.38%	65.22%	72.92%	51.08%	73.98%	88.61%	54.45%	71.54%	81.14%	67.46%	76.73%	84.79%	73.89%	83.74%	88.30%	47.04%	79.74%	93.86%	**62.91%**	**71.11%**	**79.97%**

MOVE METHOD																					
	WCT			WebMail			WEKA			Xalan			Xerces			XMojo			Mean		
Coefficient	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3	Top1	Top2	Top3
textbfBaroni-Urbani and Buser	15.84%	23.27%	30.69%	13.84%	27.04%	35.85%	7.78%	11.34%	13.72%	10.33%	12.14%	14.77%	9.89%	16.75%	19.93%	23.44%	35.94%	51.56%	**21.97%**	**29.98%**	**35.43%**
Dot-product	27.60%	39.48%	46.91%	24.53%	38.99%	53.46%	15.87%	22.48%	27.45%	21.75%	32.94%	38.11%	26.09%	36.26%	41.97%	31.25%	40.63%	57.81%	**34.24%**	**45.34%**	**52.32%**
Hamann	19.06%	26.98%	31.93%	16.98%	28.30%	34.59%	9.15%	14.19%	17.68%	15.22%	20.80%	25.42%	19.24%	24.99%	30.14%	15.63%	23.44%	29.69%	**22.96%**	**30.05%**	**34.19%**
Jaccard	29.83%	43.56%	50.74%	27.04%	46.54%	55.35%	19.25%	26.56%	31.34%	26.33%	36.93%	42.37%	30.14%	40.04%	46.89%	37.50%	46.88%	65.63%	**37.84%**	**49.01%**	**55.94%**
Kulczynski	20.67%	33.17%	41.96%	27.04%	36.48%	47.17%	11.43%	18.54%	23.89%	15.86%	23.11%	27.87%	18.04%	26.88%	34.19%	31.25%	48.44%	54.69%	**28.61%**	**40.16%**	**47.54%**
Ochiai	25.62%	38.74%	46.29%	23.90%	41.51%	54.09%	14.85%	22.17%	27.05%	19.44%	28.18%	33.12%	24.02%	33.09%	39.67%	26.56%	42.19%	57.81%	**32.61%**	**43.98%**	**51.13%**
Phi	25.37%	38.37%	46.04%	23.27%	41.51%	52.83%	14.83%	22.19%	26.94%	19.44%	28.18%	33.03%	23.93%	33.04%	39.62%	28.13%	42.19%	57.81%	**32.50%**	**43.92%**	**51.01%**
PSC	31.06%	46.16%	52.23%	30.82%	47.80%	56.60%	19.98%	29.10%	33.88%	27.50%	35.66%	41.37%	30.05%	40.54%	49.56%	39.06%	50.00%	71.88%	**38.98%**	**51.06%**	**58.31%**
Relative Matching	22.65%	34.41%	40.22%	20.13%	32.08%	44.65%	12.33%	16.24%	20.07%	16.09%	22.47%	28.45%	20.85%	30.83%	35.48%	28.13%	39.06%	53.13%	**27.59%**	**37.43%**	**43.64%**
Rogers and Tanimoto	19.31%	27.10%	31.93%	16.35%	27.67%	34.59%	9.55%	14.63%	17.97%	15.77%	21.25%	25.83%	19.37%	25.31%	30.14%	15.63%	21.88%	31.25%	**23.17%**	**30.31%**	**34.54%**
Russell and Rao	28.96%	41.96%	48.27%	28.30%	41.51%	45.28%	17.77%	23.82%	28.13%	22.20%	32.67%	39.65%	28.26%	38.10%	43.95%	35.94%	53.13%	67.19%	**34.62%**	**45.74%**	**52.49%**
Simple matching	19.18%	26.98%	31.81%	16.98%	27.04%	33.96%	9.39%	14.48%	17.83%	15.09%	20.53%	25.24%	19.33%	25.26%	30.19%	15.63%	23.44%	29.69%	**22.92%**	**30.03%**	**34.20%**
