Search and Retrieval of Source Code Using the Faceted Approach

by

Rodrigo Cavalcante Mendes

M.Sc. Dissertation

RECIFE, AUGUST/2008
RODRIGO CAVALCANTE MENDES

“SEARCH AND RETRIEVAL OF SOURCE CODE USING THE FACETED APPROACH"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR(A): SILVIO ROMERO DE LEMOS MEIRA
CO-ORIENTADOR(A): EDUARDO SANTANA DE ALMEIDA

RECIFE, AGOSTO/2008
Mendes, Rodrigo Cavalcante

106 folhas: il., fig., tab.


Inclui bibliografia e apêndice.

1. Engenharia de software. I. Título.

005.1 CDD (22.ed.) MEI2008-121
Dissertação de Mestrado apresentada por Rodrigo Cavalcante Mendes à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título “Search and Retrieval of Source Code Using the Faceted Approach”, orientada pelo Prof. Silvio Romero de Lemos Meira e aprovada pela Banca Examinadora formada pelos professores:

Prof. Carlos André Guimarães Ferraz
Centro de Informática / UFPE

Prof. Jones Oliveira de Albuquerque
Departamento de Estatística e Informática / UFRPE

Prof. Silvio Romero de Lemos Meira
Centro de Informática / UFPE

Visto e permitida a impressão.

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco.
I would like to express my deeply gratitude to everyone that supported me in this (quite long) journey that, I am sure, I would have not accomplished it alone. These last years were full of obstacles and, at the same time, of joy. My lovely son, Guilherme, was a gifted surprise and was born in a moment of life changing.

In this long road, I would like to thank my beloved family that gave me all the emotional support I needed. A special thanks to my wife, Danielle, and my parents, Jorge and Socorro, for the patience and support in hard times.

I would like to thank all the people that are part of the RiSE family, in special, my friends: Alexandre Martins, Frederico Durão, Liana Barachio, Eduardo Cruz, Vinicius Garcia for the constructive and remarkable discussions and, to bet in my potential and always cheering me up. A special thanks to my co-Advisor Eduardo Almeida (this guy will make history, take note about it) for to be my guru and friend, always pushing me and renewing my energies, and my Advisor Silvio Meira for the tricks, but also challenging advices.

I would like to thanks Dr. Ruben Pietro-Díaz and Dr. David Weiss for the remarkable and priceless discussions that guided me during the entire mastering course.

I would like to thank the UNICAP/Itautec laboratory for the comprehension and support and to Recife Center Advanced Studies and Systems (C.E.S.A.R.) software factory for providing all infrastructure for the basis of this research.
resumo

O Reuso de Software tem-se firmado como um conceito chave no aumento de produtividade e qualidade no processo de desenvolvimento de software, permitindo a reutilização dos artefatos existentes evitando assim a necessidade de construí-los do zero. No entanto, para se conseguir benefícios efetivos através do reuso de software é necessário um conjunto de recursos complementares, como: educação, forte apoio gerencial e a introdução de processos e ferramentas apropriadas.

De fato, recursos que facilitem o acesso a componentes reutilizáveis, como as ferramentas de busca e recuperação, apresentam-se como potenciais instrumentos para adoção do programa de reuso nas organizações. Um dos desafios das ferramentas de busca e recuperação é conseguir que os componentes existentes retornados possuam uma relevância significativa.

Nesse contexto, a abordagem do uso de facetas surge como uma possível alternativa. Esta abordagem se propõe a criação de um vocabulário suportado por atributos dividindo os componentes em grupos de classes baseados em palavras pré-definidas, aumentando o grau de precisão e proporcionando uma classificação mais flexível.

Assim, este trabalho apresenta uma extensão de uma ferramenta de busca e recuperação de componentes reutilizáveis, particularmente, código-fonte, utilizando a abordagem de classificação por facetas. Adicionalmente, uma ferramenta para auxiliar o especialista de domínio em suas atividades utilizando esta abordagem foi também desenvolvida. Por fim, um estudo experimental avalia a solução proposta.

**Palavras-chave:** Reuso de Software, Sistemas de Recuperação de Informação, Facetas, Estudo Experimental.
Abstract

Software Reuse has been considered a key concept to increase the quality and productivity of the software development by the reuse of existing artifacts, avoiding build new ones from scratch. However, In order to obtain effective benefits from the software reuse is necessary a set of complementary resources such as: education, active management support and the introduction of appropriate process and tools.

In fact, resources that provide mechanism to ease the access of reusable components, such as search and retrieval tools, appear as potential instruments in favor of reuse programs adoption in the organizations. One of the challenges of the search and retrieval tools is how to make that existing components returned have a significant relevance.

In this sense, the use of the faceted approach rises as a suitable alternative. This approach proposes the creation of a vocabulary supported by attributes, dividing the components into group of classes based on pre-defined keywords, increasing the level of precision and providing a more flexible classification.

Thus, this work presents an extension of search and retrieval tool of reusable components, source code in particular, using the faceted classification approach. In addition, also was developed an auxiliary tool to aid the Domain expert to perform his activities using this approach. Finally, an experimental study evaluates the proposed solution.

Keywords: Software Reuse, Information Retrieval Systems, Facets, Experimental Study.
# Table of Contents

ACKNOWLEDGMENTS ........................................................................................................... I  
RESUMO .............................................................................................................................. II  
ABSTRACT ........................................................................................................................... III  
TABLE OF CONTENTS ....................................................................................................... IV  
LIST OF FIGURES .............................................................................................................. VIII  
LIST OF TABLES ................................................................................................................ IX  
LIST OF ACRONYMS ......................................................................................................... X  

1. INTRODUCTION ........................................................................................................... 1  
   1.1. MOTIVATION ........................................................................................................... 1  
   1.2. PROBLEM STATEMENT ......................................................................................... 3  
   1.3. OVERVIEW OF THE PROPOSED SOLUTION ....................................................... 4  
   1.4. OUT OF SCOPE ..................................................................................................... 5  
   1.5. STATEMENTS OF THE CONTRIBUTIONS ............................................................ 6  
   1.6. ORGANIZATION OF THE DISSERTATION ......................................................... 7  

2. SOFTWARE REUSE: AN OVERVIEW .......................................................................... 8  
   2.1. INTRODUCTION ..................................................................................................... 8  
   2.2. SOFTWARE REUSE BENEFITS AND UNCERTAINTIES ...................................... 9  
       2.2.1. Benefits ........................................................................................................... 9  
       2.2.2. Uncertainties and Concerns ........................................................................ 10  
   2.3. REUSABLE SOFTWARE ASSETS ........................................................................... 12  
       2.3.1. Source Code Reuse ...................................................................................... 14  
       2.3.2. Reusable Software Assets Accessibility ....................................................... 16  
   2.4. CHAPTER SUMMARY ........................................................................................... 16
3. INFORMATION RETRIEVAL .................................................................................. 18

3.1. INTRODUCTION .................................................................................................. 18

3.2. INFORMATION RETRIEVAL AND DATA RETRIEVAL .......................................... 19

3.3. RELEVANT INFORMATION ................................................................................. 21

3.4. TECHNICAL SUPPORT FOR INFORMATION RETRIEVAL SYSTEMS .................. 22

3.4.1. Techniques for Query Formulation Support ................................................. 22

3.4.2. Classification Schemes for Component Retrieval Support ......................... 23

3.4.3. Source Code Retrieval for Reuse Activity ................................................. 25

3.4.4. Information Retrieval Systems for Component Reuse .................................. 26

3.5. INFORMATION RETRIEVAL DISCUSSION ...................................................... 29

3.6. CHAPTER SUMMARY ....................................................................................... 30

4. SEARCH AND RETRIEVAL OF SOURCE CODE USING THE FACETED
   APPROACH ........................................................................................................... 31

4.1. INTRODUCTION .................................................................................................. 32

4.2. FACETED CLASSIFICATION CONCEPTS ......................................................... 35

4.2.1. Facet Engineering Process ........................................................................... 36

4.2.2. Faceted Approach for Source Code Classification ....................................... 39

4.3. FACETED APPROACH APPLIED IN SEARCH ENGINES .................................... 39

4.4. SEARCH AND RETRIEVAL OF SOURCE CODE USING THE FACETED APPROACH

4.4.1. Requirements .................................................................................................. 42

4.4.1.1. Functional Requirements ....................................................................... 43

4.4.1.2. Non-Functional Requirements ............................................................... 44

4.4.2. B.A.R.T System ............................................................................................ 47

4.4.2.1. B.A.R.T Architecture ............................................................................. 47

4.4.3. Facet Classifier Engine ................................................................................. 49

4.4.3.1. Architecture ............................................................................................ 49

4.4.3.2. Implementation ....................................................................................... 56

4.4.4. B.A.R.T Search Engine Supporting Facet Classification ............................... 59

4.4.5. Implementation .............................................................................................. 60

4.5. DOMAIN FACET SUPPORTING TOOL (DoFaST) .............................................. 60

4.5.1. Main requirements ......................................................................................... 61

4.5.2. Architecture .................................................................................................. 62
5. THE EXPERIMENTAL STUDY .............................................................................. 68

5.1. INTRODUCTION ................................................................................ 68

5.2. FACET CLASSIFIER ENGINE EVALUATION ............................................ 69

5.2.1. Goal definition ................................................................................ 69

5.2.2. Planning ........................................................................................... 69

5.2.2.1. Context selection ........................................................................ 69

5.2.2.2. Hypothesis formulation ............................................................... 70

5.2.2.3. Variables selection ...................................................................... 71

5.2.2.4. Selection of subjects .................................................................... 71

5.2.2.5. Experiment design ....................................................................... 71

5.2.2.6. Instrumentation .......................................................................... 71

5.2.2.7. Validity evaluation ....................................................................... 72

5.2.3. Operation .......................................................................................... 72

5.2.3.1. Preparation .................................................................................. 72

5.2.3.2. Execution ..................................................................................... 73

5.2.3.3. Data validation ............................................................................ 73

5.2.4. Analysis and interpretation ............................................................... 73

5.2.5. Summary and conclusions ............................................................... 74

5.3. B.A.R.T SEARCH ENGINE USING FACET CLASSIFICATION EVALUATION ...... 75

5.3.1. Goal definition ................................................................................ 75

5.3.2. Planning ........................................................................................... 75

5.3.2.1. Context selection ........................................................................ 75

5.3.2.2. Hypothesis formulation ............................................................... 76

5.3.2.3. Variables selection ...................................................................... 78

5.3.2.4. Selection of subjects .................................................................... 78

5.3.2.5. Experiment design ....................................................................... 78

5.3.2.6. Instrumentation .......................................................................... 79

5.3.2.7. Validity evaluation ....................................................................... 79

5.3.3. Operation .......................................................................................... 80

5.3.3.1. Preparation .................................................................................. 80

5.3.3.2. Execution ..................................................................................... 80
List of Figures

Figure 1. The RiSE framework ................................................................. 4
Figure 2. B.A.R.T Architecture ............................................................... 48
Figure 3. Facet Classifier Engine architecture ....................................... 50
Figure 4. Class diagram UML Excerpt of the Facet description API ....... 51
Figure 5. Formatted data extracted from the source code ...................... 54
Figure 6. Link source binding problem .................................................. 55
Figure 7. Classification process result representation ............................. 56
Figure 8. Example of facet scheme for Graphics domain ....................... 56
Figure 9. Classification process flow ...................................................... 58
Figure 10. Query reformulation ............................................................. 59
Figure 11. DoFaST architecture and its modules dependencies ............... 62
Figure 12. DoFaST Classification File structure ..................................... 63
Figure 13. DoFast tool main screen ...................................................... 65
Figure 14. Classification Summary viewer ............................................. 65
Figure 15. Term detail viewer .............................................................. 66
Figure 16. Classification result of the domain facet ............................... 73
Figure 17. The evolution of B.A.R.T search mechanisms ...................... 85
List of Tables

Table 1. Information Retrieval x Data Retrieval [Durão, 2008] ............... 20
Table 2. Requirements summary ..................................................................... 45
Table 3. Java source code parsers' evaluation. .............................................. 52
Table 4. Domain terms used and its related Java package in API ................. 57
Table 5. Achieved Recall Rates. ....................................................................... 81
Table 6. Achieved Precision Rates. ................................................................. 82
Table 7. Achieved F-Measure Rates. ................................................................. 82
Table 8. Achieved Results. ............................................................................... 83
Table 9. Queries utilized in the evaluation ........................................................ 105
List of Acronyms

API: APPLICATION PROGRAMMING INTERFACE .................................................. 6
B.A.R.T: BASIC ASSET RETRIEVAL TOOL .......................................................... 5
CVS: CONCURRENT VERSION SYSTEM .............................................................. 27
DoFaST: DOMAIN FACET SUPPORTING TOOL ............................................... 7
GUI: GRAPHICAL USER INTERFACE ................................................................. 13
HFC: HIERARCHICAL FACET CATEGORIES...................................................... 33
RiSE: REUSE IN SOFTWARE ENGINEERING .................................................. 4
SVM: SUPPORT VECTOR MACHINES ............................................................... 6
XML: EXTENSIBLE MARKUP LANGUAGE ......................................................... 54
1

Introduction

Software Reuse is considered one of the most important approaches used to provide software development in a more efficient way, by engineering knowledge or artifacts from existing systems in order to build new ones from scratch systematically [Morisio et al., 2002]. Nevertheless, software developers are unable, or unwilling, to reuse if they do not know the existence of reusable assets or do not know how to find, understand, and use them [Ye and Fischer 2002]. Thus, a fundamental principle for reusing software assets is to provide ways for accessing them.

Likewise, Pietro-Diaz [Prieto-Díaz, 1991] affirmed: “to reuse an asset, first you have to find-it”. In this direction, tools such as reuse repositories and information retrieval tools play an important role on easing the search and retrieval of the reusable assets.

This chapter presents an overview about the focus of this dissertation. The motivation is described in Section 1.1 and a contextualization of the applicable area of this work is discussed in Section 1.2. A brief overview of the proposed solution is presented in Section 1.3. Section 1.4 discusses aspects that out of scope of this work. Section 1.5 presents the main contributions of this proposal and, finally, Section 1.6 outlines the structure of the remainder of this dissertation.

1.1. Motivation

The evolution of Computer Science and Information Technology has driven many organizations, in order to become more competitive in the modern society, to the adoption of computational systems. To achieve new requirements
and provide more powerful systems, software organizations are investing in more qualified professionals and constantly updating their systems which increase the cost and production time.

However, such systems started to become more complex and this requires more qualified people constantly updated with the new trends. In this scenario, it is predictable to imagine how similar applications have been developed independently, in many organizations and places and in different times in history, without sharing or reuse previously knowledge.

In this scenario, Software Reuse is considered one of the most relevant mechanisms for performing software development more efficiently [McIlroy, 1968], and emerged with the proposal of reducing rework, based on the premise that, if accurately applied, assets previously built might be reused instead of being developed from scratch. Furthermore, reusing software assets might speed up the time-to-market with the improvement of the productivity besides reducing the costs and increase the quality of the product [Devanbu et al., 1991], [Lim, 1994], [Basili et al., 1996], [Frakes and Succi, 2001].

In attempting of a more effective reuse, the adoption of a systematic reuse program, which comprehends investment on supporting tools such as source code search tools, remains necessary [Rine, 1997]. According to [Morisio et al., 2002], these tools aid the software developers to search, retrieve and reuse source code from several repositories avoiding the writing of new codes since the same solution may have been implemented by someone else.

Garcia et al. [Garcia et al., 2006b] proposed B.A.R.T, a robust search engine tool whose main goal is to provide mechanisms for search and retrieval of software components. After that initial idea, the project had experimented new trends related to this such as active search [Mascena, 2006], context [Santos et al., 2006], folksonomy [Vanderlei et al., 2007] and semantic [Durão et al., 2008].

Others efforts on development of tools focusing on additional categories such as reuse repositories [Burégio, 2006], software reengineering [Brito, 2007], and domain analysis [Lisboa et al., 2007] were reported in literature, and they also attempt to offering the reuse benefits.

This dissertation concentrates its efforts on information retrieval tools dedicated to source code retrieval and is a complementary solution for the
Chapter 1 - Introduction

previous approaches [Garcia et al., 2006b], [Mascena, 2006], [Santos et al., 2006], [Vanderlei et al., 2007], [Durão et al., 2008], since search mechanisms can be merged to empower their benefits.

1.2. Problem Statement

In attempting of contributing for reuse activity, keyword-based search engines are utilized in software development as a mechanism to access reusable source code. In a nutshell, traditional keyword-based search engines provide an user interface to formulate queries in order to retrieve an item of interest based on occurrences of the string pattern (or query) in source code. Since, in general, there is no query support to assist users to express their needs by means of natural language processing or knowledge exploration about the repository content, this approach has presented problems related the low relevance of the returned results. It occurs because a single keyword may not represent the desired functionality, whether the source code might not be well analyzed or the formulated query does not represent correctly the user expectations [Ye and Fischer, 2002].

Several techniques, such as folksonomy [Vanderlei et al., 2007] and semantics [Durão, 2008] are being applied in order to reduce the problem of find relevant, reusable source code to promote a more efficient reuse. In the same direction, facet-based approaches attempts to classify source codes with terms that best describe it [Pietro-Díaz, 1991], reducing the semantic conceptual gap (More detail in Section 3.4) and increasing the relevance of the returned reusable source code.

