University of Salamanca

Faculty of Science

Department of Computer Science and Automation

Evolutionary Visual Software Analytics

Ph.D. Dissertation

Antonio González Torres

Doctoral Advisors
Dr. Roberto Therón Sánchez
Dr. Francisco J. García Peñalvo

2015

Dr. D. Francisco José García Peẽalvo, Profesor Titular del Departamento de Informática y Automática de la Universidad de Salamanca

Dr. D. Roberto Therón Sánchez, Profesor Titular del Departamento de Informática y Automática de la Universidad de Salamanca

HACEN CONSTAR: Que D. Antonio González Torres, ha realizado bajo nuestra dirección el trabajo de investigación y la memoria de la tesis doctoral que lleva por título Evolutionary Visual Software Analytics, con el fin de obtener el grado de Doctor por la Universidad de Salamanca con mención de Doctorado Internacional. Asimismo manifestamos que dicho trabajo tiene suficientes méritos teóricos contrastados, mediante las validaciones oportunas, publicaciones y aportaciones novedosas. Por todo ello se considera que procede su defensa pública.

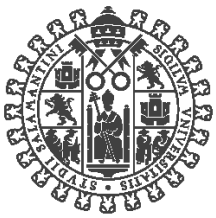
Y para que surta los efectos oportunos firmamos en Salamanca, a 21 de abril de dos mil quince.

Dr. Roberto Therón Sánchez	Dr. Francisco J. García Peñalvo
Profesor Titular	Profesor Titular
Universidad de Salamanca	Universidad de Salamanca

University of Salamanca

Faculty of Science

Department of Computer Science and Automation



VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Evolutionary Visual Software Analytics

Ph.D. Dissertation

Doctoral Advisors

Dr. Roberto Therón Sánchez

Dr. Francisco J. García Peñalvo

Ph.D. Candidate

Antonio González Torres

To my brother Ricardo
and the loving memory of my mother María Eusebia.

Acknowledgments

Every journey is formed by stretches and stations, so to make a long trip, several stretches must be walked and many stations should be reached. Thus, I have arrived to this station after walking many stretches, where several travel partners have made the trip less rough, and sometimes have also provided fresh water in the way. Therefore, it will not be possible to acknowledge in this little space to every travel partner I have had, but I want to express my gratitude for being there when I needed them.

Some of my partners in the stretch towards this station have been my doctoral advisors Roberto and Francisco, the members of the Department of Computer Science and Automation of the University of Salamanca, the scholarship programs of MICITT-CONICIT (Costa Rica), Banco Santander-USAL, the Ministry of Education (Spain) and my research mates and friends José Antonio, Cristián, Diego, Juan, Saddys, Joel, Carlos and Vadim; and my friends Javier, Luis, Yanira, Olga, José (Pepe), José María, Ricardo, Lourdes, José Miguel, Michel, Carolina, Victoria, Juanita, Carlos, Estela, Carlomagno, Andrea, Gabriela, Neli, Nieves, María Jesús, Danny, Edwin Aguilar, Néstor and many others. Other partners that have been on this journey with me for a long time and several stretches are Teresa, Rodney, Edwin Montero, Luis, Erika, Clarette, Alfonso and several people from the *UCR* (*ECCI* and *SRP*), as well as many members of my family that have been on my side all over the journey or in some critical stretches (Marielos, Maricruz, Mauricio, Dayton, Karla, George, Gabriela, Marianela, Morayma, Onocifero, Juana, Francisca, Maco and others).

Thank you to all of you!

Resumen

El desarrollo y mantenimiento de sistemas de software involucran a un gran número de complejos procesos que se extienden por largos periodos de tiempo (en algunos casos 10 años o más), e implican a grupos de personas (*e.g.*, programadores y administradores de proyectos) que pueden encontrarse en diferentes países. Por lo cual quienes participan en esos procesos requieren de herramientas que les faciliten la comprensión de los sistemas, sus componentes y las relaciones que se establecen entre estos en el tiempo.

La comprensión de los sistemas adquiere una relevancia especial cuando se toma en cuenta la rotación de personal en las organizaciones y la frecuente ausencia de documentación técnica de los sistemas. Por lo tanto, en esta tesis se llevó a cabo un análisis detallado sobre las necesidades que tienen los programadores y administradores de proyectos, se hizo un mapeo sistemático de literatura y una revisión detallada de literatura; y se efectuó una encuesta sobre el uso de herramientas de visualización en la industria de software y departamentos de informática en la comprensión de los sistemas. Con base en los resultados obtenidos de las actividades anteriores, se realizó la definición y descripción del proceso de aplicación de la Analítica Visual a la Evolución de Software (el cual recibió el nombre de Evolutionary Visual Software Analytics).

La validación del proceso mencionado se llevó a cabo en tres etapas. En la primera etapa se diseñó una arquitectura con el fin de verificar que mediante el seguimiento de la descripción del proceso es posible diseñar herramientas de Analítica Visual para facilitar la comprensión de la evolución de los sistemas de software. En la segunda etapa se validó la arquitectura mediante la implementación de *Maleku* (una herramienta basada en dicha arquitectura). En la tercera etapa, se verificó la utilidad y usabilidad de *Maleku* en la comprensión de la evolución de sistemas de software por medio de varios casos de uso, un caso de estudio y un estudio de usabilidad.

Los resultados finales de este trabajo permitieron comprobar que la aplicación de la Analítica Visual a la Evolución de Software, usando el proceso descrito en esta investigación, puede contribuir con el desarrollo y mantenimiento de software al facilitar la comprensión de los sistemas, y por tanto, las preguntas de investigación de esta tesis fueron respondidas y los objetivos planteados se cumplieron.

Palabras clave: Evolutionary Visual Software Analytics, Analítica Visual Aplicada a la Evolución de Software, Analítica Visual, Visualización de Software, Evolución de Software, Análisis de la Evolución de los Sistemas

Abstract

The development and maintenance of software systems involve a large number of complex processes (that could be extended for long periods of time) and people (*e.g.*, programmers and project managers) who may be located in different countries. Therefore, people involved in these processes require tools to understand the systems, their components and the relationships established between these in time.

Understanding systems becomes particularly relevant when taking into account staff turnover in organizations and the frequent absence of technical system documentation. Therefore, a detailed study on the needs of programmers and project managers, a systematic mapping study, a detailed literature review and a survey on the use of visualization tools in the software industry and IT departments for system understanding were carried out in this thesis. Based on the results of the above activities, the definition and description on the application of Visual Analytics to Software Evolution (which was called Evolutionary Visual Software Analytics) was performed.

The validation of this process was conducted in three stages. In the first stage, an architecture was designed to verify that by following the Evolutionary Visual Software Analytics process description it is possible to design Visual Analytics tools to facilitate the understanding of the evolution of software systems. In the second stage, the architecture was validated by implementing *Maleku* (a tool based on this architecture). In the third stage, the usefulness and usability of *Maleku* in understanding the evolution of software systems was verified through various use cases, an usability study and a case study.

The final results of this study allowed us to prove that the application of Visual Analytics to Software Evolution, using the process described in this research, can contribute to software development and maintenance to facilitate the understanding of systems, and therefore the research questions of this thesis were answered and the specified objectives were met.

Keywords: Evolutionary Visual Software Analytics, Visual Analytics, Software Visualization, Software Evolution, Software Evolution Analysis, Software Maintenance

Table of Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Nevo prometió a Güindy que harían un viaje especial con destino desconocido. El día del viaje, antes de que saliera el sol, Güindy preparó de forma diligente los menesteres indispensables para abrirse paso y sobrevivir en la selva. Con todo listo, emprendieron el viaje junto a Cucho Comecuanduay, su perro. — El viaje de Güindy, A.González

Contents

1.1 Presentation

Software systems are almost omnipresent in daily life and used in almost all devices which are utilized by people and businesses. Individuals use these contrivances and the associated software to do the following things (among many others): to work; to learn (take online courses, read and research topics of interest); entertain themselves (play, watch TV or videos, listen to music); communicate (friends, family, co-workers, participate in forums, collaborate and work meetings),to buy things; to do paperwork (banking, taxes, and other payments) and telecommuting [Charette 2005]. This has been the social reality of the recent past; is current social reality; and will continue to be everyday social reality in the future.

The omnipresence of technology has led some individuals and organizations to become socially and economically dependent on software systems [Boehm 1999b]. As a result, the percentage of profits that companies invest annually in this area, on average, is greater than 4% [Charette 2005, Hall 2013], and the expenditure in software products grows annually [Gartner 2013, Gartner 2014].

Given this environment, the software market is very attractive to investors and competition between producers is intense. It is thus necessary to bear in mind that a product is valuable if it enables or helps people and organizations to achieve particular goals or objectives by means of its use [Boehm 1999b]. The software industry in general, and more specifically, internal software development departments (being fully aware of this situation) seek to develop products which meet the requirements and functionality demanded by users with high standards of quality, in the shortest time and at the lowest cost possible. This statement is also valid when talking about open source software [Lee 2009].

At this point, it is appropriate to consider that the development and maintenance of software are complex and that the possibility of failure is present at all stages and levels of the process [Kraut 1995, Procaccino 2002, Morisio 2002, Chow 2008]. Annual reports regarding this problem are published [Group 2013] which evaluate the general performance of the software industry and the success rates of particular projects. It must be mentioned that studies have been published concerning famous cases of software development projects whose failure has cost the loss of millions [Charette 2005] and which led to a loss of prestige for some companies involved in these projects.

With these examples in mind, great efforts have been made to improve elements of the development process, such as the calculation of costs and risks, planning, the reuse of software components [García-Peñalvo 2000, Garcıa-Peñalvo 2002, Laguna 2003] and system design and maintenance [Boehm 1999b, Sullivan 2001, Royce 2009]. Many of these efforts aim to improve the technical skills of individuals and processes in order to obtain better economic results [Boehm 2000, Colomo-Palacios 2013, Buxmann 2013, Colomo-Palacios 2014]. It is worth noting that in economic terms, software engineering techniques have value if they facilitate the development of more valuable software [Boehm 1999b].

Given this more general context, this thesis seeks to contribute to the software economy by supporting the process of Software Development and Maintenance (SDM) through the definition of a Visual Analytics (VA) process in order to facilitate the analysis of software projects and their evolution.

Some factors that affect the success or failure of software projects that are

taken into account in this research are the following:

1. Control and monitoring of the quality of software by the use of metrics [Niazi 2006, Lee 2009, Nasir 2011].
2. The importance of change control and configuration [Nasir 2011].
3. The distributed location (sometimes in different countries and continents) of collaborators on the project and their level of awareness of the activities and changes carried out by project staff [Niazi 2006, Fabriek 2008].
4. The need for reliable automatic tools to obtain information about the SDM process and the changes that have been made to the project source code [Niazi 2006].

