EXAME DE QUALIFICAÇÃO (QUALIFYING EXAMINATION)

ON THE USE OF VISUALIZATION FOR SUPPORTING SOFTWARE REUSE

Marcelo Schots de Oliveira

Orientadora (advisor): Cláudia Maria Lima Werner

Rio de Janeiro

Abril de 2014

Abstract of Qualifying Examination presented to COPPE/UFRJ as a partial fulfillment of the requirements for the candidacy to the degree of Doctor of Science (D.Sc.)

ON THE USE OF VISUALIZATION FOR SUPPORTING SOFTWARE REUSE

Marcelo Schots de Oliveira

April/2014

Advisor: Cláudia Maria Lima Werner

Department: Systems Engineering and Computer Science

Software reuse is a very common and widespread concept nowadays. Reuse activities are present in the daily routine of software developers, yet mostly in an ad-hoc or a pragmatic way. A well-defined reuse program allows for reducing the effort and time spent on software development. Nevertheless, despite the extensive literature on this topic, organizations still struggle in beginning and coping with a reuse program.

A study on software visualizations targeted to reuse-related tasks pointed out that no work so far addresses a number of reuse tasks in an integrated way, and the existing ones that address particular tasks are limited in terms of collecting information from data sources and supporting the customization and selection of visualizations. Besides, most of them do not provide enough evidence on their effectiveness.

In this sense, this thesis proposal aims at providing software visualization mechanisms through an environment called APPRAiSER for supporting stakeholders (mainly reuse managers and developers) in executing software reuse tasks, such as exploring a reuse repository, obtaining and understanding information regarding reusable assets, and monitoring reuse initiatives. The long-term goal is to help introducing, instigating, establishing, and monitoring software reuse initiatives, by increasing efficiency and efficacy while decreasing the effort and time spent by stakeholders in performing such tasks.

SUMMARY

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1 – INTRODUCTION

This chapter presents the motivation for this work, the research questions that guided the proposal of the approach, as well as its goals and the adopted research methodology.

1.1 Foreword

Software systems permeate advances in all areas of knowledge, and there is an increasing participation of software in society [Brazilian Computer Society 2006]. Such systems are embedded in everyday devices such as household appliances, and support areas such as communication (e.g., mobile devices, social media etc.) healthcare (e.g., blood pressure and glucose monitors, electronic patient records, clinical exams, artificial pacemakers etc.), context-awareness (e.g., place recommenders based on location awareness, smart houses – for healthcare, energy saving etc.) and so on. In a fast-paced world that is continuously changing across all areas, there is a large demand for new functionalities, and there is a broad field of software-related opportunities that increase each day.

The demands of each of these fields (among others) and the advances of mobile devices, social media, and new technologies over the years have been strongly influencing the way software systems are built. Traditional approaches no longer meet the demands of new circumstances, and have given rise to the emergence of new forms of development, including agile methodologies [Fowler & Highsmith 2001] and the widespread open source practices [Haefliger et al. 2008]. The steady growth of distributed software development has also made the scenario more complex and, at the same time, more stimulating and challenging [Schots et al. 2012].

This scenario not only has led to a large-scale adoption of new technologies, practices and methodologies for software development, but also has required a smaller and ever decreasing time-to-market. Software organizations, in turn, need ways to make software development as efficient as possible in order to cope with the increasing demands.

One of the ways of supporting the demand for delivery of software systems in less time with less effort is through software reuse. Software reuse has become a very

common and widespread concept in software development, and has been a promising paradigm in software engineering since its inception [Benedicenti et al. 1996], given that it can be fully integrated and supported in the software development process, improving the life cycle by reducing the effort and time needed to develop a software system.

Since the beginning of software engineering, much research has been done in developing techniques and tools for supporting software reuse [Naur & Randell 1968] [Mili et al. 1995] [Frakes & Kang 2005], which include cataloging, retrieval and storage of reusable assets[1]. Moreover, several approaches have been proposed or adapted to support development for and with reuse, such as domain engineering techniques [Kang et al. 1990], software product lines [Clements & Northrop 2002] [Fernandes et al. 2011], and so on.

1.2 Motivation

Reuse activities are present in the daily routine of software developers, yet mostly in an ad-hoc or a pragmatic way. Despite the maturity of software reuse research and the extensive literature available, organizations still have difficulties in understanding and implementing reuse practices. Some issues and challenges related to implementing software reuse processes and establishing an effective reuse program[2] include:

- the lack of understanding of software reuse concepts and how to effectively apply them [Morisio et al. 2002] [Schots & Werner 2013];
- the lack of acceptance of reuse practices by the development team and top management in software organizations [Benedicenti et al. 1996] [Sametinger 1997] [Sherif & Vinze 2003];
- the lack of knowledge and experience for the creation and management of reuse repositories [Morisio et al. 2002] [Schots & Werner 2013] and the definition, identification and evaluation of reusable assets [Schots & Werner 2013], as well as making such assets available and findable [Frakes & Kang 2005];

[1] In this work, any item built for use in multiple contexts, such as software designs, specifications, source code, documentations, test cases etc. can be considered as reusable assets [IEEE 2010].
[2] A reuse program is an organizational mechanism that establishes the goals, scope, and strategies for addressing issues related to business, people, process, and technology involved in the adoption of software reuse. The term “reuse program” should not be confused with “a software application for reuse”.

- a long learning curve of understanding a software asset, i.e., its structure, behavior and functionality [Ye & Fischer 2002] [Marshall et al. 2003] [Anslow et al. 2004] [Frakes & Kang 2005];
- the lack of proper tool support for performing software reuse tasks [Benedicenti et al. 1996] [Morisio et al. 2002] [Schots & Werner 2013];
- the absence of a culture of development for reuse in development teams and the lack of systematization for the construction of reusable assets [Prieto-Diaz & Arango 1991] [Sherif & Vinze 2003];
- the "Not-Invented-Here" (NIH) syndrome [Sametinger 1997] [Sherif & Vinze 2003], i.e., the difficulty of accepting and trusting third-party developed assets, resulting in a tendency towards "reinventing the wheel" (recreating something from scratch instead of reusing) based on the belief that in-house developments are inherently better than existing implementations.

As it can be seen, introducing a software reuse program must also envision non-technical aspects, e.g., engagement of team members and managerial support [Kim & Stohr 1998]. Sherif and Vinze (2003) highlight that reuse provides better results when all stakeholders are committed to it [Sherif & Vinze 2003]. In this sense, two crucial concerns for facilitating the acceptance/consciousness and adoption of reuse are how to increase the visibility of reuse results and how to provide appropriate reuse awareness. Awareness mechanisms allow stakeholders to be percipient of what goes on in the development scenario [Treude & Storey 2010], and can provide them with the necessary information and support for performing their reuse-related tasks.

One of the ways to increase awareness is by employing visualization resources and techniques [Hattori 2010]. Software visualization has been researched as a way to assist software development activities that involve human reasoning, helping people to deal with the large amount and variety of information by providing appropriate abstractions [Lanza & Marinescu 2006] [Diehl 2007]. In the software reuse scenario, its use can allow awareness and comprehension of reuse elements (i.e., assets, users, and development projects) and their surroundings.

Despite the potential of software visualization on supporting software reuse, little work has been done with this goal, and the existing ones (e.g., [Alonso & Frakes 2000], [Marshall et al. 2003] and [Anslow et al. 2004]) do not take into account the different reuse stakeholders' information needs. Besides, they are very limited in terms

of visual customization and integration with other information sources, not providing enough evidence on their effectiveness [Schots et al. 2014].

Thus, it is believed that software visualization resources can be better investigated and explored in order to provide and increase awareness of the reuse scenario and support reuse-related tasks, such as exploring a reuse repository, obtaining, and understanding information regarding reusable assets, and monitoring reuse initiatives. Through visual abstractions, one can better comprehend the reuse elements and their surroundings. This is the focus of the environment proposed in this work.

1.3 Hypothesis and Research Questions

Considering that

i. software reuse brings several benefits to the software development scenario, reducing the cost and effort necessary for the construction of new software systems, thus allowing a "better" time-to-market and bringing a potential increase of quality;
ii. the establishment of a reuse program can facilitate reuse management in software development organizations, allowing for an increase of maturity towards systematic reuse;
iii. besides supporting stakeholders in their reuse-related tasks according to their roles, it is important to provide them with visibility of the obtained results, so that they can become committed with software reuse by perceiving the benefits brought by it;
iv. the difficulty in finding and understanding reusable assets may lead potential consumers ("reusers") to prefer to recreate from scratch an existing solution, due to the lack of available information (or lack of organization of the existing information) related to such assets;
v. software visualization techniques and resources allow awareness and comprehension of the structure, behavior and evolution of software entities and metadata, assisting software development activities that involve human reasoning;

The hypothesis of this work is:

The use of proper visualization resources can assist stakeholders in carrying out their software reuse tasks, facilitating the institutionalization of a reuse program in software development organizations.

For investigating this hypothesis, the following research questions were derived:

- ***RQ1.*** *What are the characteristics and limitations of the visualization approaches that have been proposed to support software reuse?*

- ***RQ2.*** *Which aspects (comprising stakeholders' needs, reuse tasks, and reuse-related data) should be taken into account for a visualization-based approach to support software reuse?*
- ***RQ3.*** *Are reuse-oriented visualizations feasible in helping stakeholders to be aware of the reuse scenario, allowing the execution of reuse tasks more accurately, increasing their efficiency and efficacy?*

1.4 Goals

The main goal of this work is to investigate, propose, and evaluate the use of visualization resources for supporting software reuse awareness. This generic goal can be decomposed in the following specific goals:

i. Characterize existing works that use visualization for supporting reuse somehow, in order to analyze their features, strengths and limitations;
ii. Identify the needs of stakeholders involved in software reuse tasks, taking into account their role in such tasks;
iii. Identify the necessary features for an approach that aims to support stakeholders in performing reuse tasks;
iv. Define an approach and implement an interactive visualization environment that supports software development organizations to introduce, instigate and monitor software reuse initiatives;
v. Ensure that the proposed approach meets some of the needs identified from the stakeholders, and also increases efficiency and efficacy while decreases the effort and time involved in performing reuse tasks.

This work is also a first step towards meeting the challenges described in [Schots et al. 2012] regarding awareness and comprehension in software and systems engineering, and is related to other works (developed or under development) at COPPE/UFRJ and the State University of Rio de Janeiro (UERJ), as discussed throughout the text.

1.5 Research Methodology

In order to answer the posed research questions, it is important to establish a research methodology. The methodology being adopted in this work is depicted in Figure 1.1, which shows the main research steps (on the left) and how it is intended to

accomplish them (on the right) –, provided that the chosen ways to accomplish each step may vary depending on the findings of the previous steps. The initial steps comprise the characterization of the research topic, and address both RQ1 and RQ2. The results are intended to provide and refine desirable features for proposing and developing a novel approach, which is then evaluated and improved, addressing RQ3.

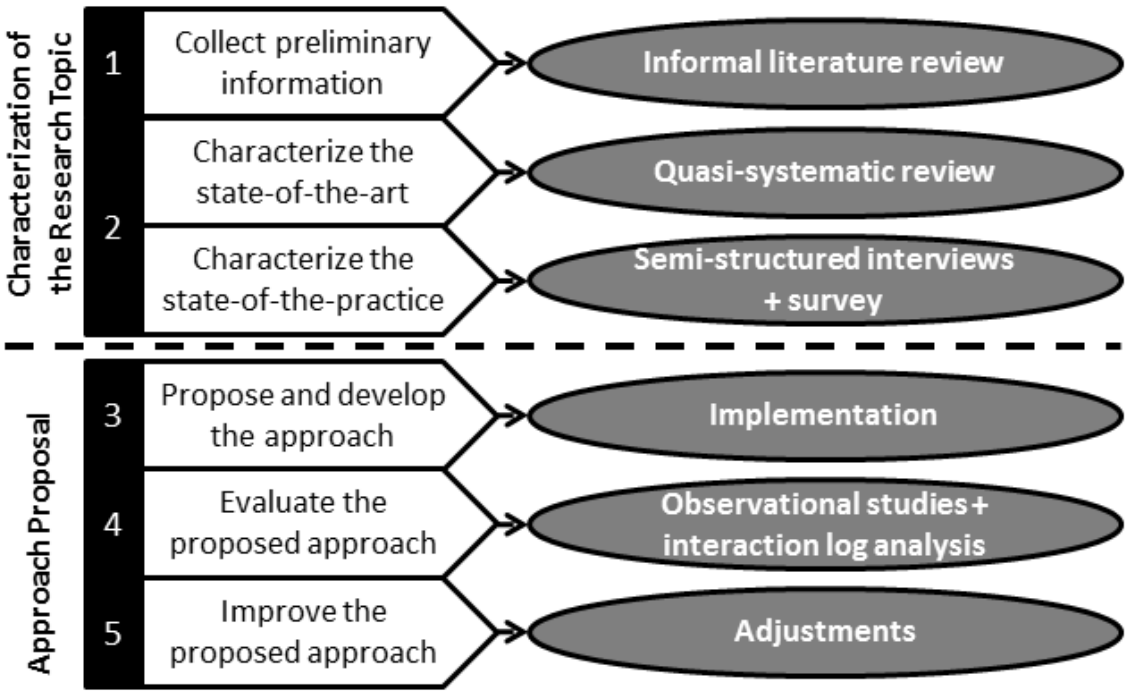

Figure 1.1 – Research methodology

In the first step (*Collect preliminary information*), an *informal literature review* provides the initial/basic knowledge about the research topic, which is subsequently improved in a *quasi-systematic review* that is conducted in the second step (*Characterize the state-of-the-art*), allowing for a broader, more comprehensive view of the topic. *Quasi*-systematic reviews use the same rigorous methodological processes from systematic reviews, looking for the identification of relevant evidence in the research field under investigation, but usually no meta-analysis can be applied [Travassos et al. 2008].

The literature must be reviewed continuously for providing subsides for enriching the definition of the topic and the proposed solution. Therefore, the first step of the research methodology is iteratively executed in this work, and the second step is going to be re-executed afterwards. The main findings of these two steps are presented in Chapter 3, and the detailed description of the planning, execution and analysis of the *quasi*-systematic review can be found in [Schots et al. 2014].

A fundamental part of this research process is to ensure the identification of the information needs of each stakeholder [Schots et al. 2012] from the state-of-the-practice, since such needs may not have been identified in the literature. This is accomplished in parallel to the characterization of the state-of-the-art in the second step of the research (*Characterize the state-of-the-practice*) by means of two primary studies: (i) *semi-structured interviews* with practitioners, and (ii) *surveys* with researchers and practitioners. Semi-structured interviews "are designed to elicit not only the information foreseen, but also unexpected types of information" [Seaman 1999], while surveys allow for a broader coverage of reuse researchers and practitioners. Details about these steps can be found in Chapter 2.

The findings from these steps provide and refine desirable features for the third step of the research methodology (*Propose and develop the approach*), and may also help building a body of knowledge on the topic, in addition to pointing out research opportunities for other works. Through the *implementation* of the approach, an environment for supporting software reuse by awareness must be concretized, according to the defined goals. Chapter 4 describes the definition of the environment and its details.

The feasibility of the proposed approach is evaluated in the fourth step (*Evaluate the proposed approach*), which is intended to assess quantitatively and qualitatively whether the perceptive and cognitive abilities of stakeholders in carrying out software reuse tasks are properly stimulated, by increasing efficiency and efficacy, while decreasing the effort and time spent on such tasks. This involves the planning and execution of studies.

Although this step can only be accomplished after the third step is done, there are some possibilities based on empirical studies on visualization presented in literature (e.g., [Kagdi & Maletic 2008]). An *observational study* is intended to capture firsthand behaviors and interactions that might not be noticed otherwise [Seaman 1999]. *Interaction log analysis* techniques are also intended to enrich the data [Seaman 1999]. More details are described in Chapter 5.

Finally, based on the evaluation results and feedback, the fifth step (*Improve the proposed approach*) takes place, for making the necessary *adjustments* and the identified improvements, if applicable.

1.6 Text Structure

The remaining of this text is organized as follows:

- Chapter 2 (*Software Reuse*) presents the main concepts related to software reuse, including how quality standards and maturity models address this topic, as well as some issues related to the establishment of a reuse program. It also presents a study conducted for characterizing the state-of-the-practice, along with its results.
- Chapter 3 (*Software Visualization*) introduces some concepts related to software visualization, some challenges in awareness and comprehension, and related works identified from the conduction of a *quasi*-systematic review.
- Chapter 4 (*Proposed Approach: The APPRAiSER Environment*) describes the proposed solution and its main elements, including their goals and details on their development.
- Finally, Chapter 5 (*Conclusion*) shows the expected contributions from this work, the results obtained so far, and a tentative schedule for the remaining steps.

CHAPTER 2 – SOFTWARE REUSE

This chapter presents an overview of software reuse, including a brief motivation, a description of how some quality standards and maturity models address this topic, and examples of software reuse tasks. Besides, some common issues related to the establishment of a reuse program are discussed, along with some problems recognized during the implementation and assessment of reuse processes.

2.1 Contextualization

Software reuse is a very common and widespread concept nowadays. According to [Holmes 2008], it has a well-established history in both research literature [Naur & Randell 1968] and industrial practice [Poulin et al. 1993]. One can state that reuse is present in the routine of software developers, yet mostly in an ad-hoc or pragmatic way.

Reuse practices allow achieving a number of benefits, such as reducing the effort and time spent on software development [Naur & Randell 1968] [Krueger 1992] [Poulin et al. 1993] [Mili et al. 1995]. Reusing assets from past projects (i.e., that have been already tested and deployed) also allows developing more reliable applications and decreasing maintenance efforts, since their quality is assured on previous experiences of use [Benedicenti et al. 1996] [Morisio et al. 2002]. Besides, the availability of reusable assets can facilitate newcomers in dealing with new technologies and domains (taking external solutions as a basis for their own development), as well as experienced developers in increasing their productivity by composing existing solutions.

Since the idea of building new software from existing pieces of preexistent software arose [Naur & Randell 1968], it was noticed that several types of artifacts can be reused in software development, such as requirements specifications, software designs, test cases and so on [IEEE 2010]. However, a recent study with Brazilian organizations pointed out that reuse of source code artifacts is still on the mainstream of software development [Schots & Werner 2013]. This has also been reported in other nationwide and worldwide studies, such as [Sá et al. 1997] and [Haefliger et al. 2008].

The concept of reuse of source code is frequently linked to software components, i.e., “self-contained, clearly identifiable artifacts that describe and/or

perform specific functions and have clear interfaces, appropriate documentation and a defined reuse status" [Sametinger 1997]. Hooper & Chester (1991) classify reusable software components into two categories: horizontal and vertical.

Horizontal reuse refers to "reuse across a broad range of application areas, such as data structures, sorting algorithms, and user-interface mechanisms". According to these authors, the assets are typically utilities that are purposely generic for comprising multiple applications [Hooper & Chester 1991].

Vertical reuse, in turn, refers to components within a given application area that can be reused in similar applications that belong to the same problem domain. Although horizontal reuse is better understood and easier to achieve (thus more frequently employed), the greatest reuse potential leverage comes from vertical reuse, due to its potential to build software product lines [Clements & Northrop 2002] and create competitive advantages to the organization [Hooper & Chester 1991].

Reuse can occur within several activities in software development. A reuse-based development model (as illustrated in Figure 2.1) divides activities into two groups [Kim & Stohr 1998]: (i) *producing activities*, involving the identification, classification and cataloging of software resources, and (ii) *consuming activities*, comprising the retrieval, understanding, modification, and integration of those resources into the software product. These groups can be referred to as *development for reuse* (i.e., build generic assets that can be reused in similar contexts) and *development with reuse* or *by reuse* (i.e., use existing assets to build [parts of the] software), respectively [Moore & Bailin 1991].

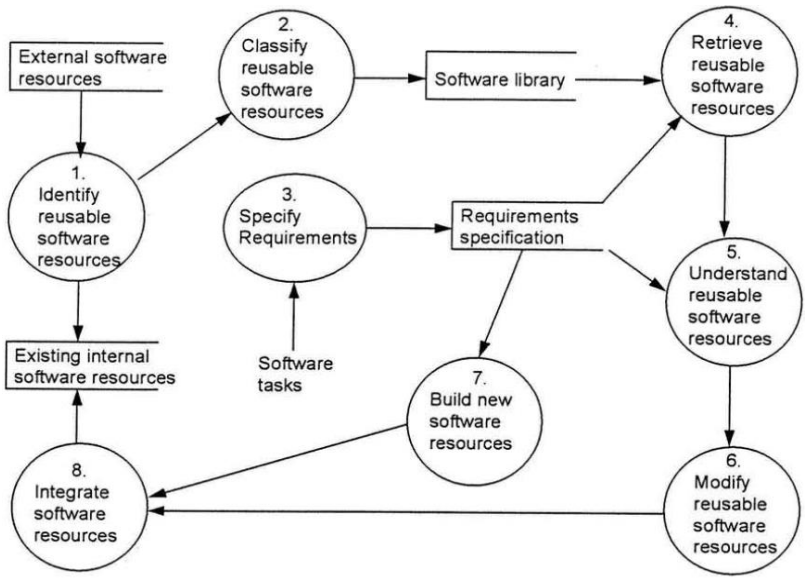

Figure 2.1 – Reuse-based development [Kim & Stohr 1998]

According to this figure, the first step (step 1) involves analyzing existing software resources (that are developed internally or externally) in order to identify potentially reusable artifacts (that may require some adjustments to this end), which must be then classified and cataloged (step 2) in a software library. These two steps have to be performed at the beginning of a reuse program and whenever a new software resource is acquired/developed [Kim & Stohr 1998]. Specifying requirements for the new system (step 3) has to be performed regardless of whether the software resource is to be developed from scratch or not.

Retrieving appropriate reusable software resources from the software library (step 4) is only necessary in a software reuse scenario [Kim & Stohr 1998]. After that, the next step (step 5) is to understand and assess the functionality of the selected resources in order to use or modify them. Modifying software resources (step 6) is necessary for adapting reusable assets to the context of a given application when the retrieved resources do not exactly match the requirements specification – which often occurs –, while building new software resources (step 7) is required when there are no similar resources in the software library for meeting some of the requirements. Finally, the last step (step 8) is the integration of both new and reusable software resources into the target software system [Kim & Stohr 1998].

The aforementioned process suggests a reuse-based software development scenario, whose advantage is to develop software assets aiming at their future reuse (if appropriate, according to the organization's goals). Developing assets without aiming their reuse beforehand (or not taking into account their reuse potential) makes it hard to fit them into other contexts beyond the original ones to which they were developed. This is partially due to the lack of systematization in the construction of reusable assets [Prieto-Díaz & Arango 1991].

Introducing reuse in an organization may require new ways of thinking about software development, given that the way software is reused has changed over the years. Developers search for existing solutions (usually source code snippets or components) based on project's needs or recommendations from another developer.

The rise and massive use of free/open source software repositories (e.g., Google Code, Maven, Github), Integrated Development Environments (IDEs) (e.g., Eclipse, NetBeans), issue tracking systems (e.g., Bugzilla, JIRA, Redmine), among other tools, have strongly influenced not only software development in general, but also software reuse. Particularly, social media tools geared to software development, such as forums,

mailing lists of development communities and Q&A (question-and-answer) websites (e.g., StackOverflow), have been playing key roles in this scenario in the last years.

All kinds of information regarding a reusable asset can be found by means of these resources, including examples of use, tutorials, documentation, support, and so on. Integrating these sources can provide relevant information to the reuse scenario, especially in terms of giving more confidence to the consumer in deciding whether to reuse an asset or not.

2.2 Software Reuse in Quality Standards and Maturity Models

Organizations need to continually seek for improving the quality of their products and services in order to stay in the competitive market. To this end, the quality of processes has been progressively targeted by software development organizations. Due to this increasing demand for software quality, a number of quality standards and maturity models – such as CMMI-DEV (Capability Maturity Model Integration for Development) [CMMI Product Team 2010] and MR-MPS-SW (Brazilian Reference Model for Software Process Improvement) [SOFTEX 2012] – have been proposed, establishing requirements for defining, evaluating and improving software processes.

In order to demonstrate its importance in the maturity of software organizations and promote ways towards its systematization, software reuse is covered by several quality standards (e.g., [ISO/IEC 2008], [IEEE 2010], and [ISO/IEC 2012]), which comprise activities related to the management of the reuse program, as well as the storage, retrieval, management, and control of the assets, among others. Such standards also contain guidelines for integrating reuse in the primary processes of the software life cycle, along with processes for reuse across projects.

According to [ISO/IEC 2008], for instance, a successful implementation of the Reuse Program Management process should provide the following outcomes as results [ISO/IEC 2008]: (i) define the organization's reuse strategy, including its purpose, scope, goals and objectives; (ii) identify the domains in which to investigate reuse opportunities or in which it intends to practice reuse; (iii) assess the organization's systematic reuse capability; (iv) assess each domain to determine its reuse potential; (v) evaluate reuse proposals to ensure the reuse product is suitable for the proposed application; (vi) implement the reuse strategy in the organization; (vii) establish feedback, communication, and notification mechanisms that operate among reuse

program administrators, asset managers, domain engineers, developers, operators, and maintainers; and (viii) monitor and evaluate the reuse program.

Reuse practices are also integrated into models that aim to measure the maturity level of organizations that produce software, such as MR-MPS [Rocha et al. 2007] [SOFTEX 2012], a program for software process improvement coordinated by the Association for Promoting the Brazilian Software Excellence (SOFTEX). This program aims to define and enhance a model for improvement and assessment of software processes focusing on micro, small, and medium enterprises (MSMEs). The MPS-SW model complies with ISO/IEC 12207 and 15504, is compatible with CMMI-DEV, adopts software engineering best practices, and is appropriate (both from the technical point of view as to costs) to the reality of Brazilian organizations [SOFTEX 2012].

MR-MPS-SW is divided into 7 maturity levels, from level G (lowest maturity level) to level A (highest maturity level), in ascending order. Since its version 1.2 released in 2007, this model indicates reuse as one of the goals to be accomplished by organizations in order to evolve their maturity levels. In this model, reuse is expressed in two processes: Reuse Management – GRU[3], required since the intermediary maturity stages (starting from level E), and Development for Reuse – DRU[4], in more advanced stages (from level C onwards).

The purpose of the Reuse Management process is to manage the life cycle of reusable assets [SOFTEX 2013a]. To make this possible, the process defines that the organizations must have a documented strategy for asset management, including criteria that govern their life cycle (i.e., criteria for acceptance, certification, classification, discontinuity and evaluation of assets) (GRU 1). In addition, there must be a mechanism for the storage and retrieval of assets (GRU 2). Modifications on these assets must be controlled throughout the life cycle (GRU 4), and usage data shall be recorded (GRU 3), so as to notify users about potential problems detected, modifications carried out, new versions available and discontinued assets (GRU 5) [SOFTEX 2013a].

The purpose of the Development for Reuse process, in turn, is to identify opportunities for systematic reuse of assets in the organization and, if possible, establish a reuse program for developing assets from the engineering of application domains [SOFTEX 2013b]. This process starts with the identification of the reuse potential

[3] Acronym for "Gerência de Reutilização", in Portuguese.

[4] Acronym for "Desenvolvimento para Reutilização", in Portuguese.

(DRU 1) and the reuse capabilities (DRU 2) of the organization. The ensuing steps are the planning (DRU 3), implementation, monitoring and evaluation (DRU 4) of a reuse program, which comprise the evaluation of proposals for reuse (DRU 5), the development of domain models and domain architectures (DRU 6, DRU 7 and DRU 8), and the specification, development (or acquisition) and maintenance of domain assets (DRU 9) [SOFTEX 2013b].

2.3 Software Reuse Tasks

Software development stakeholders need assistance in the execution of tasks related to software reuse, both at the organizational level and the project level. A discussion of some of these reuse tasks extracted from the literature is presented as follows.

Aiming to increase their productivity, developers may want to *explore the reuse repository* for an overview of which assets are available for reuse, or *search for a specific reusable asset* that they are aware of. After selecting one or more reusable assets, they should make a preliminary evaluation for *obtaining general information about it (e.g., in terms of its available metadata, evolution history, developers, reuse cases, dependencies and so on)*, which may increase reuse confidence. This acts as a filter for selecting one or more assets for an evaluation in depth.

The assets may be compared against each other in terms of their adequacy, and thus are *selected according to the project needs*. Then, developers may try to *understand detailed information of a reusable asset (in terms of the available information regarding its structure, behavior and software metrics)*, before or during the integration with the software system being developed. Eventually, developers may want to *rate* a reusable asset for stimulating or discourage its reuse and provide feedback to its developers, or *report problems on a reused asset* that did not work as expected, leading it to maintenance activities (addressed in GRU 4 [SOFTEX 2012]).

Developers may also want to *identify assets candidate to reuse*, (i.e., existing assets that can support other development projects) from version control repositories, which is a common place for storing development, keeping an evolution history. After that, such developers may need to *assess the reusability of the identified assets* for easing reuse and avoiding reuse problems, and may choose to *refactor an existing asset to make it (more) reusable* if such asset has a large reuse potential for the organization projects. If no candidate asset is found, developers may also want to *develop a reusable*

asset if reuse potential exists or *maintain a reusable asset* if an existing asset does not have a satisfactory level of reusability (e.g., it may have a high coupling that hampers its reuse).