Sokal and Sneath	18.44%	26.36%	31.06%	16.98%	26.42%	34.59%	8.80%	13.83%	17.04%	14.54%	19.26%	24.78%	19.01%	24.90%	29.91%	15.63%	21.88%	28.13%	**22.32%**	**29.23%**	**33.33%**
Sokal and Sneath 2	34.78%	47.40%	53.22%	31.45%	49.06%	59.12%	21.68%	29.50%	34.76%	29.77%	39.28%	45.13%	32.26%	42.94%	51.04%	35.94%	45.31%	62.50%	**40.79%**	**51.86%**	**58.58%**
Sokal and Sneath 4	20.54%	33.17%	41.83%	27.04%	36.48%	47.17%	11.43%	18.52%	23.80%	15.81%	22.97%	27.82%	17.99%	26.97%	34.19%	23.44%	43.75%	57.81%	**28.47%**	**40.07%**	**47.54%**
Sokal binary distance	21.41%	30.20%	36.14%	16.35%	28.93%	35.22%	11.93%	17.28%	21.10%	18.89%	25.74%	29.13%	21.68%	28.95%	33.23%	23.44%	26.56%	46.88%	**26.18%**	**33.64%**	**38.21%**
Sorenson	27.60%	39.48%	46.91%	24.53%	38.99%	53.46%	15.87%	22.48%	27.45%	21.75%	32.94%	38.11%	26.09%	36.26%	41.97%	31.25%	40.63%	57.81%	**34.24%**	**45.34%**	**52.32%**
Yule	9.41%	18.19%	25.37%	15.09%	23.27%	28.93%	3.43%	7.38%	10.17%	4.80%	7.39%	10.42%	4.42%	9.39%	13.58%	21.88%	34.38%	51.56%	**14.21%**	**22.07%**	**27.83%**
PTMM	55.43%	60.27%	64.03%	45.97%	54.11%	59.75%	41.60%	45.71%	47.87%	41.81%	50.47%	54.22%	57.11%	62.26%	66.20%	36.88%	54.74%	82.75%	**54.18%**	**62.71%**	**66.94%**

EXTRACT METHOD							
	WCT	WebMail	WEKA	Xalan	Xerces	XMojo	Mean
Coefficient	Hits	Hits	Hits	Hits	Hits	Hits	Hits
Baroni-Urbani and Buser	81.49%	59.23%	59.40%	58.93%	69.91%	31.58%	**61.74%**
Dot-product	87.84%	69.62%	68.16%	69.70%	81.33%	29.82%	**69.48%**
Hamann	17.54%	5.00%	12.14%	14.28%	25.58%	3.51%	**11.86%**
Jaccard	88.66%	71.54%	73.74%	76.36%	83.58%	29.82%	**71.74%**
Kulczynski	92.61%	75.00%	74.44%	79.86%	86.95%	29.82%	**77.62%**
Ochiai	90.75%	74.23%	70.70%	74.22%	83.60%	29.82%	**73.11%**
Phi	90.75%	73.85%	70.61%	74.15%	83.47%	29.82%	**72.70%**
PSC	90.37%	74.62%	77.17%	79.90%	86.30%	29.82%	**75.35%**
Relative Matching	94.03%	83.85%	73.65%	79.10%	85.31%	47.37%	**81.51%**
Rogers and Tanimoto	17.84%	5.00%	12.20%	14.43%	25.58%	3.51%	**12.02%**
Russell and Rao	95.90%	88.85%	79.42%	83.95%	90.46%	43.86%	**86.58%**
Simple matching	17.54%	5.00%	12.08%	14.10%	25.58%	3.51%	**11.83%**
Sokal and Sneath	17.01%	4.23%	12.01%	13.88%	25.36%	3.51%	**11.66%**
Sokal and Sneath 2	88.88%	74.62%	78.63%	80.80%	85.76%	29.82%	**74.37%**
Sokal and Sneath 4	92.31%	74.62%	74.27%	78.96%	86.71%	29.82%	**76.66%**
Sokal binary distance	22.91%	6.15%	23.48%	24.48%	33.90%	10.53%	**18.03%**
Sorenson	87.84%	69.62%	68.16%	69.70%	81.33%	29.82%	**69.48%**
Yule	81.19%	69.62%	61.62%	64.97%	69.67%	40.35%	**65.50%**
PTEM	96.14%	89.45%	79.67%	84.26%	90.69%	43.94%	**87.00%**