1.2 Research Problem

The objective of Software Evolution Analysis (SEA) is to support project managers and programmers during the process of change and evolution of software in separate geographical environments [Estublier 1999, Ogawa 2009]. Project administrators must have a general vision of the process which allows them to control the quality of the software; evaluate productivity; reduce implementation and maintenance risks as well as having the capacity to report on all these activities to higher management levels. While programmers have to learn the new code bases to understand structural changes, as well as the changes in inheritance relationships and interface implementation. Moreover, programmers need to understand the dependencies of software items and comprehend the differences of source code revisions and have access to the development history.

SEA actively looks to aid the improvement of the software process through the analysis and support to continuous change, complexity, growth and quality control [Lehman 1997]. However, SEA produces large and complex datasets, due to the number of variables involved in the evolution process and the intricacies of their relationships that are difficult to understand by humans. For these aforementioned reasons, although SEA provides valuable elements of information, it does not provide sufficient knowledge to satisfactorily carry out the tasks of understanding the changes and evolution of the software.

Accordingly, the next sections discuss some key issues and concepts that are related to the aforementioned problem. Firstly, *Big Data* is introduced and the relationship with *software evolution* is established, secondly the software development and maintenance process is explained and thirdly, the use of VA to support software developers and managers is discussed.

1.2.1 Big Data

During the last decades several changes have taken place on how organizations and individuals generate, process and share information. On the one hand, the emergence of new devices, such as smartphones and tablets, and the improvement in processing and storage capabilities of computers and servers, has increased the abilities of individuals and organizations to create new and different data content formats. While on the other hand, the data generated can be shared in real time at high speeds, thanks to the new technologies. Thus, data with more varied and complex formats is created, collected and transmitted from numerous sources including sensor networks, Global Positioning System (GPS), Radio Frequency Identification (RFID) tags, Wireless Fidelity (Wi-Fi) networks, satellites, multimedia streams and software used by mobile devices [Krishnan 2013]. Consequently, individuals and organizations have access to more data than what they are able to process and transform into knowledge and this is causing information overload (in all areas of human endeavor).

In this context, the term *Big Data* is frequently used to make reference to the volume, variety, velocity and complexity of data produced daily. The main goal of *Big Data* related research and technologies is to manage and transform available real-time and historical data into knowledge to inform decisions according to organizational requirements and needs [Hemerly 2013]. Although data characteristics have changed over time, many challenges associated with *Big Data* are not new and have been discussed in the research arena for years, even decades. Such challenges are related to the retrieval and integration of heterogeneous data sources into large data volumes, its processing and storage and the scalability of tools that perform automatic data analysis.

Although, *Big Data* represents a great opportunity for businesses to take advantages of the Data Economy, in this context, *analytics* deserves special attention in providing insight into large volumes of data [LaValle 2010], as it is located at the top of the process stack that transforms data into knowledge. According to Davenport [Davenport 2006], analytics has been an important player in the definition of strategic plans, and has helped to drive, for decades, the improvement of the ability of organizations to outperform competitors by means of analyzing the available data.

VA is a natural evolution of analytics that exploits new computational algorithms for data analysis and the ability to present and explore their results by means of visual representations. VA is now widely accepted as an essential tool in a wide range of domains such as Business Intelligence (BI), security, marketing, life sciences, and social sciences.

The next section discusses the software development and maintenance

process and the involved data elements and their relationships, whereas chapter 2 provides a detailed explanation of such process.

1.2.2 Software Development and Maintenance

One of the most critical problems for some companies is the frequent job-hoping of software developers and project managers [Fallick 2006, Laumer 2011, Owens 2011]. This causes the hiring and reallocation of personnel to projects, either within their company or a client company. Moreover, it implies that software developers and managers often have to face the maintenance of large legacy applications and software projects that they have not supported before. Nevertheless, the maintenance process is usually compromised due to the lack of proper system documentation: it is frequently incomplete, outdated or it is not present [Murphy 1997]. Under these circumstances, programmers and project managers require to understand and comprehend the project at hand, its recent changes and evolution, in a very short term for being able to carry out the most urgent changes or maintenance tasks [Sharif 2009b].

An important consideration is that software development[1] and maintenance[2] are dynamic tasks that conform to the basis of Software Evolution (SE). Changes are made to software projects and those changes are added to the change history of the project and therefore to its evolution. In this process the changes made by a developer to a software item[3] could affect a number of associated software items in which other programmers are working. Furthermore, two or more programmers could also be changing the same software item simultaneously. This increases the problem of understanding a system for someone that have just be assigned to the project, affecting its ability to make changes.

SE is a cyclic process: changes are based on the understanding of the current state of the software project, which is the accumulation of previous changes [Mens 2008]. The change process and the tracking of changes are usually managed with the assistance of a SCM tool.

A SCM tool uses revisions for storing details about changes, such as the author who made the change, the date and time of the change, the project

[1] Software development is the process that involves the design, programming and testing of software systems.

[2] Software maintenance is aimed to correct faults while improving the performance and extends the life cycle of a software system [STA 2010]. Moreover, in the maintenance phase, changes are classified as preventive (detect and correct latent faults), perfective (improve performance or maintainability), adaptive (software functionality improvements or additional requirements) and corrective [STA 2006].

[3] A software item is a source code piece (*e.g.*, a module, file, class or interface).

structure before and after the change, and the source code and the changes that were carried out on it [Estublier 2000][4].

A revision identifies the current state of the project at the moment that the change has been committed. Revisions are stored by a software repository under the control of a SCM tool and are associated to a revision number. Consequently, SE is an iterative process conformed by the accumulation of changes and revisions during software maintenance and development [Fernandez-Ramil 2008].

SE usually expands through several years, generating thousands and even millions of Lines of Source Code (LOC) [Kagdi 2007a], hundreds of software components and thousands of revisions [D'Ambros 2008]. Furthermore, within software projects exist relationships among software items in the form of inheritance, interface implementation, coupling and cohesion. In addition, source code is composed of variables, constants, programming structures, methods and relationships among those elements. Besides logs, communication systems, defect-tracking systems and LOC tools keep records with dates, comments, changes made to software projects and associated users and programmers [Hassan 2005].

Accordingly, SEA requires the assistance of automatic analysis tools for aiding the understanding of a software project. It takes into account the evaluation of individual revisions and the comparison of the output produced by such evaluation for a given number of revisions or all the existing revisions associated to the project. In this context, the analysis on an individual revision includes the comprehension of the structural characteristics of the project, the relationships among software items, the software quality metrics, source code facts, and the comprehension of the socio-technical relationships derived from the development process.

SEA also requires the retrieval of data from the source code, the software project structure, communication systems, logs and the metadata[5] records from bug tracking and SCM tools [Garcıa-Peñalvo 2011]. Thus, it makes use of a set of techniques that have the capability of recovering and analyzing software projects looking to discover patterns and relationships and calculating software quality metrics and fact extraction from the results of comparing the analysis performed on revisions [D'Ambros 2008]. Although,

[4]The IEEE Standard 828-2005 [STA 2005] states that "SCM activities include the identification and establishment of baselines; the review, approval, and control of changes; the tracking and reporting of such changes; the audits and reviews of the evolving software product; and the control of interface documentation and project supplier SCM. SCM is the means through which the integrity and traceability of the software system are recorded, communicated, and controlled during both development and maintenance."

[5]Metadata contains descriptive details about the data.

SEA is powerful for uncovering SE details, it is not capable, per se, of supporting successfully the understanding and comprehension of SE. It still depends upon other techniques and methods, such as visualization and interaction techniques, for supporting successfully software change and maintenance tasks.

1.2.3 Visual Analytics and Software Maintenance

Although the result of analysis of the evolution of software elements provides useful information, it does not provide sufficient information to carry out the tasks of understanding changes in a satisfactory fashion and therefore provide adequate support to software developers and project managers. Therefore, research has taken into account the important role of Information Visualization (IV) in recent years, providing insight from large and complex data sets through visual representations combined with interaction techniques. It is important to highlight that IV takes advantage of the human vision broad bandwidth pathway to the mind, allowing experts see, explore, and understand large amounts of information at once [Therón 2006b].

VA is a process whose goal is to provide insight into the vast amounts of scientific, forensic, academic or business data that are stored in heterogeneous data formats such as databases, HyperText Markup Language (HTML), eXtensible Markup Language (XML) files, metadata and source code. This process iteratively collects information, preprocesses data, carries out statistical analysis [Peck 2011], performs data mining, and uses machine learning [Witten 2005], knowledge representation [van Harmelen 2007], user interaction [Sharp 2011], visual representations [Leung 1994a, Johnson 1991, Robertson 1991], human cognition, perception, exploration and the human abilities for decision making [Keim 2006, Llorá 2006].

VA has been applied comprehensively to problems as diverse as avian flu [Proulx 2006], paleoceanographic conditions [Therón 2006c], organization analysis [Card 2006], eLearning [Gómez-Aguilar 2009, Gómez-Aguilar 2015b], decision making [Migut 2011, Savikhin 2008], ontology engineering [García 2012, García-Peñalvo 2012c, García-Peñalvo 2014], temporal patterns [Weaver 2006, Ziegler 2010], social networks [Perer 2011], security analysis [Harrison 2011] and software systems [Reniers 2012, González-Torres 2013b]. Therefore, one can say that knowledge discovery is an intrinsic property of VA, as it is aimed at supporting analysts and decision makers in gaining insight from large multivariate datasets [Thomas 2005].

Consequently, VA may offer solutions to the problem of supporting programmers and managers during the implementation process, taking into

account the fact that it is a process which offers a comprehensive approach which includes everything from the retrieval of relevant information and analysis to visual representation of the results of the analysis. It offers the potential to explore different levels of detail by using multiple visual representations, coordinated together and supported by the use of interaction techniques [North 2000], thus facilitating the discovery of relationships and knowledge by means of the analytic reasoning of the analyst.

Taking into account these positive factors, it can be said that one of the properties of VA is the ability to provide support for decision making [Savikhin 2008, Mane 2012] using the cognitive abilities of users, and their application to the evolution of software can offer great opportunities to support programmers and project managers. However, the application of VA to the evolution of software is new and the tasks performed by project managers and their information needs are complex [Forsberg 2005, de Oliveira Barros 2004, Munch 2004, Paul 1999].