When potentially reusable assets are found, a reuse manager[5] must *evaluate such candidate assets (or new versions of existing assets) for entering the reuse repository, in terms of organizational criteria*, in order to ensure that the reuse repository only contains assets whose quality is attested and whose relevance to the organization is verified beforehand. This is also handled in GRU 1 [SOFTEX 2012].

It may be necessary to contact people who have already contributed to the development of the asset or who have already reused it, for requesting support or clearing doubts. To this end, one must *identify experts (producers and consumers) on a given reusable asset*. Developers usually maintain an overall, broad awareness of who is who and who does what on a project, but more detailed information about people's expertise and activities is often required [Gutwin et al. 2004], and may be relevant for reusable assets as well.

Reuse managers must also *register established interested parties of a reusable asset*. If the *usage data of an asset* are available, one can semi-automatically *identify potential experts and interested parties* based on them, being confirmed later by the reuse manager. This allows *notifying interested parties about changes on the status of an asset*, and also supports meeting GRU 3 and GRU 5 [SOFTEX 2012].

Finally, reuse managers must also *evaluate and maintain the reuse repository* for discontinuing assets. This can be done based either on organizational changes (e.g., change of domain interests) or on evaluation scores and problems reported by consumers.

For an effective reuse initiative, one should *plan the reuse activities*, defining what the organization considers as reusable assets of interest, with appropriate criteria for governing the asset life cycle (i.e., for the acceptance, certification, classification, discontinuity and evaluation of assets). It is also necessary to define a periodicity of reuse repository evaluations. This is also expected in GRU 1 [SOFTEX 2012].

---

[5] A reuse manager is responsible for managing and monitoring the overall reuse program. In spite of being a mere managerial role, a technical profile is emphasized to keep up with particularities of assets and the reuse repository.

In order to keep up with the reuse initiatives, it is fundamental to *monitor the reuse activities*, so that action plans for establishing corrective actions can be made if necessary. Preferably, *reuse results should be reported to stakeholders*, especially top management, so that all of them can become committed to reuse.

This set of tasks are summarized in Table 2.1, classified into project tasks – i.e., tasks that are relevant in the context of a specific software project – and organizational tasks – i.e., tasks that either benefit all the projects or are relevant to the organizational structure (and to the reuse initiatives) as a whole.

Sometimes a given reuse task may be applicable to more than one context, but its goal varies according to the focus of analysis. For illustration purposes, inspired in [Poulin 1994], some of these tasks that vary in terms of development for and with reuse are presented (in a fine-grained level) in Table 2.2.

As it can be noticed, many tasks can be encompassed by both scenarios/contexts, but each with a different quality focus (i.e., different perspectives). Thus, it is important to provide means for achieving these quality focuses.

Table 2.1 – A list of reuse-related project tasks and organizational tasks

Project Tasks	Explore the reuse repository
	Search for a reusable asset
	Obtain general information regarding a reusable asset (in terms of its available metadata, evolution history, developers, reuse occurrences, issues and so on)
	Select an asset according to the project needs
	Understand detailed information of a reusable asset (in terms of the available information regarding its structure, behavior and software metrics)
	Rate and/or report problems on a reused asset
Organizational Tasks	Identify assets candidate to reuse
	Assess the reusability of a (candidate) asset
	Refactor an existing asset to make it (more) reusable
	Develop or maintain a reusable asset
	Evaluate a candidate asset (or a new version of an existing asset) for entering the reuse repository, in terms of organizational criteria
	Identify experts (producers and consumers) on a reusable asset
	Register usage data of an asset
	Identify potential interested parties and register established interested parties of an asset
	Notify interested parties about changes on the status of an asset
	Evaluate and maintain the reuse repository
	Plan the reuse activities
	Monitor the reuse activities
	Report reuse results to stakeholders

Table 2.2 – Examples of reuse-related tasks that vary according to the quality focus

Task	Quality focus	
	Development for Reuse	**Development with Reuse**
Evaluate asset flexibility and generality	Flexible assets are more likely to be reused, while more generic assets can be applicable to other contexts, and also improve the chances of reuse	There is a need for evaluating whether the asset fits the project and developer's needs
Understand asset dependencies	The less the number of external dependencies, the more the asset is self-contained and, therefore, the easy it is to be reused	Understanding dependencies is relevant (among other reasons) for impact analysis, e.g., in terms of asset modules that can conflict with current project configuration
Verify previous reuse occurrences	This is relevant for maintaining the assets repository, as discontinuity criteria may apply	This is important to help assessing the asset's quality and maturity
Understand the reasons why a given reusable asset is not being reused	This is useful for a decision making process regarding the asset refactoring/maintenance (adaptive, perfective or corrective) or discontinuation	This can be effort- and time-saving, since the reason can be an asset constraint (e.g., it only works on a specific operation system) or problems in reusing it
Analyze documentation	Both general and detailed documentation can be relevant for maintaining the assets	General documentation can be enough if it contains instructions on how to reuse the asset

2.4 Concerns about Software Reuse

Achieving effective software reuse is a difficult problem in itself, one that requires proper support in a number of facets, such as managerial aspects [Griss et al. 1994], the aid of tools [Marshall et al. 2003], and adequate mechanisms for retrieval of reusable assets [Braga et al. 2006], among others. In order to be acquainted with the barriers related to effective reuse, it is important to recognize some usual concerns and issues associated to software reuse initiatives.

Many reuse-related issues can be associated to technical aspects, such as the lack of tools and techniques for effectively supporting software reuse, as pointed out by [Kim & Stohr 1998], [Lucrédio et al. 2008] and other works. Particularly, wrong technology choices may considerably hamper the execution of reuse processes [Lucrédio et al. 2008] [Schots & Werner 2013].

However, it is important to emphasize that solving these aspects is not enough for the success of a reuse program; according to [Card & Comer 1994] and [Morisio et al. 2002], a misconception of the reuse needs may lead to the probability of neglecting the importance of assessing the reuse potential at the organizational level and addressing

other barriers, treating reuse as a matter of technology acquisition. Thus, as with any other software process, a crucial concern that must be taken into account is the envisioning of non-technical aspects.

Because software development processes are performed by people, attempts to introduce a software reuse program may also fail because of human issues, such as: (i) lack of management commitment, (ii) lack of understanding of reusable assets, (iii) lack of engagement of team members, (iv) absence of incentives, and (v) cognitive overload [Kim & Stohr 1998]. According to [Schmidt 1999], some of the non-technical impediments to successful reuse commonly include the following:

- *Organizational impediments*: Systematically developing, deploying, and supporting reusable software assets require a deep understanding of application developers' needs and business requirements. As the number of developers and projects employing reusable assets increases, it becomes hard to structure an organization to provide effective feedback loops between these constituencies.
- *Administrative impediments*: It is hard to catalog, archive, and retrieve reusable assets across multiple business units within large organizations. Although it is common to opportunistically scavenge small classes or functions from existing programs, developers often find it hard to locate suitable reusable assets outside of their immediate workgroups.
- *Psychological impediments*: Application developers may also perceive "top down" reuse efforts as an indication that management lacks confidence in their technical abilities. In addition, the "Not Invented Here" (NIH) syndrome [Sametinger 1997] [Sherif & Vinze 2003] is ubiquitous in many organizations, particularly among highly talented programmers.

Regarding the latter impediment, another study [Frakes & Fox 1995] pointed out that the NIH syndrome has become a minor obstacle, and has included reuse education and the perceived economic feasibility (among others) as factors that affect reuse, in accordance with [Card & Comer 1994]. However, although this syndrome has been alleviated over time, much of the phenomenon is caused by the cognitive difficulties that are inherent in the reuse process [Ye & Fischer 2000].

A crucial concern is how to facilitate the acceptance/consciousness and adoption of reuse. Reuse stakeholders must be aware of the reuse results that are relevant to them and need awareness support for reuse tasks. Monitoring activities, for instance, allows the early detection (and possibly resolution) of inconsistencies and shortcomings inside

the software process, supporting and fostering the real integration of reuse paradigm into the existing software development process, encouraging continuous process improvement [Benedicenti et al. 1996]. Since there is a lot of information involved for performing reuse tasks, the lack of awareness and understanding of such information can hinder obtaining the expected results [Selby 2005] [Gill 2006]. This non-technical aspect, however, can be partially handled with a proper support to the technical aspects.

2.5 Software Reuse in Practice: The Brazilian Scenario

Despite the extensive literature on software reuse, including several reports on how to achieve the expected benefits of reuse (and what should not be done, in order to avoid recurring failures), organizations still struggle in beginning and coping with a reuse program, as well as in selecting solutions and technology support that are suitable for the execution of organizational processes that involve activities related to reuse [Morisio et al. 2002] [Lucrédio et al. 2008].

2.5.1 Literature Reports on Software Reuse Implementations

A number of studies and reports on the implementation of reuse processes in organizations were identified in the literature (e.g., [Frakes & Fox 1995], [Frakes & Fox 1996], [Kim & Stohr 1998], [Morisio et al. 2002] and [Sherif & Vinze 2003], among others). Some of the reports in the Brazilian scenario are listed as follows.

Sá et al. (1997) report the experience of introducing software reuse in an organization, by measuring aspects related to reuse before and after the implementation. The authors mention technical and cultural obstacles identified during the process, such as: (i) reuse was only understood as code reuse; (ii) there was no technical or managerial commitment to produce reusable assets; (iii) most systems' development was going straight to the implementation phase, because stakeholders did not believe in Software Engineering as presented in the literature; and (iv) the view of profits was immediate (short-sighted) regarding the production of reusable assets [Sá et al. 1997].

Lucrédio et al. (2008) present a survey carried out with industry professionals, involving Brazilian organizations, aiming to relate organizational characteristics with the successful adoption of reuse, not taking into account the reasons why some organizations were not successful. The survey comprised several factors divided into four perspectives: organizational factors, business factors, technological factors, and processes factors. From the 200 contacted organizations, 57 answered the survey. The

main influence factors identified include the development team, the use of tools and quality models, the prior development of reusable assets, the type of these assets, and the existence of a systematic reuse process. The difficulties encountered are also related to these factors (e.g., an inadequate tool support and the lack of systematization of reuse represent negative influence factors) [Lucrédio et al. 2008].

Silva Filho et al. (2008) describe the implementation of the MR-MPS-SW Reuse Management (GRU) process at the Software Engineering Laboratory of an academic institution. Any software artifact (process asset, source code, or executable) could be considered as reusable assets; they were suggested by the team and evaluated against their quality and reuse potential. Notifications related to the assets' status were made manually by e-mail. The main difficulties mentioned were the definition of a non-intrusive strategy (i.e., which would not impact the usual activities of the organizational unit) and the choice of useful metrics to monitor and control the process. As to technical aspects, the identification of reusable assets was considered the most critical activity regarding the level of intrusion, cost, and effort. Some lessons learned include: (i) the definition of a reuse management focus (such as the minimization of projects' cost and effort) can guide to the prioritization of software process improvements, and (ii) the more mature the reuse management process is, the clearer the perception on how it can be automated [Silva Filho et al. 2008].

Santos et al. (2009) describe the experience on implementing MR-MPS-SW Reuse Management (GRU) and Development for Reuse (DRU) processes in a medium-sized, geographically distributed organization. The defined process for GRU is triggered either from the need to assess candidate assets or for implementing enhancements in a particular asset, which may arise from problems identified during the assets' use or from opportunities for improvement identified over time. Regardless of how the process starts, it ends with the notification of interested parties regarding the availability, evolution, or discontinuation of reusable assets. A research is performed for identifying people potentially interested in a given reusable asset, as well as for defining the role responsible for maintaining such asset [Santos et al. 2009].

Periodically, an assessment of the reusable assets base is made periodically for identifying assets that were less used or criticized by users – such assets can be improved or subject to discontinuation. In an initial search effort on the organization's legacy systems, 4 potential reusable assets were identified, being 3 approved on the acceptance and certification criteria, becoming part of the reusable assets library. The

authors underline the low number of identified reusable assets. Besides, the tools used for supporting the reuse program were too general, such as text editors and spreadsheets. Communications related to the reuse processes are made by e-mails sent manually [Santos et al. 2009].

During the execution of the DRU process, an initial list of nine areas of expertise was identified, corresponding to the business processes supported by the systems developed by the organization. From these, only three were rated as having some potential for systematic reuse and, therefore, were analyzed in more detail (according to the authors, the other processes did not follow a formal line of development compatible with reuse principles). The assessment of the reuse capabilities of the organization showed that there were limited resources for the establishment of an appropriate reuse program, but a plan was drawn up to overcome this problem, defining the necessary resources for its execution. Nevertheless, DRU was considered out of scope during the final assessment of the implementation, due to the lack of concrete data and results on the development of reusable assets [Santos et al. 2009].

Although the literature reports on the implementation of reuse processes in the Brazilian scenario present some problems in common, they usually describe isolated cases, and do not aim at comprehensively characterizing usual problems identified during the implementation and assessment of reuse processes. The most comprehensive one is the work of Lucrédio et al. [Lucrédio et al. 2008], but it is not based on a widely used quality standard or maturity model, i.e., it cannot be ensured that all the analyzed organizations perform a set of reuse tasks in common.

2.5.2 A Study of the Software Reuse Scenario in Brazil

In order to obtain more information on the implementation of processes related to reuse in Brazilian software organizations, semi-structured interviews were conducted with MPS.BR implementers and assessors. The choice for this population is due to the fact that there is a representative number of MPS.BR assessments on level E (60 out of the 488 organizations successfully assessed in MPS.BR[6] are in level E or above, including 38 in level C[7] or above), covering a considerable portion of the nationwide

[6] MPS-SW Published Assessments (data from August 23, 2013), extracted from [SOFTEX 2013d].

[7] It is noteworthy that the DRU process allows the exclusion of most outcomes from an assessment if the organization does not have opportunity and/or ability to perform development for reuse. Thus, one cannot state that all these organizations perform DRU.

scenario. Moreover, this ensures a common set of reuse tasks (that organizations must implement to meet the model), thus allowing for more representative results.

The interview questions were designed to obtain both technical (regarding the decisions for implementing the processes, based on the outcomes) and non-technical information (involving implementers' opinions concerning the assessed organizations, as well as difficulties and frequent problems) with respect to reuse processes. The questions for Reuse Management (GRU), Development for Reuse (DRU) and other questions that are relevant for reuse processes in general are shown in Table 2.3, Table 2.4, and Table 2.5, respectively, along with their corresponding goals.

Other aspects related to the outcomes were not directly included in the questions, such as the control of changes in assets (related to GRU 4) and the criteria for acceptance, certification, classification, evaluation and discontinuity of assets (related to GRU 1), among others. These items are very specialized; thus, they were evaluated indirectly through the general questions and the intersection with other outcomes.

For analyzing the collected data, the open coding technique [Seaman 2009] is used, by marking and categorizing snippets of interviews, relating them to questions (categories) initially defined.

Table 2.3 – Questions and Goals of the Study – Reuse Management (GRU)

ID	Question	Goal
Q1	Which kinds of assets have been considered as reusable by the organizations?	Identify which types of artifacts are considered as reusable by organizations in their projects/ processes. Related to GRU 1.
Q2	Where are the reusable assets usually stored?	Identify mechanisms (tools) used for storing reusable assets. Related to GRU 2.
Q3	Where/how are the reusable assets made available for reuse, i.e., where/how are the stored assets listed so that the interested parties can find them?	Identify the way organizations make their reusable assets available and the mechanism (tool) used to this end. Related to GRU 2.
Q4	How are the usage data about the assets logged?	Identify how organizations record reusable assets' usage data. Related to GRU 3.
Q5	How are interested parties informed of problems detected, modifications made, new versions released, and discontinued assets?	Identify the mechanisms used for notifying interested parties about changes in the status of assets. Related directly to GRU 5 and indirectly to GRU 4.

Table 2.4 – Questions and Goals of the Study – Development for Reuse (DRU)

ID	Question	Goal
Q6	What are the application domains of the organizations in which opportunities for reusing assets have been identified, or in which they have intended to practice reuse?	Identify relevant application domains from the viewpoint of the state-of-the-practice. Related to DRU1.
Q7	Are organizations able to plan and establish an effective reuse program?	Check if reuse programs have been properly established in organizations. Related to DRU3 and DRU4.
Q8	How are organizations monitoring the reuse program?	Identify monitoring mechanisms and strategies being used by organizations. Related to DRU4.
Q9	How are reuse proposals (requests for reusing existing domain assets or developing/acquiring new ones) made?	Identify how reuse proposals are made and which kinds of request are more frequent. Related to DRU5.
Q10	How are domain models and domain architectures represented in organizations?	Identify techniques being used by organizations for representing domain models and domain architectures. Related to DRU6, DRU7, and DRU8.
Q11	How are domain assets specified/ acquired/ developed and maintained?	Identify techniques being used by organizations for specifying, acquiring, and/or developing domain assets. Related to DRU9.

Table 2.5 – Questions and Goals of the Study – General Questions on Reuse Processes

ID	Question	Goal
Q12	Which comments are made by the organizations regarding the GRU and DRU processes?	Characterize general problems pointed out by organizations. Answers to this question may drive the remainder of the interview for more details (funnel strategy).
Q13	What is the point of view of the diverse stakeholders (developers, project managers, top management) about reuse?	Identify whether there is any cultural resistance by stakeholders and, if so, which roles have such resistance. This information is also relevant for DRU4.
Q14	Which GRU and DRU aspects are more difficult to understand by the organizations?	Obtain more information about difficulties in understanding (including processes, concepts, tasks, tools etc.) pointed by the respondent. This is purposely broad.
Q15	Which are the most difficult tasks (particularly, GRU and DRU tasks) for the organizations to perform?	Identify information about the most difficult tasks.
Q16	What are the problems (“required” items) usually identified on GRU and DRU during assessments?	Identify issues that organizations cannot accomplish in GRU and DRU, as well as potential difficulties in implementations.
Q17	Which aspects related to the implementations or assessments of the GRU and DRU processes would you like to add (including the moment in the MR-MPS-SW implementation when you start to implement GRU and DRU processes, and potential difficulties in implementing or evaluating these processes)?	Identify difficulties on the implementations or assessments of the GRU and DRU processes, and ultimately verify how organizations prepare themselves to assessments.
Q18	Is there anything else that has not been asked and you would like to comment on?	Obtain feedback on the process and other aspects that participants would like to add.

Invitation e-mails were sent based on the list of Implementing Institutions (IIs) and Assessment Institutions (AIs) available on the SOFTEX website [SOFTEX 2014a]. The response rate in terms of the IIs and AIs was 38.46%. The criterion for participation in the study was the experience in the implementation and/or assessment of the GRU and/or DRU processes. Participants were interviewed in person during the XII Brazilian Symposium on Software Quality (July 1 to 5, 2013), or remotely, via Skype (between July 6, 2013 and August 25, 2013).

In total, there were 10 respondents, all concomitantly MR-MPS-SW implementers and assessors, having carried out (or accompanied, as leader assessors) at least 1 implementation or assessment of the GRU process (in most cases, more than 3 assessments). Figure 2.2 (left) shows the distribution of the respondents according to the MPS assessor levels (ordered from the lowest level to the highest), while Figure 2.2 (right) shows the year of authorization[8] to perform implementations and assessments of MR-MPS-SW.

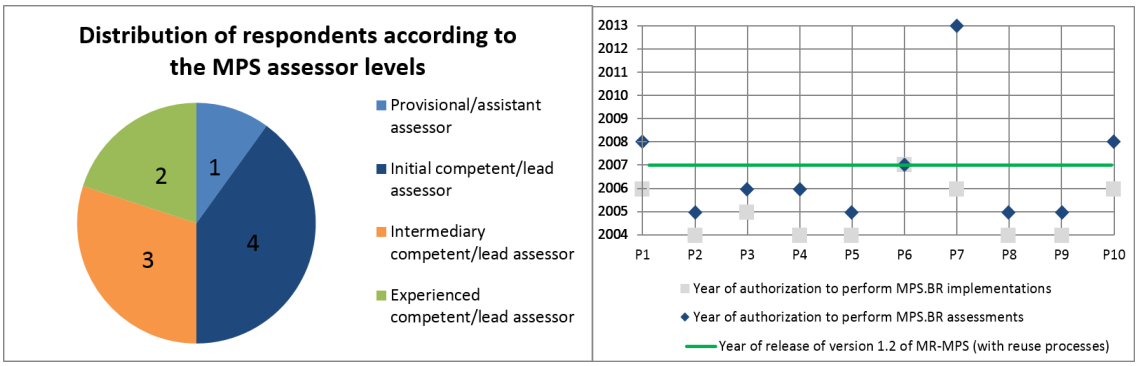

Figure 2.2 – Distribution of respondents according to the MPS assessor levels (left) and participants' experience (based on the year of authorization) in performing MR-MPS-SW implementations and assessments (right)

As it can be seen, most participants are competent/lead assessors, meaning that they received a specific training from an assessment institution and performed at least 6 assessments as provisional/assistant assessors [SOFTEX 2013c]. Moreover, 2 of them are experienced competent/lead assessors – i.e., besides having competent/lead assessor's skills, they had a specific training on statistical process control and performed at least 4 assessments in levels E, D, and C as competent/lead assessors [SOFTEX 2013c]. Additionally, all the respondents were formed implementers before the release of version 1.2 of MR-MPS, i.e., all of them are allowed to perform implementations of reuse processes (after making a complementary training course, which is mandatory

[8] Based on [SOFTEX 2014b]

when substantial changes are made in the model) since such processes were incorporated into MR-MPS-SW.

The main findings are presented in the next subsections. A full description will be published as a technical report. The number of respondents who provided the corresponding statements is omitted in this text, but will be published in the technical report that is being prepared[9].

It is worth emphasizing that the same respondent may have provided more than one answer per question. In order to ensure the confidentiality of responses, results that may allow the identification of the respondent are always reported together and in aggregated form[10]. Furthermore, no gender distinction is made in the text.

2.5.2.1 Findings from the process-specific questions

In response to Q1 (related to the kinds of reusable assets), all respondents cited source code as an asset considered by organizations to be reusable, either in the form of libraries, frameworks, individual classes (for object-oriented systems) or even code routines (for systems developed in other programming paradigms). Source code is also the asset most often identified as reusable in organizations, and was pointed out by some respondents as "most useful" and/or "most suitable". Nevertheless, respondents indicated that the number of reusable assets found in the reuse base of organizations is considerably small (2-4 assets were mentioned, in some cases).

Document templates related to processes – such as project plan templates or checklists – are also usually considered to be reusable. Other kinds of assets mentioned are standard processes[11], test scenarios, test plans, knowledge assets, business rules, and calculation spreadsheets. With respect to plans and test scenarios, one respondent mentioned that these were reused without specific adjustments for each scenario, i.e., their content was entirely reused in different projects.

Regarding Q2, related to the storage of reusable assets (in GRU 2), all respondents mentioned that version control repositories are usually employed for this purpose, being SVN the most frequent implementation. Some respondents stated that

[9] The numbers are used only for accounting purposes. It is believed that the contribution of studies like this is not the quantification, but the richness and variety of data, in accordance with [Seaman 2009].

[10] Although this makes it difficult to perform other potentially interesting correlations between answers, this decision is due to the amount of participants of the study. A different choice could allow the identification of respondents.

[11] Considering each instantiation of a standard process as a form of reuse.

the storage mechanism varies according to the type of asset. Version control repositories are most common when the reusable asset is source code (but are also used in some implementations of reuse to store other types of assets). Other forms of storage were also cited:

- Folders / directories (usually shared on the Intranet), either with or without controlled access, used for storing process templates documents and standard processes;
- Default location (local folder, in the cloud etc.) of collaborative tools and tools with social network features: Sharepoint[12], Google Docs/Drive[13] – also used for knowledge assets –, and Confluence[14];
- A tool developed by the organization.

The most cited form of making reusable assets available (Q3, also related to GRU 2) is by tools for integration with repositories, such as TortoiseSVN[15] and Maven[16]. Other cited forms of provision were:

- Wiki[17];
- Collaborative tools (aforementioned);
- Directory[18] informing the classification, type, and link of the reusable assets and list of assets available in the Intranet portal.

For collecting information about the usage data of reusable assets (Q4, concerning GRU 3), some respondents cited manual ways. In one of these, the reuse manager is responsible for capturing such information by analyzing software projects and searching for reuse occurrences, storing results in an Excel spreadsheet or a sort of list. Another form requires that, in case any asset is reused in a project, the reuse manager must be informed by the project manager or by the development team. Other

[12] http://office.microsoft.com/pt-br/sharepoint/

[13] https://drive.google.com/

[14] https://www.atlassian.com/software/confluence/

[15] http://tortoisesvn.net/

[16] http://maven.apache.org/

[17] http://www.mediawiki.org/wiki/MediaWiki

[18] In this case, unlike the concept of file folder, directories are indexes of sites, usually organized into categories and subcategories, whose main purpose is to quickly find desired websites, searching by categories. An example is Yahoo! Directory (http://dir.yahoo.com/).

forms of accounting include cross-references, number of downloads, a tool developed by the organization, and issue trackers, like Mantis[19].

Finally, most respondents cited the use of e-mails as a form of notification to interested parties about changes in the status of reusable assets (Q5, related to GRU 5). This is the only noticed way up to that moment mentioned by some of the respondents, and the most frequent one to all of them. E-mails are sent manually, either to all the members of the organization (irrespective of being interested parties) or from a list of interested parties, manually maintained based on the usage data (from GRU 3).

There are also cases in which the Wiki or the issue tracking system contains a record of consumers (considered as interested parties) of a given reusable asset, so that only these consumers are notified by e-mail (by the system) about changes in the status of assets. Other forms of communication are a tool developed by the organization and an adaptation of the concept of "followers" in social networks. In the latter, interested parties in a given asset must "follow" an asset in order to be informed of updates to the "profile" of such asset.

Most of the participants were involved in at least one assessment that included the DRU process. However, the MPS-SW model allows the exclusion of DRU 3 to DRU 9 outcomes from the scope of the assessment (when the organization shows that it does not have reuse opportunities (DRU 1) or reuse capabilities (DRU 2) [SOFTEX 2013b]). Consequently, only few assessments encompassed all the DRU outcomes. In this regard, one respondent stated that organizations often "prefer to claim that they do not have a domain of interest, in order to avoid committing to the next assessment"[20]. Another respondent affirmed that organizations frequently "state that they are not prepared" for such assessment.

Regarding the context in which DRU was implemented (Q6), organizations that implemented the entire DRU process "had the whole development structure towards reuse". The core business of one of the organizations was to "sell software that serve as a basis for other developments, like frameworks": "the organization has a product line and several different modules that support development". Another respondent stated

[19] http://www.mantisbt.org/

[20] According to [SOFTEX 2013b], when reuse opportunities exist but the organization lacks reuse capacity, it must invest in qualification, and the outcomes become mandatory in the next assessment of the organization in the same maturity level or above.

that “the organization was a software factory, which builds component line domains”, but “it was a very simple reuse, with few components in the [reuse] base”.

Two respondents stated that they did not consider the planning and establishment of the reuse program as effective (Q7): one of them stated, “it stands on the threshold of what is expected in an assessment”. Regarding the monitoring (Q8), one respondent affirmed that it happens “as a protocol formality”, i.e., it is carried out only for the record; organizations “do not really keep up with it”.

Regarding the way reuse proposals are made (Q9), one respondent stated that the analyst of the organization identifies a part of a project that could be reused in the future, and hence makes a request to the reuse group through a template (a Word form). According to the same respondent, the representation of domain models and domain architectures (Q10) were made in the Enterprise Architect tool[21], but members “did not use the most appropriate notation on domain engineering”. Finally, although the respondent stated that domain assets were usually developed internally, no specific technique for their development (Q11) was mentioned.

2.5.2.2 Other findings from the general questions on reuse processes

The answers to the questions presented in Table 2.5 and the interview data provided perceptions of respondents with respect to the implementation of reuse processes.

Regarding the comments made by organizations concerning the GRU and DRU processes (Q12), some respondents mentioned that organizations usually do not have problems regarding GRU, and it is “well received” by them. One respondent mentioned that organizations that use source code as a reusable asset consider the GRU process “very important”, while many organizations that perform process reuse “do not even realize that they are implementing GRU”, and it is “basically ‘for the record’”; thus, “it cannot properly be characterized as reuse”. Another respondent stated that organizations sometimes find it difficult to notify interested parties about changes in the status of assets.

Because the GRU process is mandatory (i.e., its outcomes are not subjected to exclusion in the context of an assessment in particular), some organizations “reuse any component just to say that they have [some] reuse”. A respondent stated that

[21] http://www.sparxsystems.com/products/ea/

organizations usually consider reuse a process "of little relevance". They get to implement a process, but do not have the "spirit of reuse".

Sometimes organizations do not see the opportunity to implement the DRU process, which is sometimes questioned regarding its viability when the organization is still not sure whether to invest for reuse. According to one respondent, the view of organizations about DRU is that "it is something that will take a lot of work and will require a high investment", which causes them to claim that they "cannot afford to do it at that moment".

Regarding the point of view of the diverse stakeholders about reuse (Q13), some respondents affirmed that top management is acquainted to the GRU process, but in a macro level. One respondent commented that they lack a vision of how much the organization has gained with reuse – "such analysis, such indicator has not been explored the way it should be".

A respondent stated that when knowledge management is more widespread, "hierarchical levels can have access to information more easily". Often the programmer does not have the necessary knowledge to decide whether it is better to reuse a component or not. Another respondent stated that he/she has suggested organizations to invest in a technical lead or architect to be responsible for selecting what will be reused. Such person would first make an analysis if the solution is appropriate to the context.