In a previous work, the RiSE group implemented the initial facets ideas in a search engine [Garcia et al., 2006b], which had interesting results when working in cooperation with keyword search approach. However, it showed that generic facets itself are not efficient to retrieve relevant source codes and that manual classification demands a lot of effort and time which makes difficult its practical use.

This problem motivated this work to revisit the faceted approach from a different perspective, by applying new techniques in favor of enhance the facets
benefits and to make it easier for practical use, as also supporting the users to express their needs and to narrow the search scope increasing the precision.

1.3. Overview of the Proposed Solution

This work is part of a set of reuse initiatives promoted by the Reuse in Software Engineering (RiSE) Group [Almeida et al., 2004]. As depicted in Figure 1, the RiSE framework is composed by initiatives in areas such as reuse processes involving domain engineering [Almeida, 2007], component certification [Alvaro et al., 2006], and repository system [Burégio, 2006]. In addition, outstanding tools also are part of the RiSE Software Reuse environment such as the Maracatu search engine [Garcia et al., 2006b] which was enhanced with the facet-based, folksonomy mechanisms [Vanderlei et al., 2007], the Admire Environment [Mascena, 2006], context-aware ideas [Santos et al., 2006], and ToolDAy, a Domain Analysis Tool [Lisboa et al., 2007].

Figure 1. The RiSE framework.

These initiatives work in cooperation to provide an integrated set of mechanisms to apply a systematic and scalable reuse solution. This work is placed on the Software Reuse Environment module and the main purpose of

http://www.rise.com.br/research
Chapter 1 - Introduction

This dissertation on the RiSE project is to provide enhancements in the B.A.R.T Search Engine by the use of the faceted approach to analyze and classify the source code assets, increasing the precision and relevance of the suggested source code in the search results.

Source code search engines are considered effective tools for promoting reuse by avoiding development of new ones from scratch and diffuse knowledge extracted from parts of the code since there is a large amount of code resided over legacy systems in organizations or spread in repositories on the Internet. Many of these are keyword-based and its efficiency relies on the quality and relevance of the query results. Nevertheless, software developers are unable, or unwilling, to reuse if they do not know the existence of reusable assets or do not know how to find, understand, and use them [Ye and Fischer, 2002].

In order to mitigate the problem related to the location of relevant source codes, this work proposes an extension of the B.A.R.T Search Engine supporting classification mechanisms based on the faceted approach. In addition, a proposition of supporting tool to aid the domain specialist on processes of analysis and classification of source code is presented.

1.4. Out of Scope

Since the main focus of the proposed solution is presenting the benefits of applying the faceted approach as a classification mechanism, some aspects were considered out of scope. Even though the provided functionalities are based on well-founded aspects of quality and performance, future enhancements to answer more efficiently its purpose are not discarded. The issues not directly addressed by this work are listed in the following:

- **Facet classification for general software asset types**: The initial proposal of this work aims on source code assets specifically. In further work, others types of assets such as documentation and component libraries may be contemplated;

- **Source code retrieval**: Although one important quality attribute is extensibility, including any programming language support, the initial implementation only contemplates Java source files;
Chapter 1 - Introduction

- **Text categorization and ranking algorithms**: In this initial version, a simple classification and ranking algorithms were implemented. Other text categorization algorithms such as naïve-bayes [Lam et al., 1999], and support vector machines (SVM) [Lam et al., 1999], are not dealt.

Although this initial implementation meets academical expectations, it also presents some limitations. For massive use, quality attributes such as modularization, readability and reusability must be refined and new supporting functions specially related to system setup as also as domain experts support must also be implemented for commercial version the of B.A.R.T Search Engine.

1.5. **Statements of the Contributions**

As result of the work presented in this dissertation, the list of contributions is enumerated as follows:

- Provide mechanisms to reduce the *semantic conceptual gap* (more detail in subsection 3.4) and increase the relevance of the returned reusable source code from the search engines;
- Presents a structure to represent and describes the facets and its terms;
- Creation of a Java source code classification engine using the faceted approach;
- Extension of the B.A.R.T system, adding supporting to facets search functions with low coupling without decreasing the system performance;
- A flexible facet description API, to support the conception of new facets classifications;
- A tool for supporting the domain expert on performing identification of new facets and, analysis and classification of source code;
- Implementation and evaluation of the faceted approach on classifying source code and its benefits applied in search and retrieval of reusable source code in an experimental study.
1.6. Organization of the Dissertation

The remainder of this dissertation is organized as follows:

**Chapter 2** contains a comprehensive revision of the *Software Reuse* field, presenting the main concepts, its benefits and major concerns about its use and a digression of source code reuse;

**Chapter 3** reviews the *Information Retrieval* field with an emphasis to the instantiation of the information retrieval techniques applied on software reuse;

**Chapter 4** presents the main proposition of this dissertation, discussing the concepts concerning the faceted approach. The extension of the *B.A.R.T Search Engine* on applying the faceted approach is detailed. In addition, is also presented *DoFaST*, a tool for supporting the Domain Expert on performing their tasks;

**Chapter 5** describes the experimentation applied on the Facet Classifier Engine and the *B.A.R.T Search Engine supporting Facets*, describing the methodologies applied, the description of goals and metrics used and, the achieved results; and

**Chapter 6** concludes this dissertation, summarizing the achieved goals and discussing related work. Finally, future enhancements to the proposed solution and some concluding remarks are also presented.
2. Software Reuse: An Overview

This chapter presents a review of the software reuse concepts required to establish the foundations of this approach. The discussion is organized as follows: Section 2 introduces the software reuse field from its beginning to the current time. The motivation and also the benefits and concerns regarding software reuse are presented in Section 2.2. Section 2.3 discusses the reusable software assets focusing on source code and means of its access. Finally, a summary of the main software reuse concepts are described in Section 2.4.

2.1. Introduction

The main ideas of reuse were introduced in 1968 by McIlroy [McIlroy, 1968] at the NATO Software Engineering Conference, in a decade of software crisis. McIlroy affirmed that reusable software components were necessary to maintain and sustain the software industry during the crisis. One of the initial approaches proposed, was the idea of constructing software from previously pieces of code already built.

Since McIlroy [McIlroy, 1968], a set of definitions about software reuse has been presented in the literature. Krueger [Krueger, 1992] affirmed that software reuse is the process of creating software systems from existing ones rather than building them from scratch. According to Frakes & Isoda [Frakes and Isoda, 1994] software reuse is the use of engineering knowledge or artifacts from existing systems to build
new ones. In [Morisio et al., 2002], Morizio et al. describe reuse as an umbrella concept, encompassing a variety of approaches and situations and its benefits must be quantified and empirically assessed.

In this direction, software reuse is considered one of the most important strategies used to provide software development in a more efficient way [Morisio et al., 2002]. Industry and academy are investing in this field, since is a key factor to improve quality and productivity working with reduction costs. Although many research reports highlighted concerns about the current state of reuse in industry since, in most of cases, reuse is still practiced in an ad hoc way, which is a reuse without any defined process [Frakes et al., 1998], [Almeida et al., 2005b].

Initiatives in areas such as reuse processes involving domain engineering [Almeida, 2007], component certification [Alvaro et al., 2006], repository system [Burégio, 2006], and supporting tools [Garcia et al., 2006a] are been developed in order to mitigate the problems identified [Morisio et al., 2002], [Almeida et al., 2004]. Most of them drive the reuse benefits in software development based on three major aspects: quality, productivity and cost reduction. These aspects are also considered key factors for achieving the level of success expected by the organizations [Lim, 1994], [Basili et al., 1996], [Frakes and Succi, 2001].

Next section discusses benefits and challenges on adoption of reuse in the software development process.

### 2.2. Software Reuse Benefits and Uncertainties

This section describes the benefits promote by a systematic and effective reuse and the concerns and impediments faced on adoption of reuse in organizations.

#### 2.2.1. Benefits

In a nutshell, the benefits provided by software reuse relies on augmenting productivity and reducing costs. The former is achieved by avoiding implement new assets from scratch through full or partial reuse of assets and, consequently, decreasing the development time and improving the organization
competitiveness. The latter is obtained due to effort reduction from the development perspective to quality aspects. Since the reused assets has probably well-tested and acceptable quality assurance due to previous usages, the costs relatives to testing and software assets correction have significant cost savings and, furthermore, redundant work concerns are decreased [Basili et al., 1996], [Sametinger, 1997], [Lim, 1998].

2.2.2. Uncertainties and Concerns

The process of component modification is considered stable when the component difference between the new and old requirements is proportional to the effort to make the implementation change. The stability of a component directly affects the maintenance effort of the software component.

In this sense, some uncertainties may be addressed to these principles that may come from the following sources [Chen and Rine, 1999]:

- **System and user requirements:** In general, the requirements are in continuously evolution along the system development cycle, since the software project must start as early as possible;

- **Hardware:** In some cases, the hardware components may not be available at the time of software development; and

- **Application environment:** Since the application environment may change after the deployment, it may causes some impediments respecting to the software maintenance.

Along the course of software development, reuse, and deployment activities, these uncertainties lead to increment of maintenance costs [Nosek and Palvia, 1990]. In order to obtain significant cost reduction some principles of software reuse are *information hiding, abstraction, encapsulation, portability,* and *modularity* [Chen and Rine, 1999]. The main purpose of these principles is to provide to the components a stable interface and easiness to modify.

Moreover, despite the benefits promoted by software reuse, some challenges are faced on its adoption in organization, since reuse is not a general purpose approach, and, in some cases, may not guarantee significant
improvements in quality and productivity [Schmidt, 1999]. Some of the software reuse impediments are detailed as follows:

- **Organizational impediments**: For a systematic use of reusable software components, the organization must consider the structure required to attend development, deployment and support concerns such as the following: while development teams are consuming reusable assets, new requirements are requested for components; or situations where the demand for reusable assets constantly increases among the projects in the organization [Sametinger, 1997];

- **Management impediments**: These concerns are one of the most important factors to success in adoption of an effective reuse [Frakes and Isoda, 1994]. Initiatives to promote reuse such as training programs and rewards must be encouraged by the managers. In addition, managers should to be in aware of the initial investments and be involved with reuse activity since it generates cost and demand investment to the whole organization [Sametinger, 1997]; and

- **Economic impediments**: Reuse, if correctly applied, can obtain expressive cost savings to organizations. Although, the adoption of reuse activities in organizations demands investments involving training, incentives, processes and infrastructure [Poulin, 1997]. Due to the return on investments does not necessarily be in a short period, some organizations avoid investments in the adoption of a systematic reuse. Moreover, the costs to produce reusable software components is higher than components built for a single use and also requires more process overhead and other concerns such as higher quality and documentation [Poulin, 1997].

Further these concerns, some other aspects must be considered such as **Not Invented Here (NIH) syndrome**, which some organizations are reluctant in use third-party components, even the ones created from other development teams in the same organization. Therefore, according to Ye and Fischer [Ye and Fischer, 2002] the major problem is that the developers are unable, or unwilling, to reuse if they do not know the existence of reusable components or do not know how to locate, understand, and use them.
Chapter 2 –Software Reuse: An Overview

2.3. Reusable Software Assets

Regarding reusable software assets some important definitions are necessary. In a research presented by Voth [Voth, 2004], a reusable software asset is anything of value to a development organization since one of the issues surrounding reuse is that it can happen on multiple levels. According to [Ezran et al., 2002], software assets encapsulate business knowledge and are of high value to a company and, it is composed of a collection of related software products that may be used among applications.

A reusable asset provides a solution to a problem for a given context and may have a variability point, which is a location in the asset that may have a value provided or customized by the asset consumer. In addition, artifacts are any work product from the software development lifecycle, such as requirements documents, models, source code files, deployment descriptors, test cases and scripts [OMG, 2002].

One of the emerging reuse paradigms being considered for the new methodology is Domain Engineering, which attempts to improve the idea of designing software maximizing reusability by analyzing the entire process of software development in a new perspective [Kean, 98], [Czarnecki et al., 2000]. The main idea of this purpose is to stimulate the creation of general libraries that solve an entire class of problems instead of writing code to solve a single problem. These classes of problems are called horizontal domains and are characterized by commonalities within the problems that are being considered.

Another advantage of Domain Engineering is the dissociation of the design of the general libraries (engineering for reuse) from the creation of programs that use these general libraries to solve a particular problem (engineering with reuse). The resulting solution is referred to as a vertical domain. The vertical domain is built using the general horizontal domains, configuration knowledge and set of rules that will be used to combine the horizontal domains in a way that is correct given the needs and limitations of the problem [Kean, 98] [Czarnecki et al., 2000].

According to Almeida [Almeida, 2007], the software assets can be classified in:
Chapter 2 – Software Reuse: An Overview

1. **Vertical assets**: Specific to an application domain, for example, financial object models, algorithms, frameworks;

2. **Horizontal assets**: Easier to identify and reuse because they represent recurrent architectural elements and can be reused independently of the application domain, demanding that the application architectural choices must be compatible with the asset. Examples of them include GUI objects, database access libraries, authentication service, and network communication libraries.

Further these two types, assets may have different sizes and granularities such as a function or a procedure, a class, a group of classes, a framework and an application or a product. A reusable asset is potentially made up of many cycle products including: requirements and architecture definition, analysis model, design models and code, test programs, test scenarios and test reports [Almeida, 2007].

In spite of software component reuse, the following major strategies can be defined [Ravichandran and Rothenberger, 2003]:

1. **White-box reuse**: It allows developers to customize the code according to their needs. This maximizes reuse potential, but it is also a source of reuse problems due to modifications applied in the code might not be documented and versioned;

2. **Black-box reuse**: It reduces the chances of introduction of modification problems, but reduces the reuse potential due to low level of customization; and

3. **COTS (Components Of The Shelf)**: It can increase the reuse rate, as the developers can search from a larger set of components and thereby are more likely to find components fitting their requirements.

According to Ravichandran and Rothenberger [Ravichandran and Rothenberger, 2003], the COTS strategies may be the most promising approach since makes software reuse a reality and advances software development to a robust industrial process. Although, the current industrial scenario is still not ready to maximize the benefits using this strategy, since it requires more investments on education and the companies achieve maturity level on reuse.
Chapter 2 – Software Reuse: An Overview

Based on the reuse practice adopted, reuse may happen relied on individual effort, where developers recognize reuse opportunities informally or it may be planned for by building and maintaining a software repository that supports the maintenance of reusable artifacts. In order to engage the reuse activity at a level closer to the organizational reuse setting, [Ravichandran and Rothenberger, 2003] focus on the systematic reuse of source code. These reuse practices may vary by extent of organizational support, integration in the overall development process, and the employed techniques. Source code, in particular, may have high degree of reusability since the implemented functionalities may be reused by other applications.

In the next section, a discussion about source code importance in reuse practices is detailed, since it is one of the fundaments of this work.

2.3.1. Source Code Reuse

The practice of large-scale source code reuse may provide potential savings in effort and improvements in quality and increase productivity. Highly reused source code tends to have better quality and requires less effort to maintain. Furthermore, if highly reused code and projects have attributes that distinguish them from the low reuse projects, some of these qualities may guide new projects that strive for their code to be reused more widely [Mokus, 2007].

In addition, according to Mili et al. [Mili et al., 1995], the potential reuse range may vary from 15 to 85 percent. Such activities as testing, assets quality assurance and ease of assets accessibility, if correctly applied, may obtain cost and production time reduction, and improvement of quality of reusable assets, thus, provides competitive advantage in terms of costs and time-to-market. Since the historical appealing of the importance of source code in the development cycle [McIlroy, 1968], source code has gained importance in reuse activity, leading the reuse industry put more effort on development of tools such as IDEs, search engines [Garcia et. al, 2006a] and reuse repositories [Burégio, 2006] to have more benefits from code reuse.

Some studies [Frakes and Fox, 1995], [Schmidt, 1999], [Glass, 2002] have discussed interesting concerns over source code reuse: NIH (Not Invented Here) syndrome, programming languages paradigms and modification costs.
Frakes and Fox [Frakes and Fox, 1995] presented a survey about software reuse using data collected from organizations in the U.S and Europe, with participation of software engineers, managers, educators and others in the software development and research community about their attitudes, beliefs, and practices in reusing code and other lifecycle objects. The obtained results unveiled some myths respecting software reuse where some of them related to source code reuse are detailed as follows:

- **Not Invented Here Syndrome (NIH):** The research demonstrated that users prefer reuse rather than coding from the scratch, regarding that the code is well-written and has a acceptable quality assurance;

- **Paradigms of Programming Languages:** The results showed that even some paradigms languages suitable to apply reuse practices, OO languages for instance, there is no directly relation between reuse and paradigms; and

- **Source Code Modification:** It was verified if modification of reused code is particularly error-prone, it is more efficient and effective to rewrite it from scratch. Nevertheless, this strategy also implies in some problems [Glass, 2002] thereby the complexity in maintaining existing software and the effort to understand the existing solution and update and maintain associated documentations. Schmidt [Schmidt, 1999] affirms that is a continuous problem due to heterogeneity of hardware architectures, several amounts of available operational systems and network platforms.

There are a plenty of mechanisms based on source code manipulation focused on improving and increasing the software development productivity such as code generators [Richards et al., 2008], code scavenging [Burd and Munro, 1997], and reengineering applications [Dar-El, 1997]. However, this does not necessarily mean systematic reuse since it is, in general, informal and opportunistic way. For effective reuse, the reuse practices must be applied in a systematic way, supported by management commitment and reuse education, in order to promote the reuse adoption throughout organization.
2.3.2. Reusable Software Assets Accessibility

One of impediments on using reusable software assets is how to access them. According to [Prieto-Díaz, 1991], [Ye and Fischer, 2002], software developers are unable, or unwilling, to reuse if they do not know the existence of reusable components or do not know how to locate, understand, and use them; which emphasizes the importance of mechanisms to find and access the assets.