This implies that there are still a great number of challenges it must overcome in order to successfully support project managers in decision making. Amongst those challenges the following are key:

* To facilitate visual analysis and evaluation of the development process.
* To provide methods to visually monitor the evolution of the quality of software elements (classes, packages and modules), taking into consideration the use of software quality metrics; with the aim of maintaining complexity and project evolution under control as well as assuring quality control.
* To provide visual mechanisms to review the measurements of task execution, and permit progress analysis and performance prediction.
* To assist, using visual methods, risk management, and to control the size and complexity of the software product.
* To keep project managers informed on patterns of collaboration between developers and about those elements which have been modified either synchronously or asynchronously as well as the implications (in terms of quality and functionality) of the changes which have been carried out.

While the challenges facing VA to support programmers in understanding SE, according to their information needs [Sillito 2006b], are:

* Offer details about the software elements upon which their work depends.
* Provide information about the software components that are being simultaneously modified by other programmers.

- Allow programmers to understand the implications of the changes made on the basis of relations (inheritance and interface implementation) and the partnerships between software elements (composition, reference, and coupling), as well as the effect on the collaboration between objects.
- Provide details about the creation of variables, access and modification of data, by means of arguments in the methods, and global variables.
- Enable programmers analyze, and compare for two or more revisions, flow control, execution and exception handling between revisions.
- Facilitate the identification of the differences between files, software elements and types of software in several revisions.

It is worth to recall that SDM [Colomo-Palacios 2012] covers a high percentage of the cost of modern software systems [Koschke 2003]. Program comprehension *tasks* [Koschke 2003] follow precisely the pattern of sensemaking by hypothesis creation, refinement, and validation, common in VA [Sun 2004, Thomas 2005, Thomas 2006]. Finally, program comprehension *tools* rely on the same combination of software analysis [Koschke 2003] and Software Visualization (SV) [Diehl 2007] components.

The application of VA to SE is a recent development, as it was mentioned before. Therefore, a further challenge facing research devoted to studying the application of VA to SE, is to define clearly the process and identification of the factors, methods and techniques which contribute to it.

1.3 Aims and Research Questions

The intention of this research is to define a process to describe and explain the application of VA to SE. The goal is to offer guidance in the design and implementation of software tools to assist programmers and project managers in software development and maintenance. Furthermore, this research anticipates aiding in the communication and understanding of research carried out by other scholars. Accordingly, the principal question posed by this research is the following:

How can a process be adequately defined to describe and explain the application of Visual Analytics to software evolution?

The definition of a process, as stated in the previous question, requires on one hand an explanation of how VA is applied to SE; and on the other hand, the identification of the roles, borders, interactions and relationships of the components, methods and techniques involved in such process. Pursuing this approach, the following are subsidiary research questions which help to adequately explain and describe the process:

1. How do the components that informed the process of applying visual analytics to software evolution interact and interrelate?
2. What is the composition of components in terms of methods and techniques and their roles and interactions in the process of applying visual analytics to software evolution?

The above research questions needed to identify the components, methods and techniques involved in the process of applying VA to SE as well to characterize the roles, relationships and interactions between these elements. Additionally, the utility of this process in the design and implementation of tools must be proved, as a necessary element in answering the main research question posed. On the basis of the above, the following subsidiary research question should be formulated thus:

3. How can it be proved that the description of the process can be followed effectively in order to design and implement an architecture to support the understanding of SE by programmers and project managers?

In order to answer Question 3, an architecture needs to be implemented, based on the process of applying VA to SE that will take into account the problems described in previous sections regarding the needs of project managers and programmers. Therefore, the implementation of this architecture will address the following research questions:

3.1 How can software project managers be supported in their decision making by deepening understanding of changes in software quality metrics, and socio-technical and collaborative relationships during project evolution or a particular time period?

3.2 How can programmers be assisted in their understanding of changes in software quality metrics, software project structures, inheritance and interface implementation for a given time period?

3.3 How can programmers and project managers be supported in their understanding of changes during software project evolution utilizing the comparison of time periods?

Additionally, the architecture will be validated through a user assessment test and use case scenarios. The objective of this validation is to test the complete cycle of applying VA to SE in the design and implementation of a tool to support programmers and project managers in the development and maintenance of software.

1.4 Methodology and Outline

The research methodology used in this thesis is an adaptation of Action Research model [Kemmis 2005] and followed the cycle detailed in Figure 1.1. The phases of this methodology and its correspondence with the organization of this study are shown in table 1.1.

The research model Action Research is a methodology that makes use of iterations to create a series of progressions in order to obtain the solution to a given problem. The goal of each iteration is to refine the solution and takes into consideration collaborative aspects of the work and the participation of the individuals interested in solving a particular problem [Kemmis 2005].

The research discussed in this thesis corresponds to the first cycle to propose models and tools that contribute to supporting in an effective manner the development and maintenance of software by means of the use of VA. When considering Figure 1.1 it can be thus seen that the methodology consists of 5 phases (Plan / Revised plan, Diagnose, Take action, Evaluate and Analyze findings) which when completed leads to a new cycle of the process that starts with a review of the research plan, goals, objectives, questions and research problem. From there on, the following steps are executed taking into account the elements of the previous iteration. Specifically, this research begins with the phase **Plan / Revised plan** that corresponds to the introductory chapter and chapters 2 and 3 (see table 1.1). In this phase, goals, objectives, research questions and research problem are constantly defined and redefined. Similarly, SE concepts, terms and the analysis process as well as the definition of the VA process are revised in each methodology iteration. The goal of chapters 2 and 3 is to define some building blocks which are aimed at contributing the principal research question of this investigation.

The process continues with the **Diagnose** phase which seeks to analyze the research papers which are published and that are related to the application of visualizations and VA to software systems (and their evolution). During this stage, surveys are conducted whose participants are professionals working in the software industry in order to become more fully aware of the current state of use of visualization tools. Subsequently, using the results obtained and with the support of the relevant bibliographical references, a detailed discussion on the state of research in this field and its impact on industry

(taking as a starting point the current use of visualization and VA to support the process of development and maintenance) is carried out.

Figure 1.1 shows the relationship between the **Diagnose** phase and the activities described. It is possible to appreciate the sequence of activities described by means of observation of the dotted blue lines. The goal of this phase (chapters 4,5, 6 and 7) is to identify the tasks supported by research, and the data elements and visualizations in current use by the academy and industry, and therefore diagnose the needs that should be addressed by the characterization of the process of applying VA to SE.

Take action is the next phase of the process. In this phase, the process of applying VA to the evolution of software is defined or redefined. Accordingly, the specification of the architecture as well as the design and implementation of the tool (which uses as a basis the specified architecture) are also defined or redefined. The dotted blue lines in Figure 1.1 show the sequence in which each one of these activities is carried out. Chapter 8 corresponds to this phase.

The next stage is **Evaluate**, which has as aim the indirect assessment of the definition of the process of applying VA to SE. The goal of the specification and implementation of the architecture is to carry out a test of the applicability of the above process. As a consequence, the evaluation and validation of the tool that makes use of this architecture is also the evaluation and validation of the process of applying VA to SE. Chapters 9, 10 and 11 present the results of assessing the tool and is associated with this phase.

The culminating phase of a research cycle is **Analyze findings**. This phase is responsible for analyzing the results of the entire cycle, and for the preparation of the principal conclusions. On the basis of the results of this phase, the plan for the next cycle is redefined. This phase is associated to chapter 12.

1.5 Research lines of this thesis

This dissertation is concerned with SDM and VA and was carried as part of a collaboration between the Interaction and eLearning Research Group (GRIAL) [García-Peñalvo 2012b] and the Visualization Group of the University of Salamanca (VisUSAL). Thus, it has a close relationship with other research works that are framed by such collaboration relationship, and which were focused in supporting eLearning [Gómez-Aguilar 2009, Gómez-Aguilar 2014, Gómez-Aguilar 2015b, Gómez-Aguilar 2015a], ontology engineering [García 2012, García-Peñalvo 2012c, García-Peñalvo 2014], drugs development [Peláez 2008, García 2009a, Pérez 2013] and bioinformatics [Santamaría 2009, Vicente 2010, Santamaría 2014] processes.

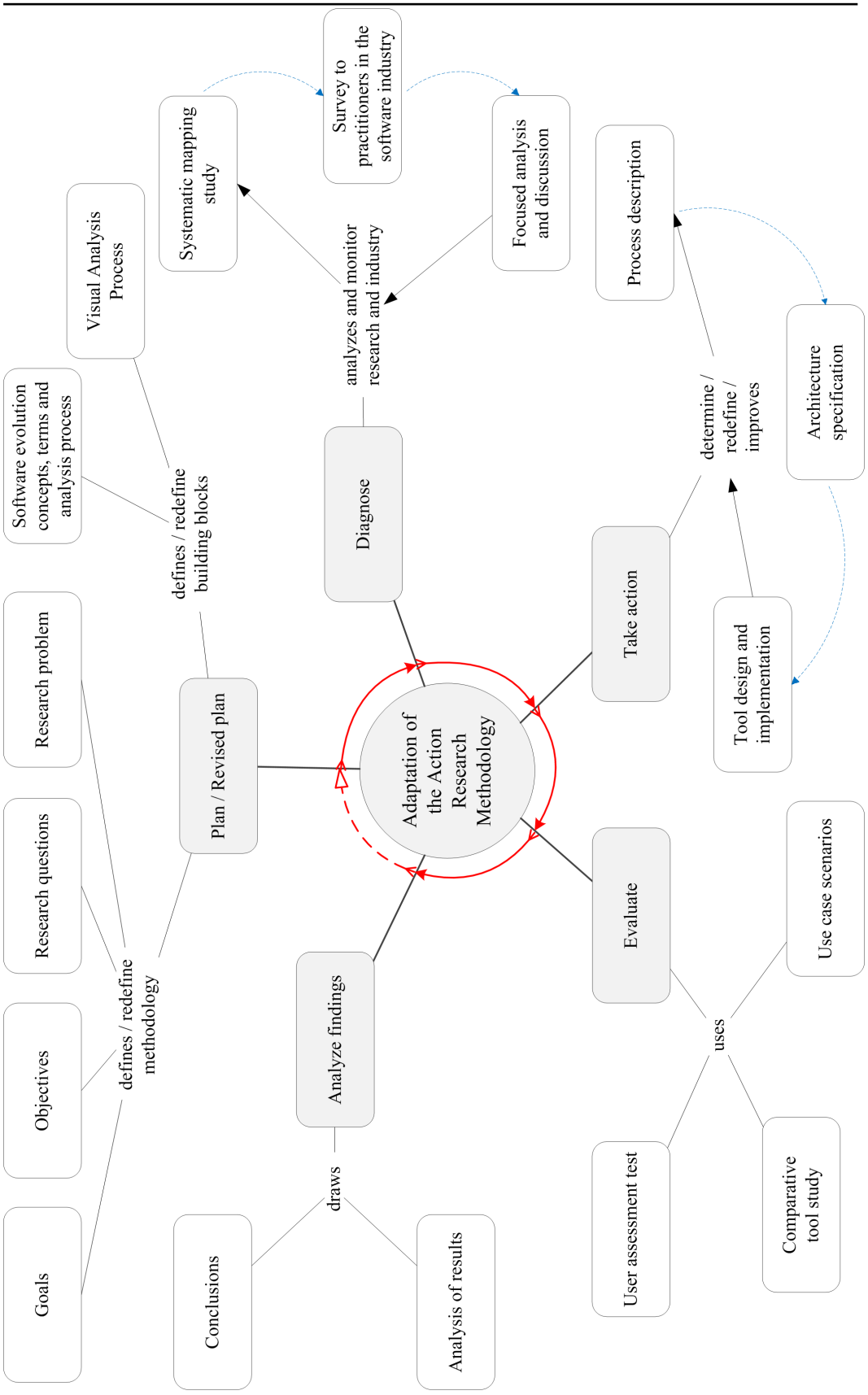


Figure 1.1: Methodology research outline.