The same respondent also stated that, "when a component has no associated documentation, the consumer can go through a trial and error activity". However, "when using the component in the wrong way and seeing that it did not work [(which is a time-consuming task)], one can give up reusing it". Moreover, prospective consumers sometimes express misgiving in reusing something that was not made by them, reinforcing the NIH syndrome (discussed in Section 2.4). Additionally, according to one of the respondents, "programmers think that, for having an effective reuse, one needs to first refactor the code, or code snippets".

With respect to the GRU and DRU aspects that are more difficult to understand by the organizations (Q14), one respondent affirmed that "it is difficult to make software development stakeholders obtain the reusable assets from a Wiki or similar mechanism, when there is a version control tool integrated to the development environment from which obtaining the assets is more practical". Another issue is that organizations are "unaware" if the reuse manager role is a more managerial or a more technical assignment.

According to one respondent, DRU can be seen as an “additional expense for small enterprises that are unable to afford it”. Organizations also may see DRU as “a great difficulty”, and “want to try to ‘go off on tangents’”. Another respondent stated that organizations “do not even know what it is”.

Concerning the most difficult tasks to be performed by the organizations (Q15), one respondent affirmed that he/she does not think that it is a matter of difficulty, but a motivation issue instead: organizations “do not see the importance” of reuse in day-to-day activities.

A difficulty indicated by other respondents is the logging of usage data (GRU 3), which often becomes error-prone, since it is made in a manual way, being very dependent on people. One of the respondents stated that, in most organizations, reuse “works in a more dynamic way”, so that a member can obtain an asset without having to undergo this “usage control”. A solution pointed out by the respondent would be the use of a tool that is able to “scavenge” the code and check for the presence of a given asset.

When something must be developed, usually the analyst – or the developer – can be reluctant in exploring a huge reusable asset database for identifying which one fits his/her needs, and may therefore prefer to develop from scratch, which might be faster in his/her discretion. “When one has about half a dozen components, this is not seen as a problem”. However, when there is a database with a large number of assets, “analyzing which one best fits the needs – including items that are not well documented – and checking ‘the whole stuff’ is not a trivial task”.

Some difficulties in implementing GRU 3 were also observed through the statements of the respondents. In the accounting of reuse data, “the probability of error is enormous, because it is not known what has happened at all”. According to the respondents, the difficulties lie both in verifying whether the asset was effectively reused (“deep down it is not known what the person has actually used”) and in checking if the reused asset was previously requested (“after [a reusable asset] that was in a repository is on a developer’s machine, the person uses [such asset] without [formally] requesting”). The latter problem is quite common when the parameter used for accounting is the number of downloads. According to another respondent, the number of downloads “is just an approximation, because a person may have downloaded it and not reused it in practice”.

Reuse itself is also a difficulty in legacy systems, according to one respondent. “The product was not originally designed for being componentized, thus reuse happens

in a way that is not correlated to the business logic", excluding the possibility of "making a business rule reusable by all the systems". Another respondent mentioned that, in DRU, difficulties lie in conducting the reuse program.

Most of the required items (Q16) involve aspects that are already mentioned throughout this text; thus, they will be discussed in more details in the technical report.

When participants were asked to add any additional aspects related to the implementations or assessments of the GRU and DRU processes (Q17), a difficulty that was mentioned regarding the implementations concerns the accounting of usage data, i.e., "to find an efficient way of accounting", "which is not only for meeting the model", but also one "that the organization can use for 'statistics', so that it can start evaluating cost-benefit". According to one respondent, such evaluation is the major issue, because "the organization wants to [perform] reuse knowing how much it is earning in return (e.g., in terms of reduction in rework)". It is noteworthy that this directly impacts in carrying out accounting (GRU 3) and monitoring tasks.

Concerning difficulties in evaluating reuse processes, one respondent pointed out difficulties in assessing the changes made to assets – in GRU 4 and its intersection with the Configuration Management (GCO)[22] process –, because "it is difficult to assess whether a developer has complied or not with the established level of control" on the evolution of reusable assets. The respondent further stated that "the assessment of communication between interested parties (GRU 5) is not trivial".

Finally, participants were asked (on Q18) if they would like to comment on anything else that. Responses ranged from kinds of reusable assets chosen by organizations, considerations about particular domains and technologies that do not favor reuse in organizations, and lack of adequate tool support.

Some of the respondents stated that they do not considered appropriate to treat knowledge assets and document templates as reusable assets (GRU 1), for different reasons: either they think that the management of the GRU process is laborious/onerous (given the high number of assets and the high frequency of use), or they think that, by using such kinds of assets, the benefits arising from reuse are not well noticeable.

Additionally, although standard processes and knowledge assets have been seen in implementations of GRU, some respondents disagree with this approach because these assets are already addressed in other processes of the same MR-MPS maturity

[22] This process is comprised in MR-MPS level F (one level before level E) [SOFTEX 2012].

level – Organizational Process Definition (DFP) and Human Resource Management (GRH), respectively –, thus not aggregating noticeable reuse results to the organizations. One can “end up improving knowledge management rather than making the implementation of a reuse process”.

Another factor mentioned by some respondents is the need for organizational culture and appropriate tools/infrastructure to implement reuse processes. Most of the respondents mentioned the “lack of tool support” for the execution of these processes. One respondent stated, “the major issue is to quickly find the component you need”, “having adequate documentation” and “having confidence that the component will work”: according to him/her, “these are the complicated things in reuse”.

The same respondent also stated that sometimes “the component works very well, but no one knows where it is, and the time it takes to find the component, study it and know how it works, depending on the size of the component, may end up giving rise for the developer to think that it is better to build his/her own component”.

Still according to the respondent, in most of the organizations whose products have more than 5 or 10 years (i.e., legacy systems), “software has a bad cohesion, high coupling etc.”. He/she suggests that “the use of a technical debt tool (such as Sonar) or the use of a heat map of the product could help to identify critical points, and thus the organization could focus on the implementation of these points so that it perceives reuse payback faster”.

One respondent posed that there is room for a “makeup” in accounting for usage data, and currently “there is no resource on the day of assessment that allows assessors to request for showing the projects that used the component”, because it “becomes too complicated to do it at that time”.

Finally, some respondents with considerable experience in implementation and/or assessment stated that many organizations “have not yet understood the benefits of GRU”, and sometimes implement this process more “to fulfill a requirement than for having understood that its goal is important”. On the other hand, it was also stated that organizations “lack a lot of knowledge” and there is a “lack of market maturity” with respect to reuse. Additionally, some implementations seem to “follow a recipe”, being necessary to invest in “training implementers and assessors” in reuse processes.

2.6 Discussion

Many of the findings identified in the study match the literature reports both in the Brazilian and worldwide scenarios, especially the lack of tool support, the lack of support for different reuse stakeholders, and the need for more engagement in reuse initiatives.

As it can be noticed, it is necessary to provide support in the execution of organizational reuse processes and in software development processes. The latter requires support not only for the identification of opportunities for reuse, i.e., assets that can be reused in the context of a given project (development with reuse), but also for the construction of software artifacts that may be reused in the future (development for reuse), through indicators that guide the development of reusable assets.

The fact that some organizations store their reusable assets in version control repositories has also been observed in other studies, such as [Lucrédio et al. 2008] – in this study, the authors suggest that this may contribute to the failure of the efforts in fostering reuse. In this context, it is noteworthy that each type of repository has features aimed at ensuring the better functioning for their intended purpose, and an inappropriate technology selection can hinder the adoption and implementation of reuse processes. Reuse repositories and configuration management repositories have different purposes (the former, for instance, is optimized for searching operations, and should only contain releases of the assets), and this must be taken into account when instantiating a repository for an organization.

As mentioned before, the programmer often does not have the necessary knowledge to decide whether it is better or not to reuse a component, and which one fits his/her needs. A huge reusable asset database to explore, the size of the asset, the time it takes to find and study it and the lack of associated documentation are aggravating factors, which makes him/her prefer to develop from scratch. One respondent explicitly pointed out that the major issue in reuse is “to quickly find the component you need”, besides “having adequate documentation” and “having confidence that the component will work”.

Thus, in software development with reuse, developers need to be able to (i) explore reuse repositories in an intuitive way, so that they can easily identify candidate assets and analyze their dependencies, (ii) compare assets that are similar in their purpose, in order to select the one(s) that best fit(s) the needs in a given context (e.g., a

nonfunctional requirement that must be met in a software project), and (iii) understand assets in terms of their properties, structure, behavior and evolution, so that they can be reused more easily and with more confidence.

With respect to the tool support, although the tools mentioned in the study partially assist GRU activities, organizations demand for applications that support the execution of these activities in an integrated way, allowing communication with other tools (e.g., configuration management tools) for more effective, visible, and reliable results.

For instance, as mentioned, it is difficult to make stakeholders obtain reusable assets from a specific mechanism, when the version control tool is already integrated to the development environment. Because reuse works in a more dynamic way in most organizations (i.e., without a strict control), one respondent cited the need for a tool to "scavenge" the code and check for the presence of a given component, thus providing more awareness of the current reuse scenario. Besides, such tool would also help to assess whether developers comply with the level of control previously established when evolving reusable assets, instead of creating blocking policies that may cause adverse effects.

It was also noted that, in many cases, notifications about changes in the status of assets are triggered without any distinction of actual interested parties. This may compromise the effectiveness of communication, leading stakeholders to ignore important notifications, depending on the frequency of reception of irrelevant information. Information overloading may also adversely affect the perceived benefits of reuse by these stakeholders. Another problem related to these notifications occurs when the maintenance of the list of interested parties and the sending of e-mails are performed manually, becoming an error-prone approach. This can be partially due to the limitations on tracking which team members reused which assets (i.e., collecting usage data, as specified in GRU 3), thus hindering communication.

Top management also needs more awareness and visibility of relevant information of the reuse process. Particularly, a vision of how much the organization has gained (or saved) in reuse is necessary, but other metrics can also be relevant for reuse initiatives. An issue pointed out in [Benedicenti et al. 1996] is that many organizations that manage to introduce reuse fail in tracking and controlling its evolution, especially if compared with organization objectives and mission.

The importance of finding what managers and developers need to understand about reuse is highlighted in [Kim & Stohr 1998]. Management needs to be able to measure and control the impact of a software reuse program. In other words, the value of reuse must be somehow established and communicated to managers [Kim & Stohr 1998], so that they can be aware and become committed to reuse initiatives. Moreover, for a better acceptance of reuse-oriented changes in stakeholders' usual activities with less impact, suitable mechanisms must be identified and developed. Particularly, because some necessary steps for implementing reuse may be challenging, organizations should try to accomplish them in a progressive way, in order to avoid resistance and allow for a better acceptance by the stakeholders, besides preventing cognitive overload.

An appropriate awareness of the reuse scenario is intrinsically correlated to monitoring. For communicating results in an effective way, monitoring mechanisms are crucial. The ways for collecting and logging information about the usage data (GRU 3) are often error-prone, since most approaches do it in a manual way, being very dependent on people. This can be one of the reasons why organizations do not keep up with monitoring practices, besides the lack of understanding the importance of such monitoring in conducting a reuse program.

Organizations do not see the opportunity to implement development for reuse; however, based on the responses related to not implementing a DRU process, a proper analysis of the reuse opportunities along with the aforementioned improvements in a reuse program may be able to overcome this scenario. Legacy systems that were not originally designed for being componentized can be gradually refactored according to the organizations' needs (which can be identified, highlighted, and communicated through visual support, e.g., a heat map or a technical debt tool, as mentioned by a respondent). This can be done with the support of software metrics (e.g., cohesion and coupling). Assets can also benefit from useful recommendations during their development, for making them more reusable.

In this sense, when it comes to development for reuse, developers need to be aware of (i) how to develop reusable solutions, i.e., what must be taken into account for increasing the chances of reuse and avoiding problems that may hamper it, (ii) how to evaluate and reengineer existing solutions for promoting their reuse when appropriate, and (iii) how development with reuse can accelerate development for reuse, i.e., reusing existing assets for building reusable assets.

It is important to highlight some limitations of the conducted study: (i) the information about the organizations were obtained by implementers and assessors, and may represent a particular point of view of the respondents; (ii) the data analysis was performed by the interviewer himself, and there may have been problems of interpretation in the analysis of responses (however, the points on which there was uncertainty were confirmed with the respondents); (iii) it stands out that the results, while comprehensive, are not generalizable; and finally, (iv) the whole study (including the interviews and the analysis) was conducted using the Portuguese language; thus, some English translation issues may cause deviations in the interpretation of the data.

It is believed that the observations of the presented study can be used as input for both research initiatives and the development of tool support for the implementation of reuse processes in software organizations. A technical report with the interview material and the full analysis will be prepared as a future work.

2.7 Final Remarks

Over the last decades, there has been a lot of progress from both industry and academic research towards systematic reuse, exploring its potential for obtaining significant gains in software development productivity and quality. Moreover, some quality standards and maturity models evidence the need for reuse processes. Nevertheless, despite the efforts undertaken in software reuse research, state-of-the-practice indicates that several software organizations still find difficulties for achieving systematic reuse in practice.

Several difficulties and barriers (both cultural and technological) occur when implementing organizational processes that involve activities related to reuse, as well as when selecting or developing appropriate solutions for the implementation of these processes, as pointed out by literature reports [Sá et al. 1997] [Lucrédio et al. 2008] [Silva Filho et al. 2008]. Interestingly, many of the identified problems in Brazilian organizations recur in the worldwide scenario over time, as can be seen in works such as [Morisio et al. 2002].

As it can be noticed from the findings of the conducted study, several respondents mentioned the lack of understanding as one of the main reasons for the problems with reuse implementations. Such lack of understanding varies in terms of understanding the need for reuse, understanding reuse benefits, understanding reuse tasks, as well as tool support for executing such tasks, so that they can be performed

more efficiently. It must be highlighted that the lack of relevant information and understanding of a reusable asset and its surroundings can also aggravate the NIH syndrome.

To this end, the application of perception and awareness techniques can be useful. For instance, visualization metaphors can represent reuse information, so that users can interact with and manipulate the corresponding data, as well as obtain answers to reuse tasks more quickly, besides decreasing the cognitive overload. Software visualization resources and techniques play an important role on awareness and comprehension, and can be used for supporting a software reuse program, especially in terms of software reuse tasks. This topic is covered in the next chapter.

CHAPTER 3 – SOFTWARE VISUALIZATION

This chapter presents the main concepts related to software visualization. It also describes how visualization resources and techniques can benefit software reuse, as well as how related works identified from the state-of-the-art have been addressing this issue.

3.1 Contextualization

The large amount and diversity of data generated throughout software development is often difficult to manage and monitor. Organizations have sought for techniques that allow not only to store and process such data, but also to exploit them in order to extract relevant information to support decision-making processes and allow increasing the quality of their services, processes, and products.

The quality and relevance of decision making heavily depend on the understanding, interpretation, and aggregation of organizational data; such factors can become critical while implementing and evaluating organizational strategies, thereby becoming a competitive edge. There is a need for appropriate models and mechanisms for analyzing and monitoring data about software processes and products, as well as studies on how the available resources can support understanding such data.

If data sources with evolution information of software development, such as repositories of version control systems (VCS) are also taken into account, software gains a dimension in time, which increases even further the mass of generated data. In addition to that, there are other data sources, such as issue trackers, measure databases etc., which bring a greater diversity on the nature of data. In order to deal with this scenario, software development requires appropriate mechanisms and tool support that can assist in the extraction and analysis of these data and allow their understanding [Schots et al. 2012]. However, such understanding is not an easy task.

According to Diehl (2007), 75% of all information from the real world is perceived visually [Diehl 2007]. On the other hand, as stated by Brooks Jr. (1987), software is very difficult to visualize; the reality of software is not inherently embedded in space [Brooks Jr. 1987]; hence, it has no ready geometric representation. One of the obstacles for visualizing software information is that data are abstract and, therefore,

have no associated physical structure [Chen 2006]. Thus, it is necessary to consider (i) the use of visual abstractions that are appropriate to the nature of data and their relationships, (ii) representation techniques that allow to emphasize what is relevant in a given context, and (iii) different forms of interaction, allowing to perform exploratory (and, therefore, richer) analyses [Schots & Werner 2012] [Schots et al. 2012].

In this context, software visualization techniques aim to provide a better and faster understanding of the structure, behavior, and evolution of software processes and products [Diehl 2007]. It can be defined as the use of information visualization techniques [Chen 2006] applied to software, i.e., as a branch of information visualization. Data are represented by means of visual metaphors for facilitating the comprehension of different scenarios and contexts, as well as the detection of underlying patterns and the creation of analogies [Lanza & Marinescu 2006] [Diehl 2007].

Software visualization has been exploited as a way to assist software development activities that involve human reasoning, helping people to deal with the large amount and variety of information by providing appropriate abstractions [Lanza & Marinescu 2006] [Diehl 2007]. Software visualization research focuses on the use of computational resources for accelerating and optimizing users' perception, understanding, and assimilation of information *of* software and *about* software, by stimulating the human cognitive capacity (derived from users' memory, perception and reasoning). Perception is the processing of sensory information and thus part of human cognition, which also includes awareness, reasoning, and learning [Diehl 2007].

Several strategies and techniques have been proposed and developed for the representation and interaction with the visual metaphors. Some of these techniques are listed in a previous work [Oliveira 2011], and a more comprehensive set are listed in a technical report being prepared. Software visualization tools make use of these techniques in order to provide a richer representation and exploration of the underlying data, thus better supporting software comprehension and correlated tasks.

In this regard, Diehl (2007) states that, in order to make visualizations effective in their goal, it is important to keep in mind that the visual metaphors and representations to be used must be adapted to the stakeholders' perceptive abilities, not the opposite (as it usually occurs) [Diehl 2007].

Some of the software engineering fields that can be supported by visualization include requirements engineering [Cooper et al. 2009], software architecture and design

[Lanza & Marinescu 2006] [Gallagher et al. 2008] [Schots et al. 2010], software measurement [Lanza & Marinescu 2006], software evolution [Wettel & Lanza 2008] [Werner et al. 2011], software maintenance, reverse engineering and reengineering [Koschke 2003] [Telea et al. 2010], among others. Software engineering education can also benefit from the use of visual metaphors and other interactive approaches to allow exploration of concepts and enhance learning [Rodrigues & Werner 2011]. One can also highlight the inherent multidisciplinary of the software visualization topic, since it integrates several computer science disciplines, such as data mining, software engineering, computer graphics and human-computer interaction.

Mukherjea & Foley state that visualization is particularly important for allowing people to use perceptual reasoning (rather than cognitive reasoning) in task-solving [Mukherjea & Foley 1996]. In this sense, it is desirable to make an explicit description of the supported tasks, in addition to the usual understanding goal, so that potential users with matching information needs can identify these visualizations easily.

3.2 The Role of Visualization in Awareness and Comprehension

Since research in software engineering is steadily expanding and investigating different methodologies, processes and techniques, it is also necessary to provide stakeholders of the software development process with a sense of what happens in the scenario in which they are involved, as well as means to explore and understand software artifacts of interest and their properties [Schots et al. 2012]. This requires appropriate awareness and comprehension resources.

Although these concepts are correlated, there is a subtle difference between them. According to [Shi et al. 2011], awareness is "the state or ability to perceive, to feel, or to be conscious of events, objects or sensory patterns", but in this level of consciousness, an observer can confirm sense data without necessarily implying understanding or comprehending. Similarly, program comprehension also encompasses the software development life cycle, but it focuses mainly on software artifacts, rather than the process and its variables. In other words, awareness is related to cognitive reactions to a condition/event (being aware of it), while comprehension involves assimilation of knowledge (understanding a fact) [Schots et al. 2012].

Enhancing awareness and understanding of software information and the software itself requires the identification of adequate abstractions according to the comprehension needs [Schots et al. 2012]. The choice of the visualization abstractions

and techniques for representing the data, as well as the interaction techniques to be employed, heavily depends on contextual information, e.g., the nature of data, the visualization constraints, and the task to be supported (e.g., selecting the most suitable assets from a set of reusable assets). Awareness and comprehension concepts are discussed with more details in the next subsections, along with a brief argument on the role of visualization.

3.2.1 On the Awareness of the Software Development Life Cycle

The concept of awareness is present in many of today's systems. Context-aware systems offer new opportunities for application developers and for end users by gathering context data and adapting systems behavior accordingly [Baldauf et al. 2007]. In the software development scenario, awareness can be characterized as "an understanding of the activities of others, which provides a context for one's own activities" [Dourish & Bellotti 1992]. Many researchers have recognized awareness as an essential part of collaborative software development and collaborative work in general [Treude & Storey 2010].

Awareness mechanisms allow software development stakeholders to be percipient of what goes on in the development scenario. Each mechanism has its specific purpose, i.e., aims at supporting a particular set of development tasks (e.g., providing information about the detection of potential conflicts in collaborative development, supporting parallel tasks in geographically distributed development, and so on), thus providing different levels of awareness according to the context. Moreover, as stated by [Treude & Storey 2010], depending on the context of the task at hand, the required granularity of awareness can vary significantly.

The inclusion of awareness mechanisms should take into account the tools most commonly used by stakeholders in their usual, daily activities, in order to ease the adoption and use of such mechanisms. For instance, among software developers, the use of IDEs is very common, and most of them are extensible by plug-in systems. Thus, developing awareness facilities as IDE plug-ins can benefit from the available IDE features, including integration with other tools [Hattori 2010] [Schots et al. 2012].

The use of visualizations can enrich development environments to promote awareness [Hattori 2010]. Awareness information can be delivered by means of visual resources especially employed to this end, e.g., dashboards [Treude & Storey 2010] that can summarize important development facts. One important aspect that must be taken

into account is the evaluation of the tradeoff between the usefulness of the visual cues and the level of distraction they may cause [Hattori 2010].

In [Ripley et al. 2007], a 3D (three-dimensional) visualization is presented for providing project managers with a comprehensive view of all project activities, allowing them to intelligently steer development and adjust task assignments. The screenshot shown in Figure 3.1 presents a snapshot of all ongoing changes taking place in a set of workspaces at a particular time [Ripley et al. 2007].

The stacks of cylinders with the most recent changes are placed in the front of the view and, as time elapses, stacks for workspaces with less recent activity start moving to the back. In the artifact mode, each stack of cylinders represents an artifact, and each cylinder in the stack represents changes to that artifact made by a workspace. In the developer mode, a stack of cylinders represents a developer's workspace.

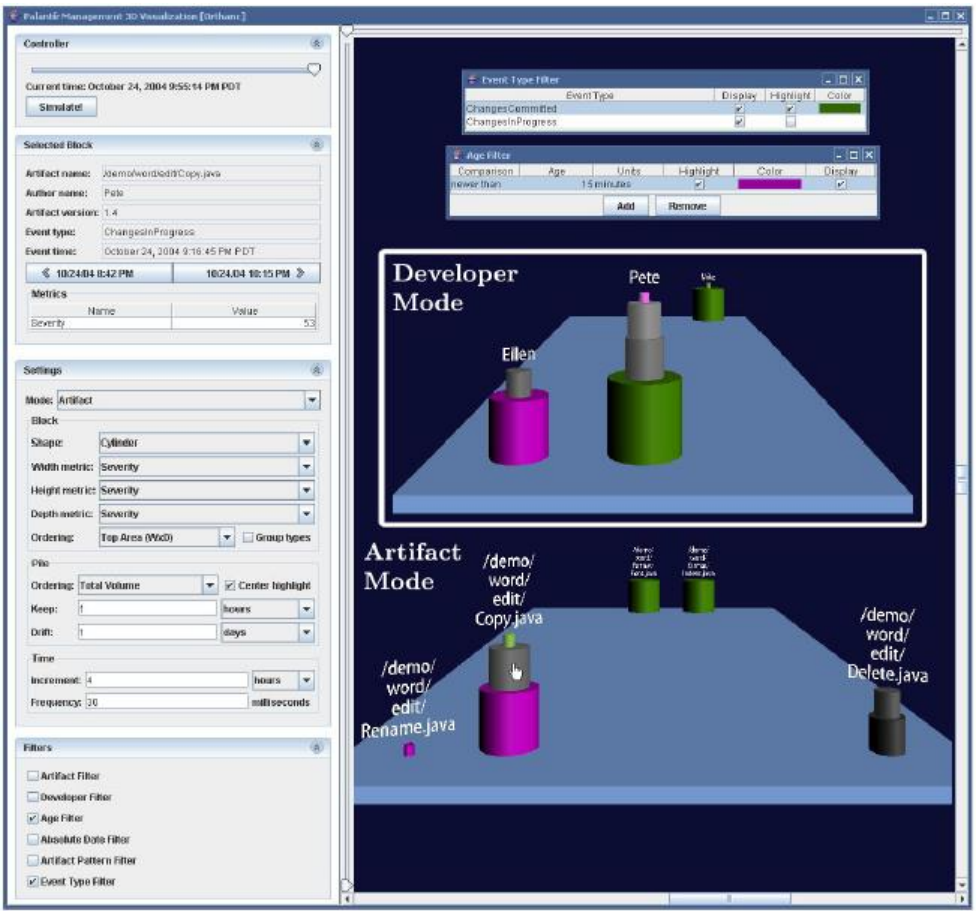

Figure 3.1 – 3D workspace visualization [Ripley et al. 2007]

Another subarea that has emerged is visual analytics, considered as "the science of analytical reasoning facilitated by interactive visual interfaces" [Thomas & Cook 2006]. Its goal is to increase insight into data through the combination of automatic analysis methods with human background knowledge and intuition [Keim et al. 2008].

3.2.2 Program Comprehension and Visualization

Program comprehension is a vital software engineering activity. It is necessary to facilitate reuse, inspection, maintenance, reverse engineering, reengineering, migration, and extension of existing software systems [Wong et al. 2007], among other software engineering practices. Particularly, it plays a crucial role in software maintenance: according to [Telea et al. 2010], about 40% of the maintenance budget is used for understanding source code.

The mapping of entities, from the software systems domain to graphical representations, aims to support comprehension and development [Gallagher et al. 2008]. In fact, many works that aim at increasing program comprehension make use of visual metaphors, by applying software visualization concepts and techniques. Such works usually try to represent software through a particular point of view, helping stakeholders to focus on the tasks being performed. Duru et al. (2013) state that software visualization tools allow users to synthesize and make sense of vast amounts of information (e.g., regarding the inner organization of software modules and the interactions between each other) [Duru et al. 2013].

An illustrating example of a software visualization tool is EvolTrack [Werner et al. 2011], an Eclipse-based extensible mechanism that combines multiple views to provide a better comprehension of the software evolution life cycle through different viewpoints. Its data source and visualization plug-ins allow performing different comprehension tasks, e.g., architectural conformance and co-evolution analyses, social network analysis, and tracking evolution of measurements. Figure 3.2 shows EvolTrack and its plug-ins PREViA (for architectural model conformance analyses over time) and SocialNetwork (for socio-technical network analyses) [Werner et al. 2011].

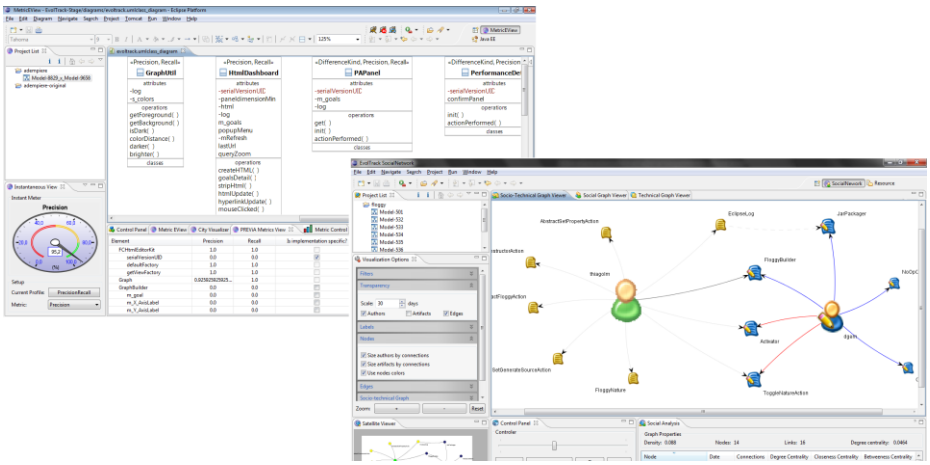

Figure 3.2 – EvolTrack and plug-ins PREViA and SocialNetwork [Werner et al. 2011]

Another illustrating example is the CodeCity tool [Wettel & Lanza 2008], presented in Figure 3.3. It displays source code information mapped into a city metaphor. The visual properties of the city artifacts reflect metric values of classes and packages (in the figure, the number of methods maps to the buildings' height and the number of attributes to their base size). The brown (darker) buildings represent the classes and the blue (lighter) districts represent the packages.