Repository tools for reuse and information retrieval tools are practical examples to achieve these requirements. More detail about each one is presented as follows:

- **Reuse repositories**: Such repositories have suitable mechanisms in favor of ease the reuse practice during the software development. Most of them are component-based and provide a catalogue of software components categorized by according to a specified scheme. In general, provide resources such as searching, browsing, versioning, certification, status reporting, metrics extraction, notification and primarily asset maintained besides provide assets life cycle management strategies [Burégio, 2006]; and

- **Information retrieval systems**: They provide effective means for accessing, sharing and retrieval knowledge from assets more efficient. More details about information retrieval are discussed in next Chapter.

2.4. Chapter Summary

This chapter presented the main concepts of software reuse, discussing its origins, motivation and benefits, concerns regarding software reuse, types of reusable source code, and mean for accessing them the in order to guide companies towards systematic software reuse.

The aspects shown insights that obtain a complete reuse infrastructure involves high initial costs as it includes well-established reuse processes, management commitment, education initiatives and tooling for supporting the reuse activity. Some considerations about technical and non-technical impediments for systematic reuse adoption also were outlined. In addition, it
Chapter 2 – Software Reuse: An Overview

was presented an overview of reusable software assets, its variances with focus on source code. Finally, it was also discussed myths of source code reuse and the use of tools such as repository systems and information retrieval tools.

In the next chapter, a review on the information retrieval field with emphasis to mechanisms and technologies useful for promoting the reuse activity will be presented.
Information Retrieval mechanisms are considered an effective mean for promoting software reuse since the amount of information available in the organizations or distributed on the Internet are constantly increasing. In the context of software organizations, the information is generally found in software assets used during the software life cycle such as documents, source code and test cases.

In this sense, information retrieval systems play a very important role on providing mechanisms to ease the access to the available assets. This chapter discusses the information retrieval field outlining applications focused on software reuse and is organized as follows: Section 3.1 presents an introduction for the information retrieval field, Section 3.2 outlines a comparison between information retrieval and data retrieval. In Section 3.3 are discussed concerns about relevant information. Section 3.4 introduces technical support for information retrieval systems and search retrieval issues focused on source code. Section 3.5 discusses some work in the area, and, finally, Section 3.6 summarizes the review about information retrieval and its application to the software reuse context.

3.1. Introduction

Software retrieval mechanisms were initially created to supply the human limitations concerning storage and retrieval [Baeza-Yates and Ribeiro-Neto,
Chapter 3 – Information Retrieval

For effective software retrieval system, it is necessary the supporting of technical mechanisms that allow accessing the data with a reasonable performance. Indexing is an example of technique used to enhance the data access performance, being widely applied in systems and composed by complex algorithms [Clarke and Cormack, 1995].

Besides performance concerns, the quality of the retrieved information must also be considered since it credits the information retrieval systems and responds for their success and popularity in the trade market [Baeza-Yates and Ribeiro-Neto, 1999].

Among the concepts related to information and retrieval, two demands a special attention since they are commonly mixed: information retrieval and data retrieval [Rijsbergen, 1979]. In the next Section, a discussion about each concept and its characteristics are detailed.

### 3.2. Information Retrieval and Data Retrieval

The retrieval activity is divided between two models: information retrieval (IR) and data retrieval (DR). According to Rijsbergen [Rijsbergen, 1979], data retrieval, in general, search for exact matches and information retrieval focus on finding similar matches satisfying the query and then, suggests the most relevant ones according to a predefined criteria. Data retrieval is more restrict and also more susceptible to errors, since a single erroneous returned item among a thousand retrieved means total failure. On the other hand, information retrieval may provide more imprecise results, but does not comprise the search capability at all. It occurs due to usually deals with unstructured natural language texts that can be semantically ambiguous while data retrieval system are more suitable to well-defined data structure such as relational database.

Table 1 summarizes a comparison between both modes of retrieval.
Table 1. Information Retrieval x Data Retrieval [Durão, 2008].

<table>
<thead>
<tr>
<th>Characteristic/Methods</th>
<th>IR</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>High Error Sensitiveness</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Semantic Treatment</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Search in unstructured data</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Deductive Inference</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Inductive Inference</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Controlled Syntax Query</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Natural Language Query</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Commonly Academic Development</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Outstanding Commercial Products</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

The information retrieval systems to be more effective have mechanisms to “interpret” the contents of the information items (documents) and rank them according to a degree of relevance to the user query. This process comprehends extracting syntactic and semantic information of the document and verifies if meets the user expectations. Further the problems related to the extracting information process, Baeza-Yates and Ribeiro-Neto [Baeza-Yates and Ribeiro-Neto, 1999] outlined the difficulty of how to decide relevance. Rijsbergen [Rijsbergen, 1979] stated that the primary goal of information retrieval is to retrieve all relevant documents and few non-relevant as possible; enforcing the notion of relevance is at the center of information retrieval.

In general, data retrieval systems are based on deductive inference, where a conclusion is logically inferred from certain premises. On the other hand, information retrieval uses inductive inference, which a conclusion is inferred from multiple observations. In addition, data retrieval systems are defined as deterministic and information retrieval as probabilistic thus, is common to find information retrieval systems using probabilistic models [Sebastiani, 2002] to infer results that satisfy the entire search process.

Regarding query language, data retrieval is more restrict, having a controlled syntax, while information retrieval systems favor the use of natural
Chapter 3 – Information Retrieval  

languages. However, there are some exceptions, such as keyword query used in traditional web search engines. Moreover, the query used for data retrieval systems is more detailed when expressing what is expected while information retrieval it is invariably incomplete, which evidence the sensitiveness to errors in data retrieval systems [Durão, 2008].

In this sense, information retrieval is the most suitable approach for the proposition of this dissertation proposes since:

- It does not demand a strict syntax on query formulation;
- The search is performed on a collection of semi-structured documents, with a controlled vocabulary; and
- The classification criteria are based on probabilistic models.

The next section enforces these premises and defines what is considered relevant information.

### 3.3. Relevant Information

In a perfect retrieval, an information system search analyses all documents, retains the relevant ones and discarding all the others, which enforces that a great challenge is to distinguish the relevant documents from the non-relevant ones, further it’s to be a key factor for the success of an information retrieval tool which intends to reach an outstanding position in the market [Baeza-Yates and Ribeiro-Neto, 1999].

In an ideal scenario, an information retrieval system may extract the information from the text (both syntactic and semantic) and uses it to decide which documents are relevant to a particular request. Some efforts in order to enhance information retrieval systems are supported by complementary mechanisms such as artificial intelligence disciplines, by providing techniques such as automatic text categorization [Lam et al., 1999], semantic inference and ontology reasoning [Bruijn, 2003], and machine learning [Sebastiani, 2002].

The faceted approach introduced by Pietro-Díaz [Pietro-Díaz, 1991] proposes the use of a facet scheme to enhance the descriptive characteristics of the documents. All of these techniques try to mitigate information retrieval concerns such as query formulation and code comprehension.
3.4. Technical Support for Information Retrieval Systems

One of the challenges to be overcome in order to meet the user expectations on retrieving information is to diminish the semantic conceptual gap. This concept is related to the difficulties in mapping the user needs in a formal representation, since the formulation of a contextual knowledge in a powerful language (e.g. natural language) may not be properly represented in a computational representation based on a formal language (e.g. programming language) [Ye and Fischer, 2002]. In addition, inappropriate keywords increase the probability of unexpected results that mismatch the user intention [Baeza-Yates and Ribeiro-Neto, 1999].

Many initiatives to reduce the semantic conceptual gap have been proposed. The next section presents some of these based on query formulation, as well as text comprehension methods.

3.4.1. Techniques for Query Formulation Support

The techniques applied in query formulation, in general, improve the original one by adding relevant information which might be useful to achieve the user expectations. Devanbu [Devanbu et al., 1991], presented an information retrieval system that utilizes hierarchical categories to identify components. In [Prieto-Díaz, 1991] was applied the concept of multiple facets which encapsulate aspects, properties and characteristics of a software component.

Belkin and Croft [Belkin and Croft, 1992] applied information filtering to remove unnecessary information from the result set. The query by reformulation mechanism was adopted by Henninger [Henninger, 1993] in which the original query is expanded by user to match additional software components. Components constraints, such as signature matching and formal specification, were used by some reuse repository systems for component retrieval [Podgurski and Pierce, 1993].

Context-based retrieval was used to make the returned components consider class syntax structures further user coding [Ye and Fischer, 2002].
Chapter 3 – Information Retrieval

Sugumaran and Storey [Sugumaran and Storey, 2003] applied domain ontology in source code search engines to correlate the domain-specific concepts with indexed classes in the repository. In [Calado and Ribeiro-Neto, 2003] was proposed the use of approximate queries formulation to rank the results according to the user expectations. In [Vanderlei et al. 2007], the folksonomy mechanism was introduced in a source code search engine to assist users to search codes through a tag cloud. Hotho et al. [Hotho et al., 2006] presented a new perspective of how apply folksonomy, called FolkRank, that exploits the structure of the folksonomy to find out communities and ranking the search results.

In general, these initiatives provide benefits and contribute to a more efficient search and retrieval of components. Nevertheless, some of them are for private use [Pietro-Díaz, 1991] and the details are not published. In the direction of increase the precision, the research presented by Belkin and Croft [Belkin and Croft, 1992] and by Sugumaran and Storey [Sugumaran and Storey, 2003] aim pruning unnecessary information from the result set. Ye and Fisher [Ye and Fischer, 2002] presents mechanisms to consider specific information related to the source code context in order to increase the precision of the returned results.

In spite of supporting the user to express his needs, the research presented in [Henninger, 1993], [Calado and Ribeiro-Neto, 2003] showed efforts in query reformulation field. In order to aid the user identify the context of his needs, Hotho et al. [Hotho et al., 2006] and Vanderley et al. [Vanderlei et al. 2007] use the folksonomy approach to introduce meta-information to the components by using tags. In the direction of classify the components, for example source code, and aiding the user narrow the search scope, some research were presented in [Devanbu et al., 1991], [Prieto-Díaz, 1991].

### 3.4.2. Classification Schemes for Component Retrieval Support

As presented in [Ugurel et al., 2002], classification schemes may be combined with query formulation techniques in order to augment the chances of successful retrieval. Classification schemes, also known as categorization
Chapter 3 – Information Retrieval

schemes, may be applied to automatically extract information about the asset content and use this to increase its decision in the retrieval process. Although manual categorization is feasible, it requires significant effort and cost to maintain and is discouraged for dynamic systems.

Although automated classification techniques are applied since the early '60s, they were exploited more effectively with the internet booming in '90s, which promoted the booming of generation and availability of information. From that time, automatic categorization began to be seen as the meeting point of machine learning and information retrieval [Durão, 2008].

In general, typical text classifiers use rules or hypothesis extracted from a knowledge base to analyze document characteristics and classify them in predefined categories. This meta-information is useful to compose the index structure providing a higher degree of semantics. The benefits of this approach provide considerable savings in terms of expert manpower and domain independence.

Some success cases reported in literature relied on code comprehension and classification. In [Ugurel et al., 2002], vector machines were applied for source code classification in a two-phase process - programming language classification followed by topic classification. Lam et al. [Lam et al., 1999] proposed a categorization approach, known as retrieval feedback, based on the combination of machine learning and text retrieval techniques where the queries performed are refined without user intervention.

In [Pietro-Díaz, 1991] was presented interesting results of classification of source code using faceted approach. A faceted scheme used in this approach may have several facets and each facet may have several terms. Faceted classification provides features that improve search and retrieval and also enhance the potential reuser's selection process and contribute to the development of a standard vocabulary for software attributes. This dissertation uses this approach to classify the source code. In next chapter, details of this approach are discussed.

In the same direction, Durão [Durão, 2008] presented an approach of source code classification using semantics and ontology based on Naive-Bayes probabilistic models for classifying the code.
3.4.3. Source Code Retrieval for Reuse Activity

Most research on information retrieval systems has centered its effort in methods for effectively retrieve relevant documents. However, according to Henninger and Belkin [Henninger and Belkin, 1996], little attention had been given to software components retrieval. Specifically about source code, there are considerable concerns such as arbitrary rules of grammar that are different from natural language as well as semantic issues attributed to its functionalities.

Source code have a controlled vocabulary and are unambiguous to the compiler and have exact syntactic structures [Ugurel et al., 2002] in opposite of documents written in natural languages. Therefore, few code searchers have embedded compiler to help in query processing. Furthermore, the syntax itself does not respond for the semantic of a particular piece of code, since the semantic is spread over functionalities covered by the class through its methods.

The lack of semantic analysis is one of the challenges faced in source code location process and is evidenced in [Ye and Fischer, 2002]. Their work shown, that if a software developer wants to draw a circle, for example, it is necessary to know that the method “drawOval” in the Java class library meets the functionality desired or at least recognizes this method also belongs to the java.awt package. Henninger and Belkin [Henninger and Belkin, 1996] emphasizes that software components must be organized in a way they can be found.

One of the strategies in this direction is to build a classification scheme where the retrieval process can be accomplished by choosing the right category and thus, augment source code representation. However, it demands a detailed understanding of the classification scheme.

These concerns about source code motivated initiatives focusing on effective source code search engines. Based on this, Garcia et al. [Garcia et al., 2006a] elaborated a review about the state-of-the-art of code search engines, outlining essential aspects to be taken into account by any code searches towards an effective support of the reuse activity. In addition, Garcia et al. have presented commercial and non-commercial search engines and how they evolved along the time.
In this direction, next section presents a range of information retrieval tools which make use of different mechanisms in order to perform searches more efficiently and consequently improves the reuse activity.

### 3.4.4. Information Retrieval Systems for Component Reuse

Since the first work regarding source code search engines in order to promote the reuse activity [Ezran et al., 2002], such engines have attracted investments from software companies [Google, 2006], [Koders, 2006], [Krugle, 2006]. This section reviews outstanding code searchers supported by different mechanisms in order to enhance its search efficiency.

Some reports in literature such as [Maarek et al., 1991] argue that manual classification of components may be imprecise, since it is a subjective process, where the interpretation may vary to describe the same asset. Maarek et al. approached the similarity problem using clustering artifacts automatically from free-text descriptors, terms or phrases that best describes a component. In 1994, Henninger [Henninger, 1994] presented the CodeFinder, a code searcher based on query-formulation methods to aid users on defining their needs when they do not know the exact terminology.

In early 90’s, Pietro-Díaz [Prieto-Díaz, 1991] proposed the utilization of a facet scheme to classify and support software components retrieval, where classes are assembled by selecting predefined keywords from faceted lists components and the components are described according to perspectives (facets), unlike the traditional hierarchical classifications, where a single class is selected since best describes the component.

In late 90’s, the Software Engineering Institute (SEI) at Carnegie Mellon University developed Agora [Seacord et al., 1998], an information retrieval system, where the user enters a keyword query and optionally specifies the type of component. These terms and other criteria are searched in an index collected by the search agents. The result set for the query is returned for user’s inspection. Each result includes meta-information containing the URL of the component and then the user can narrow or broaden the initial search criteria based on the number and quality of the matches [Seacord et al., 1998].
Chapter 3 – Information Retrieval

In 2000, Thomason et al. [Thomason et al., 2000] introduced the CLARiFi, a component-based system that provides a classification scheme that identifies component properties important in the selection for a given task. In 2002, Ye and Fischer [Ye and Fischer, 2002] presented the CodeBroker, a context-aware code searcher to retrieval source code according to information extracted from the developer environment. In their work, was proposed a process called information delivery (or active search), which consists in foresee the software engineer’s needs for components, by monitoring activities of the software engineer, and then searching automatically for the components.

In 2003, Sugumaran and Storey [Sugumaran and Storey, 2003] presented A Semantic-Based Approach to Component Retrieval to meet user’s requirement based on domain models containing the objectives, processes, actions, actors, and, an ontology of domain terms, their definitions, and relationships with other domain-specific terms.

In 2005, Holmes and Murphy [Holmes and Murphy, 2005] proposed the Strathcona, an Eclipse plug-in that finds source code examples by applying heuristics. The retrieval is based on similarities between the source code under development and others stored in the repository.

After 2005, keyword-based web search engines focused on source code retrieval also started to be available on Internet. Koders web search engine [Koders, 2006] automatically connects on different version control systems (e.g., CVS and Subversion) to search source code. Krugle [Krugle, 2006], similar to Koders, proposes to assist professional developers to solve their programming problems by searching for many different types of programming languages. Google Code [Google, 2006] searches public source code grouped by function definitions and sample code in several programming languages. In addition, it allows searches using some specific facets and provides an API called Google Code Search data API, allowing developers to build applications to search public source code on the Internet. In [Garcia et al., 2006b] was presented a keyword and facet-based component search engine called Maracatu. Designed over client-server architecture, it allowed a client Eclipse plug-in searches Java source code from CVS repositories indexed in the server side.
Chapter 3 – Information Retrieval

In 2006, Mascena et al. [Mascena, 2006] proposed an extension for Maracatu called ADMIRE, an integrated reuse environment based on the same concept of information delivery proposed in [Ye and Fischer, 2002]. Mascena et al. also introduced a new reuse metric with the goal of monitoring the reuse activities and allows the software engineer to make corrective actions across the development process. A second evolution of Maracatu was proposed by Vanderlei et al. [Vanderlei et al., 2007], where was applied the folksonomy approach, a tag-based mechanism where developers manually tag source code with related terms.