Table 1.1: Correlation of the methodology with Chapters and research activities.

Methodology phases	Chapters	Research elements and activities
Plan/Revise plan	Introduction	Goals
		Objectives
		Research questions
		Research problem
	Software Systems: Maintenance and Evolution	Description of the software development and maintenance process
		Discussion of software evolution concepts and details
	Visual Analytics	Description of the visual analytics process and its components
Diagnose	Systematic Mapping Study	Classification of research works
		Identification of tasks, data elements and visualizations in use
		Analysis and discussion
	Survey on the Use of Visual Tools in Software Development and Maintenance	Discussion and analysis of results
	Focused analysis and discussion	Detailed review of selected research works
Take action	A Visual Analytics Process for Software Evolution	Process description
		Architecture specification
		Tool design and implementation
Evaluate	Architecture Validation	Use case scenarios
		Case study
		User assessment test
Analyze findings	Conclusions	Analysis of results
		Conclusions

Part II

Background

Chapter 2

Software Systems: Maintenance and Evolution

La selva era inhóspita, compleja; lenguajes y protocolos peculiares, territorios, vasallos y reyes. Casi todo era alimento de algo o alguien. Todo elemento era una rueda dentada que encajaba con otra. Evolución natural sin reglas aparentes pero implícitas, acatadas por voluntad o impuestas por la fuerza. — El viaje de Güindy, A.González

Contents

2.1 Introduction

The objective of this chapter is to explain the process of analyzing SE and how this may support the development and maintenance of software systems. To accomplish this, some terms, concepts, techniques and methods relevant to

the processes of software development, maintenance and evolution (as well as key aspects in the understanding of changes and their effects) are studied. Moreover, user needs with respect to the comprehension of SE, and the requirements for carrying out the analysis of SE are also identified. This chapter thus seeks to answer the following research question:

How may the analysis of software evolution effectively support software development, maintenance and change tasks?

This chapter therefore explains the software process (section 2.2), and then presents the software maintenance process (section 2.2.5); next it explains the SE process including the information needs of programmers, the comprehension strategies for discerning details on changes, the basics of SEA and SCM (section 2.3) and finally the main conclusions of the chapter (section 3.5) are outlined.

2.2 The Software Process

This section explains and discusses some of the most relevant concepts and elements that concern to this research regarding the software process. Accordingly, it makes an introduction to software development models and presents a variant of the *waterfall model* [Royce 1970] (section 2.2.1); next it explains the iterative nature of the software process (section 2.2.2), then it discusses the current scenario in which software development is carried out in different geographical locations, cultures and timezones (section 2.2.3); after that it carries out a discussion on the role, tasks and skills of project managers and programmers (section 2.2.4) and finally it explains the importance of software maintenance and software system changes 2.2.5.

2.2.1 Software Development Models

The development of software is a complex process that involves a large number of activities [ISO 2014] such as:

1. Analysis of requirements.
2. System specification.
3. High and detailed level design.
4. Programming.
5. Unit testing.
6. System integration.
7. Testing system integration.
8. Training users.

9. Preparing technical documentation.
10. Maintenance and evolution.

However, not all software development models include all of the aforementioned activities. According to Sommerville, although there are many definitions of software development models, activities common to all (with possible variations in names and the combination of several steps into one) are: system specification, design and implementation (activities 3 and 4), validation (activities 5, 6 and 7) and evolution (activity 10) [Sommerville 2011]. Therefore, the *waterfall model* [Royce 1970], which is the best known software development model, also includes the activities common to all software development models (with some variations).

In the context of this research, the *spiral model* [Boehm 1988, Boehm 1999a] is of particular interest as it clearly exhibits the iterative and evolutionary nature of software development and maintenance. However, to explain the process and the components involved during the development and maintenance of software according to the aims of this research a more detailed depiction of the SDM process is required. Accordingly, Figure 2.1 shows a description of the stages and components involved in such process (following the focus of this research), which are explained as follows:

Requirement definition and analysis: This embodies activities such as the gathering, specification and analysis of requirements. Moreover, at this stage the development plan, standards, configuration management plans and software quality assurance are defined.

Preliminary design: It focuses on the tasks of high level design as well as the testing plan.

Detailed design: It is focused on designing the interface, structure and components of the system, the database and the system testing cases.

Implementation and unit testing: During this stage the programming and testing of units is carried out.

Integration and testing: Its goals are to integrate system units and test the functionality of the complete system.

Operation and maintenance: This is the stage where users may require to carry out changes to the system due to additional functionality needs or changes in the requirements [Grubb 2003]. Moreover, other causes could be derived from error detection or preventive maintenance to adapt to changing conditions such as the velocity in the production of data or the opening of new offices.

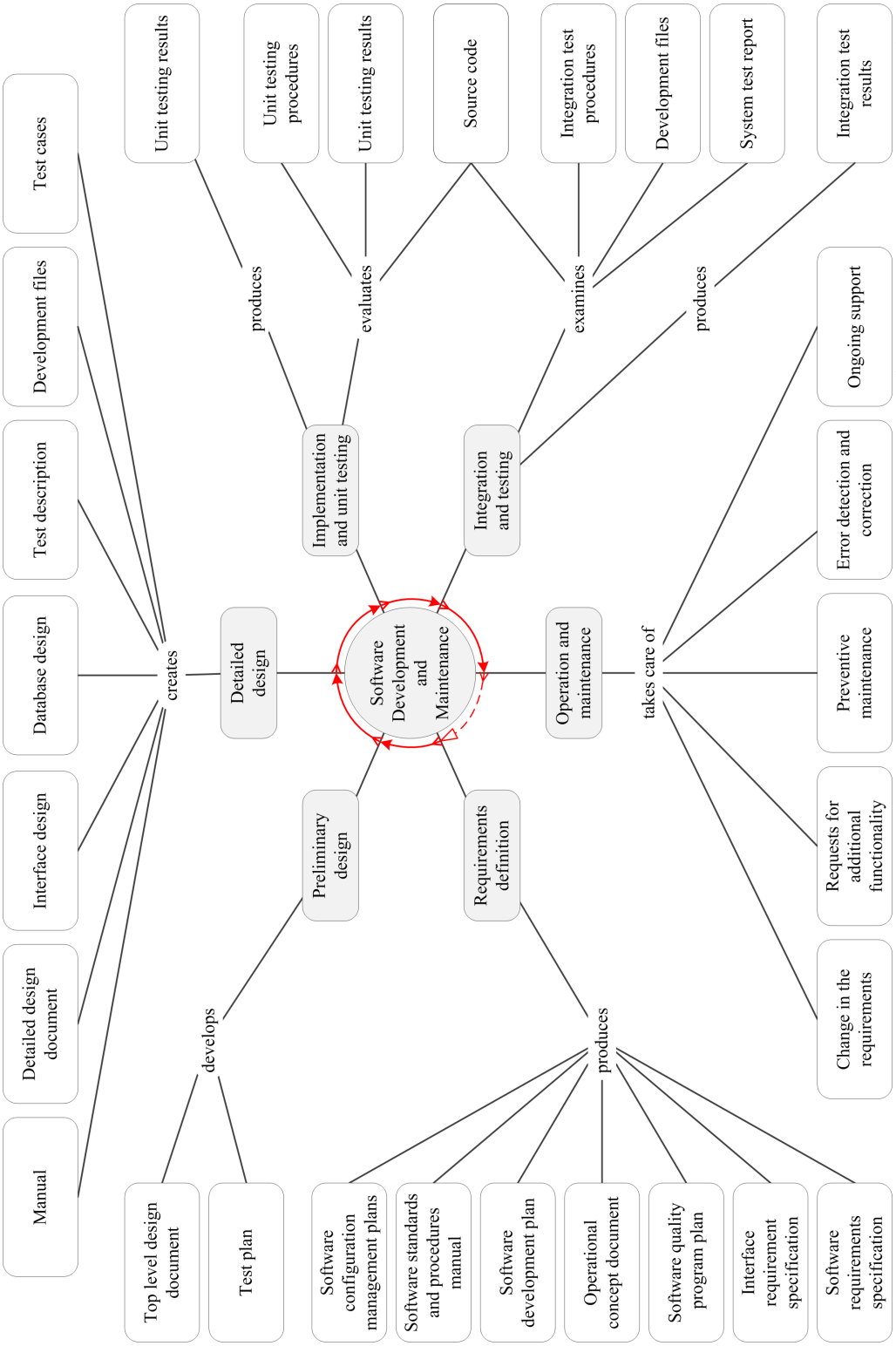


Figure 2.1: A detailed version of the software development and maintenance process (figure prepared by the author).

2.2.2 Iterative Process

The stages listed in the previous section are carried out in the order in which they appear. It is necessary to add that the stages of *Implementation and unit testing*, and *Integration and testing* produce a series of reports detailing the results of unit testing and integration. The results of each stage could be used to correct errors found in the system components and the overall system, so sometimes it becomes necessary to cycle back to one or more of the foregoing stages to update or make corrections in the documentation and source code.

The error correction process and carrying out changes to the system gives place to the evolutionary aspect of software development and maintenance. Therefore one can say that software development and maintenance conforms to the basics of SE.

Depending on the reasons why a change is made to the system, it may be necessary to return to some of the earlier stages. When changes in the requirements for additional functionality occur, such requirements for changes need to be analyzed and then perform the modifications to the whole or a part of the complete system, implement new source code and perform unit testing and system integration. In the case of single error correction may only be necessary to make changes to the code and unit testing and integration. With regard to preventive maintenance, the changes which have to be made may involve several different stages, depending on their size, and could involve the whole process, including the definition of new requirements or merely carry out small changes in implementation. The principal goal is that the system remains capable of being maintained, and thus can evolve.