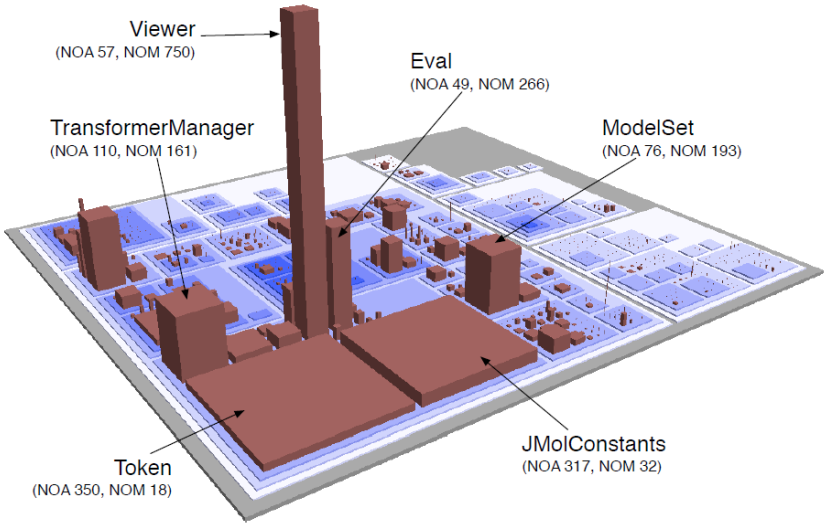

Figure 3.3 – The city metaphor presented in CodeCity [Wettel & Lanza 2008]

This figure illustrates a version of a system called Jmol. The visualization allows to easily identify outliers, such as the two large platforms (wide and short) in the foreground representing the classes Token and JmolConstants, which define many attributes (large base) and few methods (reduced height), or the "skyscraper" representing the class Viewer with a considerably high number of methods and a much lower number of attributes [Wettel & Lanza 2008].

3.2.3 Awareness and Comprehension Challenges

The creation of tools, techniques, and methodologies to support the manipulation of large data sets has been receiving special attention of both scientific and industrial communities, in order to discover new ways of dealing with the underlying information, including learning purposes, identification of patterns, decision-making support, among others. However, making use of computing resources to enhance awareness and understanding (of software information and the software itself) is still a challenge in software/systems engineering. It involves the identification of suitable mechanisms, adequate abstractions, and studies on stimulation of the human perceptive and cognitive abilities [Schots et al. 2012].

Among the grand challenges identified by the Brazilian Computer Society for the years 2006-2016 [Brazilian Computer Society 2006], the following somehow relate to these topics:

- The management of information in large volumes of distributed multimedia data, in order to develop solutions for the processing, retrieval and dissemination of relevant information, both narrative and descriptive, from the exponential growth of multimedia data;
- The computational modeling of complex systems (artificial, natural and socio-cultural) and human-nature interactions, particularly the creation of new algorithms and techniques in scientific visualization to enable visually capturing the complexity of the modeled objects and their interactions;
- The quality of technological development, which poses that systems must be available, accurate, secure, scalable, persistent, and ubiquitous – one of the research topics in this sense is the development of tools for supporting the process of implementation and evolution of software.

There is an increasing demand on how to obtain, handle/process, visualize, manipulate, and understand information, particularly data and information about software systems. Research topics that tackle this issue have the potential to deliver promising results, as well as ease and increase the quality of software processes and products. In this sense, some challenges in awareness and comprehension highlighted in [Schots et al. 2012] are listed as follows:

- General Comprehension/Awareness Challenges
  - *Use software tools to seamlessly collect rich data sets on software comprehension activities*: Kagdi & Maletic (2008) highlight the importance of automatic data collection mechanisms (e.g., eye tracking and activity logging) on software comprehension studies [Kagdi & Maletic 2008]. This can be cheaper and more reliable (with respect to data quality) than questionnaires, interviews and think aloud protocols. IDEs and VCS repositories provide means to collect this type of data. The challenge lies in associating low level (fine-grained) monitoring of actions on software engineering tools with the cognitive actions being executed (e.g., reading, searching or modifying the code). Data mining techniques (sequence and process mining) can help to achieve this.

- *Build specialized, personalized visualizations according to the comprehension needs*: Software visualizations should present a comprehensive view of the objects under analysis, based on the needs identified in industry and education.
- *Provide evidence regarding correlations involving people's profiles with respect to quality attributes on program comprehension*: There is little evidence on how (and whether) previous knowledge, skills, and abilities correlate with the efficiency and efficacy of understanding program artifacts. An example is to evaluate the influence of developers' level of expertise on how efficiently they understand code [Von Mayrhauser & Vans 1995]. Studies on stimulation of the human perceptive and cognitive abilities are welcome for understanding scenarios like this [Novais et al. 2012].
- *Identify and develop suitable mechanisms and adequate abstractions*: If an awareness/comprehension mechanism is not useful in its purpose, it will not be used in practice. It can be, among other reasons, due to its lack of flexibility and integration with other mechanisms. Thus, new tools and visualizations need to take this into account.
- *Strengthen and increase the group of researchers interested in software visualization, awareness, and related areas*: Despite the large number of studies undertaken involving software visualization, the Brazilian community in the area is still scattered. Attempts for establishing a joint research agenda are already in progress, aiming to allow the construction of a collaborative body of knowledge on software visualization and awareness, besides providing relevant solutions to the community as a whole.

- Industry-Related Challenges
  - *Understand the real needs of the software development industry stakeholders in terms of awareness and comprehension*: As one of the responsibilities of academia is to provide solutions to existing problems in industry, more studies should be conducted for identifying research opportunities. An example would be by performing primary studies, such as surveys and action-research.
  - *Evaluate the quality of existing data sources and identify relevant data*: The industry is increasingly realizing the importance of having data on the execution of their processes and metrics regarding the product, so that such data can be used to improve the performance of their activities and the quality of the final

product. However, it is necessary to ensure that (i) the data are collected, (ii) the data collected are useful and appropriate, and (iii) the data collected allow the analysis and improvement of processes and products.

- *Bridge the gap and encourage interaction between academia and industry*: Though this challenge is also pertinent for several other areas, research in software visualization and awareness lack evidence of their theories through results of studies performed in real settings. Research initiatives involving industry people with flexible formats could serve as a first step in this direction. Some potential results of such initiatives are the establishment of partnerships and exchange of human resources towards a holistic training for both communities.

As it can be seen, these challenges comprise software/systems engineering in general. Some of these challenges are expected to be addressed in the context of this work, focusing specifically on software reuse.

3.3 Software Visualization and Reuse

As mentioned in Chapter 2, introducing reuse in an organization may require new ways of thinking about software development. In order to achieve the acceptance/consciousness and successful adoption and institutionalization of software reuse, it is important to take into account how to provide appropriate reuse awareness. Awareness mechanisms allow stakeholders to be percipient of what goes on in the development scenario [Treude & Storey 2010] [Schots et al. 2012], and can provide them with the necessary information and support for performing their reuse-related tasks.

One of the ways to increase reuse awareness is by employing visualization resources and techniques. It is known that, in general, every visualization system supports understanding of one or more aspects of a software system, and this understanding process in turn supports a particular engineering activity or task [Maletic et al. 2002], such as requirements engineering, software design, or coding. It is believed that most of these software engineering tasks can also be visually supported by software reuse. Visualization resources can be used for allowing awareness and comprehension of reuse elements and their surroundings.

For instance, to reuse a software asset, stakeholders need to understand what it does, how it works, and how it can be reused; however, this is difficult in practice

[Marshall 2001] [Marshall et al. 2003]. If software engineers cannot understand assets, they will not be able to reuse them [Frakes & Fox 1996] [Alonso & Frakes 2000]. In contrast, a proper understanding can help developers to decide whether and how the asset can be reused [Marshall 2001] [Marshall et al. 2003], and visualization may play an important role in this context.

Several works aim to assist software engineering stakeholders in their day-to-day activities, but little is known on the role of visualizations in supporting software reuse tasks. Although existing visualization approaches intend to support somehow software reuse, literature lacks of a solid and comprehensive body of knowledge of software visualizations targeted to reuse. Consequently, stakeholders may not be able to choose reuse-oriented visualizations properly (i.e., based on their quality and concrete evidence on their actual effectiveness) for a given scenario.

In this sense, an informal literature review (first step of the research methodology presented in Section 1.5) was conducted for collecting preliminary information and providing the initial/basic knowledge about the research topic. This served as a basis for a secondary study (a *quasi*-systematic review), i.e., a more comprehensive study for characterizing the state-of-the-art (second step of the research methodology), aiming to identify software visualizations targeted to support reuse-related tasks.

All the details about the secondary study, including the full protocol description and the details on the analysis, are described in [Schots et al. 2014]. The next subsections present the approaches and tools identified by means of the informal literature review, as well as a framework that was created for categorizing visualizations and a brief overview of the planning and execution of the *quasi*-systematic review. An overview of the analysis and the discussion of results are presented in Section 3.4.

3.3.1 Findings from the Informal Literature Review

During the informal literature review, a number of works related to visualization and reuse were found, but some of them are not related to software development. An example is ALOCOM [Klerkx et al. 2006], which aims to visualize a large repository of learning objects in the form of small reusable content components. Such components are created by disaggregating legacy content, and some metadata are added to each component. The visualization gives an overview of the components in the repository, including how they are put together, in terms of “is part of” and “has part” relations.

The concept of component is very small-grained: examples include images, definitions, slides, and text fragments. Thus, the semantics of what can be characterized as a reusable asset is very wide. There are several other works along these lines; thus, for the sake of the scope, it was decided to focus the analysis of related works on software development.

Dy-re (Dynamic reuse) [Biddle et al. 1999] supports programming for reuse by displaying dynamic information of the internal structure of the software under development. It aims to make it easy to detect patterns of usage and patterns of dependence within a program – these patterns may help the programmer to determine how best to articulate the structure of a program using components that will be useful and independent for later reuse in other contexts [Biddle et al. 1999].

Dyno [Biddle et al. 1999] [Marshall 2001] [Marshall et al. 2003] is a tool for helping developers in reusing Java code, by means of a view based on their experience of using such code. It allows the use of visualization templates written in Java, which can be generic (for any data type) or specific (for certain data types). Developers can write their own templates. According to [Marshall 2001], the developer himself/herself must map visualizations to data, i.e., must inform "which method in the component maps to which sequence", and this can be a one-to-one or a many-to-one mapping. The author recognizes that this can be a problem, since "a developer may not know enough to know which methods should map to which sequence" [Marshall 2001].

Alonso and Frakes (2000) propose an architecture for visualizing reusable components from a software library, along with an example implementation. The architecture is based on two architectural styles: (i) pipes and filters, and (ii) repository. The repository stores and manages the assets and their metadata; the visual representation displays the data using a visualization metaphor; finally, the intermediate representation enables data interchange between the repository and the visualization [Alonso & Frakes 2000]. There is a strong dependency of the search input query, which indicates that the usefulness of the results is closely related to the quality of the search.

The Variant Analysis approach [Duszynski et al. 2011] focuses on recovering and visualizing information about commonalities and differences in the source code of multiple similar software systems (delivering quantitative information about similarities across system variants). By identifying parts suitable for transformation into reusable assets and planning necessary implementation steps, it aims at supporting the reuse

potential assessment and the migration to systematic software reuse, besides providing an overview of commonality distribution in the whole analyzed system family.

These publications compounded an initial data set that served as a preliminary input of control for the *quasi*-systematic review. In addition, the need for organizing the information from the findings motivated the extension of a framework for categorizing visualization approaches. This framework is presented as follows, instantiated to the software reuse scenario.

3.3.2 An Extended Framework for Categorizing Visualization Approaches

In order to identify the set of data to be extracted from the findings, this work uses the five dimensions of software visualization from [Maletic et al. 2002]. Other researchers have been using such dimensions for classifying their visualization tools (e.g. [Ripley et al. 2007]). The dimensions reflect the why, who, where, what, and how of the software visualization [Maletic et al. 2002].

In order to encompass other aspects related to the findings, two additional, complementary dimensions are proposed and used in this work: one related to the **requirements** of the visualization approaches (***which***) and other related to **evidence** on their use (***worthwhile***). All the dimensions are depicted in Figure 3.4.

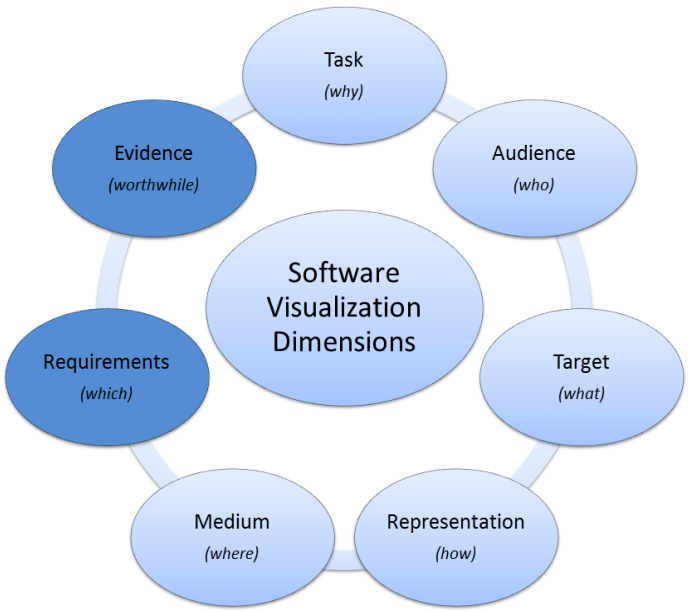

Figure 3.4 – Dimensions of software visualizations (extended from [Maletic et al. 2002])

Details on the dimensions and their corresponding information fields are described in the next subsections. Such dimensions are used in the *quasi*-systematic review for better organizing the findings [Schots et al. 2014].

3.3.2.1 Task – *why* is the visualization needed?

This dimension indicates what particular software engineering tasks are supported by the software visualization system [Maletic et al. 2002]. In order to understand why each of the visualizations is needed in the reuse scenario, it is important to identify which problems, motivations, or issues leaded to the development of such approach. In this sense, the following fields are used in this work:

- Approach motivation/Assumptions
- Approach goals
- Visualizations' reuse-specific goals

Besides identifying which are the supported tasks, one important aspect that can be relevant is *when* – i.e., in which software engineering activities and stages of the development process – the visualization can be used. Particularly in this work, the purpose is to identify the software engineering activities (in which there are reuse opportunities) that are somehow supported by the software visualizations. This can provide an overview of current tools' coverage, as well as identify opportunities for research and improvements.

Although this could be considered as an additional dimension (*when* is it visualized?), it was decided to keep it along with the *task* dimension (in which the supported tasks are identified), so that both can provide complementary information. Thus, this work resorts to the following fields:

- Software engineering activities addressed by the visualizations
- Reuse-related tasks supported by the visualizations

3.3.2.2 Audience – *who* will use the visualization?

The audience dimension defines the attributes of the users of the visualization system [Maletic et al. 2002]. As stated in Section 3.1, in order to make visualizations effective in their goal, it is noteworthy to keep in mind that the representations and visual metaphors must be adapted to the stakeholders' perception abilities, not the opposite [Diehl 2007]. The audience is represented in this work by the following field:

- Visualizations' audience (stakeholders who can benefit from the visualizations)

3.3.2.3 Target – *what* is the data source to represent?

The target of a software visualization system defines which (low level) aspects of the software are visualized, i.e., the work product, artifact, or part of the environment of the software system [Maletic et al. 2002]. Examples include architecture, design, algorithm, source code, data, execution/trace information etc. Other types of target source data are measurements and metrics extracted from software, process information, and documentation; this type of information can support the software process and team management activities [Maletic et al. 2002].

Software development surroundings themselves also provide several aspects that can be visualized. The aspects related to this dimension are described in this work by the following fields:

- Visualized items/data (what is visualized)
- Source of visualized items/data
- Collection procedure/method of visualized items/data

3.3.2.4 Representation – *how* to represent the data?

This dimension shows how the visualization is constructed based on the available information [Maletic et al. 2002]. According to [Few 2009], one of the aspects on which the effectiveness of information visualization hinges is the visualizations' ability to represent information clearly and accurately. In this work, the fields used for depicting this dimension are:

- Visualization metaphors used (how it is visualized)
- Data-to-visualization mapping (input/output)
- Visualization strategies and techniques

3.3.2.5 Medium – *where* to present the visualization?

The medium is where the visualization is rendered. The user must interact and perceive the visualization from some technology. Each medium has different characteristics and in consequence is suited for different tasks [Maletic et al. 2002]. Some of today's software visualization tools do not even exploit the graphics power of an average PC or laptop [Diehl 2007]. Moreover, the medium dictates how interactions may occur, and the effectiveness of visualizations also relies on the humans' ability to interact with them to figure out what the information means [Few 2009].

In this work, such dimension is comprised by the following fields:

- Device and/or environment used for displaying the visualizations (where it is visualized)
- Resources used for interacting with the visualizations

3.3.2.6 Requirements – *which* resources are required by or used in the visualization?

An important consideration in any interactive visualization system is performance, in particular responsiveness to changes triggered by direct manipulation of images [Bavoil et al. 2005]. The use of certain visualizations may become costly, not only in terms of processing but also due to specific hardware and software solutions which they depend on. This "cost" can be expressed in terms of the visualizations' hardware and software requirements, besides the programming languages, application programming interfaces (APIs) and frameworks reused for building it. For capturing information of this new dimension, the following fields are used:

- Hardware and software requirements
- Programming languages, APIs and frameworks used for building the visualization

3.3.2.7 Evidence – are the proposed visualizations *worthwhile*?

Software visualization tools are continuously being built by researchers and software development organizations [Sensalire et al. 2009]. However, according to these authors, many of the tool developers perform very limited or no evaluation at all of their tools. As a result, as stated by [Mulholland 1997], it may become unclear how effective such visualization tools are, either for students or professional programmers; moreover, there are some concerns on whether lessons in successive designs of software visualization tools, or whether the application of new technologies (e.g., 3D animation and the internet) has become the primary goal, rather than the true goal of making computer programs easier to understand [Mulholland 1997].

In order to determine if visualizations are worthwhile, i.e., effective in helping their target users, it is desirable that they are exposed to a proper evaluation [Sensalire et al. 2009]. Revealing how software visualization contributes to software understanding "requires a systematic investigation of the relationship between cognitive processes underlying software comprehension and the software visualization techniques" [Duru et al. 2013].

Thus, for providing an insight of the visualizations' worthiness beforehand, this dimension aims at characterizing which kinds of evaluations and assessment were carried out with the visualization (if any), as well as any indication (or identified limitations) for its use. This can be a useful indication for people interested in making use of the visualization.

Because quantitative evaluations that involve humans can be very time-consuming, at least qualitative evaluations should be performed during the design of visualization tools or posthoc [Diehl 2007]. For instance, two criteria that can be used for evaluating the mapping of data to visual metaphors are expressiveness and effectiveness [Mackinlay 1986] [Maletic et al. 2002].

The former refers to the capability of the metaphor to visually represent all the information to be visualized (e.g., if the number of visual parameters available in the metaphor for displaying information is equal to or greater than the number of data values to be visualized). The latter relates to the efficacy of the metaphor as a means of representing the information, and can be further distinguished in effectiveness regarding the information passing as visually perceived, regarding aesthetic concerns etc. [Mackinlay 1986] [Maletic et al. 2002].

According to Diehl (2007), typical problems with evaluations of visualization techniques are the use of toy data sets and the generation of visual artifacts that suggest nonexistent relations; but, above all, many evaluations are biased because they have been done by the developers of the visualization [Diehl 2007]. The author also states that the lack of empirical studies is a shortcoming not only of software visualization research, but also of software engineering and computer science in general, in accordance with [Tichy 1998].

Therefore, for describing this new dimension, the following fields are included in this work:

- Visualization evaluation methods
- Application scenarios of the visualizations
- Evaluated aspects
- Visualization evaluation results/outcomes

3.3.3 Outline of the Secondary Study (*Quasi*-Systematic Review)

Since this study aims mainly at characterizing the state-of-the-art, it is performed by means of a *quasi*-systematic review [Travassos et al. 2008]. This kind of study is

also known as systematic mapping study, i.e., a study that aims to identify and categorize the research in a fairly broad topic area [Kitchenham et al. 2009]. However, since this study must explore the same rigor and formalism for the methodological phases of protocol preparation and running (except for the fact that no meta-analysis in principle can be applied), the *quasi*-systematic literature review denomination seems to be more appropriate [Travassos et al. 2008].

The conducted secondary study aims at characterizing and identifying visualization approaches that can be used for somehow supporting software reuse, i.e., regardless of the focus of support. The study goals are described in the Goal-Question-Metric (GQM) format [Basili et al. 1994] as follows:

Analyze tools and approaches described in publications

For the purpose of characterizing

With respect to visualizations for supporting software reuse

Under the point of view of the researchers

In the context of software development tasks and organizational tasks

The objects of this study are the publications that present visualizations supporting software reuse. The expected results are (i) the identification of visualizations that can be used for supporting software reuse, as well as their features and limitations, and (ii) the establishment of a solid body of knowledge on visualizations for software reuse. Based on the findings, it is also expected to identify desirable features for novel approaches.

The research questions are partially inspired by the work of Maletic et al. (2002), and are decomposed into primary (*PQ*), secondary (*SQ*) and tertiary (*TQ*) questions, as follows:

- *PQ*: Which visualization approaches have been proposed to support software reuse?
  - *SQ1*: How do visualizations support software reuse?
    - *TQ1.1*: Which software engineering activities are addressed by the visualizations?
    - *TQ1.2*: Which reuse-related tasks are supported by these visualizations?
  - *SQ2*: To which stakeholders are these visualizations intended/targeted?
  - *SQ3*: Which items/data are visually represented?
    - *TQ3.1*: Where do these items/data come from?
    - *TQ3.2*: How are these items/data collected?
  - *SQ4*: Which visualization metaphors are used?

        - *TQ4.1*: How are data mapped to the visualizations?
        - *TQ4.2*: Which visualization strategies and techniques are employed?
    - *SQ5*: Where are the visualizations displayed?
        - *TQ5.1*: Which resources can be used for interacting with the visualizations?
    - *SQ6*: Which hardware/software resources are needed to deploy and execute the visualization tools?
        - *TQ6.1*: Which programming languages, APIs, and frameworks are used?
    - *SQ7*: Which methods are used for assessing the quality[23] of the visualizations (if any)?
        - *TQ7.1*: In which scenarios are the visualizations employed (if any)?
        - *TQ7.2*: Which aspects of the visualizations are evaluated (if any)?
        - *TQ7.3*: What are the results/outcomes of the conducted evaluations (if any)?

The research questions are related to the categorizing information presented in Section 3.3.2, according to the mapping presented in Table 3.1.

Table 3.1 – Mapping between research questions and information to be extracted

Task (why)	**Approach motivation/Assumptions** *(SQ1)*
	Approach goals *(SQ1)*
	Visualizations' reuse-specific goals *(SQ1)*
	Software engineering activities addressed by the visualizations *(TQ1.1)*
	Reuse-related tasks supported by the visualizations *(TQ1.2)*
Audience (who)	**Visualizations' audience (stakeholders who can benefit from the visualizations)** *(SQ2)*
Target (what)	**Visualized items/data (what is visualized)** *(SQ3)*
	Source of visualized items/data *(TQ3.1)*
	Collection procedure/method of visualized items/data *(TQ3.2)*
Representation (how)	**Visualization metaphors used (how it is visualized)** *(SQ4)*
	Data-to-visualization mapping (input/output) *(TQ4.1)*
	Visualization strategies and techniques *(TQ4.2)*
Medium (where)	**Device and/or environment used for displaying the visualizations (where it is visualized)** *(SQ5)*
	Resources used for interacting with the visualizations *(TQ5.1)*
Requirements (which)	**Hardware and software requirements** *(SQ6)*
	Programming languages, APIs, and frameworks used for building the visualization *(TQ6.1)*
Evidence (worthwhile)	**Visualization evaluation methods** *(SQ7)*
	Application scenarios of the visualizations *(TQ7.1)*
	Evaluated aspects *(TQ7.2)*
	Visualization evaluation results/outcomes *(TQ 7.3)*

The chosen search engine for carrying out the review is Scopus[24], due to its well-known stability, reliability, interoperability with different referencing systems, and

[23] Quality evaluation/assessment encompasses any quality attributes, such as effectiveness, efficacy, amongst others.

high coverage – its database indexes most of the publications that are available in different digital libraries or other search engines (e.g., Compendex, IEEE Xplore, ACM Digital Library, Springer, Web of Science etc.) [Santa Isabel 2011]. Besides, it indexes relevant journals and proceedings from the main software engineering conferences that comprise software reuse as a topic of interest. Examples of such conferences include:

- International Conference on Software Reuse (ICSR);
- International Conference on Software Maintenance (ICSM);
- European Conference on Software Maintenance and Reengineering (CSMR);
- International Conference on Information Reuse and Integration (IRI);
- International Conference on Program Comprehension (ICPC);
- International Conference on Software Engineering (ICSE).
- etc.

Since ACM is the only digital library that contains two of the control publications, it was decided to partially overcome this limitation by visiting the ACM Author Profile Page[25] of the respective authors and searching for the search string terms in the titles, abstracts and keywords of each listed publication. This decision was taken because the research described in these publications belongs to a specific research group and contains a set of related works (in terms of goals and features). More details on this issue are discussed in [Schots et al. 2014].

Because Portuguese is the native language of the researchers involved in this study, it was decided that publications in Portuguese should be analyzed as well. The following conferences were considered relevant for the purpose of this research:

- Brazilian Symposium on Software Engineering (SBES);
- Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS) and its predecessor Workshop on Component-Based Development (WDBC);
- Brazilian Symposium on Software Quality (SBQS).

Given that the Brazilian digital library (BDBComp[26]) did not index all the proceedings from any of these conferences (until the date of creation of the research

---

[24] http://www.scopus.com/

[25] See http://www.acm.org/publications/acm-author-profile-page for details (checked in November 30, 2013).

[26] http://www.lbd.dcc.ufmg.br/bdbcomp/

protocol), a manual search was required, following the same selection procedure (described in [Schots et al. 2014]).

The search string used was *((software OR system OR program OR asset) AND (reuse OR reusability OR reusable)) AND (visual OR visualization OR visualisation)*. Details on the definition of the search string can be found in [Schots et al. 2014]. Although a large number of publications were obtained, it was decided not to constrain the search string, due to the exploratory nature of this study.

A Portuguese version of the search string was also built: *((software OR sistema OR programa OR ativo) AND (reuso OR reúso OR reutilização OR reusabilidade OR reusável OR reutilizável)) AND (visual OR visualização)*. However, no results were found in the search engine through this search string. Thus, only the manual search on the identified sources should be performed for this language.

The manual search was performed in the following Brazilian proceedings:

- Brazilian Symposium on Software Engineering (SBES): from 1987 (1st edition) to 2012 (26th edition) (including);
- Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS): from 2007 (1st edition) to 2012 (6th edition) (including);
- Workshop on Component-Based Development (WDBC): from 2002 (2nd edition)[27] to 2006 (6th edition) (including);
- Brazilian Symposium on Software Quality (SBQS): from 2002 (1st edition) to 2012 (11th edition) (including).

The search results are listed in Table 3.2.

Table 3.2 – Study selection data (manual search)

	SBES	**WDBC/SBCARS**	**SBQS**
Title and abstract reading	556	158	315
Number of accepted publications	30	42	26
Number of rejected publications	526	116	289
Number of duplicate publications	0	0	0
Full reading	30	42	26
Number of accepted publications	0	0	0
Number of rejected publications	30	42	26
Number of duplicate publications	0	0	0

[27] The first edition of WDBC has no proceedings; selected works evaluated by the program committee were invited for publication in a book: Gimenes, I. M. S., Huzita, E. H. M. (2005). Component-Based Development: Concepts and Techniques [Desenvolvimento baseado em componentes: conceitos e técnicas] (in Portuguese), 1st ed., 304p., Ciência Moderna.

As it can be seen, from the 1030 analyzed publications, no one was selected. Most of the publications selected during the title/abstract reading (98) were related to software reuse; however, in the full reading, it was noticed that no publication mentions the use of visualization resources with the goal of supporting software reuse.

Regarding the search engines, the searches were performed on October 1st, 2012 at 3PM local time (UTC/GMT -3) in both the Scopus search engine and the selected ACM Author Profile Pages. Although no time constraint was set, it is believed from the search results that the publications were obtained in the range between 1980 and September 2012. However, publications that had not been indexed until the date of search may have been added to the digital libraries afterwards.

In total, 1159 publications were retrieved from Scopus by performing the search with the chosen search string. The publications were exported from this search engine and formatted in tables. The search performed on the ACM Author Profile Pages was conducted in a different way: all the publications listed in the pages of each key author identified from the control publications (as discussed in [Schots et al. 2014]) were manually exported, and their title, authors and keywords were extracted using regular expressions in a text editing tool (Notepad++). After that, duplicates were semi-automatically identified and removed, resulting in 304 results. Then, a semi-automatic search was performed using the search string terms, and 6 publications were returned.

Table 3.3 summarizes the study selection stages in terms of accepted, rejected, and duplicate publications. From this point, a M.Sc. student (referred to as second researcher) supported the selection stages listed in this table.

Table 3.3 – Study selection data (search engines)

	Scopus		**ACM**
	1st researcher	**2nd researcher**	**Both researchers**
Title reading	1159	1159	6
Number of accepted publications	411	320	6
Number of rejected publications	740	831	0
Number of duplicate publications	8	8	0
Abstract reading	411	320	6
Number of accepted publications	77	45	6
Number of rejected publications	326	275	0
Number of duplicate publications	8	0	0
Full reading	77	45	6
Number of accepted publications	**29**	**19**	**5**
Number of rejected publications	47	26	0
Number of duplicate publications	1	0	1

During the consensus stage (for conflict resolution), both researchers selected 19 publications, so these did not need to be reanalyzed. From the 15 publications selected only by the first researcher, 13 were included after discussion, and 2 were rejected. From the 5 publications selected only by the second researcher, 2 were included after discussion, and 3 were rejected (being 1 by a third researcher, since consensus had not been achieved). Details on the consensus stage can be found in [Schots et al. 2014].