The RiSE group and C.E.S.A.R² (Recife Center for Advanced Studies and Systems) have sponsored the development of the B.A.R.T, the Basic Asset Retrieval Tool, a commercial version of Maracatu search engine [Garcia et al., 2006b]. Further searching software assets, the B.A.R.T agenda envisage the addition of other search techniques - beyond the current keyword-based - such as facet-based (proposal of this dissertation), folksonomy among others.

In the same direction, Merobase [Merobase, 2007], [Hummel, 2008] was proposed to allow users to find, remember and share components on the Internet. Different of first-generation code search engines, Merobase deals with the source code modules as first class abstractions rather than chunks of text. In particular, Merobase specializes in searching for components from their interface (or API) rather than the terms inside the code.

The search engines presented so far have employed their efforts in source code retrieval based on keyword search. However, some challenges in order to attend and to reduce the semantic conceptual gap are still not completely solved by these solutions and also do not exclude complementary approaches such as the faceted approach. Nevertheless, the next chapter outlines how this approach can contribute to increase the recall and precision of search engines results.

² http://www.cesar.org.br.
3.5. Information Retrieval Discussion

Prieto-Díaz [Prieto-Díaz, 1991] introduced the concepts of facets in order to a more effective reuse. Although, the creation of a efficient a scheme composed by facets that best describes the components is a difficult task, since the context and related information may vary, being useful for ones but not for others. In this direction, new approaches of automating the generation of a facet scheme as the use of WordNet [Stoica and Hearst, 2006] and proposition of algorithms [Stoica et al., 2007] were suggested.

Maarak et al. [Maarek et al., 1991] and the CLARiFi [Thomason et al., 2000], also presented techniques to perform this process automatically by clustering artifacts from free-text descriptors, terms or phrases that best describes a component. A complement of these mechanisms was suggested in the CodeFinder [Henninger, 1994], by using query-construction methods for assisting users to define their needs when they do not know the exact terminology. Alternative initiatives such as evidenced in [Sugumaran and Storey, 2003], present the use of ontologies in order to identify proper terms to be applied in the query formulation. A merge of all these approaches may result in a powerful and efficient search engine.

Proposals such as Maracatu [Garcia et al., 2006b] and Strathcona [Holmes and Murphy, 2005] must have special attention since they take advantage because were developed to be embedded in the user environment which facilitates their usage. The use of folksonomy technique as implemented by Vanderlei et al. [Vanderlei et al., 2007] also contributes towards free hand classification by use of tags, approach that has been applied in many important information retrieval tools available especially on internet such as Del.icio.us3. Web search engines such as Krugle, Merobase and Google must also be considered, especially due to the availability quality attribute.

---

3 A social bookmarks manager. More detail in http://del.icio.us/
3.6. Chapter Summary

This chapter introduced the information retrieval field and discussed its relationship to software reuse. A comparison between data retrieval and information retrieval system also was presented and, some questions regarding relevance search, and related work in favor of software reuse were also described.

The next chapter outlines the proposition of this dissertation: an extension of the B.A.R.T Search Engine using the faceted approach to classify source code assets. In addition, the next chapter also introduces the DoFaST, a tool to aid the domain expert on performing activities such as analysis and source code classification based on this approach.
Search and Retrieval of Source Code Using The Faceted Approach

One of main challenges in software reuse is how to organize collections of assets for effective search and retrieval. Facets appear as a suitable approach to provide a high representative and descriptive representation of assets in categories. In this sense, the main proposition of this work is the use of faceted approach to classify source code in order to increase the precision of source code search engines. In this chapter, main concepts related to the faceted approach, and an extension of the B.A.R.T Search Engine and a supporting tool to aid the expert on performing his activities are also presented.

This chapter is organized as follows: Section 4.1 presents an introduction about the challenges found in search and retrieval systems and the motivation of using the faceted approach. Section 4.2 introduces the main concepts regarding facets. Section 4.3 presents some related facet-based search engines. Section 4.4 discusses the main proposal of this dissertation: the Facet-based Source Code Search Engine. Section 4.5 outlines the DoFaST, a supporting tool. Section 4.5.3 presents some functions of DoFaST, and finally Section 4.6 finishes the chapter with a brief summary about what was presented.
4.1. Introduction

Many researchers have highlighted that search mechanisms have a key role on information handling. Singer et al. [Singer et al., 1997] found out that activities regarding the search for information are the most dominant daily activity of the software developers. According to Henninger [Henninger, 1997], activities related to processes of searching and accessing different types of information, accounts for around 40% of the time of people in software organizations.

In the context of reusable components search and retrieval, one of main challenges is how organizing collections of reusable components for effective search and retrieval. A classification scheme is a tool for the production of systematic order based on a controlled and structured index vocabulary. This index vocabulary is called the classification schedule and it consists of a set of names or terms representing concepts or classes, listed in systematic order, to display relationships among classes [Prieto-Díaz, 2004].

In spite of information retrieval techniques, they can be classified in two major categories [Prieto-Díaz, 1991]:

1. **Free-text analysis**: relies on evaluate word frequencies in natural text. Relevant keywords are derived automatically by their statistical and positional properties, thus resulting in what is called automatic indexing. This technique has proven relatively effective for text intensive assets; and

2. **Controlled vocabulary**: consists on a predefined set of keywords used as indexing terms. These keywords are conceived by domain experts and are designed to represent concepts relevant to the domain of discourse.

In a context of source code assets, some features of controlled vocabulary make this technique a more suitable approach over free-text analysis. Some of these features are described as follows [Prieto-Díaz, 1991]:

1. Source code is very low on free text;
2. Keyword meanings are usually assigned by convention or by programmer preference, since it is not obvious "what" components do and "how" they do it; and
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

3. Human intervention is typical when describing component functionality.

In a controlled vocabulary, concepts are organized as a classification scheme. Hearst [Hearst, 2006a] reported a comparison between two commonly used techniques of controlled vocabulary: clustering and Hierarchical Facet Categories (HFC) approach. The pros and cons of each approach are detailed as follows:

**Clustering**

- **Pros:**
  - Clustering can be useful for clarifying and sharpening a vague query, by showing users the dominant themes of the returned results;
  - Clustering also works well for disambiguating ambiguous queries; particularly acronyms. For example, ACL can stand for Anterior Cruciate Ligament, Association for Computational Linguistics, Atlantic Coast Line Railroad, among others; and
  - An underappreciated aspect of clusters is their utility for eliminating groups of documents from consideration. For example, if most documents in a set are written in one language, clustering will very quickly reveal if a subset of the documents is written in another language.

- **Cons:**
  - Because clustering algorithms are imperfect, they do not neatly group all occurrences of each acronym into one cluster, nor do they allow users to issue follow-up queries that only return documents from the intended sense (for example, "ACL meeting" will return meetings for multiple senses of the term);
  - Lack of predictability, their conflation of many dimensions simultaneously, the difficulty of labeling the groups. Some algorithms build clusters around dominant phrases, that make for understandable labels, but whose contents do not necessarily correspond to those labels; and
Disorderly grouping: usability results show that users do not like disorderly groupings, preferring understandable hierarchies in which categories are presented at uniform levels of granularity.

**HFC (Hierarchical Faceted Categories)**

**Pros:**
- A category system is a set of meaningful labels organized in such a way as to reflect the concepts relevant to a domain;
- Provides flexible ways to access the contents of the underlying collection. Navigating within the hierarchy naturally builds up a complex query that is a conjunction of disjunctions over subsets of hierarchies;
- An interface using HFC simultaneously shows previews of where to go next, and how to return to previous states in the exploration, while seamlessly integrating free text search within the category structure;
- Reduces mental work by promoting recognition over recall and suggesting logical but perhaps unexpected alternatives at every turn, while at the same time avoiding empty results sets. This organizing structure for results and for subsequent queries can act as scaffolding for exploration and discovery; and
- HFC-enabled interfaces are overwhelmingly preferred over the standard keyword-and-results listing interfaces used in Web search engines. Participants in a study [Hearst, 2006a] found the design easy to understand, flexible, and less likely to result in dead ends.

**Cons:**
- Categories of interest must be known in advance, and so important trends in the data may not be shown; and
- In most cases, the category hierarchies are built by hand and automated assignment of categories to items is only partly successful.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

In this sense, the faceted classification approach proposes mechanisms to **synthesize** such assets according to a classification scheme. More details about the faceted approach and its concepts are discussed in next section.

### 4.2. Faceted Classification Concepts

The **Faceted Approach** was introduced by Ranganathan [Ranganathan, 1967] and promotes synthetic classification where classes are assembled by selecting predefined keywords from a facet scheme.

Consider a common example, wine. Each wine has a certain color, from a certain place, is made from a particular kind (or blend) of grape, the vintage’s year is known, comes in a container of a given volume and has a price. Using facet-based approach, a set of categories is created to best describe the wines characteristics such as: color, origin, grape, year, volume, and price. Each category contains terms to organize the wine’s list in an appropriate way, where each bottle of wine is classified by picking and choosing the right terms from each category.

This is a faceted classification: a set of mutually exclusive and jointly exhaustive categories, each made by isolating one perspective on the terms (a facet), that combine to completely describe all the objects in question, and which users can use, by searching and browsing, to find what they need [Kwasnick, 1999].

Faceted classification consists of two components: the facet scheme containing *facets* and *categories*, and the association between each document and the categories in the facet schema. For collections growing in both volume and variety, a major challenge is to evolve the classification: continuously classify new objects, modify the facet scheme, and reclassify existing objects into the modified facet schema. Existing classification systems are typically centrally managed and difficult to evolve [Wu et al., 2007].

According to Kwasnick [Kwasnick, 1999], there are several aspects in favor of faceted classification, especially regarding source code classification context, such as:

- Does not require complete knowledge of the entities or their relationships;
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

- Are hospitable (can accommodate new entities easily);
- Are flexible and expressive;
- Can be ad-hoc and free-form; and
- Allow many different perspectives and approaches to the things classified.

Nevertheless, some impediments about the creation of a handful and descriptive facet scheme that represents in a more adequate way the relevant concepts of an asset are presented as follows [Kwasnick, 1999]:

- Difficulty of choosing the right facets;
- Lack of the ability to express the relationships among them; and
- Difficulty on visualization.

The process of selecting the right facets to represent the facets is crucial, and requires good knowledge of the assets being classified. In this direction, some research in the literature propose methodologies to the process of conception of a well-descriptive facet scheme [Spiteri, 1998]. In order to reduce problems concerning visualization of the assets, this dissertation proposes an extension of a web search engine, B.A.R.T, to deal with search and retrieval of source code. The web architecture of B.A.R.T is a suitable mechanism to dispose the assets in order to ease the reuse activity as evidenced in [English et al., 2002]. In the direction of diminishing the manpower required in the process of analyze and classify the source code, is also proposed a tool to support the domain expert in such activities, besides activity of identifying new facets is also proposed in this dissertation and it is named DoFaST.

4.2.1. Facet Engineering Process

In spite of the classification process of assets by using the faceted approach, there are basically three main concepts to be achieved [Damiani et al., 1999]:

1. **Facets Scheme Conception**: Elaboration of a set of categories, each one made by isolating one perspective of the items (a facet), that combine to completely describe all the objects in question;
2. **Asset Analysis**: The domain expert analyzes the features and characteristics of the assets to be used in the faceted classification step; and

3. **Ranking/Classification**: Based on the analysis of the asset, the domain expert relates the asset to a set of terms of facets according to the facets scheme.

In order to systematize the process of facets scheme creation, Spiteri [Spiteri, 1998] presented a work based on Ranganathan’s [Ranganathan, 1967] and divides classification into the following principles:

- **Idea Plane**: Which involves the process of analyzing a subject field into its component part;

- **Verbal Plane**: The process of choosing appropriate terminology to express those component part; and

- **Notational Plane**: It involves the process of expressing these component parts by means of a notational device.

Each one of these principles has several aspects to be taken in account and listed as follows [Spiteri, 1998]:

**Idea Plane**

- **Differentiation**: when dividing an entity into its component parts, it is important to use characteristics of division (i.e., facets) that will distinguish clearly among these component parts. For example, dividing humans by sex;

- **Relevance**: when choosing facets by which to divide entities, it is important to make sure that the facets reflect the purpose, subject, and scope of the classification system;

- **Ascertainability**: it is important to choose facets that are well defined and can be ascertained;

- **Mutual Exclusivity**: facets must be mutually exclusive, in other words, the contents of any two facets cannot overlap; and

- **Fundamental Categories**: there are no categories that are fundamental to all subjects, since categories should be derived based upon the nature and context of the subject being classified.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

Verbal Plane

- **Context:** the meaning of an individual term is given its context based upon its position in the classification system. For example, *São Paulo*, the *city*, and *São Paulo*, the *state*, can be identified as *São Paulo* and the distinction will be whether it is *city* or *state*; and

- **Currency:** the terminology used in a classification system should reflect current usage in the subject field. This means the system will demand constantly revision.

Notational Plane

- **Synonym:** each subject can be represented by only one unique class number;
- **Homonym:** each class number can represent only one unique subject;
- **Hospitality:** notation should allow for the addition of new facets at any point in the classification system; and
- **Filing Order:** a notational system should reflect the filing order of subjects. This type of notation would reflect the citation order underlying the classification system.

Despite recommendations of how creating facets schemes, a considerable impediment for a wider adoption of the faceted approach on classification of assets is the manpower dispended in the process [Stoica et al., 2007].

In this direction, some research in the literature have suggested automated approaches to generate the facets scheme. Damiani [Damiani et al., 1999] presented the *Corrigenda*, a hierarchy-aware classification schema for object-oriented code that uses the facets to classify software components according to their behavioral characteristics, such as provided services, employed algorithms, and needed data.

In [Stoica et al., 2007] is proposed the *Castanet*, an algorithm used to build automated hierarchical metadata, where suggestions for metadata terms are automatically generated and grouped into hierarchies and then presented to information architects for limited pruning and editing [Stoica, 2004].

In [Wong et al, 2007] is proposed data mining to select some suggestions (facets) according to numerical and nominal attributes using an approach
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

named Skyline. Dakka et al. [Dakka et al., 2008] introduces an approach to generate facets in a three-step algorithm based on interaction between external/on-line search engines and applications.

4.2.2. Faceted Approach for Source Code Classification

In a source code classification context, the faceted approach appears as an alternative to meet most of the requirements necessary to classify reusable components. Faceted approach provides features [Prieto-Díaz, 1991] such as:

- Flexibility to deal with expanding collections;
- Must provide in its result similar matches than just exact matches on searching and retrieval of components;
- High precision and descriptive power;
- Low level of effort to maintenance; and
- Can be partially automated.

According to Damiani et al. [Damiani et al., 1999], the faceted approach is based on software descriptors whose purpose is to describe meta-information about code characteristics. These software descriptors rely on semi-automatic extraction of terms from code. Hearst [Hearst, 2006a], define facets as a recommended approach to promote a coherent and a quite complete classification, providing a set of meaningful labels reflecting the concepts relevant to a domain.

4.3. Faceted Approach Applied in Search Engines

The faceted approach is been used in different contexts regarding to classification. Categories systems may apply the faceted approach to organize and classify the data to be stored. Some of these systems are presented as follows:

ClassServer and ClassSearcher proposed by [Mili et al., 1994] are experimental library tools that focus on evaluating issues in software reuse, especially information retrieval. In these tools, the data is structured in
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

Multifaceted categorization of components, where attributes are represented in two properties: text, which is a textual description in natural language; and values, which are a collection of key words or phrases. Nevertheless, this library lacks of mechanisms to ease the creation of the facet scheme, which is fundamental on classification process using the faceted approach. In addition, the ClassSearcher has a complex usability and thereby demands considerable effort to understand how to express and formulate queries.

An interesting work was realized in [English et al., 2002] and describes the use of faceted approach to build site search interfaces, where facet categories where applied to drive the users on narrowing their scope search. This approach was enhanced in the Flamenco framework [Hearst, 2006b].

Flamenco, presented by Hearst [Hearst, 2006b], is a search interface framework whose main purpose is allowing users to browse through large information spaces in a flexible manner, by the exposure of category metadata based on hierarchical faceted metadata approach. The disposition of facets to the user favors a better understand of how the facet categories are structured and consequently aids the user to reformulate queries to attend his expectations. This kind of browsing approach may provide effective benefits and can be adopted in further versions of the B.A.R.T Search Engine extension proposed in this work. However, there is no mention if its classification mechanisms are applicable to source code context since it focus on free-text documents. Likewise, the facet scheme conception and classification process of the documents are still performed manually. In a nutshell, the main benefits of this framework are to increase the usability of the system and a new perspective of how to present the data to the user. This framework is currently an open-source project and it first release version is available for download.

Wu et al. [Wu et al., 2007] present a system that proposes evolve a scheme with facets and categories, and to classify documents into this scheme based on the premise that a diverse, large group of people can do better than a small team of experts. Despite interesting concepts were introduced, some of them similar to the folksonomy approach [Vanderlei et al., 2007], this study still requires more refinement such as pruning of redundant facets created by the users and the increasing of the accuracy of the automated classification.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

Moreover, faceted approach is used as the classification mechanisms of organizational systems that details about development and strategic decisions are rarely published. Therefore, success cases such as the GTE Data Services' Asset Management Program developed at the Contel Technology Center and described in [Prieto-Díaz, 1991] are available in literature. Further the organizational process adopted to promote reuse in GTE Program, the supporting tool adopted and used for search and retrieval of assets was facet-based. However, as the previous analyzed proposals, it lacks of mechanisms to ease the updating and maintenance of the facet scheme.