2.2.3 Global Software Development

In this context it is relevant to remark the current tendency to carry out the development and maintenance of software projects[1] in a distributed form (with members of the development and maintenance teams located in different geographical areas) [Estublier 1999, Herbsleb 2001b, Jiménez 2009, Ogawa 2009], which impedes the fluidity of communication and understanding of the state of the project and the activities which are carried out during its implementation and maintenance [Prikladnicki1 2003, Herbsleb 2003, Omoronyia 2010, Talaei-Khoei 2012].

In this regard, it is worth mentioning that the distances involved are geographical, temporal, and socio-cultural and therefore that

[1] A software project complies with the general definition of any type of project because of its temporary nature with a determinate beginning and end (when the objectives are reached or there is no longer any need for the project [PMI 2002].

the challenges that need to be overcome involve problems of communication, coordination and control [Carmel 1999, Herbsleb 2001a, Conchúir 2009, Misra 2013, Colomo-Palacios 2012]. This requires the use of effective mechanisms [Carmel 1999, Mockus 2001, Prikladnicki1 2003, Colomo-Palacios 2014]:

* Documentation systems (requirements, specification, design and manuals).
* Appropriate development methodologies.
* Email.
* Telephone.
* Instant messengers.
* Video conference facilities.
* Collaborative technology and tools for sharing details of the activities carried out by team members.
* Team building strategies.

In this context, another factor that is present is the type of organization used by GSD teams (independent of the organizational structure [Mintzberg 1991] used by the company), the following being the most common:

* Virtual teams [Karolak 1999, Carmel 1999].
* Coherent and collocated teams of fully allocated engineers [Ebert 2001a, Ebert 2001b].
* Loosely coupled teams [Herbsleb 2001a, Pinelle 2005].
* Scattered software development [Vrhoveca 2013].

2.2.4 The Role of Project Managers and Programmers

Software development teams include personnel working the following roles: project managers, analysts, designers, programmers, testers and manual and documentation writers. However, this research takes into account all these roles it usually makes reference to Project Managers (PMs) and programmers as the former are the responsible of the success of software projects in administrative terms and the latter are central to the software process as they are in charge of performing the development, maintenance and evolution of software systems and thus the recipients to which the designs, test results and change requests are delivered in order for them to carry out the appropriate actions. The following two sections thus discuss the tasks that these people perform and the capacities which are required from them [Cegielski 2006] to successfully carry out their duties.

Tasks and skills of PMs: According to Thayer project management is "a system of procedures, practices, technologies, and know-how[2] that provides the planning, organizing, staffing, directing, and controlling necessary to successfully manage an engineering project" [Thayer 1988]. Accordingly (and taking into account that the term know-how is referred to the use of best practices in carrying out human activities) the aforementioned tasks must be performed by adequately skilled and trained individuals [Sarewitz 2008]. Thus, PMs are the individuals who have the proper training and skills to carry out the project management tasks. Consequently, the abilities of PMs, specially when working in GSDs [Saldaña-Ramos 2014], should include:

* Team coordination.
* Management skills.
* Project monitoring and tracking.
* Result evaluation abilities.

Moreover, some desirable soft skills of PMs [Sukhoo 2005] are:

* Communication skills.
* Team building.
* Flexibility and creativity.
* Leadership.
* Organizational effectiveness.
* Stress management.
* Time management.
* Change management.
* Trustworthiness.
* Conflict management.

Tasks and skills of programmers: Software programmers are present in nearly al the stages of the software process: programming, testing, system integration, training users, preparing user and technical documentation, and maintenance and evolution. Therefore, their role is central to the software process. Some of the most common tasks [Ko 2007] for which they are responsible are the following:

* Writing new source code.
* Testing and debugging source code.
* Carrying out system changes.
* Reproducing failures and fixing system bugs.

[2]The Merriam-Webster dictionary defines know-how as the "knowledge of how to do something smoothly and efficiently" [Merriam-Webster Online 2009].

* Understand execution behavior.
* Maintaining and updating source code.
* Preparing user and technical documentation.
* Training users.

Consequently, the set of skills and knowledge that software programmers require to carry out the tasks within their area of responsibility [Tockey 1999] may include the following:

* Programming language concepts.
* Database system concepts.
* Software architectures.
* Requirements analysis.
* Software design.
* Code optimization.
* Debugging techniques.
* Software project audits.
* Software testing techniques.
* Customer support techniques.
* Abilities for writing user and technical documentation.
* Configuration management.
* Software quality assurance and metrics.
* Effective communication skills.

Moreover, a study conducted by Turley [Turley 1995] found that the following competencies are attributes of exceptional software engineers:

* Mastery of skills and techniques.
* Maintains a *big picture* view.
* Desire to do/bias for action.
* Driven by a sense of mission.
* Help others.

In the next section we discuss in more detail the process of software maintenance and its implications for the evolution of software systems.

2.2.5 Software Maintenance

The operation of many organizations depends on the proper functioning of their systems and therefore the maintenance of these is a critical element [Bennett 2000]. It is thus important the fact that software maintenance usually takes place over several years, according to project size

and the role played in the organization, and demands more resources than the implementation phase [Grubb 2003]. This makes it likely that during maintenance multiple challenges will have to be confronted in order for it to be carried out successfully. Some of these challenges are caused by factors which are external to the organization while others are caused by internal factors. The following are some of the most common challenges:

Frequent job-hoping: The software industry is very competitive and many companies are engaged in attracting and hiring the most talented employees in the market, thus job-hoping of software developers and managers between companies is frequent and mostly motivated by better job offers from the competing companies [Fallick 2006, Laumer 2011, Owens 2011].

Frequent reallocation of personnel: It is common that programmers are assigned to other projects, either within their own company or a client company, according to organizational needs and priorities.

Maintenance of large legacy systems: Programmers and managers frequently face the maintenance of large legacy applications and software projects that they have not supported before.

Lack of proper system documentation: The maintenance process is usually compromised because system documentation is frequently incomplete, outdated or it is not present [Murphy 1997].

It is noteworthy in this context that software maintenance[3] is a dynamic process which begins from the moment in which a software system is conceived and initiated.

The tasks performed by project managers, designers, architects and programmers look to satisfy the organizational needs and requirements by making preventive (for improving source code quality), adaptive (due to software functionality improvements or additional requirements) and corrective changes[4] to the software system. Change management [STA 2005] is, therefore particularly important and is opportune to take into account Figure 2.2. Change management includes the following list of tasks:

Change requests: These must specify the type of change (corrective, adaptive or perfective), its description, as well as justify the necessity and urgency in carrying a change.

[3] According to the ISO/IEC/IEEE 24765 standard software maintenance is "the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment" [STA 2010].

[4] The ISO/IEC/IEEE 24765 standard defines a change as "the modification of an existing application comprising additions, changes and deletions" [STA 2010]

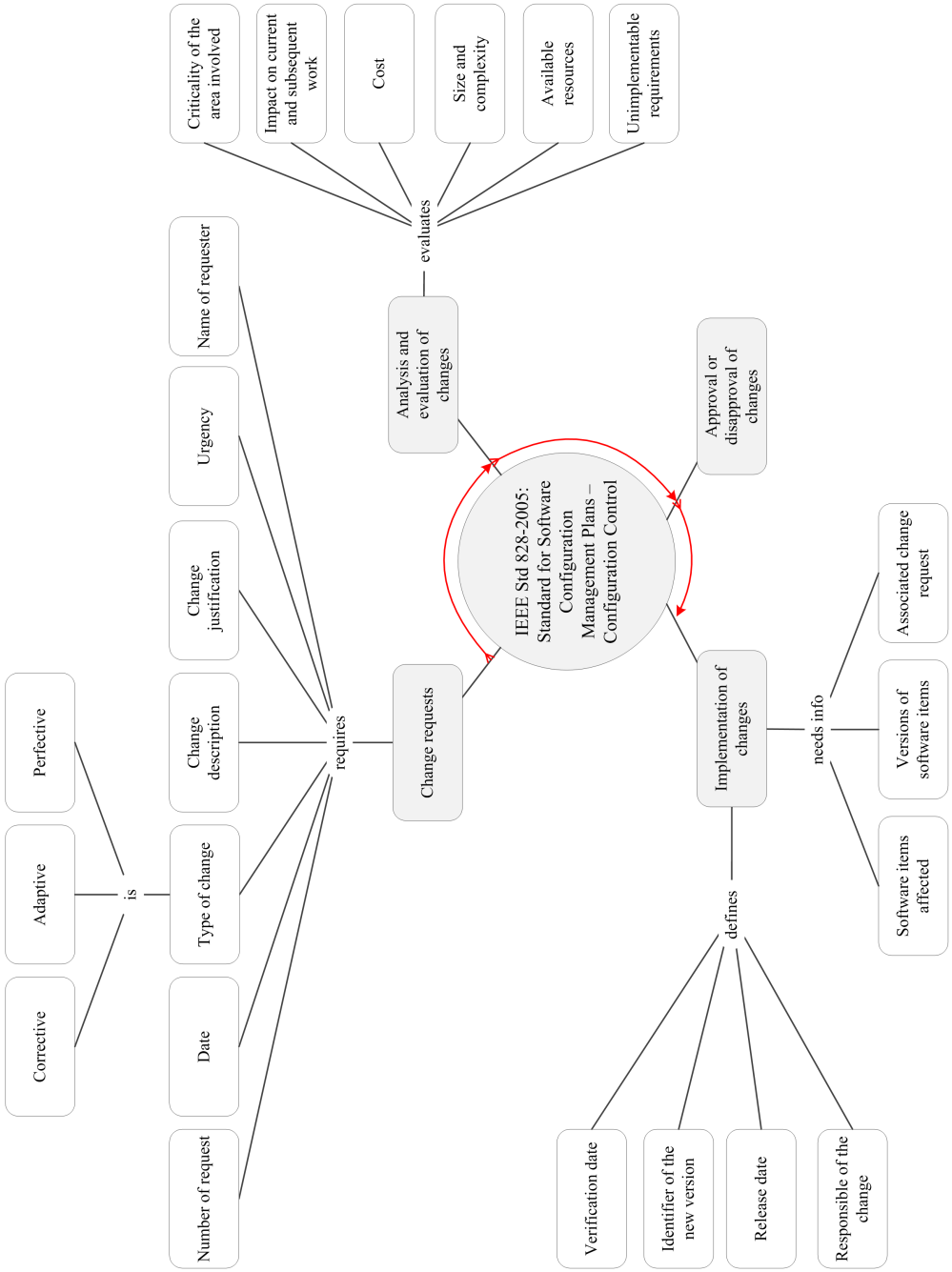


Figure 2.2: Management of changes in software maintenance [STA 2005] (figure prepared by the author).