Beyond one of the control publications that was manually added (because it was not indexed by any academic search engine), another related publication [Anslow et al. 2004] was manually added. It was found based on the citations of the ACM key authors. It was agreed in the consensus stage to include it, along with the control publications previously included. In total, 36 publications were selected, describing 34 approach proposals. They are listed in [Schots et al. 2014].

3.4 Discussion

Regarding the way visualizations aim to support software reuse (*SQ1*), it was noticed that ever since the first identified works were published, there was already a concern on supporting reuse of a variety of artifacts [Mancoridis et al. 1993] [Constantopoulos et al. 1995]. It can also be noticed that most approach goals are artifact-oriented, not taking into account the dynamics of reuse in an organization (i.e., correlating projects, assets, users/developers and the relationship between these core elements).

Although approaches somehow encompass diverse software engineering activities (*TQ1.1*), it can be noticed that only a few of them present integration among different activities. Particularly, in software development for and with reuse, it is relevant not only to understand assets in terms of their properties and metadata, but also to understand how they are being used and maintained, in order to increase reuse confidence. Another concern about the approaches is the lack of integration with development environments that can provide interfaces to other activities.

There is support for a variety of reuse tasks (*TQ1.2*), and understanding assets is by far the most supported one. This is indeed expected, since understanding is a likely benefit in employing visualizations. Nevertheless, research on using visualization for restructuring assets for reuse is still much underexploited: the identified works are very specific, with limited customizability and integration support.

In terms of the different aspects that can be the focus of comprehension, most approaches support understanding the structure of an asset. Evolution information about reusable assets is a particular absence of existing works – the only work found deals with comparing refinement sets of different versions of feature models, and it is based on a trace repository; no other evolution aspects are taken into account by any approach.

Creating a modular architecture (including repositories and visualizations) for supporting different stages of the analysis process (e.g., [Anslow et al. 2004]) may give more flexibility to the approaches, allowing new functionalities to be added and, consequently more support for software engineering activities and tasks.

Although there is a reasonable variety of stakeholder support (*SQ2*), only a few works support more than one kind of stakeholder simultaneously. The lack of multi-stakeholder approaches hamper the evaluation of how well organization's goals related to reuse are being accomplished, under the perspectives of each reuse stakeholder (e.g., producers, consumers and reuse managers).

Particularly, the only approach that mentions support for managers [Mancoridis et al. 1993] states that the visualization is targeted to developers, but actions that could eventually be performed by managers are pointed out. However, since it only presents technical details about a software project, it does not seem feasible for the reality of project management (and top management), which is a key role in reuse initiatives.

The majority of the visualized items and data (*SQ3*) are source code artifacts (object-oriented entities, such as classes and relationships, or software components). In spite of this imbalance, there are many different kinds of artifacts (from different software development stages) that can be visualized, such as design artifacts, requirements, and web services.

There are few approaches for visualizing software repositories with the intention to promote reuse (providing relevant reuse data), and no repository information or metadata are visually represented aiming to increase awareness. According to [Orso et al. 2000], approaches that employ reuse repositories must store not only the reusable assets, but also the information about them (usually called metadata), e.g., a short description of their functionality and non-functional requirements. Understanding a reuse repository beyond its structure (e.g., in terms of how its assets have been used) may improve reuse planning and considerably increase the chances of reuse.

It is interesting to observe that the data sources (*TQ3.1*) are usually the source code of a program and databases. Only a few approaches combine information from

different sources (e.g., [Kelleher 2005]), and some are compatible with a limited set of data types. Although it is known that several kinds of information may be used for supporting reuse, it was noticed that some common data sources are not explored by any of the works (e.g., VCS repositories, issue trackers etc.). Moreover, although many assets have additional related data available online, such data are usually underexplored.

Other possibilities for the use of data should also be explored in the reuse scenario. Evolution information, for instance, can be relevant since it allows assessing an asset's stability and frequency of updates (i.e., how active the development community is), supporting the decision making regarding its reuse. Projects' history is also useful for analyzing projects in which there have been reuse attempts and, from those, which were successful and why. These sources also allow identifying new assets candidate to reuse. Social information (e.g., who developed/reused what) seems to be important as well, and may be quite relevant for supporting reuse decisions, but this is also not well exploited.

Since each visualization technique may have some constraints, each collection procedure (*TQ3.2*) must deal with this issue and make the proper arrangements. For instance, in [Kelleher 2005], some format conversions are mentioned in order to make the data ready to be represented by the intended visualization. During the data collection procedure, the source may still require some transformations to have the data set in the correct format to be used by different representations. Some authors also defend the use of intermediary formats for storing the collected data (e.g., [Alonso & Frakes 2000] and [Anslow et al. 2004]) in order to make them reusable in different visualizations.

Regarding the representation of data (*SQ4*), as expected, different abstractions are used for representing different data. Although several types of abstractions are used, publications lack a discussion on how/why a given metaphor was chosen and, more importantly, whether it is effective or not in its purpose. The mapping between data and visualizations (*TQ4.1*) is barely described in most of the publications. Some do not even mention how data maps to visualization, so the reader/user has to "guess" it, which sometimes can be risky and lead to wrong interpretations of data.

Moreover, although several visualization strategies and techniques are used (*TQ4.2*), only a few approaches make a comprehensive use of them. This does not mean that every possible technique should be employed, but some approaches may require more interaction facilities, which may increase their usability.

In this scenario, customizable approaches may support different perception needs and give more flexibility to the user, in order to tackle a wider range of software engineering tasks. However, there is a lack of mechanisms that offer flexibility to software stakeholders in customizing their visualizations, so one can focus on relevant data and information to improve the understanding of their activities (as already stated in [Silva et al. 2012]). In the other hand, letting the user decide which visualization to use may not be adequate, as he/she may not have experience with the metaphors or may not know which metaphors better fit the structure to be visualized (as discussed in Section 3.3.1).

In [Marshall 2001], for instance, the amount of required mapping information that the user needs to supply was intended to be minimized, but the user must map visualizations to data. As mentioned in Section 3.3.1, the author recognizes that this can be a problem, as the purpose is to understand the component, and "a developer may not know enough to know which methods should map to which sequence". However, there is no tool support for this task. According to the author, "it is bordering on the impossible for a tool to be able to automatically create mappings from one arbitrary name to another arbitrary name, so it is necessary for the developer to say which method in the component maps to which sequence", and this can be a one-to-one or a many-to-one mapping [Marshall 2001]. Nevertheless, there should be at least some kind of support for filtering inappropriate visualizations according to underlying restrictions associated with the data.

The flexibility may also be compromised due to some approach restrictions. In [Marshall et al. 2003], the collected information for creating visualizations as a complement for documentation is mostly based on the developers' experiences of using the components, and creating visualizations "does require some prior knowledge of the component and its important features and uses (i.e., knowing what to focus on in the visualization)" [Marshall et al. 2003]. This means that "any configuration is better left to experienced users who wish to create visualizations of that component for other developers". Nevertheless, there does not seem to be any support for either the developer or other users in creating, choosing, or selecting visualizations.

Some works try to generate flexible visualizations, but they usually require expertise knowledge (e.g., programming skills for configuring/mapping views and data) for stakeholders to operate them. A list of works in this regard can be found in [Silva et

al. 2012]. When an approach makes an assumption of a particular technical knowledge for creating the visualizations, it may potentially inhibit some stakeholders to use it.

Regardless of the number of occurrences for each of the strategies, it is unwise to affirm that certain techniques are more important than others. Visualization strategies and techniques must be chosen according to the goals. Moreover, the available data must meet the representation constraints associated to the employed visualizations.

From the approaches that use computers as a medium for displaying information (*SQ5*), only a few explicitly mention that they work in (or are integrated with) a web environment. Depending on the focus of the task and the sources of data, web environments may be more appropriate, since computers are usually equipped with web browsers, not requiring any additional installation procedure. Moreover, in spite of the existence of these web-based approaches, it cannot be stated that they are supported on multiple devices, since some (such as smartphones and tablets) contain displaying and interaction constraints that must be accounted for when designing visualizations.

Additionally, even among more recent approaches, none mentions or focuses on mobile devices as an alternative to execute and interact with the visualizations. They can be useful for monitoring-related activities for some stakeholders (e.g., managers and top management, who may not access a computer all the time, and are often "on the go" for business issues).

Regarding the resources used for interacting with the visualizations (*TQ5.1*), it is not surprising that mouse and keyboard are the main ones, as current information visualization systems are still largely focused on these peripherals for interacting with data [Lee et al. 2012].

In spite of that, there has been a constantly growing interest in other research areas for incorporating other natural forms of interaction such as touch, speech, gestures, handwriting, and vision. However, these new forms of interaction need to "follow the basic rules of interaction design, which means well-defined modes of expression, a clear conceptual model of the way they interact with the system, their consequences, and means of navigating unintended consequences" [Norman 2010].

According to the same author, because gesturing is a natural, automatic behavior, systems also have to "be tuned to avoid false responses to movements that were not intended to be system inputs". Thus, as an interaction technique, gestures "need time to be better developed", so that interaction designers can "understand how best to deploy them" [Norman 2010].

Software and hardware requirements (*SQ6*) are not discussed properly in the publications, which hampers the proper evaluation of the feasibility of the approaches to particular contexts. The same occurs with information about programming languages, APIs, and frameworks (*TQ6.1*), which would help evaluate how up-to-date a tool is, as well as to identify any potential integration constraint. It can be noticed that some of the technologies used by the approaches are already in disuse.

As mentioned previously, a particular concern is the lack of integration with development environments: this hampers integration with other tools, and may require additional efforts from stakeholders to cope with reuse tasks. IDEs have an enormous amount of information about the developer and his/her system, and using the IDE as the source of information is the closest way to understand the developer's intentions [Robbes & Lanza 2006]. An effective integration with existing IDEs could provide useful information for each approach purpose.

Finally, in the evaluation dimension, it was observed that the majority of the works does not present a proper evaluation on their use (*SQ7*): some of them do not present any at all. This can be partially explained by the lack of demand for evidence in publications (a scenario that has been changing in the last years). In many cases, the evaluation is done by the authors themselves, which is subjective and may bring some bias. The absence of proper evaluations may raise questions as regards to meeting the purpose to which the approaches were proposed/constructed. This can be seen as a major downside.

Moreover, the reported data on the evaluations in general lack more useful details, so that it cannot be understood in which scenarios they have been conducted (*TQ7.1*), which aspects have been evaluated and why (*TQ7.2*), how the analysis has been made and which strengths and opportunities for improvements have been identified (*TQ7.3*). Recent works present a proper experimental soundness that helps to understand the identified limitations, so that other researches aiming to support reuse can use their evaluation report as a basis.

An interesting finding is that there is a balance between the evaluation scenarios (*TQ7.1*), since not only academic projects are used, but open source projects are also taken into account (which allows the verification of results), as well as commercial/industrial (thus strengthening the interaction with industry). Still, the field lacks studies on whether the perceptive and cognitive abilities of the stakeholders in carrying out software reuse tasks are properly stimulated. Particularly, since industry

stakeholders can directly benefit from the results of such studies, experiments in industry are strongly recommended for strengthening interaction with academia.

Some limitations of this study include: (i) the chosen search string, which may have not captured some relevant related work; (ii) the scope of analysis, which was based solely on the content of the retrieved publications (i.e., no other source was taken into account); (iii) the lack of variety of search engines used, which may not be representative, and (iv) the publication selection and the data analysis, which were made from the viewpoint of the researchers, and may be biased.

More details on the analysis and the *quasi*-systematic review process as a whole can be found in [Schots et al. 2014].

3.5 Final Remarks

Software visualization can be a useful resource for supporting software reuse demands, especially the ones related to awareness and comprehension. In spite of that, as shown in this chapter, its potential has not yet been thoroughly explored, i.e., there is room for research and development in this regard. Although there are publications in the literature that propose visualization approaches geared specifically for software reuse, few approaches aim at assisting reuse management as a whole, i.e., providing the necessary support to carry on a range of software reuse tasks.

The software engineering community can use the results found in the *quasi*-systematic review as a starting point for future research directions that can be addressed when choosing, instantiating, or developing visualization-based approaches for supporting software reuse. Besides, the presented information can be used as a body of knowledge not only to support the decision making regarding the choice of visualization approaches for software reuse, but also to conduct other secondary studies on software visualization applied to another field of interest (e.g., software maintenance). This study can also be seen as a summarized catalog of the approaches, whose further information can be obtained from the corresponding original publications.

The identified limitations of the current findings and the unexploited research opportunities point out directions and desirable features for a novel approach for providing awareness and visualization support to activities related to software reuse. Such approach is presented in the next chapter.

CHAPTER 4 – PROPOSED APPROACH: THE APPRAISER ENVIRONMENT

This chapter introduces the approach proposed in this work (called APPRAiSER), which uses visualization resources for supporting software reuse tasks. The environment that concretizes the approach and its elements are presented along with some aspects about their implementation.

4.1 Introduction

Based in the literature reports and the semi-structured interviews (both described in Chapter 2), some issues in implementing a reuse program were identified. These issues allow the identification of a number of desirable features for an environment to support software reuse. However, dealing with all these issues would not be possible in the context of a single thesis.

For illustration purposes, the Odyssey environment[28], developed by the Software Reuse team at COPPE/UFRJ, comprises a number of Undergraduate Final Projects, M.Sc. dissertations and Ph.D. theses since its inception, which resulted in several publications (e.g., [Braga et al. 2006], [Blois et al. 2006] and [Fernandes et al. 2011]) and research collaborations (e.g., [Mello et al. 2014]).

In fact, based on the results from the *quasi*-systematic review, it can be noticed that some of the identified works have addressed (even partially) part of the identified issues. For instance, Ali (2009) supports the understanding of Java software libraries in terms of their structure [Ali 2009], while in [Biddle et al. 1999] the support is also given in terms of understanding behavior. However, despite the limitations of these approaches (discussed in Section 3.4), it turns out that they focus on individual problems, which may not be effective, since the effective implementation of reuse initiatives requires dealing with other activities that are in some ways interdependent.

Moreover, the performed research could not identify any work trying to address in an integrated way a number of reuse tasks in which visualization and comprehension

[28] http://reuse.cos.ufrj.br/odyssey

aspects seem to be important, especially regarding the exploration of a reuse repository as a whole (searching for a reusable assets or analyzing how a given asset has been reused) and the collection of information from data sources that can provide additional information for reuse.

As discussed in Chapter 2, it is necessary to provide visualization metaphors for representing reuse data, so that stakeholders can interact with and manipulate such data and obtain answers to their tasks quickly, besides decreasing the cognitive overload. In this sense, this work aims to define and implement an interactive visualization approach that supports introducing, instigating, and monitoring software reuse initiatives. This is accomplished by assisting stakeholders in awareness and comprehension aspects related to executing some software reuse tasks, such as exploring a reuse repository, obtaining and understanding information regarding reusable assets, and monitoring reuse initiatives.

The realization of the proposed approach (third step of the research methodology presented in Section 1.5) is achieved through an environment called APPRAiSER: an **A**pproach for **P**erceiving and **P**romoting **R**euse by **A**wareness **i**n **S**oftware **E**ngineering and **R**eengineering [Schots 2014]. An appraiser "has the knowledge and expertise necessary to estimate the value of an asset, or the likelihood of an event occurring, and the cost of such an occurrence"[29], which relate to the expectations of this work.

APPRAiSER aims to assist the execution of some tasks related to software reuse, both at the organizational level and the project level. At the organizational level, it focuses on supporting the management of assets and their surroundings, as well as monitoring reuse initiatives. At the project level, APPRAiSER aims to support the search and selection of assets to reuse, as well as the understanding of such assets and their properties. Moreover, it supports the reengineering of existing assets for reuse. Regarding reuse stakeholders, it is expected to allow:

- *developers* to better explore a reuse repository and its assets, accessing all the available information about them (when developing with reuse) and better understand how reusable their developed assets are, and realize how to improve their assets for future reuse (when developing for reuse);

[29] http://www.investopedia.com/terms/a/appraiser.asp

- *reuse managers* to evaluate assets candidate to reuse and the reuse repository as a whole (e.g., in order to examine possible reasons why a given asset is not being reused and whether it should be discontinued), as well as identify experts (producers and consumers) on a reusable asset; and
- *all stakeholders* to monitor the progress of reuse initiatives and be aware of how reuse can help achieving some development goals with less effort and time, increasing their efficiency and efficacy.

Through the APPRAiSER environment, it is expected to provide reuse awareness to stakeholders through visualization resources that can help them be aware of the reuse scenario as a whole and perform reuse tasks more accurately. To this end, this work also uses and extends some academic (Undergraduate and Master) projects.

It must be emphasized that it is not assumed that the proposed environment is sufficient to address all the issues, since there are several non-technical aspects related to a successful reuse program (as mentioned in Chapter 2). Thus, they are in no way intended to exhaust the corresponding issues, but just a way of approaching them. Moreover, although it is believed that the use of APPRAiSER can contribute to dealing with some non-technical aspects, this work focuses only on some technical aspects of a reuse initiative.

It is known that there are several kinds of reusable assets. However, as stated in Section 2.5.2.2, using source code as reusable assets allows the benefits arising from reuse to be more noticeable by organizations. Moreover, reuse of source code artifacts is still on the mainstream of software development [Schots & Werner 2013]. Thus, in this work, APPRAiSER focuses mainly on object-oriented source code artifacts (including frameworks and libraries, assuming that they are packaged somehow).

4.2 APPRAiSER Elements

Since visualization tools become more useful when they operate in an environment that can provide integration with other tools and resources [Werner et al. 2011], the APPRAiSER environment comprises a suite of elements/tools[30], which are briefly described in the next subsections. An overview on how these tools are integrated is presented in Figure 4.1.

[30] Most of these tools were or are being developed in collaboration with other students, as described later.

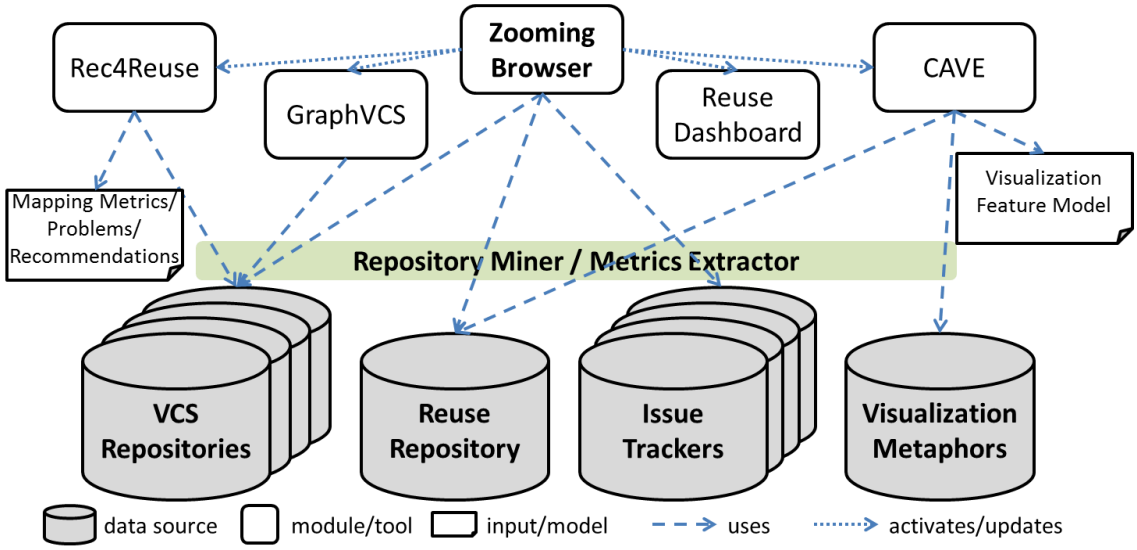

Figure 4.1 – APPRAiSER Overview

Figure 4.2, in turn, presents in a nutshell a mapping between the supported stakeholders, the tools, and the reuse tasks they are supposed to meet (based on the tasks listed in Table 2.1), in order to allow a better understanding of the expected benefits of the approach. Although other stakeholders may benefit from the approach features, the focus of this work are the ones depicted in this figure. Moreover, as stated in Section 2.3, a given task may be executed with a different focus by each kind of stakeholder.

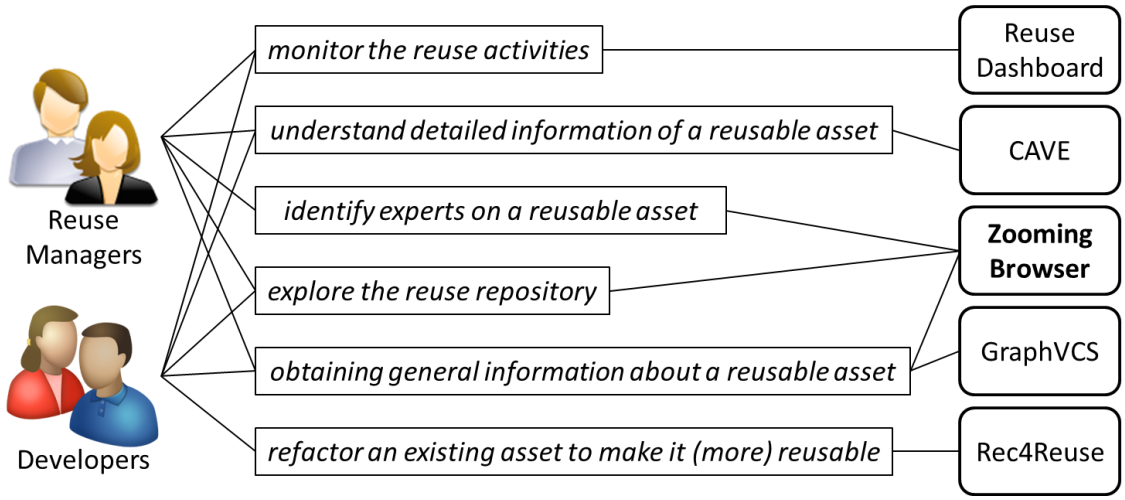

Figure 4.2 – Mapping between stakeholders, reuse tasks and APPRAiSER tools

Details on the each of the APPRAiSER elements are discussed in the next subsections.

4.2.1 Repository Miner

As mentioned in Chapter 2, software reuse can be sometimes restrained by the loss or overlook of relevant information. Moreover, as stated in Section 3.4, some

common data sources are underestimated. It is important to consider that an asset of good quality may not be reused due to the lack of information about it. Such information is important to ensure that they can be retrieved according to the developer's needs [Hadji et al. 2008].

In this sense, the *Repository Miner* mechanism is responsible for mining software repositories, both organizational (internal) and external, searching for different kinds of reuse-related information, including evolution, social, and metadata information, which is delivered to the tools that compose the APPRAiSER environment and the *Metrics Extractor* mechanism.

Three submodules compose the Repository Miner, as follows:

- The *Reuse Repository Miner* collects assets and their metadata from reuse repositories, supporting the maintenance of reuse initiatives. It provides social information regarding producers (organization, developers, contact information, website), as well as assets' descriptions, licenses, dependencies, among others.
- The *VCS Miner* captures information from VCS repositories, such as commits, authors, dates, log messages and so on. According to the repository layouts, it can also obtain additional information, such as the number and frequency of releases, among others. Moreover, it allows analyzing projects' history for identifying reuse occurrences, which can indicate (i) how assets present in the reuse repository are being reused, and (ii) assets that are frequently reused and can be suggested as candidate assets to the reuse repository (depending on organizational criteria).
- The *Issue Tracker Miner* obtains issues (such as reported problems and feature requests) from issue tracker repositories. It also collects metadata regarding the issues' status, priority, start and due date, percentage of work done and interested parties.

Among the Repository Miner submodules, the VCS Miner is the most critical in terms of interchangeability, since there are several software projects stored in different version control repository technologies, such as CVS, SVN, Git, and Mercurial. The diversity of such technologies would require specific connectors to be implemented. To overcome this limitation and gather historical information from different VCSs, EvolTrack-VCS connector [Werner et al. 2011] will be incorporated to APPRAiSER. It makes use of the Maven SCM API, which provides mechanisms for accessing version control repositories via generic interfaces. EvolTrack-VCS also aims to provide extensibility, so that other VCSs can be added over time, requiring only the

implementation of some interfaces. Currently, the connector communicates with 12 popular (commercial and open source) VCSs [Werner et al. 2011].

The *Reuse Repository Miner* uses the Maven API for accessing the open source Maven Repository[31] and for creating a local repository for the organization. This decision was made due to the large variety of artifacts of this repository (around 72,978 artifacts and 628,615 versions)[32] and due to the technology stability. The communication with Maven repositories is made through the REST (Representational State Transfer) API. An existing implementation [Chaves 2013] will be adapted and integrated to APPRAiSER.

The *Issue Tracker Miner*, in turn, will use the Redmine REST API[33], assuming the use of the Redmine issue tracker system, also based in an existing implementation [Queiroz et al. 2012]. It is intended to reuse a more generic and comprehensive solution, such as Mylyn API connectors[34]; however, a deeper analysis is required to this end.

A kernel module will orchestrate the flow of information between the APPRAiSER modules and specify the required interfaces to be implemented by plug-ins. All the process can be triggered by the Repository Miner, which keeps listening to software repositories periodically seeking for new data[35]. Repository Miner mechanisms can also be mere observers, being activated by triggers or hooks – some data providers (such as VCS repositories) implement such features.

It is intended to implement the data interchange between most of the modules using REST and JSON (JavaScript Object Notation)[36] solutions, since they are lightweight and offer a more human-friendly way of representing data. Besides, this allows data to be reused in different visualizations (as discussed in Section 3.4). Although JSON is derived from JavaScript, it can be read and written by several programming languages.

---

[31] http://search.maven.org/

[32] http://search.maven.org/#stats (data from March 13, 2014).

[33] http://www.redmine.org/projects/redmine/wiki/Rest_api

[34] https://wiki.eclipse.org/Mylyn/Integrator_Reference#Tasks_API

[35] EvolTrack-VCS, for instance, can operate in two built-in modes of extraction: (i) real-time mode, in which the connector searches for a new version (if any) in the repository to add it immediately to the evolution flow, and (ii) traditional mode (real-time option off), only working with previously selected versions [Werner et al. 2011].

[36] http://json.org/

It is also intended to incorporate additional data from Github, through the Github API[37]. Particularly, source code data can be analyzed through an Abstract Syntax Tree (AST), which organizes the source code in a tree representation, allowing the creation of links between pieces of code. Eclipse provides an AST implementation.

4.2.2 Metrics Extractor

The Metrics Extractor mechanism is responsible for handling and manipulating the input from the Repository Miner, making the necessary calculations and aggregations. It is not embedded into the Repository Miner so that their particular operations can be performed separately, allowing reuse of the same "raw" data provided by each of them.

The current implementation of the Metrics Extractor is based on Metrics2[38] Eclipse plug-in, whose modules responsible for reading the metrics and extracting the measurements were reused [Palmieri et al. 2013]. This plug-in was chosen because it is easy to reuse, it is open source, and it allows the inclusion of new metrics, in addition to providing a large number of metrics that can support reusability assessment (e.g., lack of cohesion, depth of inheritance tree and number of children).

In order to effectively understand a metric value, one must need to look at some reference point, i.e., thresholds [Lanza & Marinescu 2006], which can be based on statistical information or a "common sense" (i.e., implicitly given by observations). Because there can be specific organizational parameters or other impact factors, thresholds can be defined/obtained in three different ways: (i) based on metrics datasets (such as COMETS [Couto et al. 2013] [Oliveira et al. 2014]), (ii) based on calculated values from organization's data (e.g., obtained from the Repository Miner), and (iii) customized values, manually informed by the organization.

The composition of metrics and thresholds characterizes an *evaluation plan*, inspired by the detection strategies found in [Lanza & Marinescu 2006]. It consists of a set of comparative associations between measures and thresholds that work as a "rule" that characterizes a particular situation. If the result of all measure-threshold comparisons is true, this indicates the occurrence of the specified situation [Palmieri et al. 2013].

---

[37] http://developer.github.com/v3/

[38] http://metrics2.sourceforge.net/

Both the information related to the metrics extraction and the evaluation plans are stored in a database, which serves as a bridge between the Metrics Extractor and APPRAiSER tools, sharing the data required by the visualizations. The steps of extracting data and metadata can be accomplished through data mining techniques.

For allowing stakeholders to create their own evaluation plans, as well as reuse (and customize) previously existing plans, a GUI was implemented in the Eclipse IDE [Palmieri et al. 2013]. A web GUI is going to be implemented as well. Currently, such plans are used for analyzing assets and projects developed in Java. Figure 4.3 displays the selection and creation of an evaluation plan.