In this sense, the proposition of this work attempts empower the benefits proposed by these tools, applying the faceted approach to classify source code in a robust search engine named B.A.R.T. Regarding the efforts to reduce the manpower required to create the facet scheme evidenced in Flamenco and ClassServer, this dissertation aims two directions:

1. Proposition of a tool, named DoFaST, to aid the expert on the process of analysis and classification of source code, that are activities related to the creation of the facet scheme; and
2. The presentation of a Facet Description API, which allows the developers extend the classification algorithms and the use of new mechanisms to become the process of the facet scheme’s creation more automated. In spite of search interface and query formulation concerns, the extension of B.A.R.T suggests user-friendly interfaces to increase the usability of the system and diminishing the learning curve obtained in the ClassSearcher.

4.4. Search and Retrieval of Source Code Using The Faceted Approach

There are a plenty of source code files and open source sites on the Internet and in organizational repositories. If the programs in such sites are correctly classified in an efficient way, software reuse would be greatly facilitated. Although, in most cases, program files are classified according to the programming language and application topic instead of attempt to organize
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

them based on the source code itself, and related documentation such as design specifications and the requirements documentation [Ugurel et al., 2002].

In this sense, to relate the category within the code is recommended in order to gather information over natural language resources such as source code comments and related documents in order to increase the descriptive power. However, natural language resources do not contain all the available information, since some of them are intricacy related to the code itself such as the programming language and algorithms. In order to take advantage of the benefits more effectively, not only of the natural language information available in documentation, but also the code must be considered [Ugurel et al., 2002].

Therefore, searching itself is not always the only desirable and sufficient method for accessing large repositories due to their simplicity. Often, other access methods are necessary or preferred and, for such scenarios, users cannot rely on search alone. In fact, ranking is not feasible in these scenarios because the absence of a concrete user query mechanism and every asset is a candidate of interest to the curious or unfamiliar users.

In order to support such exploratory interactions, the facet hierarchies’ concept is been widely adopted [Dakka et al., 2008]. Compared to searches, faceted classification allows users retrieve information through recognition of category names instead of recall of query keywords. Faceted classification has been shown to be effective for exploration and discovery in large collections of documents [Hearst, 2006a].

In this direction, next sections present in the details the proposed solution based on these concepts.

4.4.1. Requirements

The functional requirements and quality attributes proposed for the B.A.R.T Search Engine extension using the faceted approach to classify source code are based on aspects extracted from the analysis of tools developed so far [Prieto-Díaz, 1991], [Mili et al., 1994], [Garcia et al., 2006b], [Vanderlei et al., 2007], [Hearst, 2006b], and reviews of the state-of-the-art [Lucrédio et al., 2004], [Garcia et al., 2006a]. In addition, some of the stated requirements relied on reuse practices and discussions in the RiSE group. In this context, a set of
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

The functional and non-functional requirements for the Source Code Search Engine supporting the Faceted Approach are detailed in the following Subsections.

4.4.1.1. Functional Requirements

The functional requirements for effective search and retrieval source code search engine supporting facets are described as follows:

**Keyword Search and Retrieval**: The search mechanism should be performed through keywords usage, with a user friendly interface such as web search tools. The input data must accept text and support logical operators such as “AND” and “OR”;

**Query Formulation**: The Source Code Search Engine supporting the Faceted Approach must provide means to help the user to formulate the queries in order to reduce the gap between the problem stated by the user and the solution proposed, since there is a natural information loss when the user is formulating a query as described in [Ye and Fischer, 2002]; and

**Source Code Filtering**: As the main focus is source code, the search engine must filter assets that are not compliant with source code, such as documents and other kinds of artifacts. Moreover, low quality source code must also not be recommended.

Specifically about faceted approach, the functional requirements stated are described as follows and are considered core requirements for the search engine.

**Support to Facets Description**: The search engine must provide a facet description API to support extension of the facets repository, which will be used in classification process;

**Set of Default Facets in Facet Knowledge Base**: In order to provide an initial set of facets, the repository of facets, also known as Facet Knowledge Base, must be composed by default facet descriptions;

**Faceted Query Formulation**: The search mechanism must expand the formulated query by the composition of the keywords of the query and the selected facet terms available to narrow the scope search;
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

**Source Code Classification**: The source code of the reuse repository must be evaluated and classified into the facets relied in the facet repository;

**Visualization of the Source Code Classification**: The source code classification results shall be available for the domain expert analysis in order to validate and balance the classification engine module; and

**Manual Facet Classification Adjustments**: For more complex source code data classification, or adjustments in the automated classification, the system must provide supporting features to aid the domain experts in such activities.

The non-functional requirements, also known as quality attributes [Sadana and Liu 2007], are described in the next subsection.

### 4.4.1.2. Non-Functional Requirements

The quality attributes stated as non-functional requirements are:

**Reusability**: the proposed faceted approach must follow the Component-Based Development premises, producing reusable self-contained components instead of an integrated application. Such features may allow further extension and ease the integration with others information retrieval systems;

**Modifiability**: in order to provide a structure able to support future growth, the system must be extensible, considering the addition of new functionalities without impact the current characteristics and performance of the system. In this sense, the system must be well-designed, composed by high cohesive and loose coupling modules;

**Usability**: this qualitative attribute refers to how easy and intuitive user interfaces are. The system must apply methods in favor of these aspects, since they directly imply on popularity and attractiveness of a system. In a nutshell, the user interface must be intuitive and efficient;

**Performance**: Regarding search and retrieval systems, performance is a key factor to be considered. The performance of those systems is measured taking into account the achieved precision and recall rates. High precision is met if most of relevant elements are returned in the search results and high
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

recall is achieved when few relevant elements are not covered into the results. Response time must also be considered in this quality attribute;

**Interoperability**: This attributes drives the faceted approach applied on the search engine to work along with others search and retrieval techniques such as keyword match, folksonomy and semantics; and

**Platform Independence**: The development platform and environment usually vary among organizations. In this sense, in order to support more distinct contexts, cross platform solutions are recommended.

The development of requirements in the implementation was driven according to the order of importance stated. According to the RiSE group experience, each one was classified according to the criteria described as follows:

- **Essential**: It represents the key requirements that must be achieved. Since are considered core features, the lack of them turns the application useless;
- **Important**: It represents the medium-priority requirements that are strongly advisable for better usage of the tool; and
- **Desirable**: It represents desirable requirements that its use is related to a specific context or to enhancements about current development.

For the *situation of realization*, three criteria were adopted:

- **Achieved**: It means that the requirement was completely carried out and tested;
- **Partially Achieved**: It means that the requirement was implemented but there was not opportunity for testing or validation; and
- **Not Achieved**: It represents the requirements that were not definitely implemented.

Table 2 summarizes the requirements showing the priority and the situation of the proposed requirements.

**Table 2. Requirements summary.**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Priority</th>
<th>Situation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensibility</td>
<td>Essential</td>
<td>Partially Achieved</td>
</tr>
<tr>
<td>Faceted Query Formulation</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Desirable</td>
<td>Partially Achieved</td>
</tr>
<tr>
<td>Keyword Search and Retrieval</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>Manual Facet Classification Adjustments</td>
<td>Important</td>
<td>Not Achieved</td>
</tr>
<tr>
<td>Modifiability</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>Performance</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>Platform Independence</td>
<td>Desirable</td>
<td>Achieved</td>
</tr>
<tr>
<td>Query formulation</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>Reusability</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>Set of Default Facets in Facet Knowledge Base</td>
<td>Aimed</td>
<td>Achieved</td>
</tr>
<tr>
<td>Source Code Classification</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>Source Code Filtering</td>
<td>Desirable</td>
<td>Partially Achieved</td>
</tr>
<tr>
<td>Support to Facets Description</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>Usability</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>Visualization of the Source Code Classification</td>
<td>Important</td>
<td>Achieved</td>
</tr>
</tbody>
</table>

As discussed before, the proposal of this dissertation regarding search and retrieval of source, in a nutshell, is to integrate the faceted approach into B.A.R.T Search Engine. More detail about the B.A.R.T system architecture and description of its modules are described in next section.
4.4.2. B.A.R.T System

Based on the idea that software reuse can be performed in a systematic way [Frakes et al., 1998], and supported by an environment to support the software development process activities, the B.A.R.T (Basic Asset Retrieval Tool) was developed. B.A.R.T is a search engine to aid the software engineer to find components, which implement the problem that needs to be solved [Garcia et al., 2006a]. The main idea of the B.A.R.T project is that the environment evolves in an incremental and systematic way [Frakes et al., 1998], across the phases of the whole software development life cycle, through the integration of different techniques and methods that act to improve the effectiveness and the results of the environment.

4.4.2.1. B.A.R.T Architecture

The B.A.R.T tool is based on a client-server architecture, where the client side communicates to the server side through a web service layer. The client side is represented by Eclipse and MS Word plug-ins besides a brand new web interface, which searches reusable assets on the server side [Garcia et al., 2006b]. Its architecture was designed to be extensible by providing the capacity to add new features by including new components. Initially, this proposal concentrates its efforts only on an external client and in architectural modules of server side. The basic modules that comprise the entire architecture are: searcher, retriever, indexier, filter, and repository manager.

Figure 2 shows the current B.A.R.T architecture. The main components of B.A.R.T are detailed as follows:
Figure 2. B.A.R.T Architecture.

**Repository manager**: this module checkouts assets from Concurrent Version System (CVS) repositories on the Internet and maintains the entire repository scheme of the system. It is also possible to schedule updates from previous checkouts in order to capture brand new assets.

**Filter**: after automatic checkout, this module filters those assets which do not satisfy some conditions such as unnecessary extension and lack of documentation (particularly applied to source codes).

**Indexer**: this module indexes the reuse repository and creates the index base which is essential for the search occurrence.

**Searcher**: this module is responsible for processing the user query and prepares it to seek the assets through the index base answering selected facets constraints.
Ch. 4 – Search and Retrieval of Source Code Using The Faceted Approach

**Retriever:** after a successful search, the user can download assets listed in the B.A.R.T client application. This module is responsible for transferring them to the user directory or to a specific Eclipse project.

Respecting to search and retrieval specifically, the current version of B.A.R.T presents functionalities such as keyword, semantic and folksonomy search. Therefore, a robust infrastructure was implemented to manage users, extract statistics and, registering logs.

As an alternative to promote a more efficient search and retrieval process the faceted approach appears as a suitable mechanism to classify assets in particular, source codes handled by the B.A.R.T engine.

### 4.4.3. Facet Classifier Engine

The Artificial Intelligence field has contributed with learning techniques for text categorization such as: bayesian models, nearest neighbor classifiers, decision trees, support vector machines (SVMs) and neural networks. In text classification, each document is represented as a vector of words and new documents are assigned to predefined categories using textual content [Ugurel et al., 2002].

Although programming languages are written in a different manner from natural language and have some commented information, they also have specific keywords and features that can be identified. Thus, these aspects indicate that text categorization techniques may also be effective for source code classification.

In addition, since such techniques are more applicable for a specific context, this dissertation proposes a flexible structure where classification algorithms can be connected in order to categorize a source code input into facets.

#### 4.4.3.1. Architecture

The *Facet Classifier Engine* structure relies on the architectures proposed by Pietro-Díaz [Prieto-Díaz, 1991], Damiani [Damiani et al., 1999] and Ugurel et al.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach [Ugurel et al., 2002]. As showed in the Figure 3, the engine is composed by the following modules:

- **Analyzer Module**: it manipulates the evaluated source code in order to retrieve information for use in classification and scoring processes. It is composed by two sub-modules: *AST Parser*, which creates a syntactic tree representation of extracted information, and *Source Code Analyzer*, which analyze the source code in order to identify relevant information; and

- **Reasoner Module**: it is responsible for evaluate and classify the source code according to information provided by the analyzer module. The sub-modules that compose this module are presented as follows:
  - Facet database, in other words, the Facet Knowledge Base;
  - Source code evaluation; and
  - Scoring data and classification process.

Figure 3. Facet Classifier Engine architecture.

More detail about the modules and the interaction among them are discussed as follows.

For the initial version of this proposal, was selected the *Java* programming language, *since* object-oriented (OO) paradigms provide resources that may cause a strong impact of reuse on product productivity and, especially, on product quality, or defect density and rework density [Basili et al., 1996]. In Java, the concept of class and packages suggests a representation of a
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

Concise and cohesive objects, and components, in general, are composed by set of these entities.

In order to provide a more flexible classification engine, was built a facet description API (Application Programming Interface). The main purpose of this mechanism is to ease the manipulation of the facets by allowing creation of brand new ones or the updating of features and terms of existing facets. In addition, this approach promotes the standardization of the interfaces that share facets among others facet repository systems.

The facet API is composed by a set of classes in Java that define specific contracts to be achieved by any facet of the Knowledge Base. Figure 4 shows the class diagram UML excerpt core classes of the Facet description API. Basically, the *Facet* class defines the contract to be implemented by any facet class of the Knowledge Base. Each realization of this class has to implement its own specific evaluation method, which allows customized implementation of facet evaluation algorithms or more refined, efficient ones.

![Class diagram UML Excerpt of the Facet description API.](image)

Each *Facet* is formed by a set of *Term*. A *Term* is a concept that best define a subset belonging to a *Facet*. For instance, a Domain facet may contains terms such as Database, Audio, and Network among others. The criteria of evaluating a specific snippet of code as a *term* were also implementation-dependent, enjoying of the benefits of customization.

One of the most important steps on building a classifier is the extraction of features. In a source code context is evidenced a controlled vocabulary due to use programming language in essence, since the feature set consists of 'tokens' in the source code [Ugurel et al. 2002].
In this direction, for source code evaluation, this work suggests in its initial support API, a simplified strategy, which focus on extracting more representative information related to syntax and semantics provided by programming languages than just deal with source code as a set of words. Representative information in this context is any code containing class elements in use such as class name, imports and methods declarations.

To achieve this, several frameworks supporting Java source code parsing and manipulation were evaluated. All the frameworks are free for use with no restrictions. According to the evaluation about each framework was adopted the AntLR framework with the Java 5 grammar, which provided a more suitable API to our expectation and needs. More detail is depicted in Table 1.

**Table 3. Java source code parsers’ evaluation.**

<table>
<thead>
<tr>
<th>Framework</th>
<th>Description</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>QDox</td>
<td>Parser for extracting class, interface and method definitions from Java source files.</td>
<td>Lacks of resources for extracting data inside methods and specific syntaxes such as method calls and expression commands.</td>
</tr>
<tr>
<td>JaxMe-JS</td>
<td>Framework used for generating Java source code, providing a rich API.</td>
<td>Has a restrict API for extracting data in the code.</td>
</tr>
<tr>
<td>Eclipse JDT</td>
<td>Project for developing Eclipse plug-ins. Its UI module contains an AST viewer and an API for handling the parsed data.</td>
<td>Powerful API and resources such as AST viewers but has a high dependency of the Eclipse platform.</td>
</tr>
<tr>
<td>AntLR</td>
<td>Language tool that provides a framework for constructing</td>
<td>Provide a rich API, but has a higher learning curve.</td>
</tr>
</tbody>
</table>

---

4 Eclipse is an open source community whose projects are focused on building an open development platform. More details in http://www.eclipse.org.
recognizers and interpreters from grammatical descriptions of a language. It provides support for tree construction, and tree walking.

Thus, a data extractor that uses an AST (Abstract Syntax Tree) was built to represent the analyzed source code. The AST structure allows browsing through the Java source code syntax. From this structure, it is able to retrieve specific blocks of code associated to the use of Java classes declared in the code. The main reason of choosing analyze the class entities applied in the code instead of keyword based approaches for instance, was because in well designed object-oriented based components, the classes are the most representative component due to the semantics associated.

For the initial version of the data extractor were retrieved, from the source code, the use of class entities in the following contexts:

- New instance’s sentence e.g. Object obj = new Object();
- Method call’s sentence; e.g. obj.toString();
- Variable declaration’s sentence; e.g. Object obj;
- Import's sentence. e.g. import Java.sql.Connection;

These sentences are formatted and structured in a component named ClassStructure as showed in Figure 4.
Comment blocks are ignored since no quality control is assured due to its subjective content meaning, since it is written in natural language and eventually wrong descriptions about the method functionality may puzzle the classification.

In addition, parts not related to class entities handling also were ignored to reduce deviation of classification caused by snippets of code that are common and elementary among different classes such as primitive type declaration, loop commands and reserved words of the programming language. In order to reduce ambiguities in the identification of class types, it was created a simple type binding between class declarations through the code and import class declarations.

One problem in the source code analysis was to identify the correct source package of the declared class and the class used over the code itself since source code are not compiled classes and, therefore, there is no binding between a variable of a specific class type and the source package of its related classes. This problem is exemplified in Figure 6 as follows.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

```java
package example.b;

import example.a.*;
import example.c.*;

public class B {
    public void foo() {
        A x = new A();
        x.toString();
    }
}
```

**Figure 6. Link source binding problem.**

In this example shows if the source code import declaration block import refers to all classes in a package (`example.a.*`), in compiling time the Java Virtual Machine (JVM) links used classes over the code (class `A` *type of X variable in the foo method*) within its respective class in package. However, before compile time, this link still not exists and makes difficult the identification of which package a declared class in the code belongs to, since is not able to define if a class are in the same package of the class that are being developed or if are in another import declaration package or if it belongs to native package (`java.lang`).