Analysis and evaluation of changes: It consists in evaluating change requests taking into account if the change is implementable, the criticality of the area involved, the impact on current and subsequent work, the size and complexity of the change, the available resources and the projected cost of the change.

Approval or disapproval of changes: The request for changes is approved or disapproved according to the analysis and evaluation carried out and the organization priorities.

Implementation of changes: This is responsible for determining which software items and the various versions that will be affected by the changes. During this task one or more persons must be assigned to make the change, the date by which it is expected that the change will be completed must be specified, as must the identifier of the new version which will originate the change as well as its date of verification.

In short, once a request for change has been made is analyzed and evaluated, then based on the result of that evaluation the process will proceed to its approval or disapproval. If the change is approved the process continues to the implementation phase, otherwise the request is filed. Taking into account the challenges mentioned above and the management of changes, programmers and project managers require to understand and comprehend the current project and its evolution in a very short time in order to carry out the most urgent maintenance tasks. This involves an understanding of the accumulation of changes, from the last point at which accurate documentation was available and may involve hundreds of software components as well a necessity to clarify the relationships between them in the form of inheritance, interface implementation, coupling and cohesion. Consequently, the understanding of changes and thus the evolution of software projects is a crucial task for software maintenance.

Accordingly, the following section reviews related concepts in SE and analysis that are necessary to take into consideration in order to effectively support the process of understanding software systems during the performance of task change.

2.3 Software Evolution

SE describes the process of software change and improvement over years and releases [Bennett 2000]. This produces vast amounts of details which is frequently collected using automated mechanisms that report the changes made and the tasks carried out by means of Integrated Development

Environment (IDE), SCM, defect-tracking and system testing tools. The collected data is stored in source control repositories, bug repositories, archived communications, testing logs and deployment logs [D'Ambros 2009a]. However, it is often true that most of the tools and storage mechanisms mentioned above are not fully integrated and the recording and access to data has to be performed on an individual basis.

SEA is concerned with aiding the understanding of software changes: their causes and effects [D'Ambros 2008]. Its principal objectives are to provide information which contributes to the maintainability of the project thus supporting its improvement through the analysis of continuous, increasing complexity, continuous growth but causing quality to decline [Lehman 1997]. The goal, which is to allow the implementation of the appropriate actions to make additional changes and whether monitor the quality or functionality of the software project is compromised in the short or long term. In summary, it seeks to support new changes because modifications are based on the understanding of the current state of the software project, which is the accumulation of previous changes made by the software development or maintenance activity.

The main techniques that are commonly used to understand a software project for a particular revision are outlined. Following this, the process of automatic analysis, the main elements involved in it, and their relationship to SEA are explored.

Consequently, section 2.3.1 describes the main concepts of SCM and how it is used to collect data from the software development and maintenance processes, next section 2.3.2 discusses the information needs of software project managers and programmers, and finally Section 2.3.3 is concerned with the process of software evolution analysis.

2.3.1 Software Configuration Management

SCM plays an important role because it controls the evolution of complex systems [Estublier 2000] or in a more detailed fashion: it is the process that manages the evolution of a software project, taking into account all the levels of communication in the organization and including all modifications that programmers have made to the code. Figure 2.3 shows the activities of SCM plans according to the standard *IEEE 828-2005* and its activities are the following:

1. **Configuration identification** involves the identification and naming of the configuration items to be controlled, as well as the repository where they will be stored.

Figure 2.3: IEEE Standard for Software Configuration Management plans [STA 2005] (figure prepared by the author).

2. **Configuration control** requires to take into account change requests, the analysis and evaluation of requests, the approval or disapproval of the requests and the implementation of changes if these are approved.

3. **Configuration status accounting** defines how information is managed, from its collection to storage and protection, as well as the reporting of the status of configuration items.

4. **Configuration evaluations and reviews** determines how configuration items are managed and identify deficiencies and report corrective actions.

5. **Interface control** coordinate changes between items under management and items outside the management scope.

6. **Subcontrator/contractor vendor control** incorporates into the management scope those items or software that have been developed by contractors. It involves their monitoring, evaluation and acceptance.

7. **Release management and delivery** controls the build, release and delivery of software products.

Consequently, SCM systems must provide services for managing the software repository and assistance to the configuration control process. According to the definition of the *IEEE Standard 828-2005* [STA 2005] SCM systems "provide methods and tools to identify and control the software throughout its development and use". Moreover, the same standard states that SCM activities include "the identification and establishment of baselines; the review, approval, and control of changes; the tracking and reporting of such changes; the audits and reviews of the evolving software product; and the control of interface documentation and project supplier".

SCM systems use distributed or client-server architectures, and the latter are the most common of the two nowadays.

Tools with client-server architectures provide each programmer with his own workspace and repository into which software items are copied when she checks elements out[5] from the software repository to be modified. After the changes have been made, the software item is copied back to the software repository and the check-in[6] operation is finalized. Where more than one member of the team has worked on the same component, the combination is performed on the copies of the component using the mechanism files copy-modify-merge.

[5] A check-out indicates that a modification to a file is going to be performed, so that it can make a copy and deliver it to the user.

[6] A check-in is executed to copy files into the repository to produce a new version.

Alternatively, distributed software configuration architectures provide the developer with a work environment with a complete repository of components. So, when changes are made to components, these are copied from one repository to the others [Estublier 2000]. Table 2.1 shows a summary of the most popular open source and commercial SCM systems using client-server or distributed architectures. Additionally, it is valuable to mention that the selection of the right tool for individual needs could be assisted by frameworks and methodologies that have been proposed previously, such as the one proposed by Kilpi [Kilpi 1997].

Table 2.1: SCM systems and architectures.

Tool	Architecture	Open source/ commercial
Concurrent Versions Systems (CVS)	Client-server	Open source
CVSNT	Client-server	Open source
OpenCVS	Client-server	Open source
Subversion	Client-server	Open source
Vesta	Client-server	Open source
AccuRev	Client-server	Commercial
Aldon	Client-server	Commercial
AVS	Client-server	Commercial
Accrue	Client-server	Commercial
Alien brain	Client-server	Commercial
ClearCase	Client-server	Commercial
CollabNet Subversion	Client-server	Commercial
Perforce	Client-server	Commercial
Plastic SCM	Client-server	Commercial
Polarion	Client-server	Commercial
StarTeam	Client-server	Commercial
Telelogic Synergy	Client-server	Commercial
Team Foundation Server (TFS)	Client-server	Commercial
Aegis	Distributed	Open source
ARCS	Distributed	Open source
Bazzar	Distributed	Open source
Codeville	Distributed	Open source
Darcs	Distributed	Open source
Git	Distributed	Open source
Mercurial	Distributed	Open source
Monotone	Distributed	Open source
SVK	Distributed	Open source
BitKeeper	Distributed	Commercial
Code Co-op	Distributed	Commercial
TeamWare	Distributed	Commercial
Wandisco	Distributed	Commercial

Moreover, SCM tools have traditionally been used to record changes in software repositories; including time, date, affected modules, how long the modification took and information about who performed the change [Estublier 2000].

The structure of software repositories vary from system to system: some use relational databases whereas others use a file system. Two examples of the latter are *SVK* and *Subversion* both of whom use a file system with a tree structure. In these software repositories all the changes carried out to each file and folder within the structure are recorded. Furthermore, each time SCM tools read information from the software repository, they read the latest stored revision, but they also allow access to all the previous revisions of the source code base. In this structure the repository stores information from multiple projects and each project is a subdirectory in the file system which means that when the developer checks-out code, a copy is made of the subdirectory of the project on which she was working in her workspace. Each time the user updates the structure and sends the changes back to the repository a new revision of the repository is created, in effect, that's to say, a new *snapshot* of the state of the repository is produced. Figure 2.4 shows an example of the visualization of the repository used by *Subversion* [Collins-Sussman 2004]. In the case of the tools mentioned earlier, as in many others, the management of revisions is an atomic process in which the revision number is changed even if only one file has been modified.

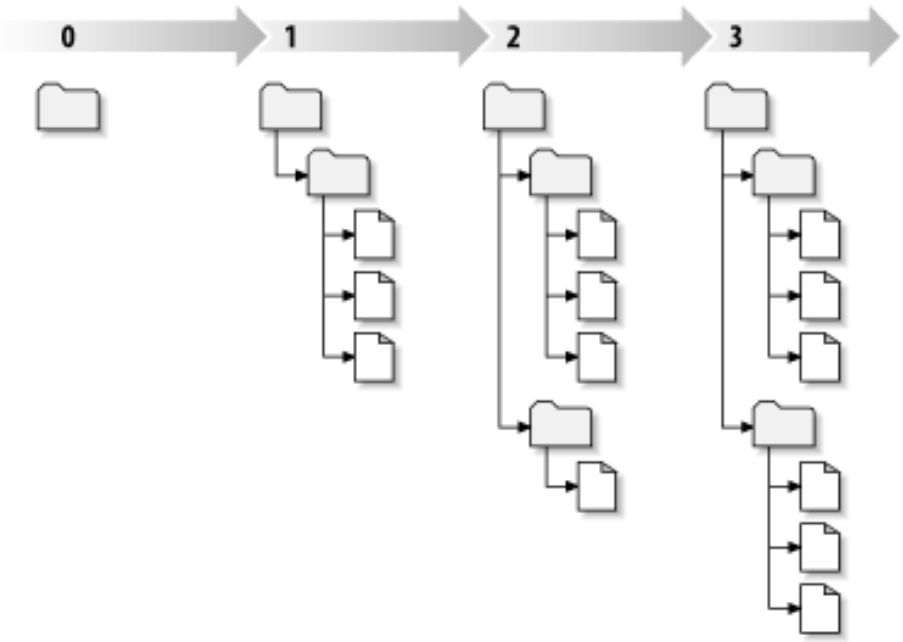


Figure 2.4: Source code snapshots stored in the software repository [Collins-Sussman 2004].

SCM systems provide a means of collaboration among software developers, support the developer's workspace and manage the collaboration among multiple users who are trying to make changes to the same software item. The most common methods for managing changes and the collaboration among team members are:

1. **Block simultaneous changes to the same software item**. The software item is blocked by a programmer who wishes to make changes. Then, the software item is unblocked once it has been modified.

2. **Allow simultaneous changes to the same software item**. The software item is changed simultaneously by several developers and the changes are subsequently combined. If there is no conflict among the changes, the tool combines them automatically, but if a conflict appears, the tool will ask programmers to resolve the conflict.