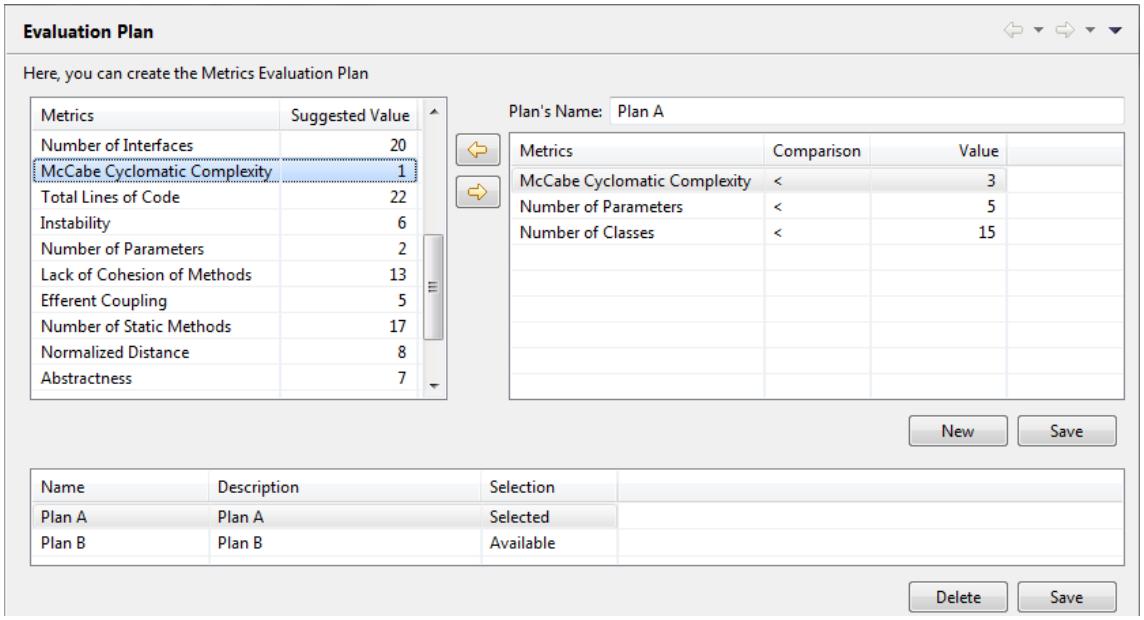

Figure 4.3 – An example of Evaluation Plan in Eclipse [Palmieri et al. 2013]

In order to provide an up-to-date status of the reuse scenario, all the integrated visualizations must be automatically updated (in real time or in configured intervals), displaying metric value changes. For real-time updating of the visualization tools, the Observer pattern is used, so that views are notified and updated in an event occurrence (such as changes performed in another view or changes in metric values), if desired.

4.2.3 GraphVCS

Visualizing a project evolution can facilitate comprehension and support decision-making by the development team, as well as support the execution of more specific tasks, such as configuration management audits, thus complementing VCS client tools. In this sense, GraphVCS [Pereira & Schots 2011] aims at providing a better comprehension of the evolution of versioned software projects, using visualization

resources and techniques to allow exploring the structure of software repositories and facilitate the identification of potential problems.

The repository structure is represented in GraphVCS through visual graphs, in which each operation commit and project milestones (tags) are represented as nodes, while edges represent the main line of development (trunk) or its derivations (branches). The tool allows pan and zoom operations, as well as drill-down to a given version. In order to facilitate the location of project information, a search tool is integrated with the visualization features, allowing to filter the displayed information according to their relevance. This facilitates the exhibition of details on demand, and allows for maintaining context without losing the focus of the task. The search can be performed by date, version ID/number, commit messages, author, and file. A commit statistics view is also displayed for providing additional information, such as commit frequency.

In terms of software reuse tasks, a useful resource for evaluating reusable assets is their history, which is usually stored in VCS repositories. Such information is relevant because it allows assessing assets' stability and frequency of updates (i.e., how active the development community is), supporting the decision making regarding their reuse.

As mentioned in Section 3.4, projects' history is also useful for analyzing projects in which there has been reuse attempts and, from those, which were successful and why. It also supports identifying new assets candidate to reuse. Thus, GraphVCS can be integrated to the APPRAiSER environment for supporting these needs.

A proof of concept of GraphVCS approach was implemented using the Subversion version control system. By integrating GraphVCS with APPRAiSER, the repository implementation will become VCS-independent, enabling communication with different types of repository. For the generation of visualization, the Prefuse[39] visualization library is used. A graph layout implementation was extended in order to display trunks and branches in a more intuitive way. For the statistics view, the JFreeChart[40] library is used.

Figure 4.4 shows an excerpt of the tool, demonstrating the search for a given commit message. The project committers (listed on the left) can be used as filtering criteria. Commit activity is shown on the bottom left.

---

[39] http://www.prefuse.org/

[40] http://www.jfree.org/jfreechart/

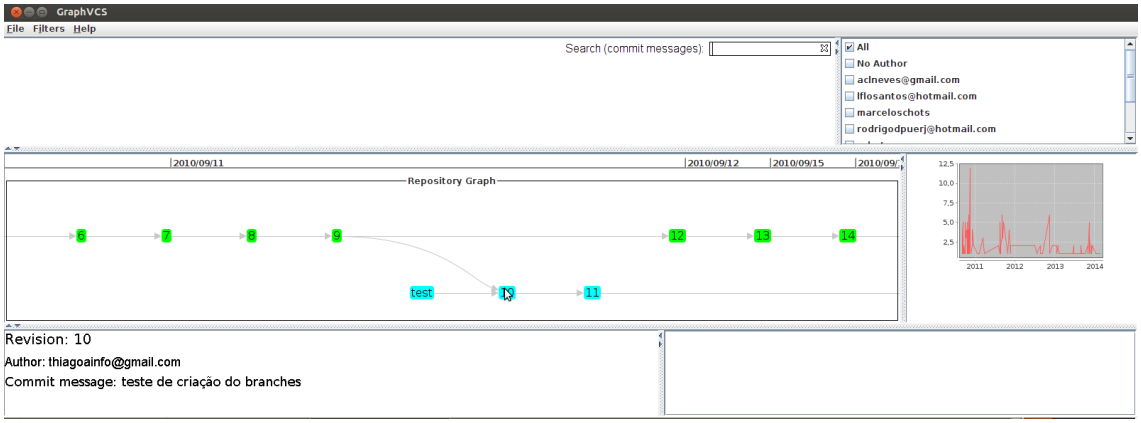

Figure 4.4 – GraphVCS Screenshot [Pereira & Schots 2011]

4.2.4 Context-Aware Visualization Engine (CAVE)

As stated in Section 3.2, enhancing awareness and understanding of software information and the software itself requires the identification of adequate abstractions according to the comprehension needs [Schots et al. 2012]. The choice of the visualization abstractions and techniques for representing the data, as well as the interaction techniques to be employed, heavily depends on contextual information, e.g., the nature of data, the visualization constraints, and the task to be supported (e.g., selecting the most suitable asset from a set of reusable assets) [Queiroz et al. 2013].

Specific visualizations must be built in order to better represent an asset and its peculiarities, so that one can be able to focus on small-scale as well as large-scale entities. For analyzing these assets, a crucial point is to define appropriate visual metaphors. To this end, some factors must be considered, such as the reuse task context being performed and the stakeholder's profile.

An approach for supporting and improving reuse must provide visual means to explore assets in order to allow a better understanding of their structure, behavior, history information, and so on. This can also support the decision regarding the asset reusability. Such approach must support evaluating an asset using a set of visualizations and the data obtained by the Repository Miner.

In this sense, CAVE (Context-Aware Visualization Engine) [Vasconcelos et al. 2013] [Schots 2014] is a mechanism for selecting visualizations according to the contextual information. In the Zooming Browser (described in Section 4.2.7), CAVE can be invoked to depict assets according to the available data and user tasks. The integration with the Repository Miner allows assets of different granularity levels.

CAVE allows to analyze and compare properties of the reusable assets. For instance, in cases where more than one component fits the user needs, this can support the decision making regarding which component to reuse. With respect to suitability of visualizations, a study was performed for correlating types of task and types of visualization [Queiroz et al. 2013], aiming to allow better visualization choices.

The CAVE engine is going to be implemented by a M.Sc. student [Vasconcelos et al. 2013] using the D3.js framework[41] for the visualization features. A context-aware feature model (for mapping context information and rules) is being modeled in the Odyssey environment [Fernandes et al. 2011]. The implemented visualization metaphors will be stored in a separate repository, as presented in Figure 4.1.

4.2.4.1 Visualization Feature Model

There are several visualization strategies and techniques available, and it is not a simple task to choose a visualization that will meet all the expectations and represent everything needed [Vasconcelos et al. 2014a]. As mentioned in Section 3.4, although there are mechanisms that offer flexibility in customizing visualizations, letting the user decide by himself/herself which visualization to use may not be adequate, as he/she may not have experience with the metaphors or may not know which metaphors better fit the structure to be visualized.

To this end, a feature model that encompasses these elements and their associated rules is being constructed [Vasconcelos et al. 2014a], in order to avoid mistakes when mapping visual abstractions to data. By allowing to select only the necessary features, visualizations can be composed with less effort.

A domain analysis has been carried out in the context of this work, in order to identify different characteristics in visualizations, organizing them into the feature model for easing their selection and organization. In APPRAiSER, the feature model serves as input for CAVE.

The adopted methodology for feature analysis is based on two steps: (i) an informal literature review for identifying a set of visualization and interaction elements, and (ii) the *quasi*-systematic review [Schots et al. 2014] that is used for confirming the use of the already classified elements and for complementing the model with new candidates. Although the object of investigation of the latter step was restricted to

[41] http://d3js.org/

visualization approaches that have been proposed to support reuse, such a literature review enabled to gather some initial knowledge from software engineering researches.

By analyzing the visualization elements, the results obtained from the *quasi*-systematic literature review point out some interesting findings. For instance, from 34 approaches selected in 36 publications, 6 visualization and interaction elements were mentioned simultaneously in more than 10 approaches, namely: *Selection*, *Navigation*, *Drill-Down*, *Clustering*, *Highlighting*, and *Labeling*. More details can be found in [Schots et al. 2014]. A total of 53 elements were extracted from the feature analysis process.

Aiming at structuring the feature model, the Odyssey-FEX notation [Blois et al. 2006] is used, due to the researchers' previous knowledge on its syntax and to its wide-scope representation model, comprising characteristics such as category, variability, and optionality.

For better structuring the model, some categories were defined to group similar elements. Although all the elements are identified based on works that present visualizations, some were strictly applied regarding interaction functionalities (*Interaction* category). Another group of visualization techniques was interpreted as an alternative for presenting different visualizations (*Presentation* category). Finally, the third proposed group was related only to changing the exhibition mode (*Information Visualization* category). These categories led to the creation of three corresponding high-level, conceptual features.

After all the elements have been mapped and organized in the feature model, the next phase is the selection of features in order to build a visualization. A wizard (under development by an Undergraduate student) supports the manual selection of such features. In CAVE, this process is intended to be automatized, as described in [Vasconcelos et al. 2014a].

Figure 4.5 shows the most recent version of the feature model. It is being modeled in the Odyssey environment [Fernandes et al. 2011] with the support of a M.Sc. student, and will be exported through the Odyssey-XMI facility, also serving as input for CAVE.

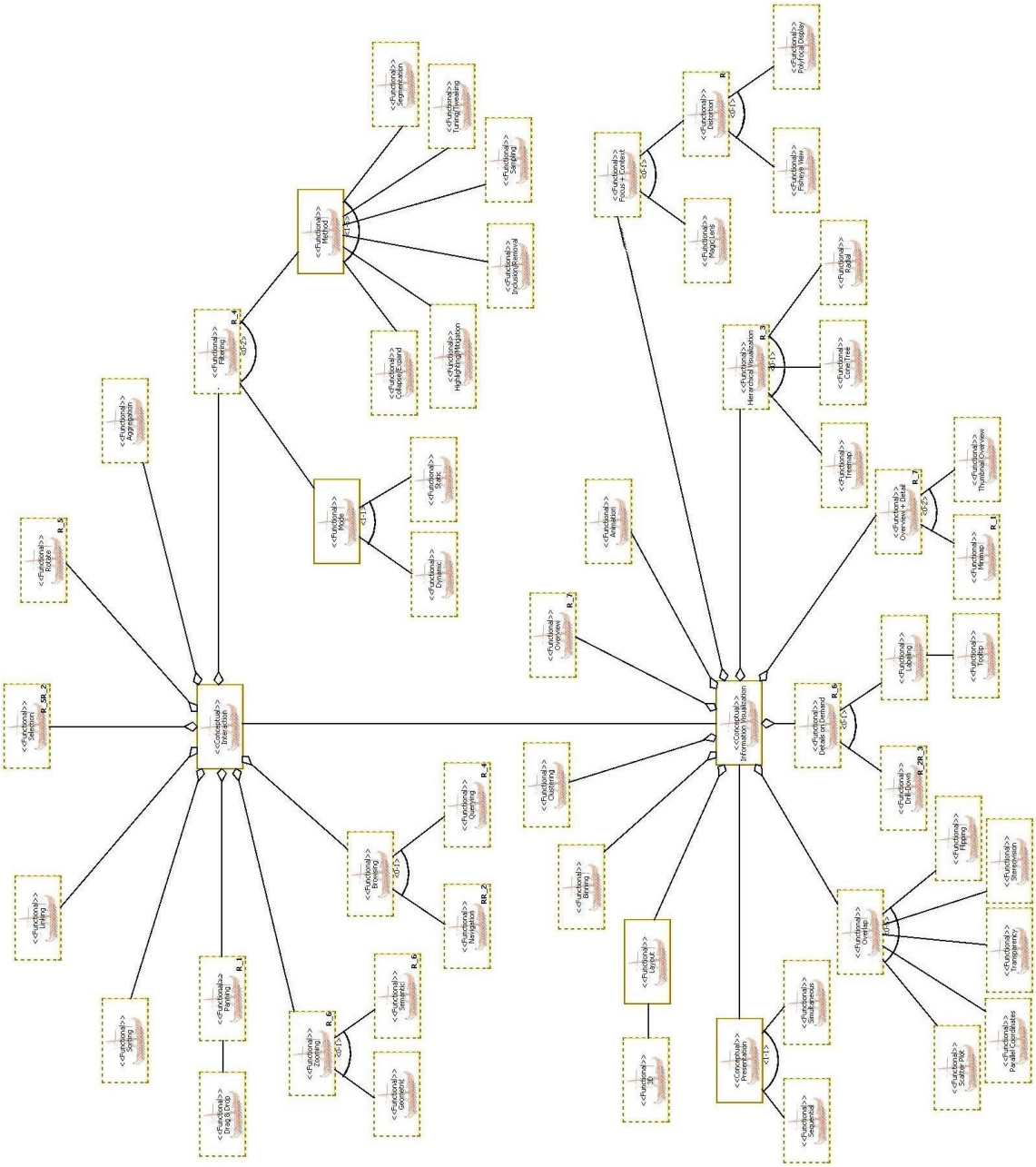

Figure 4.5 – Information Visualization Feature Model [Vasconcelos et al. 2014a]

4.2.5 ReuseDashboard

According to Frakes & Terry (1994), as organizations implement software reuse programs to improve productivity and quality, they must be able to measure their progress and identify the most effective reuse strategies [Frakes & Terry 1994]. This can be achieved by choosing and using the proper software metrics for supporting reuse monitoring. However, a metric alone cannot help to answer all the questions: metrics must be used in combination in order to provide relevant information [Lanza & Marinescu 2006].

To this end, dashboards can be useful by giving a high-level overview of the project status, besides providing peripheral awareness, among other benefits. They are intended to provide information at a glance and to allow easy navigation to more complete information [Treude & Storey 2010].

In this sense, the ReuseDashboard [Palmieri et al. 2013] is an interactive mechanism geared to software reuse programs, which uses metrics and visual analytics for communicating reuse results in a fast and effective way. It aims at stimulating the engagement of stakeholders, providing both high- and low-level relevant information.

Based on the identification and extraction of reuse-related metrics from software projects, along with their established thresholds defined in the evaluation plan (described in Section 4.2.2), the ReuseDashboard presents a multi-device visualization of the extracted measures and analysis results. Concerning the visualization suitability, ReuseDashboard allows choosing different metrics and visualizations according to the stakeholder profile.

For presenting information, ReuseDashboard uses two technologies: Java EE for the server side, and the Ext JS framework[42] for the client side. This framework was chosen because of its HTML 5 features that allow for a rich interaction with visual objects, besides having several types of graphics that are used in the ReuseDashboard visualization [Palmieri et al. 2013].

Figure 4.6 illustrates the visualizations of metrics that inform the flexibility of the code of a given component. Deviations from the evaluation plan are shown at the bottom, pointing items in which problems were detected.

ReuseDashboard offers a web-based visualization of the metrics, which allows it to be accessed from web browsers and mobile devices, besides any IDE with a web browser view, such as Eclipse. The metrics are sent from Eclipse to the dashboard through a websocket[43].

[42] http://www.sencha.com/products/extjs/

[43] http://dev.w3.org/html5/websockets/

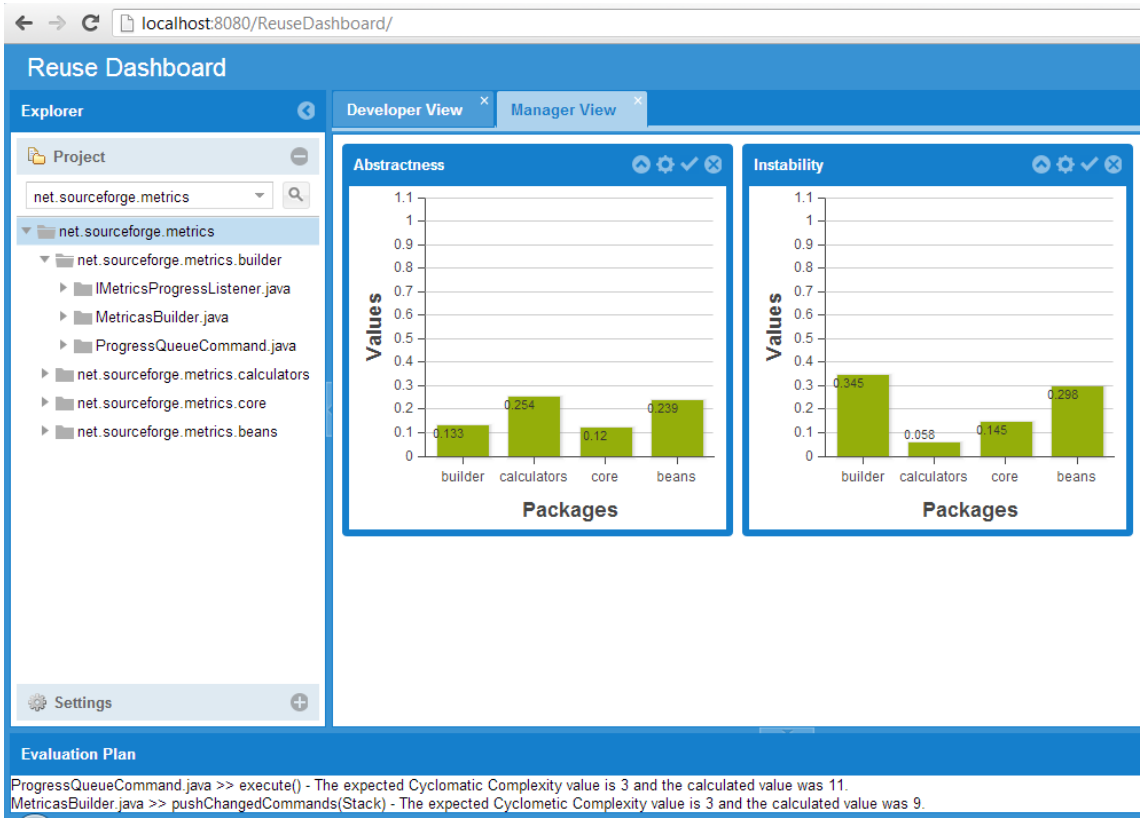

Figure 4.6 – ReuseDashboard screenshot [Palmieri et al. 2013]

4.2.6 Rec4Reuse

In the development for reuse, some properties and characteristics that make an asset to be a candidate to reuse (e.g., [Poulin 1994]) can be evaluated, and some feedback can be provided (e.g., in terms of tips) in order to assist the developer to meet those properties and characteristics. The evaluation of these properties, among others, can denote the assets' degree of reusability, and the evaluation results may indicate the need to make some adjustments in the code in order to make it more reusable. Such adjustments can be made by means of refactoring.

In this sense, Rec4Reuse [Vital & Krause 2013] aims to (i) identify problems in the source code of a software project that may affect its reusability, based on software metrics, and (ii) propose (through recommendations [Robillard et al. 2010]) refactorings that allow increasing its potential to be reused. It provides contextual suggestions for creating and refactoring reusable assets, in order to help developers to be aware of desirable properties of the assets (e.g., low number of external dependencies, flexibility, and generality), supporting development for reuse.

Rec4Reuse evaluates the reusability in terms of software metrics. Its maintenance support is limited to assessing and increasing reusability, providing

awareness of problems to software developers. For visual support on other maintenance tasks (e.g., fixing asset problems reported by users), one must resort to CAVE (presented in Section 4.2.4).

The approach is not intended to perform refactorings automatically. Firstly, it is believed that the judgment of the applicability of the refactoring should be done by the developers. Even among reuse candidates, not everything should be refactored for reuse, because the costs may outweigh the benefits in some cases, i.e., they may affect other software properties or nonfunctional requirements (such as performance). Another reason is that there are several available solutions that automate refactoring, some of which are already integrated into IDEs. These reasons make automatic refactorings for reuse a potential hindrance in software development.

The process performed by the approach consists of the following steps [Vital & Krause 2013]: after selecting a project for analysis, source code files are read by a parser that creates a parse tree to represent them; for each node of the tree, metrics are calculated according to its type (e.g., class, attribute, method, etc.); the metric analysis is inspired in the detection strategies (discussed in Section 4.2.2), and indicates that there may be problems for reuse; when a deviation from an expected metric value is observed, one or more problems are associated with this metric; and finally, each type of problem has one or more associated solutions. The solutions are refactorings that aim to increase reusability. After these steps, developers should decide, for each software project, which refactorings must take place, and which ones must not.

In order to establish the relationship between reuse problems and possible solutions, a study was performed that maps each problem with its detection metrics and refactoring actions that allow for an increase in the software reusability [Vital & Krause 2013]. Problems that prevent the reuse of software (which can be considered as bad-smells) are associated with possible refactoring solutions.

As a proof of concept, an IDE plug-in for Eclipse was developed. The parser framework used is JaxMe JavaSource (JaxMe JS), which creates abstractions of the Java source code structure for manipulating it. An Eclipse view lists all the reuse problems identified by the Rec4Reuse approach, along with their corresponding refactoring recommendations for supporting developers to decide which refactoring may actually be performed. Figure 4.7 shows a screenshot of the tool integrated to the Eclipse IDE. The problems view, shown at the bottom, presents the identified problems, grouped by type. Recommendations are presented through small snippets.

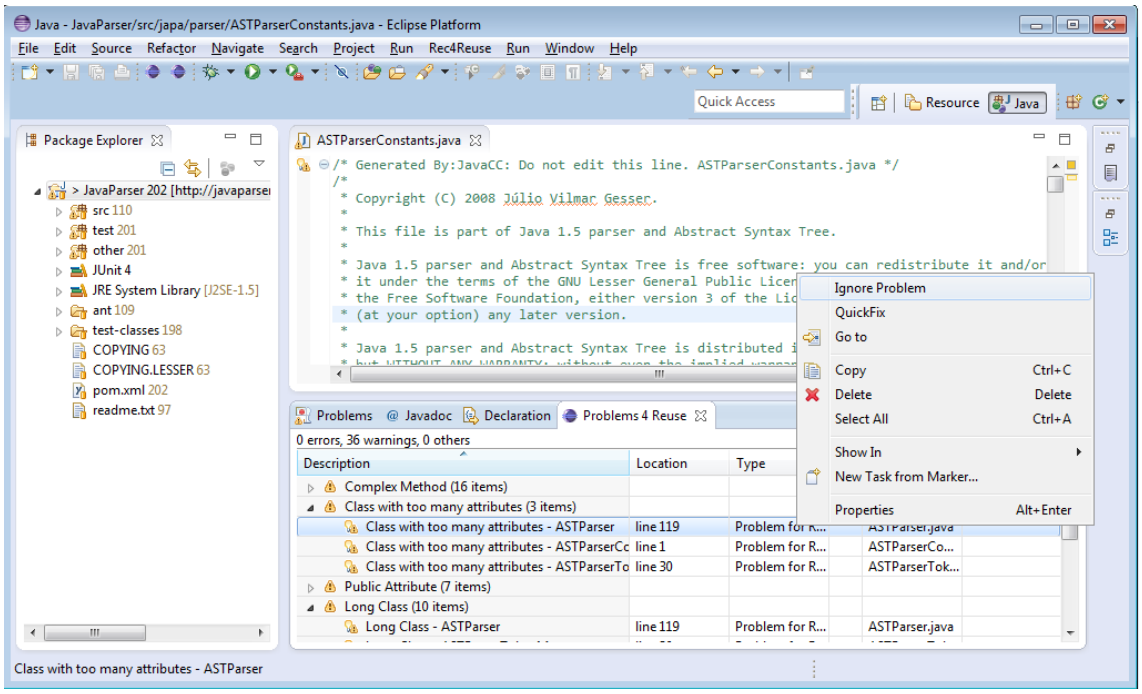

Figure 4.7 – Rec4Reuse screenshot [Vital & Krause 2013]

After configuring the plug-in in the IDE, developers have the option to enable the evaluation of each of their projects individually. Such "recommendation request" is necessary because not all projects need reuse recommendations. After recommendations are activated, they are provided automatically to the developer (push mode [Robillard et al. 2010]). Currently, Rec4Reuse always assumes that the developer is not aware of the refactoring opportunities, but allows him/her to ignore non-applicable recommendations. Ignored recommendations are kept in a separate list so that they can be presented in a report.

4.2.7 Zooming Browser

As mentioned in Section 3.4, none of the analyzed visualization approaches aims to support the dynamics of software reuse in terms of software projects, assets and users and the relationship between these core elements. Information about these elements must be objective, avoiding misinterpretation, so that developers can be aware of the asset with which they will work.

The Zooming Browser [Schots 2014] aims at enabling stakeholders to quickly search, navigate, and browse the contents of the reuse repository and its surrounding elements. It provides basic reuse information along with other information that supports reuse awareness. Aiming to avoid information overloading, it uses semantic zooming for presenting the reusable assets' contextual information in different levels of detail

(zoom levels). All the elements can also be highlighted, filtered, expanded/collapsed and drilled-down to their raw format (source code, document etc.) or to a visual metaphor (as described in Section 4.2.4) in order to be explored with more details.

The high-level mode (activated by zooming out) displays an overview of the main high-level elements of the approach: users, VCS repositories (which contain software projects), and the reuse repository (which contains reusable assets). Users and repositories are linked according to reuse data. The low-level mode (activated by zooming into a particular location) presents assets, users, and projects along with their connections (surroundings, as depicted in Figure 4.8) and supplementary information, all displayed on demand.

The Zooming Browser allows analyzing assets by applying specific visual filters that can help to obtain repository-related information (e.g., represent size by "asset age", filter by number of access, use different colors for representing the asset stability etc.). This is particularly helpful for repository maintenance. In addition, the Zooming Browser can indicate relevant assets (which can also be obtained by retrieval mechanisms), supporting development with reuse, based on the user profile, previous searches and previous reuses. Assets can be recommended in terms of domain similarity and other relevant properties. The Zooming Browser also allows uploading candidate assets to be posteriorly evaluated and included in the reuse repository.

Multiple views are suitable and relevant for both the kind of task being performed and the stakeholder's role in the reuse scenario. The Zooming Browser makes use of multi-perspective views [Wu & Storey 2000], in which the stakeholder uses a main view for general comprehension tasks and, in certain contexts or activities, can make use of auxiliary views for additional exploration tasks. For a consistent interaction pattern, changes in one view are propagated to the other associated views.

The supplementary information presented in the Zooming Browser is categorized into four *perspectives* that change according to the active *viewpoint*, as described in the following subsections. The Zooming Browser visualizations are intended to be implemented with the D3.js framework, while user forms and other integrations will be made with Java EE.

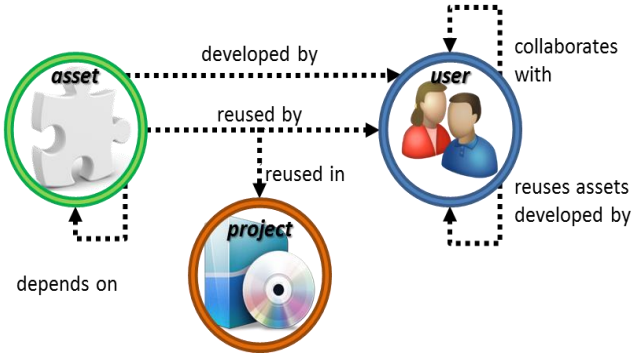

Figure 4.8 – Core elements and their surroundings

4.2.7.1 Overview

The *overview* perspective presents general information of the core elements. The *asset-centric* viewpoint shows its status (available, discontinued etc.), metadata (size, type, notation/programming language etc.), and users' ratings. *User-centric* information comprises the status (active, inactive etc.), role(s) (reuse manager, developer etc.) and ratings (based on developed assets' ratings). The *project-centric* viewpoint shows the project status (in progress, finished, aborted etc.) and the percentage of finished tasks.

4.2.7.2 Statistics

Increasing the visibility of results achieved through reuse not only stimulates developers to integrate reuse in their daily development activities, but also encourages managers in endorsing it. The evaluation of how well organization's goals are being accomplished relies on the selection of an appropriate set of metrics, along with effective measurement practices for collecting such metrics [Benedicenti et al. 1996] [Lanza & Marinescu 2006]. For instance, statistics about reuse, numbers of hits and percentage of code reused can help analyzing an asset's reuse history [Alonso & Frakes 2000], supporting the management of the reuse repository and its assets.