In spite of the **Scoring Processor**, the scoring criteria were designed to be defined for each facet type, allowing to the experts/specialists implement and manipulate the extracted data according to their expectations. For each facet loaded in the Knowledge Base, the scoring process produces as result a classification structure as represented in Figure 7. This classification consists in a set of terms of the facet that qualifies the input data language. For example, for a Domain facet, some related terms may be Database, Network, Graphic among others. Thus, if the evaluated data have characteristics of Database and Network terms and attend the expect score for both, this data will be classified, according to the Domain facet perspective, as Database and Network.
4.4.3.2. Implementation

For initial evaluation, was developed one facet focused on classification of source code written in Java language from the Domain perspective. For the implementation of the facet Domain, was used the Facet Descriptor API. The strategy used in the development process was split the component in two sub-components: Domain Knowledge Base and Scoring and Classification.

**Domain Knowledge Base component:** This component is responsible for loading and assembling domains that compose the facet. In the facet domain classifier, the domains that pertain to the facet were structured in a XML file with the schema showed in Figure 8. The XML structure is a proposal based on default pair-values classification suggested by Pietro-Díaz [Prieto-Díaz, 1991].

```
<facet>
  <name>Domain</name>
  <terms>
    <term acceptanceScore="6">
      <name>Graphics (AWT)</name>
      <package-reference>java.awt</package-reference>
      <description>
        Classes based on J2SE 'java.awt' Package
      </description>
      <dictionary>
        <word>ActiveEvent</word>
        <word>Adjustable</word>
        ...
        <word>Paper</word>
        <word>PrinterJob</word>
      </dictionary>
    </term>
  </terms>
</facet>
```

*Figure 8. Example of facet scheme for Graphics domain.*
In a nutshell, the scheme contains a set of elements focused on to represent facets as a set of related words and properties. For domain facet, was adopted this representation to handle extracted data from Java source code. The facet name in structure is the name of the facet that, in our case is Domain.

The term element has an attribute that is acceptanceScore which will be described in details in next subsection. The name represents a domain category itself. The package-reference element is optional and it is being used to define the most representative Java package for the classes listed as word in the dictionary element. The description element can be used to give extra information about the domain. The domain evaluation algorithm is based on a knowledge base formed by previously subscribed terms of a domain. Term is a category of domain and it is composed by a set of classes API.

For the initial knowledge base, some Java API packages were subscribed and related to a term belonging to a domain as follows in Table 4:

<table>
<thead>
<tr>
<th>Domain term</th>
<th>Java API packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML, XML (DOM), XML (JDOM), XML (SAX)</td>
<td>java.xml, org.w3c.dom, org.jdom, org.xml.sax</td>
</tr>
<tr>
<td>Security</td>
<td>java.security</td>
</tr>
<tr>
<td>Remote Services</td>
<td>java.rmi</td>
</tr>
<tr>
<td>Network</td>
<td>java.net</td>
</tr>
<tr>
<td>Math</td>
<td>java.math</td>
</tr>
<tr>
<td>I/O</td>
<td>java.io</td>
</tr>
<tr>
<td>Graphics (AWT)</td>
<td>java.awt</td>
</tr>
<tr>
<td>Database, Database (HIBERNATE)</td>
<td>Java.sql, org.hibernate</td>
</tr>
</tbody>
</table>

The terms of the domain facet are loaded by the Domain Knowledge Base component and are available for queries made by the scoring component.

**Scoring and Classification component**: After loading the terms of the facet domain, the scoring component evaluates each Java source code input by using these terms and the acceptanceScore. The acceptanceScore represents the score that a Java source code needs to achieve to be classified as the specific term.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

The facet domain scoring component uses the simple class type binding algorithm of the Facet Descriptor API to reduce Java class ambiguities. Moreover, from this API, was used the class `ClassStructure` to represent and extract information used for our scoring process.

Keyword match are used, which in this case are Java class names, to score the data input. If the package name and class name matches, it is scored 2 points. If only the latter case matches, it scores 1 point. The reference values are given by our experience justified by the link source problem stated in 4.4.3.1 subsection. Thus, the values considered that the package and class name match are more precise than a class name match only.

After scores the source code input, the component verifies if the `acceptanceScore` criterion is satisfied to classify the input as the evaluated term.

The `Facet Classifier Engine` was implemented during 3 months and took 140 hours approximately in a total of 15,739 lines of code and 7 Java packages. As development environment, the Eclipse Europa Tool was adopted as IDE and the Java programming language was selected to guarantee the compatibility with B.A.R.T Search Engine.

**Summary**

The overall process of classification is depicted in Figure 9.

![Figure 9. Classification process flow.](image)

For each Java source code, the **Facet Classifier Engine** extracts key information using the **Data Extractor** (Figure 9 (A)) that will be used as input
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

for the evaluation process in (Figure 9 (C)). In this process, the facets of the 
**Facet Knowledge Base** (Figure 9 (B)) evaluate the data and the result is consumed by the **Scoring Processor** (Figure 9 (C)), which verifies and scores it. According to the **acceptanceScore** criteria the classification result is presented in **FacetResult** structure.

### 4.4.4. B.A.R.T Search Engine Supporting Facet Classification

This subsection presents a general overview of how the faceted approach was integrated into the **B.A.R.T Search Engine** and how it works on searching and retrieval process.

Since the facet classification process is an expensive process in terms of time and, in order to perform a pre-processing of the source code, was added an **analyzer** module, responsible for analyzing and classifying source code into facets according to the facet scheme. The analyzer uses the Facet Classifier Engine to perform the classification. In addition, the classification process adopted the *on-the-fly* approach, which occurs when a new source code asset is added into the search engine repository.

As depicted in Figure 10, the user searches for codes which contain the keyword “connection” and select the “Domain” facet term “Database”.

![Figure 10. Query Reformulation.](image)

The B.A.R.T server receives the query and redirects it to **searcher** module. This module was extended to support filtering by facet and its respective terms. The **searcher** module retrieves all codes which contain the
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

keyword “connection” and, if any term of a facet is selected as filtering criteria, the searcher uses the filtering criteria to narrow the scope search results. In this sense, the **faceted approach**, used as a search constraint, aims to **increase** the **precision** of the search since the **recall** is provided by the **keyword-based** mechanism.

### 4.4.5. Implementation

The extension of the *B.A.R.T Search Engine* was implemented during 1 month and took 40 hours approximately in a total of 274 lines of code and 3 Java classes. This effort was reduced significantly due the help provided by a current developer of the *B.A.R.T Search Engine*. As development environment, the Eclipse Europa Tool was adopted as IDE and the Java programming language was selected to guarantee the compatibility with *B.A.R.T Search Engine*.

Some of challenges that arose during the development are related of the learning curve demanded to understand the *B.A.R.T Search Engine* structure and analysis of how to integrate the *Facet Classifier Engine* in the classification module and how to display and to make available the facets to the user formulate queries. More details of results are presented in chapters 5 and 6.

### 4.5. Domain Facet Supporting Tool (*DoFaST*)

Although the *Facet Classifier Engine* has achieved significant results respecting to semi-automation of Java source code classification by using the faceted approach (chapter 6), it still presents some difficulties regarding the extraction of meta-information from the source code. Moreover, the effort required to the domain expert performs activities such as analysis and classification in manual way have other limitations. In this context, was developed an auxiliary tool: *DoFaST*. 
4.5.1. Main requirements

*DoFaST* - **Domain Facet Supporting Tool** - is a tool for aiding the **domain experts** on performing source code analysis and classification activities, by providing a set of functional resources such as graphical components management and facet classifier engine integration. Most of the requirements proposed for the tool were based in improvements outlined in the classifier engine development process.

In addition, the classification of some scenarios requires a complex analysis of the source code and thus the automated classification did not obtain satisfactory classification results. Some of these are described as follows:

**Source code based on primitive types**: source code that are mainly structured over primitives types demands a more complex analysis to identify facets characteristics due to not present latent features and semantics as those based on classes from APIs. For instance, a source code that contains a specific encryption algorithm, such as MD5, are based on elementary mathematical operations not presenting latent behavior nor properties to be easily detected by an automated classifier. In this sense, the automated classifier would require a previous knowledge about encryption algorithms to classify properly.

**Imprecise import declarations**: since source code still not be in compiled state, the detection and statement of bindings between the referred source code and other classes are more difficult, especially when, in the import declaration block, the declared classes were not defined with their full package name which hardens class-relations analysis and the identification of the source classes’ packages. More detail about this problem is mentioned in 4.4.3.1 subsection.

**Relevant terms not covered in facet scheme**: Since the classification of source code is directly related to facets and terms covered by the facet scheme, source code based on terms that are still not covered by the facet scheme lacks of an expressive classification, causing an improperly classification.

In order to meet the stated requirements, was proposed an architecture of ease integration with the facet classifier engine and able to provide graphical
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

interface resources. Moreover, in order to increase the benefits promoted by the classifier engine further become a more suitable and flexible proposition, the proposed solution focused on a multi-platform programming language that were well-known by the domain experts. In this sense, source code written in Java was adopted as assets.

4.5.2. Architecture

The architecture of the **DoFaST** is structured in three main modules as depicted in Figure 11.

![Diagram of DoFaST architecture](image)

**Figure 11. DoFaST architecture and its modules dependencies.**

According to [Happel et al., 2008], one of the strategies in order to ease the source code search and manipulation is to provide graphical representations. In this sense, the **Graphics Module** manages the output results obtained in Classifier Module and display them in graphic component forms. The facets terms, java files and the relations among them are displayed as visual components as detailed in Figure 13 (B).

The **Classifier Module** is responsible for gathering all information of the loaded facets in the tool and apply them into the classification process of the add set of Java source code files. The evaluation process is delegated to the **Facet Classifier Engine** and its output is interpreted by the **Graphics Module** to be displayed as graphical components.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

The **Persistence Module** is responsible for store and retrieve files containing information about previous classifications, facet domain descriptions and Java source code files to be handled by the tool. The facet domain description file structure is in XML format, based on the proposition stated in the *Facet Classifier Engine*. For storing classification results, was developed a structure presented in Figure 12. This structure and its elements were driven by the **Classifier Module** output and it contains information about the analyzed source code file and the acquired score to each term of the facets used in the evaluation/classification process.

![Figure 12. DoFaST Classification File structure.](image)

In addition, some features to aid the domain expert to interpret and manage the files and facet scheme were provided by the *DoFaST* tool and are detailed in next subsection.

### 4.5.3. DoFaST in Action

The core features of the tool are focused on ease the domain expert activities. In this sense, the provided features are described as follows:
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

**Facets scheme loading:** Facets can be loaded from the facet description files and their terms are displayed as visual components (Figure 13(B));

**Java source code files loading:** Java source code files can be loaded and displayed as visual components (Figure 13(B));

**Facet Classifier Engine:** The classifier engine is responsible for classify loaded Java source code files in the tool. This engine performs the classification according to the facet scheme loaded in the tool. This feature is accessible from the menu application (Figure 13 (A)) and, if the source code meets the *acceptance score* to be classified as a specific term of a facet, the relation between facet terms and source code files are displayed as a weighted (calculated score value) arrow (Figure 13 (B));

**Visual Component finder:** This feature highlights terms and source code files, in agreement with the specified filter checkers (Figure 13 (C)), containing part of the search text. It is useful for select specific components and also to locate them in the graph;

**Graph display:** The graph display has two advanced features: Zooming and Layout Switcher (Figure 13 (D)). The former allows the application user apply zoom in and out effects on the components graph (Figure 13 (B)). The latter rearranges the component graph until five different perspectives available by the graph generator API: *Circle, KKLayout, FRLayout, SpringLayout* and *ISOMLayout*;

**Facet and Term filters:** It filters the visual components by the association to a specific facet or a specific term of a facet (Figure 13 (D)). Source code files and terms not related to the selected facets and/or terms are not displayed in the component graph. It is useful for focus the analysis over a specific facet or a specific term;

**Console Output:** The main actions applied in the application are detailed in the console output as shown in Figure 13 (E).
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

Figure 13. DoFast tool main screen. (A) Menu of the application to manage loading of facets and Java source code files and other features. (B) Representation of source code files and facet terms connected by a weighted association. (C) Search element box. (D) Filter and Zoom box. (E) Output console box.

Despite of the visual components, there are advanced features for deep analysis according to each visual component type. Each functional resource is described as follows:

**Classification Summary Viewer**: It presents the classification result summary of the selected source code file and its score is depicted in agreement with each facet and respective terms as shown in Figure 14.

Figure 14. Classification Summary Viewer.

**Source Code Viewer**: It enables to view the source code of the Java file visual component.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

**Term Detail Viewer:** It displays the description of the selected facet term, presenting the parent facet, its name and description, and the dictionary of keywords related to the term and used in classification process is showed in Figure 15.

![Figure 15. Term Detail Viewer.](image)

**Components Binding:** This feature is specially used to perform manual adjustments in automated classification result, allowing the domain expert classify the source code files to a facet term.

### 4.5.4. Implementation

The initial version of the tool was implemented during 2 months and demanded 45 hours approximately in a total of 2,189 lines of code and 4 Java packages. As development environment were used: as Concurrence Version System (CVS), the Subversion\(^5\); the Eclipse Europa Tool was adopted as IDE and the Java as the programming language. Java was selected due to be a rich multi-platform programming language with many compatible libraries available in the community.

---

\(^5\) Subversion is an open-source revision control system, which aims to be a compelling replacement for CVS. For more detail see http://subversion.tigris.org.
Chapter 4 – Search and Retrieval of Source Code Using The Faceted Approach

The JUNG\textsuperscript{6} API was adopted to manage the visualization of the components. JUNG was adopted because it provides a well-documented API and is widely used to build network graphs. Furthermore, it is free for academic use and provides an extensible API and implemented features as a set of graph layout, labeler displayers and input event handling.

Some of challenges faced during in the development are related of how to provide mechanisms that really contribute to the Domain Expert perform his activities. One of them was to define the strategy to be used in extracting relevant information from the source code. Likewise, the scoring strategy definition was also difficult since it is directly related to the extracting information strategy. Moreover, the selection of the visual library required the evaluation of several available libraries, which requires an expressive effort time. After the conclusion of the initial implementation, some aspects to improve and to make the tool more efficient were identified. More details of these aspects are presented in chapters 5 and 6.

4.6. Chapter Summary

In this chapter, it was discussed some challenges found in the search and retrieval field and how the facet-based approach can contribute. The main concepts regarding the facet-based approach and some related facet search engines were presented. In addition, was presented an extension of B.A.R.T Search Engine using the faceted approach and an auxiliary tool, named DoFaST, to aid the expert on performing his activities. In the next chapter, empirical studies to evaluate the Facet Classifier Engine and the B.A.R.T Search Engine extension using facets are described.

---

\textsuperscript{6} JUNG (Java Universal Network/Graph Framework) is a software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. For more detail see http://jung.sourceforge.net.
Once the proposition of the *B.A.R.T Search Engine* extension and the *DoFaST* tool were described and its initial implementation was detailed, some experiments were performed in order to evaluate the benefits of the *faceted approach* applied on *source code search engine*. The evaluation process was divided in two phases: *Facet Classifier Engine* evaluation and then the *B.A.R.T Search Engine* supporting the faceted approach. In this sense, some open source projects were selected from remote repositories on the Internet in order to evaluate if this proposal is suitable for practical usage.

This chapter is organized as follows: Section 5.1 describes the methodology applied on the experimentations. Section 5.2 presents the experimentation performed to evaluate the Facet Classifier Engine. Section 5.3 discusses the experimentation applied to evaluate the *B.A.R.T Search Engine* supporting the faceted approach. Section 5.4 outlines some conclusions on the findings of the evaluation.

### 5.1. Introduction

The experiment followed the methodology presented by Wholin et al. [Wholin et al., 2000]. This experimentation approach aims to provide guidelines of how to perform experiments to evaluate methods, techniques and tools in software engineering, turning the research activity more experimental.
Chapter 5 – The Experimental Study

5.2. Facet Classifier Engine Evaluation

5.2.1. Goal definition

The goal of this experiment is to analyze and evaluate the Facet Classifier Engine in terms of precision and classification correctness, by comparing the automatic classification and manual classification result.

Object of study: The object of study is the classification of Java source code from the Domain perspective using the faceted approach.

Purpose: The purpose of this experiment is to evaluate the performance of the Facet Classifier Engine during the process of classification of source code from the Domain perspective, in terms of precision and classification correctness.

Perspective: The perspective applied is from the Domain expert point of view.

Quality focus: The main effects studied in the experiment are the precision and correctness of the Facet Classifier Engine on classifying Java source code into facets.

Context: The context of this experimentation is the classification, from the Domain perspective, of Java source code using the faceted approach.

5.2.2. Planning

5.2.2.1. Context selection

The faceted classification efficiency relied on a study applied in an evaluation data analyzed and classified manually by RiSE members. During 1 month, 10 domain specialists collaborate with about 100 Java source code classes. Each source code was classified in the following domains: XML, Database, Security, GUI, Network, Math and I/O.
Chapter 5 – The Experimental Study

5.2.2.2. Hypothesis formulation

In order to evaluate if the proposed solution attend the goal expectations, the following hypotheses were formulated:

1. At least 60% of the component classification must match with the empirical classification.
2. No more than 30% of the component classification should be wrong.

These values were based on values expected by the B.A.R.T engine, according to the acceptance percentage level of precision stated by the engine [Durão et al., 2008].