The above methods are referred to the terminology coined by Collins-Sussman *et al.* as *lock-modify-unlock* and *copy-modify-merge*. The former method consists in blocking the software item, modifying it, and then unblocking it, whereas the latter consists in copying the software item, modifying it, and then combining the changes made by another programmer.

Figure 2.5 shows the activities of SCM systems. The activities that are highlighted with dotted blue lines are activities that are performed according to the decisions taken by the programmer for a particular situation or according to the functionality of the specific system used. Finally, table 2.2 lists the key terms (and a brief explanation of these) which are used by the process and SCM systems.

2.3.2 Information Needs of Software Project Managers and Programmers

In this section it is assumed that in a large number of cases only source code (which lacks supportive documentation) is made available to programmers and managers. Therefore, there is a high likelihood that these individuals could be confronted with three problems that the scope of this research takes into account:

1. Old applications that require maintenance.
2. Being hired to do maintenance on another company's applications.
3. Being hired by a company and thus having the need to understand the existing applications for carrying out maintenance.

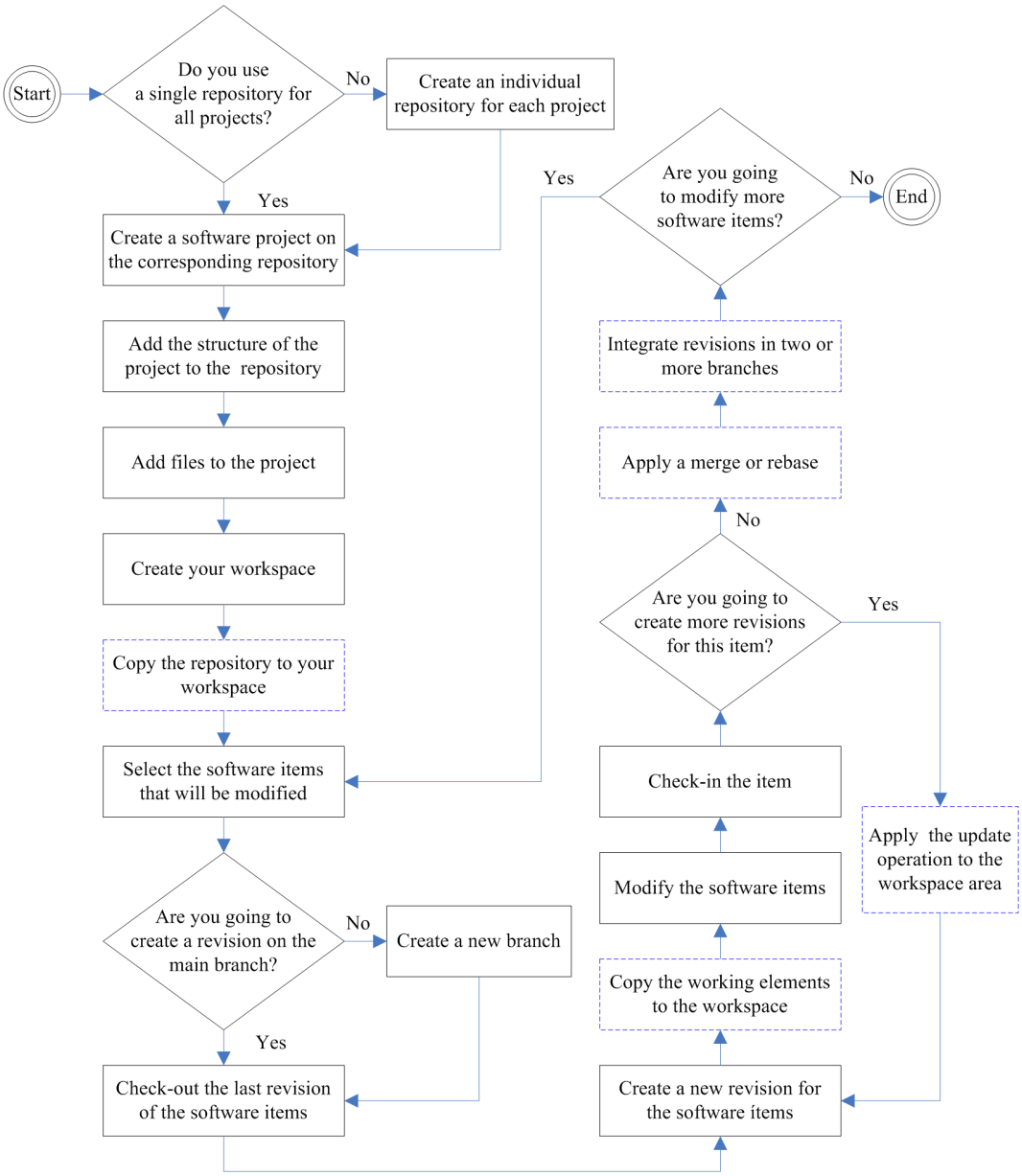


Figure 2.5: Flowchart of activities involved in the operation of SCM systems (figure prepared by the author).

Table 2.2: Terms and concepts used in Software Configuration Management.

Term	Concept
Version	A system version number is an identifier for a group of elements that have been changed successively during several revisions.
Revision	It is a minor change which is made to the system or software items that are identified by a number in the form X.Y where Y is the revision number and X is the version number.

Continued on next page..

Table 2.2 Terms and concepts – continued from previous page.

Term	Concept
Baseline	A baseline is used in a manner analogous to a version, with the difference that it is applied to changes in the project structure.
Repository	It is a data structure that stores the details of the evolution of projects and software items.
Project	A project is comprised by the source code of a program and related documents to the requirements, design, development and testing.
Workspace	This is the local working area that stores programmer software items and project documents during the process of development and maintenance.
Configuration item	It refers to a basic component of software or software item, as a class, interface or document managed by the SCM system.
Check-out	This is an operation that allows software items to be copied from the repository to the programmer's workspace, at the same time the attributes are changed from read-only to modify, in order to create a new revision.
Check-in	This operation sends updates from a group of software items to the software repository and converts the revision which is generated when a checkout is made into a regular revision.
Branch	A branch is used to create editions of a program and to make changes concurrently: each developer works in his own branch and then integrates it into the main branch.
Main branch	This is the main branch of the evolution of the project. It integrates all branches or revisions of the evolution of the project and serves as the main reference of the development.
Update	Is the update of the working areas with the changes carried out by other programmers and takes place at the programmer's request in order to allow the latest modifications to be taken into account.
Merge	This is the fusion of multiple revisions of a software item that has been modified in parallel by several programmers.
Integration	It is the merger of two branches in which there have been several revisions of a group of software items.
Rebase	This is a very similar operation to merge, only that the revision history is not associated with several branches but rather one branch as a linear succession of revisions.

Accordingly, the next sections describe some common information needs of programmers, project managers and both groups when these are confronted with the problems mentioned above or with maintenance tasks that needs to be carried under normal circumstances. Moreover, the strategies to understand software systems using a manual approach are also discussed.

2.3.2.1 Information Needs of Project Managers

The information needs of PMs are diverse [Jedlitschka 2009] and evolve as the overall software process is scrutinized and controlled by them. PMs need

information to understand high level details of the project in order to take better quality decisions.

In the light of the above, programmers must cooperate and coordinate their activities with one another in order to better understand the activities which are carried out in order to be aware of the general state of progress of the software project. Cooperation and coordination are essential to the process of development and maintenance of software [Omoronyia 2010, Talaei-Khoei 2012]. With regard to this subject, there are several studies that indicate that adequate information is an essential requirement for those who participate in software projects. The primary information that participants require comes from the need to be fully aware of what the other people involved in the project are doing [Ko 2007, Kim 2011]. This is true in the case of both collocated and GSD settings, but is more relevant in the latter case because of geographical, cultural and time differences.

Additionally, the execution of these projects which use either of the aforementioned approaches (whether collocated or GSDs [Colomo-Palacios 2012]) requires that the changes which are made are both informed as well as adequately and continuously measured. With regard to the continuous measurement of the development process, it is worth mentioning that the results of the study conducted by Buse suggest that PMs consider metrics to be the most important factor in decision making and monitoring of the evolution of a project; above all with regard to quality control. Furthermore, the same research suggests that PMs may have become aware of the potential benefits of using analytical tools and they would therefore be more willing to use them if these tools can satisfy their requirements [Buse 2012].

Taking into account the current trends concerning GSD [Jiménez 2009] (outlined in Section 2.2.3) as well as the results of the study conducted by Buse [Buse 2012]; this research is of particular interest with regard to the information needs of PMs whose concerns are the following: the efficient management of human resources [Misra 2013]; collaboration between team members [Rodríguez 2004, Servant 2010]; evolution of quality control; task assignment [Predonzani 1998]; and the necessity to remain informed of all the activities carried out by team members (awareness of activities carried out). The following topics are thus of particular interest to project managers:

1. Information about the programmers that were championing the software project in its early stages or at some point in the project evolution.
2. The programmers that have left the software project or company.
3. The names of the programmers who have worked on the software project, a module or a software item derived from source code changes.

4. The correlation of programmers and software items to determine who is responsible of making changes in which software items.
5. The collaboration network that is formed by programmers which can be inferred from the software items that have changed in common, and in particular, which programmers have changed the most software items in common.
6. The programmers that have committed most revisions or created most files and software items during the evolution of the software project or over a particular period of time.

Additionally, other topics concerning the information needs of PMs are discussed in Section 2.3.2.3, as well as those that are of interest to programmers.

2.3.2.2 Information Needs of Programmers

It is of great importance to understand the effects of the changes and meet the specific needs of developers in order to enhance their effectiveness [Xu 2009, Tao 2012]. In this respect, Sillito carried out two research studies to thoroughly obtain details of programmer's information needs when they carry out tasks of change [Sillito 2006a, Sillito 2006c, Sillito 2008]. Thus, these studies were aimed at investigating on the questions that programmers usually ask while they work in change tasks. One of the studies was conducted with programmers who were new to a given software project whereas the other study was carried out with programmers that were familiar with the software project they were given. The first group of programmers worked for the study in a laboratory and the latter group in software industry settings. The questions posed by participants were group into 5 set of questions, which are summarized below:

Finding focus points: This group is composed of 5 kinds of questions that were posed by participants with little or no previous knowledge about the project. The questions were general and sought to find where errors were located or a software item was located.

Expanding focus points: The questions in this group (15 kinds of questions) were posed by participants that had some previous knowledge of the project and were interested in finding more information relevant to carrying out a change task. These questions focused on more specific details such as class hierarchy, siblings location, composition, interface implementation, class instantiation, access to variables, and method calling and arguments.