In the *statistics* perspective of the Zooming Browser, the *asset-centric* viewpoint shows the number of views (based on selection), the number of reuse intentions (based on downloads), and the number of effective reuses (based on the projects' VCS that reused the asset). The *user-centric* viewpoint shows this information (user-oriented), in addition to the number of assets developed by him/her and the number of users who collaborated with him/her in assets' development. The *project-centric* viewpoint presents the number of reused assets (based on VCS history), the number of effectively reused assets (based on their releases), and the number of users who added assets to the

project. Stakeholders navigating through the Zooming Browser will have a “minimalist view” of the ReuseDashboard, whose information is automatically updated from based on user interactions.

4.2.7.3 History

In the *history* perspective of the Zooming Browser, the *asset-centric* viewpoint presents the asset’s development history (based on its VCS repository, if any) and its release history (semantic versioning) based on public release repositories (e.g., Maven), and VCS tags, depending on their semantics. The *user-centric* viewpoint shows his/her reuse timeline history (based on actions performed in the reuse repository and/or the VCS repositories, if any). The *project-centric* viewpoint presents its development history and release history.

VCS information can be drilled-down to GraphVCS, which presents a visualization of the project’s history, e.g., highlighting project versions in which an asset is present, for understanding the reuse context. Project information is obtained both from the organization’s portfolio and open source projects. The reasons are twofold. In the beginning of a reuse program, it is unlikely to find enough information about an asset (that encourages its reuse) on local configuration management systems. On the other hand, experiences from the organization itself (even if a reuse program has not yet been established) are essential for promoting reuse.

Although these sources of information complement each other, they are handled separately since they allow for different kinds of insight. The fact that an asset is widely reused in open source projects may give more confidence in reusing it, while an asset reused in the organization’s projects (successfully or not) gives a clue of the chances of success/failure based on such previous experiences.

4.2.7.4 Issues and Opportunities for Improvement

The *issues* perspective provides information primarily extracted from issue tracking systems that complements the *history* perspective, by displaying issues that affect reusable assets and their surroundings. In the *asset-centric* viewpoint, reported issues related to an asset (e.g., problems and feature requests) are shown. The *user-centric* viewpoint shows issues reported by the user and issues in which he/she is involved. The *project-centric* viewpoint depicts issues related to the project and its reusable assets.

For implementation assets, this perspective also indicates code metrics whose values may negatively impact the asset's reusability (based on customized thresholds or calculated values). Developers can drill down to the source code (if available) for using Rec4Reuse. In the context of the APPRAiSER environment, it will support development for reuse and reengineering for reuse.

4.3 Other Architectural and Implementation Aspects

Figure 4.9 summarizes the information of the APPRAiSER tools in terms of a conceptual model of their main elements and the main relationships between these elements. Some of these relationships (such as the ones presented in Figure 4.8) were omitted for a matter of simplicity.

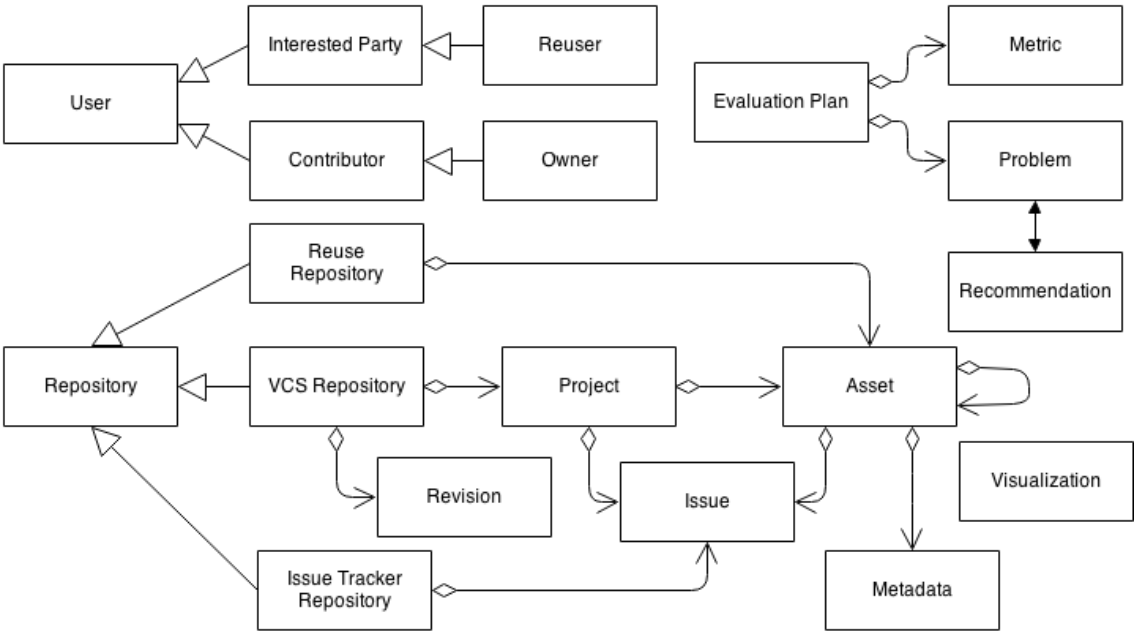

Figure 4.9 – APPRAiSER Conceptual Model

There are several demands that must be taken into account for supporting and improving software reuse with visualizations. It is worth highlighting that some requirements can be fulfilled in many different ways, which brings up the need for a flexible architecture that allows interoperability and seamless integration of its modules. As discussed in Section 3.4, a modular architecture allows new functionalities to be added and, consequently, more support for software engineering and reuse tasks.

As mentioned previously, a primary concern on the APPRAiSER architecture is flexibility. All the conceptual elements and their connections are going to be concretized in terms of well-defined interfaces, in order to allow their easy interchange without impacting the implementation. For instance, one can choose to use a different

reuse repository. By implementing a reuse repository connector conforming to the interfaces required by APPRAiSER, communication occurs seamlessly. Analogously, if one needs to include new metrics or add new visualizations, it is just a matter of implementing the corresponding interfaces – for instance, methods that allow metrics to be recalculated (or visualizations to update themselves) when required.

Since development for/with reuse may already represent a change in the daily routine of those developers, any additional external tool involves adopting new technologies, whose issues may hinder the establishment of a reuse program. The integration of a tool suite in an IDE also facilitates the adoption of the tool among developers [Mariani & Pastore 2011]. In this sense, developing plug-ins can benefit from the IDE's available features and allow integration with other tools, since the interaction among the environment resources usually occurs through a plug-in architecture.

The Eclipse IDE was chosen for integration with APPRAiSER, because it is designed as a modular platform, with the ability to install, use, and develop plug-ins to extend the core functionality. Besides its widely used plug-in architecture, it also has a robust and extensible event notification system. Experiences with previous works in developing Eclipse plug-ins [Oliveira 2011] [Werner et al. 2011] are also an influencing factor.

Eclipse extension points and extensions are used for achieving the flexibility to add connector plug-ins. Extension points defines a contact on how other plug-ins can contribute, so that a plug-in that implements a given functionality may contribute to (or receive contributions from) other plug-ins. In other words, plug-ins that define extension points open themselves up for other plug-ins. The Eclipse IDE also allows to dynamically link plug-ins together at runtime.

4.4 Final Remarks

The APPRAiSER environment is being built based on the issues and limitations identified from both state-of-the-art and state-of-the-practice. It comprises a set of tools that aim at visually supporting reuse stakeholders (especially reuse managers and developers) in performing reuse tasks.

As mentioned previously, APPRAiSER aims at promoting software reuse in a progressive way, so that cultural barriers can be gradually overcome and all stakeholders can become committed with the reuse initiatives by perceiving the benefits

brought by them, without causing cognitive overload. In this sense, Table 4.1 presents a suggestion on how APPRAiSER and its tools could be used in different stages of reuse initiatives, i.e., the expectations with its use.

Table 4.1 – The use of APPRAiSER in different stages of reuse initiatives

	Initial stages	**Intermediate stages**	**Advanced stages**
ReuseDashboard	Support awareness and communication of first reuse achievements, communicating results in a fast and effective way.	Keep stakeholders' motivation and help institutionalizing a reuse program.	Provide data for supporting decision-making.
GraphVCS	Support finding occurrences of candidate reusable assets, helping to populate the reuse repository and stimulating further reuse.	Support deeper analyses of the assets' history, when the organization is already used to the reuse practices.	Support identifying kinds of projects in which assets are usually reused, for identifying domains with more reuse potential.
Zooming Browser	Provide information about reusable assets in a centralized way, easing and stimulating their reuse.	Support exploring the reuse repository, avoiding user disorientation when there are a reasonable number of reusable assets.	Understand and explore relationships between users, assets, and projects, while avoiding information overloading (when the reuse repository contains many reusable assets).
Rec4Reuse	Help evaluating the reusability of assets.	Help developers in refactoring candidate reusable assets and understanding how to increase assets' reusability.	Provide useful recommendations in development for reuse, when development with reuse is well established in the organization.
CAVE	Help understanding underlying details (e.g., structure) of assets, in order to [better] reuse them.	Help choosing between reusable assets, when there is more than one asset that fits a given need, and potentially detecting refactoring opportunities.	Help understanding bottlenecks in the reuse of specific assets.
The APPRAiSER environment as a whole	Help engaging stakeholders in the establishment of a reuse program.	Help identifying key consumers and domains in which there are more reuse opportunities, towards consolidating a reuse program.	Provide information for supporting continuous improvement of reuse activities and processes.

All the APPRAiSER tools were or are being developed under the supervision of the author of this work, who also collaborated to the development of some of them (e.g., [Werner et al. 2011]). In order to integrate the tools to APPRAiSER, some implementation changes need to be performed. Table 4.2 presents the current stage of

each tool and the necessary modifications (to be performed by the author of this work) for the integration to the environment.

Table 4.2 – APPRAiSER tools: current status and necessary modifications

Tool	Current Status	Modifications
ReuseDashboard	Partially implemented, but the current implementation has some device issues[44].	Fix the device issues, maybe changing the visualization framework in use[45].
GraphVCS	Fully implemented.	Since its original goal is to support comprehension of VCS repositories, it will be extended for displaying relevant evolution information regarding the reusable assets.
Zooming Browser	Not implemented yet. It is being implemented by the author of this work.	It will be integrated "as is".
Rec4Reuse	Fully implemented.	It will be integrated "as is".
CAVE	Not implemented yet. It is being implemented by a M.Sc. student. The feature model, in turn, is already under development by the author of this work along with the M.Sc. student.	It will be integrated "as is".
Repository Miner	Fully implemented.	The submodules must be integrated in order to provide the required information, using REST and JSON formats.
Metrics Extractor	The evaluation plan is fully implemented (GUI for the IDE), but the mechanism for extracting metrics has some implementation problems.	Implement a web GUI for the evaluation plan and fix the mechanism for extracting metrics. Depending on the progress of the research, additional metrics must be implemented.
All the tools	--	Adapt their architecture for making them depend on the Repository Miner and Metrics Extractor mechanisms.

Although there are other application settings to which this approach might be useful, these examples can give an insight of the benefits that APPRAiSER awareness resources can bring in the reuse context.

The next chapter presents the current stage of the research, as well as the planning of the next steps.

[44] Although the presentation of the visualization is compatible with mobile and desktop platforms, there is limited interactivity on mobile devices, due to the framework currently used.

[45] The visualization can incorporate the features provided by the Sencha Touch framework (http://www.sencha.com/products/touch), which allows porting web applications written in Ext JS to mobile platforms. Another possibility is to replace all the visualization by another framework, e.g., D3.js.

CHAPTER 5 – CONCLUSION

This chapter presents the current contributions and some expected results from this work, as well as its current stage and the planned next steps for its conclusion.

5.1 Epilogue

Software reuse provides several benefits throughout the software development process, such as the decrease of implementation efforts, the reduction on time-to-market, and the amortization of test and inspection costs, favoring an increase of quality. Nevertheless, organizations still find difficulties in implementing reuse due to several reasons, including technical and non-technical aspects. It can be noticed, though, that many of these difficulties, if not most of them, are recurring throughout the years.

Findings from literature reports and a study (described in Chapter 2) showed that a frequent issue is the lack of understanding, not only in terms of the potential benefits of software reuse – which may lead to a lack of incentives from top management –, but also in performing reuse tasks. The lack of a proper tool support for executing such tasks is also a usual complaint.

In order to achieve the acceptance/consciousness and successful adoption of software reuse, it is important to take into account how to better provide appropriate reuse awareness. Awareness mechanisms can provide stakeholders with the necessary information and support for performing their reuse-related tasks. One of the ways to increase such awareness is by employing visualization resources and techniques.

In this sense, a comprehensive study (described in Chapter 3) was conducted for identifying software visualization approaches targeted to reuse-related tasks. Results pointed out that no work so far addresses a number of reuse tasks in an integrated way, and the existing ones that address particular tasks are limited in terms of collecting information from different data sources and supporting the customization and selection of visualizations. Besides, most of them do not provide evidence on their effectiveness.

To this end, this thesis proposal (presented in Chapter 4) aims to define an interactive visualization approach that assist stakeholders (mainly reuse managers and developers) in executing software reuse tasks, such as exploring a reuse repository,

obtaining and understanding information regarding the three main identified elements (reusable assets, users, and projects), and monitoring reuse initiatives. The realization of the proposed approach is being achieved through the implementation of the APPRAiSER environment and its integrated tools, which provide stakeholders with different visualization mechanisms suited for their day-to-day tasks. Its architecture contemplates elements for gathering, processing and visually presenting information that is relevant for software reuse.

5.2 Expected Results and Contributions

The main expected result from the thesis is to assist software development stakeholders in carrying out software reuse tasks, providing them with the necessary awareness through visualizations, thereby helping them to achieve the expected benefits from reuse. In this regard, the current contributions of this research are:

- *A primary study on issues related to software reuse in some Brazilian organizations* (Section 2.5.2): The results from the semi-structured interviews conducted with Brazilian implementers and assessors of MR-MPS-SW allowed the definition of some reuse tasks that need more support. This can be used not only for other research initiatives, but also for the development of additional tool support for the implementation of reuse processes in software organizations.
- *A secondary study on visualization approaches geared to software reuse* (Section 3.3.3): The results from the *quasi*-systematic review can be used as a starting point for future research directions to be addressed by the software engineering community when choosing, instantiating, or developing visualization-based approaches for supporting software reuse. Besides, the presented information can be used as a body of knowledge to support decision making regarding the choice of visualization approaches for software reuse.

Other contributions from the thesis that are expected to be accomplished are:

- The implementation of Zooming Browser and the APPRAiSER architecture.
- The extension and integration of existing tools to the APPRAiSER environment.
- A survey for mapping which data can be relevant to which stakeholders (described in Section 5.4.1).
- Evidence on the use of APPRAiSER, through experimental evaluations (described in Section 5.4.2).

Although not considered as a contribution, the framework extension for categorizing visualization approaches (presented in Section 3.3.2), with the two new dimensions and the definition of research questions for all dimensions, can guide researchers in terms of (i) aspects that must be taken into account for novel approaches, and (ii) information that should ideally be described in publications. Such framework may also support the conduction of other secondary studies on software visualization applied to another field of interest (e.g., software maintenance), providing useful information regarding visualization approaches.

Regarding the awareness and comprehension challenges listed in Section 3.2.3, the listed contributions aim to address the following ones in the context of this work:

- The semi-structured interviews and the survey, as well as potential experiments on industry, represent a step towards *understanding the real needs of the software development industry stakeholders in terms of awareness and comprehension*, and *bridging the gap and encouraging interaction between academia and industry*.
- In terms of *identifying and developing suitable mechanisms and adequate abstractions* and *building specialized, personalized visualizations according to the comprehension needs*, APPRAiSER and its integrated tools are expected to provide adequate abstractions that are useful in their purposes (i.e., supporting reuse tasks), allowing users to customize them according to their needs.
- The APPRAiSER Repository Miner and Metrics Extractor mechanisms, along with the survey, allow *evaluating the quality of existing data sources and identifying relevant data*. Such mechanisms also address the *use of software tools to seamlessly collect rich data sets on software comprehension activities*, in addition to an activity logging mechanism that is expected to be built or reused for supporting the evaluation of APPRAiSER.

5.3 Current Stage

The research methodology of this work, presented in Section 1.5, comprises five steps. The *informal literature review* (Section 3.3.1) and the *quasi-systematic review* (Section 3.3.3) – planned in the first step (*Collect preliminary information*) and part of the second step (*Characterize the state-of-the-art*), respectively – have been performed. Nevertheless, as mentioned in Section 1.5, the literature must be reviewed continuously for providing subsides for enriching the definition of the topic and the proposed

solution. Thus, the first step is executed iteratively, and the second step is going to be re-executed afterwards.

Still in the second step of the research (*Characterize the state-of-the-practice*), two primary studies were planned: the *semi-structured interviews* (which were performed and presented in Section 2.5.2), and (ii) *surveys* with researchers and practitioners (which are going to be performed, as described in Section 5.4.1).

The third step of the research (*Propose and develop the approach*) has been partially achieved. As described in Section 4.4, some of the APPRAiSER tools are already implemented, but need to be adapted (e.g., ReuseDashboard, Repository Miner and Metrics Extractor) or extended (e.g., GraphVCS) in order to be integrated to the environment. Other tools are still going to be developed (e.g., Zooming Browser and CAVE – the latter by a M.Sc. student).

Details on the fourth step (*Evaluate the proposed approach*) are described in Section 5.4.2. Based on the evaluation results and feedback, the fifth step (*Improve the proposed approach*) will take place, by making the necessary *adjustments* and the identified improvements, if applicable.

5.3.1 Research Achievements

The conduction of this research allowed the following research achievements:

- Publications and research projects about the APPRAiSER tools:
  - The *VCS Miner* uses an infrastructure previously developed [Werner et al. 2011] in collaboration with the author of this work.
  - The *Reuse Repository Miner* was inspired in MPS-Reuse [Chaves 2013], an Undergraduate Final Project at UERJ) advised by the author of this work.
  - The *Issue Tracker Miner* was produced in the context of an Undergraduate Research at COPPE/UFRJ [Queiroz et al. 2012].
  - *GraphVCS* was presented in [Pereira & Schots 2011] and developed in the context of an Undergraduate Final Project at UERJ advised by the author of this work.
  - *Rec4Reuse* was also built in the context of an Undergraduate Final Project [Vital & Krause 2013] at UERJ advised by the author of this work.
  - *CAVE* is part of an ongoing M.Sc. thesis at COPPE/UFRJ being developed in the context of this work, and some preliminary results focusing on the use of a

context model for supporting context-aware visualizations are described in [Vasconcelos et al. 2013].

  - The first prototype of ReuseDashboard is described in [Palmieri et al. 2013], in the context of a M.Sc. thesis proposal.

- Research on software visualization:
  - The research on software visualization allowed the identification of awareness and comprehension challenges in a special track of the Brazilian Symposium on Software Engineering (SBES) [Schots et al. 2012].
  - Additionally, an introductory tutorial in software visualization was presented at the Brazilian Conference on Software (CBSoft) [Schots & Werner 2012].
- Performed studies:
  - The preliminary results from the semi-structured interviews are described in [Schots & Werner 2013]. A technical report will soon be prepared.
  - A preliminary study on mapping visualizations according to the focus of analysis was developed in the context of an Undergraduate Research [Queiroz et al. 2013].
  - The full protocol and the results from the *quasi*-systematic review are described in a technical report [Schots et al. 2014].
  - Details on the construction of the feature model for information visualization that supports CAVE are described in [Vasconcelos et al. 2014a], and a technical report is being prepared.
- Research proposal:
  - The research proposal [Schots 2014] will be presented at the Doctoral Symposium of the 36th International Conference on Software Engineering.

5.4 Next Steps and Schedule

Besides the tool development and extensions activities (as described in Table 4.2), the following subsections describe the next steps for concluding this research.

5.4.1 Survey with software reuse researchers and practitioners

Some aspects must be taken into account in order to provide relevant information to stakeholders, such as: (i) which software metrics are indicated to

summarize the reuse characteristics of a reusable asset (depending on the type of asset), and (ii) which metadata a stakeholder needs to analyze for considering reusing an asset.

In order to identify which reuse metrics and metadata information are relevant, an initial set of metrics and metadata related to reusable assets (mostly reusable software components) was collected in an ad-hoc way from the literature and from previous experiences in implementing reuse processes.

From this initial set, a survey (part of the second step of the research methodology) is intended to be conducted with worldwide software reuse researchers and industry practitioners in software organizations that implemented reuse processes, aiming to evaluate these and other metrics and metadata as regards to their relevance in facilitating the identification, documentation and evaluation of reusable assets in general.

In addition to evaluating the relevance of metrics and metadata, the survey also aims at obtaining feedback regarding which kinds of metadata can be important for which stakeholders, based on a mapping that is in progress in conjunction with a M.Sc. student at COPPE/UFRJ [Vasconcelos et al. 2013]. This can be useful for recommending assets in terms of domain similarity and other relevant properties (as mentioned in Section 4.2.7). Furthermore, the survey will help validating some findings from the *quasi*-systematic review and interviews.

5.4.2 Approach Evaluation

The evaluation of the approach (fourth step of the research methodology) involves the planning and execution of experimental studies. The evaluation will assess quantitatively and qualitatively whether the perceptive and cognitive abilities of the stakeholders in carrying out software reuse tasks are properly stimulated, increasing efficiency and efficacy, while decreasing the effort and time spent on such tasks.

Regarding the evaluation scenario, an ideal setting would be the use of APPRAiSER tools in software development organizations that are implementing reuse processes and/or aim to institutionalize a reuse program. However, this depends on the availability and interest of organizations at the time of the evaluation. Another shortcoming is that a proper evaluation of the use of APPRAiSER would require much time for verifying if the expected benefits were effectively achieved, due to their long-term nature.

Although it is believed that an evaluation over time seems to be unfeasible in the context of a Ph.D. thesis, it is expected to investigate the use of APPRAiSER by stakeholders who are used to reuse practices, so that they can qualitatively evaluate the feasibility and adequacy of the environment and its supporting tools, based on the following hypothesis: *the execution of reuse tasks with the use of visualizations can be performed more efficiently and with more efficacy than the execution of such tasks without visual support*.

In principle, participants will be divided into two groups. Each group will carry out the same tasks, but one will be supported by APPRAiSER visualizations, while the other will use other usual tools without visualization support – or no tool at all, if the focus of evaluation is based solely on the contribution of the visualization (in a similar format to [Oliveira 2011]). The time spent, the efficiency and the efficacy of each participant/group will be analyzed quantitatively. Other nonfunctional features may also be subject to evaluation, as well as qualitative aspects of the tools (based on participants' opinion, as described in some studies in the literature [Schots et al. 2014]).

The focus of evaluation of each tool is based on the tasks they aim to support (as depicted in Figure 4.2). A scenario that involves the execution of such tasks will be set up to this end. The evaluation of each APPRAiSER tool separately may be based on the following focuses:

- GraphVCS and its support for analyzing a repository evolution can be evaluated from two perspectives: (i) the projects' history, allowing the identification of successful and unsuccessful reuse attempts, and (ii) the reusable asset's history, allowing the analysis of its stability and frequency of updates. The evaluation will focus on the ease of use, time spent for answering evolution questions related to reuse, and correctness of answers.
- The visualization feature model will follow an evaluation methodology based on two steps: (i) verifying the feature model syntax and semantics by using a feature model checklist [Mello et al. 2014], and (ii) carrying out a peer review method. In the latter step, aiming at checking both the visualization aspects and the feature model notation, experts from both areas will participate in the review. These experts are already selected and the review is being prepared. After the CAVE tool itself is implemented, a qualitative evaluation will focus on its customizability of visualizations and data, as well as its support in understanding reusable assets, while

a quantitative evaluation may be based on answering questions related to detailed information of reusable assets, as well as the correctness of such answers.

- Concerning the ReuseDashboard, an experiment for evaluating the effectiveness in monitoring a reuse program can be set in terms of the creation of an evaluation plan and, from it, the analysis and monitoring of reuse metrics and reusable assets, allowing the identification of problems in the implementation of a reuse program. To this end, three fictitious scenarios over time will be set up and presented to the participants. The evaluation will take into account items such as the ease in interpreting the visual information, the time spent in being aware of a change and in analyzing such change, and the decisions taken (which will be compared both comparatively and based on a template of expected decisions).
- Regarding Rec4Reuse, an evaluation is intended to be performed using the source code of open source components that were designed to be reused, aiming to identify potential reuse issues and indicate which refactorings may be applied in order to increase its reusability. Participants of the study will be asked if the recommendations given by the approach are easy to understand and relevant (i.e., if they actually help to observe reuse-related properties that otherwise would remain unnoticed). Moreover, a report generated by Rec4Reuse will be used to evaluate which suggestions were mostly ignored and which of them were accomplished.
- Finally, Zooming Browser will be evaluated in terms of the time spent in answering questions related to reuse management and reusable assets, as well as the correctness of such answers. Some aforementioned qualitative aspects are also intended to be evaluated. The questions will be made in terms of the tasks supported by Zooming Browser (i.e., identify experts on a reusable asset, explore the reuse repository and obtaining general information about a reusable asset).

For all the approaches, an *observational study* is intended for capturing firsthand behaviors and interactions that might not be noticed otherwise [Seaman 1999]. *Interaction log analysis* techniques are also intended for enriching the data [Seaman 1999]. To this end, an activity logging mechanism can be developed or reused, as suggested in [Kagdi & Maletic 2008].

5.4.3 Expected Targets for Publications

During the conduction of this research, its results are expected to be published in correlated conferences, such as: International Conference on Software Engineering

(ICSE), International Conference on Software Reuse (ICSR), International Conference on Program Comprehension (ICPC), International Conference on Software Maintenance (ICSM), Working Conference on Software Visualization (VISSOFT), Mining Software Repositories (MSR), European Conference on Software Maintenance and Reengineering (CSMR) and Brazilian Symposium on Software Engineering (SBES), among other possibilities.

Correlated journals will also be targeted for publications, such as the Journal on Software Maintenance and Evolution (JSME), IEEE Software, IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology (TOSEM), among others.

5.4.4 Schedule

Table 5.1 presents the planning of the next activities for concluding this work.

Table 5.1 – Schedule of the next steps

Activities	2014									2015								
	04	05	06	07	08	09	10	11	12	01	02	03	04	05	06	07	08	09
Informal literature review																		
Execution of a new round of the *quasi*-systematic review																		
Integrate Repository Miner submodules																		
Fix and evolve the Metrics Extractor																		
Implement the Zooming Browser																		
Extend GraphVCS for displaying reuse information																		
Fix ReuseDashboard device issues																		
Integrate APPRAiSER tools																		
Planning of evaluations (including the survey)																		
Execution of evaluations and analysis of results																		
Implementation of potential improvements																		
Writing of publications with research results																		
Thesis writing																		
Ph.D. defense																		

REFERENCES

[Ali 2009] Ali, J. (2009). "Cognitive support through visualization and focus specification for understanding large class libraries". *Journal of Visual Languages and Computing*, v. 20, n. 1, pp. 50-59.

[Alonso & Frakes 2000] Alonso, O., Frakes, W. B. (2000). "Visualization of Reusable Software Assets". In: *6th International Conference on Software Reuse (ICSR 2000)*, Vienna, Austria, pp. 251-265, June.

[Anslow et al. 2004] Anslow, C., Marshall, S., Noble, J., Biddle, R. (2004). "Software visualization tools for component reuse". In: *2nd Workshop on Method Engineering for Object-Oriented and Component-Based Development, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2004)*, Vancouver, Canada, pp. 1-11, October .

[Baldauf et al. 2007] Baldauf, M., Dustdar, S., Rosenberg, F. (2007). "A survey on context-aware systems". *International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC)*, v. 2, n. 4, pp. 263-277.

[Basili et al. 1994] Basili, V., Caldiera, G., Rombach, H. (1994). "Goal Question Metric Paradigm", *Encyclopedia of Software Engineering*, v. 1, edited by John J. Marciniak, John Wiley & Sons, pp. 528-532.

[Bavoil et al. 2005] Bavoil, L., Callahan, S. P., Crossno, P. J., Freire, J., Scheidegger, C. E., Silva, C. T., Vo, H. T. (2005). "Vistrails: Enabling interactive multiple-view visualizations". In: *IEEE Visualization* (*IEEE VIS 2005)*, Minneapolis, USA, pp. 135-142, October.

[Benedicenti et al. 1996] Benedicenti, L., Succi, G., Valerio, A., Vernazza, T. (1996). "Monitoring the efficiency of a reuse program". *SIGAPP Applied Computing Review*, v. 4, n. 2, pp. 8-14, September.

[Biddle et al. 1999] Biddle, R., Marshall, S., Miller-Williams, J., Tempero, E. (1999). "Reuse of debuggers for visualization of reuse". In: *Proceedings of the 5th Symposium on Software Reusability (SSR 1999)*, Los Angeles, USA, pp. 92-100, May.

[Blois et al. 2006] Blois, A. P. T. B., Oliveira, R. F., Maia, N., Werner, C., Becker, K. (2006). "Variability modeling in a component-based domain engineering process". *Reuse of Off-the-Shelf Components*, pp. 395-398. Springer Berlin Heidelberg.

[Braga et al. 2006] Braga, R. M. M., Werner, C. M. L., Mattoso, M. (2006). "Odyssey-Search: A Multi-Agent System for Component Information Search and Retrieval". *Journal of Systems and Software*, v. 79, n. 2, pp. 204-215, February.

[Brazilian Computer Society 2006] Brazilian Computer Society (2006). "Grand Challenges of Computing Research in Brazil – 2006-2016". Available at http://www.sbc.org.br/.