In addition, in order to verify the precision and correctness of the classification were adopted the following concepts:

- **Match**: If the classification result of the faceted approach full matches the expected empirical result;
- **Confusion**: When the domains identified by the component differs at least in one of the empirical results; and
- **Fail**: If the component does not identify any of the domains classified empirically.

Based on this informal statement of the hypotheses, two hypotheses were formalized and their respective measures were stated using the null hypothesis approach.

**Null hypothesis** is the hypothesis that the experimenter wants to reject with as high significance as possible. The rejection of the null hypotheses defines if the classification results of facet based on domain are minimally acceptable for its use in the B.A.R.T Search Engine. Thus, the hypotheses defined were:

- **Hoa**: $\mu$ successful matches $< 60\%$.
- **Hob**: $\mu$ fail matches $> 30\%$.
Chapter 5 – The Experimental Study

5.2.2.3. Variables selection

The variables were selected in order to verify if the proposed solution attends the classification acceptance criteria in the B.A.R.T Search Engine [Durão et al., 2008]. In this sense, the independent variable is the classification result yield by the Facet Classifier Engine and the dependent variables are successful matches and fail matches.

5.2.2.4. Selection of subjects

The study was dealt as a single object study, in which studies are performed on a single team and in a single project. The participants were selected based on convenience - all are RiSE members and are the M.Sc. and Ph.D. students in Software Engineering from the Federal University of Pernambuco, Brazil.

5.2.2.5. Experiment design

The design of the experiment was based in the following principles:

- **Randomization**: All the Java source code were evaluated and classified by a Domain Expert manually and also by Facet Classifier Engine automatically;
- **Blocking**: No systematic approach to blocking was applied since all the source code elements should be evaluated; and
- **Balancing**: Although a balanced data set is preferable, some domains have more related Java source code than others. However it does not influence the results since in a real environment is not guaranteed that elements will be available equally balanced.

5.2.2.6. Instrumentation

Since the focus of this evaluation is the precision of the faceted classification by domain, the instrument used to validate was based on a comparison between a manual classification and the automated by the facet engine.
5.2.2.7. Validity evaluation

To provide a valid test result, were defined several threats to the validity of the experiment. Internal validity dependent of two aspects: Number of domain experts and the size of the evaluated data set. Regarding these aspects, was assumed that the quantity and quality of the subjects and the evaluated data set chosen provided a good internal validity.

Respecting to external validity, a risk related was the expert’s motivation, since they could perform the study without responsibility or without a real interest in performing the project with a good quality as it could happen in an industrial project. Since the goal was to evaluate the precision of the automated classification, the external validity was considered sufficient.

As threat to the conclusion validity, one was identified and it is related to the result not satisfies the expectations and then the classification engine may not be integrated in B.A.R.T engine. Moreover, the construction validity included one threat related to expert knowledge and their experience in the stated domains on collecting and classifying the data set. However, once the selected members are experienced developers and domain experts about the stated domains in the study, it was not considered this a threat able to impact in the proposed evaluation.

5.2.3. Operation

5.2.3.1. Preparation

The experimental study was conducted in two main activities: build and classify the dataset to be evaluated and apply automated classification and compare the result.

Both activities were conducted during from October 2007 to April 2008, at Federal University of Pernambuco and C.E.S.A.R. The experts were trained to use the criteria defined as key information by the Data Extractor on Java source code domain classification process. All the information was published and shared through collaborative tools.
5.2.3.2. **Execution**

The first block of activities - *build* and *classify*, were performed by 10 RiSE members domain experts. The dataset to be collected was built from collaboration of the experts and meets the stated requirements referring to the covered domains for the experiment.

The second activity block attempts to verify the efficiency of the faceted classifier by comparing the manual results and the automated result.

5.2.3.3. **Data validation**

Each specialist was oriented to collect and classify 10 Java source code classes, describing the source repository and, if applicable, the related project. All the experts performed all the activities, but 2 classes were invalidated due to its code syntax not be well structured. Thus, only 98 collected source code files were evaluated in experiment.

5.2.4. **Analysis and interpretation**

This section describes the analysis and interpretation of the experimental study. The automated classification results are depicted in Figure 16 for a better comprehension.

![Figure 16. Classification result of the Domain facet.](image-url)
Chapter 5 – The Experimental Study

**Successful matches**

Analyzing the classification result in Figure 16, about 66.67% of the source code dataset matches the expected empirical result. Thus, this rejects the null hypothesis (H₀a).

**Fail matches**

As shown in Figure 16, about 9.09% of the source code dataset failed on matching the empirical result. Therefore, this rejects the null hypothesis (H₀a).

### 5.2.5. Summary and conclusions

Relying on the obtained results, the experimental study indicates that the faceted approach has a considerable efficiency in classify Java source code from domain perspective, achieving the requirements to be able to **B.A.R.T Search Engine**.

Although positive results were obtained regarding **precision**, some difficulties and potential improvements were identified. Regarding to the results, an expressive classification percentage was considered as **confusion** type. Moreover, the need of provide a flexible structure for updating the data of the knowledge base allows the **reduction** of **confusion** and **fail** classification types.

According to the feedback extracted by the results and suggestion made by the experts, were identified new requirements in order to improve the efficiency of the **Facet Classifier Engine**. The data extractor has difficult in identify correctly classes that belongs to native Java packages such as **Java.lang** or that are not explicitly imported in import declaration block. In addition, in order to provide a more robust framework for facets conception, which comprehend the creation of more supporting class for the **Facet Descriptor API**, such as keywords analyzers, the implementation of supporting tools to aid the expert on analyzing and conception of new facets were evidenced.
5.3. B.A.R.T Search Engine Using Facet Classification Evaluation

5.3.1. Goal definition

The goal of this experiment is to analyze the B.A.R.T Search Engine using the faceted approach results and compare the previous result of search mechanisms (keyword and semantics layer) applied on B.A.R.T Search Engine in [Durão, 2008] under precision and recall perspectives.

Object of study: The object of study is the result provided by the extension of B.A.R.T Search Engine using the faceted approach.

Purpose: The purpose of this experiment is to evaluate the precision of B.A.R.T Search Engine extension using the faceted approach, in order to attend the user formulated query expectations.

Perspective: The perspective adopted is the classification of source code from the Domain point of view.

Quality focus: The main effect studied in the experiment is the precision of the B.A.R.T extension results.

Context: The context of this experimentation is the result suggested by the B.A.R.T extension in search and retrieval of Java source code.

5.3.2. Planning

5.3.2.1. Context selection

The context of the experiment relied on a study applied for evaluating obtained results of the B.A.R.T Search Engine on searching and retrieval of source code by using different search mechanisms, where some query were formulate in other to verify the quality of the results in terms of precision and recall.
5.3.2.2. Hypothesis formulation

In order to achieve the proposed goal, were elected these questions to be answered:

1. Does the faceted-approach adopted in the classifier engine have contributed for increasing the recall of B.A.R.T Search Engine?
2. Does the faceted-approach adopted in the classifier engine have contributed for increasing the precision of B.A.R.T Search Engine?
3. Is the proposed solution based on the faceted approach viable and practical mechanism to be part of the B.A.R.T Search Engine, especially for commercial purposes?

In order to answer these stated questions, some hypotheses were raised. In this sense, some criteria were elected to guide the answers proposition and are detailed in the following.

The usual approach for evaluating the performance of information retrieval systems is based on precision and recall metrics by utilizing a large dataset along with a set of queries and expected responses [Baeza-Yates and Ribeiro-Neto, 1999]. In this sense, the metrics adopted to evaluate and verify these questions are: precision, recall and f-measure that correspond to the harmonic mean of precision and recall. The precision and recall are based on sets of retrieved documents (list of documents produced by the search engine for a query) and a set of relevant documents (list of all documents that are relevant for a certain topic). Each metric as well the formula for its respective calculus are detailed as follows:

**Precision** is the fraction of a search output that is relevant for a particular query [Makhoul et al., 1999]. Its calculation is based on the knowledge of the relevant and non-relevant hits in the evaluated set of documents [Clarke and Willett, 1997]. Thus it is possible to calculate absolute precision of search engines which provides an indication of the relevance of the system. In the context of the present study precision is defined as [Makhoul et al., 1999]:

\[
\text{Precision} = \frac{\text{Number of Relevant Documents}}{\text{Total Number of Documents Retrieved}}
\]
Recall is the ability of a retrieval system to obtain all or most of the relevant documents in the collection. Thus it requires knowledge not just of the relevant and retrieved but also those not retrieved [Clarke and Willett, 1997]. The recall value is thus defined as [Makhoul et al., 1999]:

\[
\text{recall} = \frac{|\text{relevant documents} \cap \text{retrieved documents}|}{|\text{relevant documents}|}
\]

**F-measure** is the weighted harmonic mean of precision and recall. In the traditional F-measure, the recall and precision are evenly weighted. The formula of F-measure is as follows [Makhoul et al., 1999]:

\[
F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.
\]

To this experiment, the precision and recall rates have the same factor of importance and, as much as the closer a search mechanism is of 1.0, better it is. However, this only happens if both precision and recall are high. In case of high disparity between precision and recall rates, f-measure tends to zero, indicating that this mechanism does not play one of these criteria very well.

The null hypotheses (Ho) approach was applied and the hypotheses intended to be rejected will indicate that the hypotheses were not fulfilled. For each elected metrics the following hypotheses were stated:

**Recall Hypotheses**
- **Ho\text{a}:** the keyword-based mechanism has higher recall than the facet-based + keyword;
- **Ho\text{b}:** the semantics has higher recall than the facet-based + keyword;

**Precision Hypotheses**
- **Hoc:** the keyword-based mechanism has higher precision than facet-based + keyword;
Chapter 5 – The Experimental Study

- **H₀d**: the semantic mechanism has higher precision than the facet-based + keyword;

**F-measure Hypotheses**

- **H₀e**: the keyword-based mechanism has higher f-measure than the facet-based + keyword;
- **H₀f**: the semantic mechanism has higher f-measure than the facet-based + keyword;

From the optimistic view, **H₀e** hypothesis is expected to be favored, besides the entire **null hypotheses** being rejected, since the semantic and keyword-based + facet-based mechanisms tend to obtain equivalent results.

### 5.3.2.3. Variables selection

In the context of this experiment, the independent variables refer to the search mechanisms evaluated in the experiment such as: **keyword-based, semantic** and the combination between **keyword and facet**. The results from previous search mechanism (**keyword and semantic**) will be compared against the combination between **keyword and facet** to compare the evolution of them.

Thus, the independent variables are: keyword, semantic, keyword + facet-based. The dependent variables are the **precision, recall** and **f-measure** that correspond to the harmonic mean of **precision** and **recall**.

### 5.3.2.4. Selection of subjects

The study was dealt as a single object study, whose studies are performed on a single team and in a single project. The participants were selected based on convenience - all are RiSE members and are the M.Sc. and Ph.D. students in Software Engineering from the Federal University of Pernambuco, Brazil. In addition, the query entries proposed were stated by these participants.

### 5.3.2.5. Experiment design

The design of the experiment was based in the following principles:
Chapter 5 – The Experimental Study

- **Randomization**: All query entries were performed for each search mechanism: keyword-base, semantics and keyword + facet-based.
- **Blocking**: No systematic approach to blocking was applied since all the source code elements should be evaluated; and
- **Balancing**: Although a balanced data set is preferable, some domains have more related Java source code than others. However it does not influence the results since in a real environment is not guaranteed that elements will be available equally balanced.

5.3.2.6. Instrumentation

Since the focus of this evaluation is the precision of the faceted classification by domain, the instrument used to validating was based on a comparison among search mechanisms supported by the *B.A.R.T Search Engine*.

5.3.2.7. Validity evaluation

To provide a valid test result, were defined several threats to the validity of the experiment. Internal validity is dependent of two aspects: query entries stated and the size of the evaluated data set. Regarding these aspects, was assumed that the quantity and quality of the queries and the evaluated data set chosen provided a good internal validity.

Respecting to external validity, a risk related was the expert’s motivation, since they could perform the study without responsibility or without a real interest in performing the project with a good quality as it could happen in an industrial project. Since the goal was to evaluate the precision of search engine and the expected results were already established in [Durão, 2008], the external validity was considered sufficient.

Regarding threats to the conclusion validity, one threat was identified related to the result not satisfies the expectations and then the faceted approach adopted may require more refinement. However, once the query entries and expected results were already defined, it was not considered a potential threat to impact in the proposed evaluation.
5.3.3. Operation

5.3.3.1. Preparation

The experimental study was conducted in two main activities: Classify the dataset using the Facet Classifier Engine and compare the B.A.R.T Search Engine extension supporting facets with the results obtained by previous keyword-base and semantics mechanisms presented in [Durão, 2008].

5.3.3.2. Execution

Both activities were conducted during from July 2008, at Federal University of Pernambuco. In order to establish an equivalent experimentation scenario, the query entries applied in the B.A.R.T Search Engine extension as also the evaluated collection data were the same as used in [Durão, 2008]. In the Appendix, more details about the experiments resources are available.

5.3.3.3. Data validation

In order to compare the search mechanism evaluated in the experiment, 10 query entries were formulate and applied in the B.A.R.T Search Engine using each mechanisms. More details are described in Appendix.

5.3.4. Analysis and interpretation

This section describes the analysis and interpretation of the experimental study. For each metrics applied, recall, precision and f-measure, were calculated the arithmetic average, the standard deviance and the variance for the total of queries performed.

- **Arithmetic Average**: the mean is a way to describe the location of a distribution.
- **Standard Deviance**: is a measure of the dispersion of the results;
- **Variance**: is one measure of statistical dispersion, and is a way to capture its scale or degree of being spread out.
Chapter 5 – The Experimental Study

5.3.4.1. Recall Results

In Table 5 is presented the obtained results regarding the recall measure according to arithmetic average, standard deviance and variance values.

**Table 5. Achieved Recall Rates.**

<table>
<thead>
<tr>
<th>Search Mechanism</th>
<th>Recall</th>
<th>Standard Deviance</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keyword</strong></td>
<td>0,81</td>
<td>0,21</td>
<td>0,04</td>
</tr>
<tr>
<td><strong>keyword + facet-based</strong></td>
<td>0,65</td>
<td>0,29</td>
<td>0,09</td>
</tr>
<tr>
<td><strong>Semantic</strong></td>
<td>0,76</td>
<td>0,15</td>
<td>0,02</td>
</tr>
</tbody>
</table>

According to Table 5, the keyword mechanism has achieved the highest recall followed by the semantic and the *facet-based + keyword* mechanism. All the obtained values are considered significant since are higher than values obtained and also considered satisfactory by other authors (50% for the recall) [Ye and Fischer, 2002], [Frakes and Pole, 1994]. In general, the evaluated mechanisms had 15-30% of standard deviance and very low variance (< 10%) which means that their findings were close to the recall average and not disparity was evidenced. Although the recall of the *keyword + facet-based* mechanism is slightly lower than the others, which indicates that *keyword + facet-based* mechanism, even with the restriction of the domain, did not left many source codes behind. In this sense, high precision is expected since the search is focused on a specific set of classes.

According to the results obtained and formal hypotheses which analyze the recall, both null hypotheses *H₀a* and *H₀b* were not rejected.
5.3.4.2. Precision Results

Table 6 shows the precision measure according to arithmetic average, standard deviation and variance values.

<table>
<thead>
<tr>
<th>Search Mechanism</th>
<th>Precision</th>
<th>Standard Deviance</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
<td>0.57</td>
<td>0.17</td>
<td>0.044</td>
</tr>
<tr>
<td><em>keyword + facet-based</em></td>
<td>0.84</td>
<td>0.32</td>
<td>0.10</td>
</tr>
<tr>
<td>Semantic</td>
<td>0.79</td>
<td>0.14</td>
<td>0.02</td>
</tr>
</tbody>
</table>

According to Table 6, the *facet-based + keyword* mechanism has achieved the highest precision followed by the semantics and keyword mechanism. This mechanism had obtained higher results than the referenced values obtained by other authors (20% for precision) [Ye and Fischer, 2002] [Frakes and Pole, 1994]. Furthermore, it also had low standard deviance (no more than 32%) and very low variance, which means that the findings were close to the precision average and not disparity among the obtained results was evidenced.

Expectations over the keyword + facet-based mechanism were achieved. In this direction, it seems that searches focused on the specific domain are closer to user needs, but to achieve this is necessary an efficiency classification.

According to the results obtained and formal hypotheses which analyze the precision, the *null hypotheses* $H_0c$ and $H_0d$ were rejected.

5.3.4.3. F-Measure Results

Table 7 shows the $f$-measure measure according to arithmetic average, standard deviation and variance values.

<table>
<thead>
<tr>
<th>Search Mechanism</th>
<th>F-</th>
<th>Standard Deviance</th>
<th>Variance</th>
</tr>
</thead>
</table>

Table 7. Achieved F-Measure Rates.
According to Table 7, the semantic mechanism achieved the highest f-measure rate followed by the facet-based + keyword and keyword mechanism, since in this experimentation the recall and variance have equally significance for the f-measure value. In addition, to the pleasing values achieved for recall and precision, the low discrepancy among them has contributed for a highest f-measure rate.

Although no referenced value is used to compare with the obtained values, all mechanisms have achieved values above 50% of the maximum expected and then may be considered as satisfactory. Again, all mechanisms have obtained low standard deviance and variance values, which means that no disparity between the f-measure findings and its average has happen. According to the results obtained and formal hypotheses which analyze the f-measure, the null hypotheses $H_{0e}$ was rejected and $H_{0f}$ was not rejected.