[Brooks Jr. 1987] Brooks, Frederick P. (1987). "No Silver Bullet: Essence and Accidents of Software Engineering". *Computer*, v. 20, n. 4, pp. 10-19, April.

[Caldiera & Basili 1991] Caldiera, G., Basili, V. R. (1991). "Identifying and qualifying reusable software components". *Computer*, v. 24, n. 2, pp. 61-70, February.

[Card & Comer 1994] Card, D., Comer, E. (1994). "Why do so many reuse programs fail?" *IEEE Software*, v. 11, n. 5, pp. 114-115, September.

[Chaves 2013] Chaves, V. B. C. (2013. "MPS-Reuse: A tool for supporting the execution of reuse management tasks and reusable assets management" [MPS-Reuse: Uma ferramenta de apoio à execução de tarefas de gerência de reutilização e à gestão de ativos reutilizáveis] (in Portuguese). Undergraduate Final Project, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, Brazil.

[Chen 2006] Chen, C. (2006). *Information Visualization: Beyond the Horizon*. 2nd ed. Springer.

[Clements & Northrop 2002] Clements, P., Northrop, L. (2002). *Software product lines*. Addison-Wesley, Boston.

[CMMI Product Team 2010] CMMI Product Team (2010). "CMMI for Development, Version 1.3", Technical Report CMU/SEI-2010-TR-033, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA. Available at http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm.

[Constantopoulos et al. 1995] Constantopoulos, P., Jarke, M., Mylopoulos, J., Vassiliou, Y. (1995). "The software information base: A server for reuse". *The VLDB Journal*, v. 4, n. 1, pp. 1-43.

[Cooper et al. 2009] Cooper, J. R., Lee, S.-W., Gandhi, R. A., Gotel, O. (2009). "Requirements Engineering Visualization: A Survey on the State-of-the-Art". In: *4th International Workshop on Requirements Engineering Visualization (REV 2009)*, Atlanta, USA, pp. 46-55.

[Couto et al. 2013] Couto, C., Maffort, C., Garcia, R. Valente, M. T. (2013). "COMETS: A Dataset for Empirical Research on Software Evolution Using Source Code Metrics and Time Series Analysis". *SIGSOFT Software Engineering Notes*, v. 38, n. 1, pp. 1-3, January.

[Diehl 2007] Diehl, S. (2007). *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*, 1st ed., Springer-Verlag Heidelberg New York.

[Dourish & Bellotti 1992] Dourish, P., Bellotti, V. (1992). "Awareness and Coordination in Shared Workspaces". In: *1992 ACM Conference on Computer-Supported Cooperative Work (CSCW 1992)*, Toronto, Canada, pp. 107-114, November.

[Duru et al. 2013] Duru, H. A., Çakır, M. P., İşler, V. (2013). "How Does Software Visualization Contribute to Software Comprehension? A Grounded Theory Approach". *International Journal of Human-Computer Interaction*, v. 29, n. 11, pp. 743-763, November.

[Duszynski et al. 2011] Duszynski, S., Knodel, J. Becker, M (2011). "Analyzing the source code of multiple software variants for reuse potential". In: *Proceedings of the 18th Working Conference on Reverse Engineering (WCRE 2011)*, Limerick, Ireland, pp. 303-307, October.

[Fernandes et al. 2011] Fernandes, P., Werner, C., Teixeira, E. (2011). "An Approach for Feature Modeling of Context-Aware Software Product Line". *Journal of Universal Computer Science*, v. 17, n. 5, pp. 807-829, March.

[Few 2009] Few, S. (2009). *Now You See it: Simple Visualization Techniques for Quantitative Analysis*, 1st ed., Analytics Press, 327p.

[Fowler & Highsmith 2001] Fowler, M., Highsmith, J. (2001). "The agile manifesto". *Software Development*, v. 9, n. 8, pp. 28-35.

[Frakes & Fox 1995] Frakes, W. B., Fox, C. J. (1995). “Sixteen questions about software reuse”. *Communications of the ACM*, v. 38, n. 6, pp. 75-87, June.

[Frakes & Fox 1996] Frakes, W., Fox, C. (1996). “Quality Improvement Using a Software Reuse Failure Modes Model”. *IEEE Transactions on Software Engineering*, v. 22, n. 4, pp. 274-279, April.

[Frakes & Kang 2005] Frakes, W. B., Kang, K. (2005). “Software reuse research: Status and future”. *IEEE Transactions on Software Engineering*, v. 31, n. 7, pp. 529-536.

[Frakes & Terry 1994] Frakes, W., Terry, C. (1994). “Reuse Level Metrics”. In: *3rd International Conference on Software Reuse (ICSR 1994)*, Rio de Janeiro, Brazil, pp. 139-148, November.

[Gallagher et al. 2008] Gallagher, K., Hatch, A., Munro, M. (2008), “Software Architecture Visualization: An Evaluation Framework and its Application”, *IEEE Transactions on Software Engineering*, v. 34, n. 2, pp. 260-270.

[Gill 2006] Gill, N. S. (2006). “Importance of Software Component Characterization for Better Software Reusability”. *SIGSOFT Software Engineering Notes*, v. 31, n. 1, pp. 1-3, January.

[Griss et al. 1994] Griss, M. L., Favaro, J., Walton, P. (1994). “Managerial and organizational issues – Starting and Running a Software Reuse Program”. In: Schäfer, W., Prieto-Díaz, R., Matsumoto, M. (eds.), *Software Reusability*, pp. 51-78, Ellis Horwood Ltd.

[Gutwin et al. 2004] Gutwin, C., Penner, R., Schneider, K. A. (2004). “Group Awareness in Distributed Software Development”. In: *ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*, Chicago, USA, pp. 72-81, November.

[Hadji et al. 2008] Hadji, H. B., Kim. S. K., Choi, H. J. (2008). “A Representation Model for Reusable Assets to Support User Context”. In: *4th International Symposium on Service-Oriented System Engineering*, Jhongli, Taiwan, pp. 91-96, December.

[Haefliger et al. 2008] Haefliger, S., von Krogh, G., Spaeth, S. (2008). “Code Reuse in Open Source Software”. *Management Science*, v. 54, n. 1, pp. 180-193, January.

[Hattori 2010] Hattori, L. (2010). “Enhancing collaboration of multi-developer projects with synchronous changes”. In: *32nd ACM/IEEE International Conference on Software Engineering*, Cape Town, South Africa, pp. 377-380, May.

[Holmes 2008] Holmes, R. (2008). “Pragmatic software reuse”. Ph.D. Thesis, University of Calgary, Calgary, Canada, November.

[Hooper & Chester 1991] Hooper, J. W., Chester, R. O. (1991). “Software Reuse: Guidelines and Methods”, Springer, 180p.

[IEEE 2010] IEEE (2010). “IEEE Std. 1517-2010: IEEE Standard for Information Technology – System and Software Life Cycle Processes – Reuse Processes”, 39p, Institute of Electrical and Electronics Engineers.

[ISO/IEC 2008] ISO/IEC (2008). “ISO/IEC 12207:2008 – Systems and software engineering – Software life cycle processes”, 123p, International Organization for Standardization and the International Electrotechnical Commission, Geneva, Switzerland.

[ISO/IEC 2012] ISO/IEC (2012). “ISO/IEC 15504-5:2012 – Process assessment, Part 5: An exemplar software life cycle process assessment model”, 196p, International Organization for Standardization and the International Electrotechnical Commission, Geneva, Switzerland.

[Kagdi & Maletic 2008] Kagdi, H., Maletic, J. I. (2008). “Expressiveness and effectiveness of program comprehension: Thoughts on future research directions”. In: *Proceedings of the IEEE Frontiers of Software Maintenance (FoSM 2008)*, Beijing, China, pp. 31-37, October.

[Kang et al. 1990] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., Peterson, A. S. (1990). *Feature-oriented domain analysis (FODA) feasibility study* (No. CMU/SEI-90-TR-21). Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, USA.

[Keim et al. 2008] Keim, D. A., Mansmann, F., Schneidewind, J., Thomas, J., Ziegler, H. (2008). “Visual analytics: Scope and challenges”. In: Simoff, S. J., Böhlen, M. H., Mazeika, A. (eds.), *Visual Data Mining*, pp. 76-90, Springer Berlin Heidelberg.

[Kelleher 2005] Kelleher, J. (2005). “A reusable traceability framework using patterns”. In: *Proceedings of the 3rd International Workshop on Traceability in Emerging*

Forms of Software Engineering (TEFSE 2005), Long Beach, USA, pp. 50-55, November.

[Kim & Stohr 1998] Kim, Y., Stohr, E.A. (1998). "Software reuse: survey and research directions". *Journal of Management Information Systems*, v. 14, n. 4, pp. 113-147, March.

[Kitchenham et al. 2009] Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., Linkman, S. (2009). "Systematic literature reviews in software engineering – A systematic literature review", *Information and Software Technology*, v. 51, n. 1, pp. 7-15, January.

[Klerkx et al. 2006] Klerkx, J., Verbert, K., Duval, E. (2006). "Visualizing Reuse: More than Meets the Eye". In: *6th International Conference on Knowledge Management (I-KNOW)*, Graz, Austria, pp. 489-497.

[Koschke 2003] Koschke, R. (2003). "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey," *Journal of Software Maintenance and Evolution: Research and Practice*, v. 15, n. 2, pp. 87-109.

[Krueger 1992] Krueger, C. W. (1992). "Software reuse". *ACM Computing Surveys*, v. 24, n. 2, pp. 131-183, June.

[Lanza & Marinescu 2006] Lanza, M., Marinescu, R. (2006). *Object-Oriented Metrics in Practice*. Springer-Verlag Berlin Heidelberg.

[Lee et al. 2012] Lee, B., Isenberg, P., Riche, N. H., Carpendale, S. (2012). "Beyond Mouse and Keyboard: Expanding Design Considerations for Information Visualization Interactions". *IEEE Transactions on Visualization and Computer Graphics*, v. 18, n. 12, pp. 2689-2698, December.

[Lucrédio et al. 2008] Lucrédio, D., Brito, K. S., Alvaro, A., Garcia, V. C., Almeida, E. S., Fortes, R. P. M., Meira, S. L. (2008). "Software reuse: The Brazilian industry scenario". *Journal of Systems and Software*, v. 81, n. 6, pp. 996-1013, June.

[Mackinlay 1986] Mackinlay, J. D. (1986). "Automating the design of graphical presentation of relational information". *ACM Transaction on Graphics*, v. 5, n. 2, pp. 110-141, April.

[Maletic et al. 2002] Maletic, J. I., Marcus, A., Collard, M. L. (2002). "A task oriented view of software visualization". In: *Proceedings of the 1st International Workshop*

on Visualizing Software for Understanding and Analysis (VISSOFT 2002), Paris, France, pp. 32-40, June.

[Mancoridis et al. 1993] Mancoridis, S., Holt, R. C., Penny, D. A. (1993). "Conceptual framework for software development". In: *Proceedings of the 1993 ACM Computer Science Conference*, Indianapolis, USA, pp. 74-80, February.

[Mariani & Pastore 2011] Mariani, L., Pastore, F. (2011). "Supporting plug-in mashes to ease tool integration". In: *1st Workshop on Developing Tools as Plug-ins (TOPI 2011)*, Cape Town, Africa, pp. 1-4, May.

[Marshall 2001] Marshall, S. (2001). "Using and Visualizing Reusable Code: Position Paper for Software Visualization Workshop". In: *Workshop on Software Visualization, 2001 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2001)*, Tampa, USA, pp. 1-4, October.

[Marshall et al. 2003] Marshall, S. Jackson, K., Anslow, C., Biddle, R. (2003). "Aspects to visualising reusable components". In: *Proceedings of the Australasian Symposium on Information Visualisation (InVis.au 2003)*, Adelaide, Australia, pp. 81-88, February.

[Mello et al. 2014] Mello, R. M., Teixeira, E. N., Schots, M., Werner, C. M. L., Travassos, G. H. (2014). "Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections". *Journal of Universal Computer Science (J.UCS)* (to appear).

[Mili et al. 1995] Mili, H., Mili, F., Mili, A. (1995). "Reusing software: Issues and research directions". *IEEE Transactions on Software Engineering*, v. 21. n. 6, pp. 528-562.

[Moore & Bailin 1991] Moore, J. M., Bailin, S. C. (1991). "Domain Analysis: Framework for reuse". In: Prieto-Díaz, R., Arango, G. (eds.), *Domain Analysis and Software System Modeling*, pp. 179-202, IEEE Computer Society Press, Los Alamitos, USA.

[Morisio et al. 2002] Morisio, M., Ezran, M. and Tully, C. (2002). Success and failure factors in software reuse. *IEEE Transactions on Software Engineering*, v. 28, n. 4, pp. 340-357.

[Mukherjea & Foley 1996] Mukherjea, S., Foley, J. (1996). “Requirements and Architecture of an Information Visualization Tool”. In: *Database Issues for Data Visualization*, Springer Berlin/Heidelberg, pp. 57-75.

[Mulholland 1997] Mulholland, P. (1997). “Using a fine-grained comparative evaluation technique to understand and design software visualization tools”. In: *7th Workshop on Empirical Studies of Programmers*, Alexandria, USA, pp. 91-108, October.

[Naur & Randell 1968] Naur, P., Randell, B. (1968). “Software Engineering”, Scientific Affairs Division, NATO, Brussels, Garmisch, Germany, Report on a conference sponsored by the NATO Science Committee, October.

[Norman 2010] Norman, D. A. (2010). “Natural User Interfaces Are Not Natural”. *Interactions*, v. 17, n. 3, pp. 6-10, May.

[Novais et al. 2012] Novais, R., Nunes, C., Lima, C., Cirilo, E., Dantas, F., Garcia, A., Mendonça, M. (2012). “On the proactive and interactive visualization for feature evolution comprehension: An industrial investigation”. In: *34th International Conference on Software Engineering (ICSE 2012) – Software Engineering in Practice (SEIP) Track*, Zürich, Switzerland, pp. 1044-1053, June.

[Oliveira 2011] Oliveira, M. S. (2011). “PREViA: An Approach for Visualizing the Evolution of Software Models” [PREViA: Uma Abordagem para a Visualização da Evolução de Modelos de Software] (in Portuguese). M.Sc. Thesis, COPPE/UFRJ, Rio de Janeiro, Brazil, March.

[Oliveira et al. 2014] Oliveira, P., Valente, M. T., Lima, F. P. (2014). “Extracting relative thresholds for source code metrics”. In: *1st Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE 2014)*, Antwerp, Belgium, pp. 254-263, February.

[Orso et al. 2000] Orso, A., Harrold, M. J., Rosenblum, D. S. (2000). “Component Metadata for Software Engineering Tasks”, *2nd International Workshop on Engineering Distributed Objects*, Davis, USA, pp. 126-140, November.

[Palmieri et al. 2013] Palmieri, M., Schots, M., Werner, C. (2013). “ReuseDashboard: Supporting Stakeholders in Monitoring Software Reuse Programs” [ReuseDashboard: Apoiando Stakeholders na Monitoração de Programas de

Reutilização de Software] (in Portuguese). In: *1st Brazilian Workshop on Software Visualization, Evolution, and Maintenance (VEM 2013)*, Brasília, Brazil, pp. 54-61, September.

[Pereira & Schots 2011] Pereira, T. A., Schots, M. (2011). "GraphVCS: An Approach for Visualizing and Understanding Version Control Repositories" [GraphVCS: Uma Abordagem para a Visualização e Compreensão de Repositórios de Controle de Versão] (in Portuguese). In: *1st Brazilian Workshop on Software Visualization (WBVS 2011)*, São Paulo, Brazil, pp. 1-8.

[Poulin et al. 1993] Poulin, J. S., Caruso, J. M., Hancock, D. R. (1993). "The Business Case for Software Reuse". *IBM Systems Journal*, v. 32, n. 4, pp. 567-594, October.

[Poulin 1994] Poulin, J. S. (1994). "Measuring Software Reusability". In: *3rd International Conference on Software Reuse (ICSR 1994)*, Rio de Janeiro, Brazil, pp. 126-138, November.

[Prieto-Díaz & Arango 1991] Prieto-Díaz, R., Arango, G. F. (1991). *Domain Analysis and Software Systems Modeling*. IEEE Computer Society Press.

[Queiroz et al. 2012] Queiroz, A. R., Oliveira, M. S., Werner, C. M. L. (2012). "Supporting Project Management through Visual Analytics of Scenario Data" [Apoio ao Gerenciamento de Projetos por Meio da Análise Visual de Dados de Cenários]. In: *XXXIV Jornada Giulio Massarani de Iniciação Científica, Tecnológica, Artística e Cultural UFRJ*, pp. 229.

[Queiroz et al. 2013] Queiroz, A. R., Oliveira, M. S., Werner, C. M. L. (2013). "Systematic Support for the Choice of Information Visualizations based on Representation Constraints" [Apoio Sistemático à Escolha de Visualizações de Informação Baseada em Restrições de Representação]. In: *XXXV Jornada Giulio Massarani de Iniciação Científica, Tecnológica, Artística e Cultural UFRJ*, pp. 102.

[Ripley et al. 2007] Ripley, R. M., Sarma, A., Van der Hoek, A. (2007). "A Visualization for Software Project Awareness and Evolution". In: *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007)*, Banff, Canada, pp. 137-144, June.

[Robbes & Lanza 2006] Robbes, R., Lanza, M. (2006). "Change-Based Software Evolution". In: *Proceedings of the 2nd International ERCIM Workshop on Software Evolution (EVOL 2006)*, Lille, France, pp. 159-164, April.

[Robillard et al. 2010] Robillard, M., Walker, R., Zimmermann, T. (2010). "Recommendation System for Software Engineering". *IEEE Software*, v. 27, n. 4, pp. 80-86, July.

[Rocha et al. 2007] Rocha, A. R. C., Montoni, M., Weber, K. C., Araujo, E. E. R. (2007). "A Nationwide Program for Software Process Improvement in Brazil". In: *6th International Conference on Quality of Information and Communications Technology (QUATIC 2007)*, Lisbon, Portugal, pp. 167-176, September.

[Rodrigues & Werner 2011] Rodrigues, C. S. C., Werner, C. M. L. (2011). "Making the comprehension of software architecture attractive". In: *24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T 2011)*, Honolulu, Hawaii, pp. 416-420.

[Sá et al. 1997] Sá, M. L. B., Werner, C. M. L., Goldman, I. (1997). "Introduction of Reuse in a Brazilian Enterprise on Software Production" [Introdução da Reutilização em uma Empresa Brasileira de Produção de Software] (in Portuguese). In: *11th Brazilian Symposium on Software Engineering (SBES 1997)*, Fortaleza, Brazil, pp. 233-248, October.

[Sametinger 1997] Sametinger, J. (1997). *Software Engineering with Reusable Components*. Ed. 1997. Springer-Berlin, 288p.

[Santa Isabel 2011] Santa Isabel, S. L. (2011). "Selection of Testing Approaches for Web Applications" [Seleção de Abordagens de Teste para Aplicações Web] (in Portuguese). M.Sc. Thesis, COPPE/UFRJ, Rio de Janeiro, Brazil, July.

[Santos et al. 2009] Santos, G., Zanetti, D., Maciel, M., Simões, C. A., Werner, C., Rocha, A. R. (2009). "The Experience of the Implementation of Reuse Management and Development for Reuse Processes in Synapsis-Brazil" [A Experiência de Implantação dos Processos Gerência de Reutilização e Desenvolvimento para Reutilização na Synapsis-Brasil] (in Portuguese). In: *Proceedings of the 5th Annual Workshop of MPS (WAMPS)*, Campinas, Brazil, pp. 128-135, October.

[Schmidt 1999] Schmidt, D. C. 1999. “Why software reuse has failed and how to make it work for you”. *C++ Report*, SIGS Publications Group, January.

[Schots et al. 2010] Schots, M., Silva, M. A., Murta, L. G. P., Werner, C. M. L. (2010). “Adherence Checking between Conceptual and Emerging Architectures by Using the PREViA Approach” [Verificação de Aderência entre Arquiteturas Conceituais e Emergentes Utilizando a Abordagem PREViA] (in Portuguese). In: *7th Brazilian Workshop on Modern Software Maintenance (WMSWM 2010)*, Belém, Brazil, pp 1-8, June.

[Schots & Werner 2012] Schots, M., Werner, C. (2012). “Exploiting the Intangible: An Overview of Software Visualization and its Applications” [Explorando o Intangível: Um Panorama da Visualização de Software e suas Aplicações] (in Portuguese). Tutorial. In: *III Brazilian Congress on Software: Theory and Practice (CBSoft 2012)*, Natal, Brazil.

[Schots et al. 2012] Schots, M., Werner, C., Mendonça, M. (2012). “Awareness and Comprehension in Software/Systems Engineering Practice and Education: Trends and Research Directions”. In: *26th Brazilian Symposium on Software Engineering (SBES)*, Natal, Brazil, pp. 186-190.

[Schots & Werner 2013] Schots, M., Werner, C. (2013). “Characterizing the Implementation of MR-MPS-SW Reuse Processes: Preliminary Results” [Caracterizando a Implementação de Processos de Reutilização do MR-MPS-SW: Resultados Preliminares]. In: *Proceedings of the 9th Annual Workshop of MPS (WAMPS 2013)*, Campinas, Brazil, pp. 44-53, October.

[Schots 2014] Schots, M. (2014). “On the Use of Visualization for Supporting Software Reuse”. In: *36th International Conference on Software Engineering (ICSE 2014), Doctoral Symposium*, Hyderabad, India, pp. 694-697, June (to appear).

[Schots et al. 2014] Schots, M., Vasconcelos, R., Werner, C. (2014). “A *Quasi*-Systematic Review on Software Visualization Approaches for Software Reuse”, Technical Report, COPPE/UFRJ, Rio de Janeiro, Brazil (to appear).

[Seaman 1999] Seaman, C. B. (1999). “Qualitative methods in empirical studies of software engineering”. *IEEE Transactions on Software Engineering*, v. 25, n. 4, pp. 557-572, July.

[Seaman 2009] Seaman, C. (2009). “Using Qualitative Methods in Empirical Studies of Software Engineering”. Short course. In: *VI Experimental Software Engineering Latin American Workshop (ESELAW 2009)*, São Carlos, Brazil, November.

[Selby 2005] Selby, R. W. (2005). “Enabling reuse-based software development of large-scale systems”. *IEEE Transactions on Software Engineering*, v. 31, n. 6, pp. 495-510, June.

[Sensalire et al. 2009] Sensalire, M., Ogao, P., Telea, A. (2009). “Evaluation of software visualization tools: Lessons learned”. In: *5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2009)*, Edmonton, Canada, pp. 19-26, September.

[Sherif & Vinze 2003] Sherif, K., Vinze, A. (2003). “Barriers to adoption of software reuse: A qualitative study”. *Information & Management*, v. 41, n. 2, pp. 159-175, December.

[Shi et al. 2011] Shi, Z., Wang, X., Yue, J. (2011) “Cognitive Cycle in Mind Model CAM”. *International Journal of Intelligence Science*, v. 1, n. 2, pp. 25-34.

[Silva et al. 2012] Silva, M., Schots, M., Werner, C. (2012). “Supporting Software Maintenance Activities through a Software Visualization Product Line Infrastructure”. In: *9th Workshop on Modern Software Maintenance (WMSWM 2012)*, Fortaleza, Brazil, pp. 1-8, June.

[Silva Filho et al. 2008] Silva Filho, R. C., Katsurayama, A. E., Santos, G., Murta, L., Rocha, A. R. C. (2008). “Deploying Software Reuse Management at COPPE/UFRJ Software Engineering Laboratory”. In: *1st Workshop on Software Reuse Efforts (WSRE), 2nd RiSE Summer School on Software Product Lines (RiSS 2008)*, Recife, Brazil, pp. 1-5, November.

[SOFTEX 2012] SOFTEX (2012). “MPS.BR – Brazilian Software Process Improvement – General Guide” [MPS.BR – Melhoria de Processo do Software Brasileiro – Guia Geral] (in Portuguese), August. Available at http://www.softex.br/mpsbr.

[SOFTEX 2013a] SOFTEX (2013a). “Implementation Guide – Part 3: Reasoning for the Implementation of Level E of MR-MPS-SW:2012” [Guia de Implementação –

Parte 3: Fundamentação para Implementação do Nível E do MR-MPS-SW:2012] (in Portuguese), September. Available at http://www.softex.br/mpsbr/guias/.

[SOFTEX 2013b] SOFTEX (2013b). "Implementation Guide – Part 5: Reasoning for the Implementation of Level C of MR-MPS-SW:2012" [Guia de Implementação – Parte 5: Fundamentação para Implementação do Nível C do MR-MPS-SW:2012] (in Portuguese), September. Available at http://www.softex.br/mpsbr/guias/.

[SOFTEX 2013c] SOFTEX (2013c). "Assessment Guide", August. Available at http://www.softex.br/mpsbr/english-guides/.

[SOFTEX 2013d] SOFTEX (2013d). "MPS-SW Published Assessments" [Avaliações MPS-SW (Software) Publicadas] (in Portuguese), August. Available at http://www.softex.br/wp-content/uploads/2013/07/Avalia%C3%A7%C3%B5es-MPS-SW.pdf.

[SOFTEX 2014a] SOFTEX (2014a). "Authorized Institutions" [Instituições Autorizadas] (in Portuguese). Available at: http://www.softex.br/mpsbr/instituicoes-autorizadas/ia/.

[SOFTEX 2014b] SOFTEX (2014b). "Authorized Professionals" [Profissionais Habilitados] (in Portuguese). Available at: http://www.softex.br/mpsbr/profissionais-habilitados-2/.

[Telea et al. 2010] Telea, A., Ersoy, O., Voinea, L., (2010). "Visual Analytics in Software Maintenance: Challenges and Opportunities". In: *1st European Symposium on Visual Analytics (EuroVAST)*, Bordeaux, France, pp. 65-70, June.

[Thomas & Cook 2006] Thomas, J. J, Cook, K. A. (2006) "A visual analytics agenda". *IEEE Computer Graphics and Applications*, v. 26, n. 1, pp. 10-13, January.

[Tichy 1998] Tichy, W. F. (1998). "Should computer scientists experiment more?", *IEEE Computer*, v. 31, n. 5, pp. 32-40, May.

[Travassos et al. 2008] Travassos, G. H., Santos, P. S. M., Mian, P. G., Dias Neto, A. C., Biolchini, J. (2008). "An environment to support large scale experimentation in software engineering". In: *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008)*, Belfast, Northern Ireland, pp. 193-202, April.

[Treude & Storey 2010] Treude, C., Storey, M.-A. (2010). “Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds”. In: *32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, Cape Town, South Africa, pp. 365-374, May.

[Vasconcelos et al. 2013] Vasconcelos, R. R., Schots, M., Werner, C. (2013). “Recommendations for Context-Aware Visualizations in Software Development”. In: *10th Workshop on Modern Software Maintenance (WMSWM 2013)*, Salvador, Brazil, pp. 41-48, July.

[Vasconcelos et al. 2014a] Vasconcelos, R., Schots, M., Werner, C. (2014). “An Information Visualization Feature Model for Supporting the Selection of Software Visualizations”. In: *22nd International Conference on Program Comprehension (ICPC 2014), Early Research Achievements Track*, Hyderabad, India, pp. 122-125, June (to appear).

[Vital & Krause 2013] Vital, G. B., Krause, V. S. (2013). “Rec4Reuse: A system for performing evaluations and recommendations based on desirable properties of reusable software” [Rec4Reuse: Um sistema de avaliação e recomendação baseado em propriedades desejáveis a software reutilizável] (in Portuguese). Undergraduate Final Project, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, Brazil.

[Von Mayrhauser & Vans 1995] Von Mayrhauser, A., Vans, A. M. (1995). “Program comprehension during software maintenance and evolution”, *Computer*, v. 28, n. 8, pp. 44-55, August.

[Werner et al. 2011] Werner, C., Murta, L., Schots, M., Magdaleno, A., Silva, M., Cepeda, R., Vahia, C. (2011). “EvolTrack: A Plug-in-Based Infrastructure for Visualizing Software Evolution”. In: *1st Brazilian Workshop on Software Visualization (WBVS 2011)*, São Paulo, Brazil, pp. 1-8.

[Wettel & Lanza 2008] Wettel, R., Lanza, M. (2008). “Visual Exploration of Large-Scale System Evolution”. In: *15th Working Conference on Reverse Engineering (WCRE 2008)*, Antwerp, Belgium, pp. 219-228, October.

[Wong et al. 2007] Wong, K., Stroulia, E., Tonella, P. (2007). “Message from the Chairs”. In: *15th IEEE International Conference on Program Comprehension (ICPC 2007)*, Banff, Canada, pp. ix, June.

[Wu & Storey 2000] Wu, J., Storey, M.-A. D. (2000). “A Multi-Perspective Software Visualization Environment”. In: *Conference of the Centre for Advanced Studies on Collaborative Research*, Mississauga, Canada, pp. 1-10, November.

[Ye & Fischer 2000] Ye, Y., Fischer, G., Reeves, B. (2000). “Integrating Active Information Delivery and Reuse Repository Systems”. In: *ACM SIGSOFT Symposium on Foundations on Software Engineering (FSE 2000)*, San Diego, USA, pp. 60-68, November.

[Ye & Fischer 2002] Ye, Y., Fischer, G. (2002). “Supporting reuse by delivering task-relevant and personalized information”. In: *24th International Conference on Software Engineering (ICSE 2002)*, Orlando, USA, pp. 513-523, May.