5.3.5. Summary and conclusions

According to the presented results, the $H_{0a}$, $H_{0b}$ and $H_{0f}$ were the null hypothesis not rejected and $H_{0e}$, $H_{0c}$ and $H_{0d}$ were the null hypothesis rejected. The obtained results indicate that keyword + facet-based mechanism provide a more efficient search and retrieval mechanism than keyword approach and equivalent efficiency comparing with semantics approach.

Table 8, summarizes the mean of recall, precision and f-measure achieved during the experiment evaluation. From the precision value perspective, the keyword + facet-based approach obtained the highest precision rate among the mechanisms compared. On the other hand, the recall of the keyword + facet-based mechanism was the lowest, which means that adjustment must be done.

Table 8. Achieved Results.
Chapter 5 – The Experimental Study

<table>
<thead>
<tr>
<th>Search Mechanism</th>
<th>Recall</th>
<th>Precision</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
<td>0.81</td>
<td>0.57</td>
<td>0.65</td>
</tr>
<tr>
<td><strong>keyword + facet-based</strong></td>
<td>0.65</td>
<td>0.84</td>
<td>0.71</td>
</tr>
<tr>
<td>Semantic</td>
<td>0.76</td>
<td>0.79</td>
<td>0.75</td>
</tr>
</tbody>
</table>

In a general overview, the recall and precision of the *keyword + facet-based* approach rates have reached reference values obtained by other authors (50% recall and 20% for precision) [Frakes and Pole, 1994] [Ye and Fischer, 2002]. This indicates that the *keyword + facet-based* place the *B.A.R.T Search Engine* in the same level of others search engines allowing be comparable in terms of recall and precision. Although this significant results, it still premature to consider this approach recommended for integration in B.A.R.T for commercial purposes, since more massive tests are needed.

In addition, it is not safe to assure which mechanism is the best or that the *keyword + facet-based* mechanism exclude the usage of other mechanism which had less performance in the experiment, requiring further experiments with other dataset (with different projects, domains and size) and queries. In this sense, continuous evaluations can determine the real effectiveness of the tool, but the obtained results indicate the research can provide potential benefits.

Figure 17 shows line graphic that outlines the evolution of search mechanisms of *B.A.R.T Search Engine* in terms of recall, precision and f-measure. According to the results, in scenarios where the precision is more valuable than the recall, the *keyword + facet-based* mechanism is more suitable than the others approaches. Likewise, in scenarios were higher recall is preferable, keywords and semantics seems more recommendable.
Chapter 5 – The Experimental Study

Figure 17. The evolution of B.A.R.T search mechanisms.

The B.A.R.T evolution has been evidenced since the search mechanisms incorporated have contributed to converge the recall and precision rate which yields the augmentation of the f-measure.

5.4. Chapter Summary

In this chapter, main aspects related to the experiment regarding the application of facet-based approach to the B.A.R.T in terms of the precision and recall on a set of real projects. The results evidenced that more refinement is necessary for the information retrieval engine, but also showed the proposed approach consists on a valuable resource for increasing the precision of B.A.R.T Search Engine and consequently in achieving a higher reuse activity level.

Next chapter concludes this dissertation by summarizing the experimentation on this chapter, reviewing some related work, pointing directions for future enhancements to the environment and presenting some final considerations.
In previous chapters were discussed the pillars of this work: *Software Reuse*, *Information Retrieval* and *Faceted Approach*. Moreover, details of the conception and implementation of the tools that are part of the contributions were presented. In this chapter, some conclusions and comparisons can be drawn and directions to future work are pointed out.

This chapter is organized as follows: Section 6.1 summarizes the achieved goals of the work and Section 6.2 presents a comparison with some related work. Section 6.3 points out some directions for future work unexplored by this work and, finally, Section 6.4 contains a closing discussion on the topics covered in this dissertation.

### 6.1. Achieved Goals

The main contributions of this dissertation were a proposal of classification scheme based on the faceted approach, a tool to support the domain specialist on analyze source code and classify into facets and, integration of the facet categories with a keyword-based search engine in order to increase the precision of search returns. The proposed solution utilizes the faceted approach for enhancing the search and retrieval process by narrowing the search scope and a Java desktop tool to aid the domain experts. The implemented proposal, with some refinements and adjustments, configures an *initial* and *practical* solution for being utilized in an industrial scenario in favor of source code reuse.

According to the experimentation results, was noticed a sensible increase of precision during the code searches. In addition, this work showed that the synergy among the areas applied in the study such as *Software Reuse*,
**Information Retrieval** and **Faceted Classification Approach** may provide efficient solutions intended for practical usage in software development factories. The faceted classification approach reduces the problems evidenced in assets location process and, in a certain perspective, reduces the conceptual gap since the facets contains semantic since are composed by terms that best describes the assets. The facet scheme vocabulary was used to expand the user query and then increase the precision of the search.

The extension of B.A.R.T, a search engine tool, contributed to the reuse activity, specially source code reuse, since enhance the search and retrieval process. The *Facet Classifier Engine* was designed to attend **extensibility** and **reusability** quality attributes, which allows implement more robust facet classification algorithms and also extend the facet scheme.

The *DoFaST* tool, showed an initiative focused on ease the domain expert activities especially analysis and facet classification of source code, by providing visual and statistical resources. Although valuable contribution, was evidenced that more complex and valuable features such as other classification engines support may be implemented in order to enhance the tool contributions.

The evaluation results of the *B.A.R.T Search Engine* extended with faceted approach has obtained interesting results, but some fine tuning are still needed; and the evaluation of the proposed solution based on a formal experiment that can be repeated and extended with other dataset in terms of content and size.

### 6.2. Related Work and Research Foundations

The basis of this contribution relied on **Software Reuse** and **Information Retrieval** and **Facet Classification Approach**. The *B.A.R.T Search Engine* supporting *Faceted Classification* and the *DoFaST* application have concentrated their efforts on providing an integrated set of resources aiming to increase the reuse activity in organizations.

In this section some related work are discussed concerning these pillars that were the motivation of the proposed solution.
6.2.1. Software Reuse

Since the first initiatives of software reuse [McIlroy, 1968], this field is been widely explored by research community and also the industry. The proposition of increase productivity, reduction of costs and still attend the time-to-market expectations makes reuse practice a key factor to be more competitive. These premises were the main motivation to proposals in this direction, such as Component-Based Development Process [Heineman and Council, 2001], [Neto et al., 2004], Domain Engineering [Frakes et al., 1998], [Almeida, 2007], and Software Product Lines [Atkinson et al., 2000].

In addition, a set of tools in promoting for facilitating the creation and accessing to the reusable software assets also have being developed such as Component Search Engines [Ye and Fischer, 2002], [Garcia et al., 2006b], Reuse Repositories [Burégio, 2006], Reengineering Tools [Brito, 2007], Domain Analysis Tools [Lisboa et al., 2007].

Initiatives concerning the quality and liability of reusable component are under research such as systematic policies, reuse best practices and certification process [Alvaro et al., 2006].

Regarding source code reuse, since the historical appealing of the importance of source code in the development cycle [McIlroy, 1968], source code has gained importance in reuse activity, leading the reuse industry put more effort on development of tools such as IDEs, search engines [Garcia et. al, 2006a] and reuse repositories [Burégio, 2006] to have more benefits from code reuse. In addition, according to Mili et al. [Mili et al., 1995], the potential reuse range may vary from **15 to 85 percent**. Moreover, the practice of large-scale source code reuse may provide potential savings in effort and improvements in quality and increase productivity [Mokus, 2007].

In this direction, one of the propositions of this work was to extend the B.A.R.T Search Engine for supporting the faceted approach as an alternative solution to classify source code. In addition the main idea of this proposition relies on providing a complementary solution in order to increase the precision of the existing returned source code assets, in order to promote the reuse activity.
6.2.2. Information Retrieval

The synergy of software reuse and information retrieval, has been yield many work [Henninger, 1994], [Thomason et al., 2000], [Ye and Fischer, 2002], [Sugumaran and Storey, 2003]. Some of them are presented in this work and a resume of each one is described as follows:

The CodeFinder is a source code search engine proposed by Henninger (Henninger 1994) and uses query-construction methods to aid the user on defining the query search. The main goal is to ease location of software items, since even for well-informed software designers, when searching in large, complex, and the library is continuously growing, finding components becomes more difficult. Facet-based approaches are suitable for this scenario since are flexible and contains a high descriptive power [Pietro-Díaz, 1991].

In 2000, [Thomason et al. 2000] was presented CLARiFi, a component-based system that provides an classification scheme that is responsible for identifying relevant component properties that meets the user searcher expectations. Classification schemes implemented by automated categorization techniques are practical instruments for augmenting the knowledge about what is retrieved. In this dissertation, the use of the faceted approach on classification of the source code goes in same direction, since the facet scheme, once conceived, automatically classify the source code and it classification is helpful to narrow or broaden the user search queries.

In 2002, Ye and Fisher [Ye and Fisher, 2002] have proposed a context-based retrieval tool, CodeBroker, based on information retrieved from user environment and return source codes that have similarities with the code under development. The main purpose of the CodeBroker is to reduce the “Semantic Conceptual Gap” existing between the user needs and its computational representation. Based on this premise, the Facet Classifier Engine proposed uses facets that best describes the component and thus, reducing this conceptual gap.
Chapter 6 – Conclusion

6.2.3. Faceted Approach and B.A.R.T Search Engine Extension

Many researchers have pointed that search mechanisms have a key role on information handling [Singer et al., 1997], [Henninger, 1997]. The **Faceted Approach** was introduced by Ranganathan [Ranganathan, 1967] and promotes synthetic classification where classes are assembled by selecting predefined keywords from facet scheme. A **classification scheme** is a tool for the production of systematic order based on a controlled and structured index vocabulary [Prieto-Díaz, 2004].

Kwasnick [Kwasnick, 1999] defined faceted classification as a set of mutually exclusive and jointly exhaustive categories, each made by isolating one perspective on the terms (a facet), that combine to completely describe all the objects in question, and which users can use, by searching and browsing, to find what they need.

In spite of the classification process of assets by using the facet-based approach, there are basically three main concepts to be achieved [Damiani et al., 1999]:

- Facets Scheme Conception;
- Asset Analysis; and
- Ranking/Classification.

In order to systematize the process of facets scheme creation, Spiteri [Spiteri, 1998] divided the facet scheme creation process into three principles:

- Idea Plane;
- Verbal Plane; and
- Notational Plane.

Some reports in literature have suggested automated approaches to generate the facets scheme. In [Stoica et al., 2007] is proposed the Castanet, Damiani [Damiani et al., 1999] presented the Corrigenda, [Wong et al, 2007] presented Skyline, a research on using data mining to select facets. Dakka et al. [Dakka et al., 2008] introduces an approach to generate facets in a three-step
algorithm based on interaction between external/on-line search engines and applications.

In spite of category systems that support uses faceted approach to classify the data, ClassServer proposed by [Mili et al., 1994] is an experimental library tool developed that focuses on evaluating issues in software reuse. Flamenco presented by Hearst [Hearst, 2006b] and based on English et al. work [English et al., 2002], is search interface framework that its main purpose is allowing users to move through large information spaces in a flexible manner.

Wu et al. [Wu et al., 2007] presents a system that proposes evolve a scheme with facets and categories based on large group contributions.

In a source code classification context, the faceted approach provides suitable mechanisms to classify reusable components [Prieto-Díaz, 1991] such as:

- Flexibility to deal with expanding collections;
- High precision and descriptive power;
- Low level of effort to maintenance; and
- Can be partially automated.

Based on the pillars of this work (Software Reuse, Information Retrieval and Facet Classification Approach) and also on aspects to be improved evidenced from the analysis of tools developed so far [Prieto-Díaz, 1991], [Mili et al., 1994], [Hearst, 2006b], [Garcia et al., 2006b], [Vanderlei et al., 2007], and reviews of the state-of-the-art [Lucrédio et al., 2004], this dissertation proposed a extension of B.A.R.T Search Engine, supporting facet-based approach to classify the source code assets.

Another contribution of this work is the DoFaST, a tool proposition for aiding the domain expert by providing a set of functional resources such as graphical components management and facet classifier engine integration.

The next section outlines future work proposed for the current development.
6.3. Future Work

In spite of the proposition of applying facet-based approaches in a version of the B.A.R.T Search Engine, some enhancements are evidenced, since the initial goal was to demonstrate the viability of this approach in an academic level. In this direction, some aspects take place:

**Facet Browsing** – Some reports have showed that the faceted approach can also provide expressive benefits in the interface field, such exploratory interactions and discovery in large collections [Hearst, 2006a], [Dakka et al., 2008];

**Artifact Types Support** - New artifact types and formats must be supported in order to provide the search with a broader coverage range. Documents written in natural language, in particular, certainly will necessity the extension of the new facets and new classification algorithms more suitable to each kind of artifact;

**Other Text Categorization Techniques Support** - Several initiatives have applied different learning techniques for text categorization: bayesian models, nearest neighbor classifiers, decision trees, support vector machines (SVMs) and neural networks. In this direction, SVMs have been shown promising results for text categorization [Krovetz, 2003]. New methodologies and strategies regarding the analysis source code process can also be applied for more effective classification engines [Binkley, 2007];

**Search Engine refinements** – In spite of the search engine, some reports [Happel et al., 2008] presents more efficient approaches that can also be applied in order to provide more query formulation, ranking, and other fields. The combination of different approaches can also provide expressive results as presented by Pietro-Díaz [Pietro-Díaz, 2004]. Likewise, the integration with context-aware mechanisms [Santos et al., 2006] may empower the faceted approach since, for example, can be used to suggest potential facets regarding the context identified; and

**New Experiments** - New experiment configurations for further evaluation may be tacked involving the repetition of the experiment with new
dataset through the use of new projects or comparing the *B.A.R.T Search Engine* using the *faceted approach* against related tools by sharing a common experiment environment.

### 6.4. Concluding Remarks

The contributions of this dissertation are a result of a consistent research relied on three pillars: *Software Reuse*, *Information Retrieval* and *Facet Classification Approach*. In this chapter, a general overview about these three fields was presented, although no existing solution is integrally equivalent to the proposed solution. In addition, some related work that inspired some aspects adopted to the development of this work were presented in this chapter and also introduced the future work planned to be realized in the future versions.

The *extension of B.A.R.T Search Engine*, contributed to the reuse activity, source code reuse in particular, since enhances the search and retrieval process. The *Facet Classifier Engine* attempts to provide source code classification more precise and consequently increase the practice of reuse activities by the user by reducing the semantic gap.

The *DoFaST*, showed an initiative focused on ease the *Domain Expert* activities related to source code classification using the *faceted approach*, by providing visual and statistical resources. Although significant contribution, more refinement is required in order to obtain more effective benefits. Likewise, in the experimentation study shown that the evaluation results of the *B.A.R.T Search Engine extended with faceted approach* has obtained interesting results, but some fine tuning is still needed to commercial purposes.


References


References


References


References


References


References


References


References


Towards an Effective Context-Aware Proactive Asset Search and Retrieval Tool. *Sixth Workshop on Component-Based Development (WDBC)*, Recife, Brazil, p. 105--112.


References

**Brokerage.** *International Workshop on Component-Based Software Engineering (CBSE 2000)*. Limerick, Ireland.


As previously stated in Chapter 5, each search mechanism had a specific filter query during the test execution. Table 9 presents 10 queries utilized during the tests for each search mechanism: keyword-based, semantic and keyword + facet-based. In addition, the objective of each query was established with the purpose of avoiding eventual ambiguity about the expected functionalities.

The choice of the queries was based on common needs shared among developers; however, it is already thought to raise run queries derived from statistic analysis from the most frequent ones.

**Table 9. Queries utilized in the evaluation.**

<table>
<thead>
<tr>
<th>Keyword Query</th>
<th>Objective</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>To return source code that performs reading, writing, compressing, file stream transference and byte handling.</td>
<td>i/o</td>
</tr>
<tr>
<td>resultSet</td>
<td>To return source code that performs database operations such as insert, update or select.</td>
<td>Database</td>
</tr>
<tr>
<td>Dialog</td>
<td>To return source code that builds a comprehensive dialog box to be used as a user graphical interface.</td>
<td>GUI</td>
</tr>
<tr>
<td>connection</td>
<td>To return source code that performs database connection and catch applicable exceptions for that.</td>
<td>Database</td>
</tr>
<tr>
<td>Request</td>
<td>To return source code that handle network request and session management.</td>
<td>Network</td>
</tr>
</tbody>
</table>
Appendix

<table>
<thead>
<tr>
<th>Hash</th>
<th>To return source code that performs any hash calculus with security purpose.</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>To return source code that performs parse operation over xml files to access and manipulate its content.</td>
<td>Xml</td>
</tr>
<tr>
<td>Decode</td>
<td>To return source code that performs calculus for decoding messages.</td>
<td>Math</td>
</tr>
<tr>
<td>Buffer</td>
<td>To return source code that performs buffering of file stream.</td>
<td>I/O</td>
</tr>
<tr>
<td>http</td>
<td>To return source code that manipulates the protocol for establishing a connection and exchange information through a network.</td>
<td>network</td>
</tr>
</tbody>
</table>

Although Table 9 does not show the “expected results” column, this information was carefully analyzed in order to compare it with returned source codes after the test execution.

The open-source projects accessed to extract the classes which compound the dataset are: SoapUI, MvCase, Jackrabbit, Java Pilot-DB, RIFE, Protomatter, Ostermiller, Jeeves, Maracatu e Ganttproject. These projects can be downloaded in http://www.cin.ufpe.br/~rcm2/testBase.zip.