Understanding a subgraph: The questions in this group (13 kinds of questions) were concerned with building an understanding of the software items involved in the change task and their relationships. The goal of these questions was to determine how objects are created, assembled and related, as well as their behavior, how control is passed from one point to other and how data structures are passed and accessed at different points in the code.

Questions over groups of subgraphs: These questions (11 kinds) were asked by participants interested in understanding the interaction of the subgraph, described in the previous question, and the rest of the system. The aim of the questions in this group is to get insight on the direct and indirect impact of a change and whether after carrying it out the problem is completely solved or additional changes are needed.

Specifically, when programmers make a change or study the evolution of a software project (correlating two or more revisions) are interested in obtaining information about the following aspects [Ko 2007, Sillito 2008, Tao 2012, Buse 2012]:

* Inheritance relationships.
* Siblings in the hierarchy.
* Interface implementation.
* Composition.
* Architecture and project structure changes.
* Reference relationships.
* Associations.
* Objects collaboration.
* Instances creation and access.
* Data access (methods arguments, variables and data structures).
* Data flow.
* Control flow, execution and exception handling.
* Source code differencing and comparison.
* Change ripple effects.
* Code cloning[7].
* Direct, indirect and logical coupling.
* Program execution.
* Design patters in use.
* Details about the complexity of the software items and the overall project.

[7] A clone is a segment of code that has been created through duplication of another piece of code.

* The cohesion and coupling between software items, measured with the use of metrics.

It is also necessary to be aware of the dynamic behavior of the software items and the program during its execution. Thus, programmers need to get information about the effects produced by the changes in other software items and other elements of the overall program.

2.3.2.3 Information Needs Common to Project Managers and Programmers

It is important to highlight that while the needs of project managers and developers may differ, both groups require methods and tools that enable them to compare and correlate the evolution of the software project and derived metrics. Taking this into account, the following list presents some of the metrics that have received special attention from both groups, programmers and managers:

* Code smells[8] [Lanza 2005b].
* Complexity measures [McCabe 1976].
* Maintainability measures [Aggarwal 2005, Heitlager 2007].
* Evolution metrics [Lehman 1997, Mens 2001].
* Size measures [Lanza 2005b].
* Coupling: Direct Coupling Between Objects (CBO) [Chidamber 1994], indirect and logical [Gall 2003, German 2006, D'Ambros 2009b].
* Lack of Cohesion Metric (LCOM) [Chidamber 1994].
* Depth of Inheritance Tree (DIT) [Sheldon 2002].

Additionally, both developers and project managers are interested in details about the activities carried out on the project or software items [Kim 2011] and therefore these details are of interest to both groups:

* The number of revisions associated to a software project.
* The time periods during the project evolution or other time period under study which exhibit an abnormal level of activity due to a dramatic increase or decrease in the number of commits.
* The time periods where the activity level demonstrates that the project has become stable.
* The revision that does not adjust to the general pattern in terms of the average of files created or updated.

[8] Code smells are characteristics of software that indicate that code may have a design problem. These have been proposed as a way for programmers to recognize the need for restructuring their software.

* The baseline that has most revisions associated with it.
* The branch that have most commits been carried out.

2.3.3 Software Evolution Analysis

The process of understanding the evolution of a software project can be carried out using a comprehensive or partial approach, according to user's profile (software project manager or programmer) and the task she is performing. Software project managers are interested in the comprehensive approach for obtaining information to support decision making and assuring software quality. Basically, they try to understand the overall evolution of the project through the tracking of project advances and quality measurement, the socio-technical relationships and the collaboration among software developers. Software developers, on the other hand, are usually most interested in using the partial approach. Their main concern is understanding only a small number of the revisions to track recent changes and make modifications.

SEA requires data collected by adequate logging tools, such as IDEs using plugins, SCMs systems, defect-tracking and system testing tools, during the process of development and maintenance, and the mechanisms that these tools provide to extract details of software repositories, databases and the logs they use.

The analysis of the evolution of software, which can be static or dynamic, consists in analyzing two or more versions of a software project individually and then carry out a comparative analysis of the results. Such analysis can be performed on one or more levels of granularity: source code line, class, file, package or module and system level. Overall, one or more of the methods listed in table 2.3 can be considered to carry out such analysis which may produce results as those listed in table 2.4.

In this context, the process of extracting and analyzing data from software repositories (known as software repository mining) has been an active research area for several years [Kagdi 2007a]. Software repository mining focuses on extracting details from metadata and source code for analyzing software changes. It analyzes the dissimilarities between revisions and changes to artifacts on different granularity levels, such as classes and methods [Kagdi 2007b]. The results of software repository mining are fed into SEA. At this point it is relevant to highlight the work of Kagdi [Kagdi 2007a], which is a detailed survey of the literature on the purpose of mining software repositories, the methodology employed in the mining process and the evaluation of the mining approaches currently in use. Hassan [Hassan 2005, Hassan 2006] also discusses in depth the extraction of information from software repositories to assist developers and support

managers.

Table 2.3: Correlation of references with analysis methods.

Method	References
Static source code analysis	[Jackson 2000, Robillard 2003]
Metadata analysis	[Kagdi 2007a, Gethers 2012]
Execution and system traces analysis	[Fischer 2005, Gethers 2012]
Software metrics extraction	[Lincke 2008, Vasa 2009]
Information retrieval methods	[Baysal 2007, Gethers 2012]
Social network analysis	[Scacchi 2004, Ducheneaut 2005, Sack 2006]
Association rules	[Chawla 2003, Morisaki 2007]
Analysis of linked data sources	[Keivanloo 2011]
Program and architectural slicing	[Weiser 1981, Agrawal 1990, Hassine 2005]
Origin analysis and refactoring	[Zou 2003, Godfrey 2005, Green 2011]

Therefore SEA has a closed relationship with software repository mining as it is aimed at identifying relevant facts that could be used to provide support to software developers and managers in the discovery and comprehension of evolution details. It looks to support software developers and manager tasks such as process improvement, fault prediction, productivity estimation, comparing the actual and desired architectures of a product, and planning future development activities. This type of analysis supports project managers in decision-making, which is affected by factors such as the dynamics of software quality measured by quality metrics; the need to control the contribution frequency and contribution patterns of programmers to software projects for team and productivity assessments; and reporting activities to upper management.

2.4 Conclusions

Programmers and PMs are frequently confronted with projects for which little or no adequate documentation is available and about which they have no little or no prior knowledge. Moreover, as well as problems related to fully understanding the projects when they are initially confronted with them, they also need to understand the effects of changes which are made later during the processes of change and maintenance [Benestad 2009]. The aim is to make the changes which are necessary and that the project continues to evolve while it is still subject to maintenance or capable of being maintained. It is thus of great value to be aware of the information needs of project managers and

programmers during the process of evolution, which involves understanding the project and the effects of the changes which are made to one or more of the revisions.

Table 2.4: Correlation of references with analysis results.

Analysis results	References
Defect classification	[Chillarege 1992]
Contributions and socio-technical relationships	[Scacchi 2004, Ducheneaut 2005, Baysal 2007]
Software prediction models	[Fenton 2000, Goulão 2012]
Metrics and evolution metrics	[Fenton 2000, Mens 2001, Meyers 2007] [Vasa 2009]
Change classification and analysis	[Bohner 2002, Hassine 2005, Gethers 2012]
Software item lifelines	[Zou 2003, Godfrey 2005]
Dependencies	[Bohner 2002]
Architecture/structure changes	[Tu 1992, Zhao 2002, LaMantia 2008]
Exception structures	[Robillard 2003]
Direct, indirect and logical coupling	[Hitz 1995, Gall 1998, Gall 2003, Yu 2004] [Yang 2005, Yang 2007, Meyers 2007]
Cohesion	[Hitz 1995, Gall 1998, Meyers 2007]
Frequent patterns	[Yu 2004, Kim 2006]
Source code differencing	[Collard 2004, Maletic 2004, Fluri 2007]
Program and web services and execution	[Pauw 2006, Treiber 2009]
Clone detection	[Baxter 1998, Kamiya 2002, Roy 2009]
Code smells	[Lanza 2005b]
Reverse engineering	[Buss 1994, Gîrba 2004]

It has been discussed earlier in this chapter, when a large number of changes have to be understood and multiple revisions of a software project have to be correlated, it is not practical to use manual strategies and, as a result, the use of automatic analysis methods, such as SEA, is required.

The analysis of software evolution is carried out using the information which is recorded by tools that support the process of software development and maintenance, and that allows the automatic extraction of relevant knowledge facts[9]. The following are examples of these tools: software

[9]Knowledge Facts is a term used in this context for making reference to the results of advanced data analysis.

repositories of SCM tools; bug tracking tools, existing revisions of the source code of the system and information gathered by IDEs.

Therefore, taking into consideration the above and what has already been said (with respect to the software process, the current trend towards developing systems with teams distributed in different geographic areas as well as the process of software maintenance and evolution), this type of analysis may effectively support software development, maintenance and change tasks when information needs of PMs and programmers are adequately addressed in order for them to carry out their duties. In other words, the purpose of SEA is to meet the information needs of project managers and programmers so that they can develop and maintain software systems adequately.

Finally, it is worth mentioning that software evolution is usually difficult to analyze due to the large volume of relevant facts extracted and the large number of relationships between the elements involved. Thus, carrying out SEA is often not sufficient to provide adequate support to the development, maintenance and evolution of software systems. Consequently, the following chapters discuss the next steps in the process of supporting project managers and programmers in the tasks which enable the evolution of software systems.

Chapter 3

Visual Analytics

Pero a pesar del entorno salvaje, la belleza era desbordante, los ríos decoraban la selva y animaban el ambiente con sus cantos, al que se unían el canto de las aves y los congos con sus gritos. Pero lo que más llamó la atención de Güindy fue un río cuyas aguas teñidas de celeste por un volcán cercano creaban pequeños y misterios remolinos en los cuales se perdía su mirada. Era un mundo de fantasía, en donde la naturaleza lo llenaba todo; todo lo extasiaba con su creatividad. — El viaje de Güindy, A.González

Contents

3.1 Introduction

The goals of this chapter are to introduce VA, and explain the underlying processes that permit the transformation of data into a useful knowledge. The aim is to contribute to answer the principal research question (see Section 1.3), and in accordance with it the following contributory question is sought to be answered:

How can the Visual Analytics process be defined from the interactions, roles and composition (in terms of methods and techniques) of its components?

Accordingly, this chapter describes and explains some of the techniques and methods used by the VA components, with an special emphasis on those of IV and Human-Computer Interaction (HCI). It therefore first presents an overview of VA (section 3.2), next it describes some of the most well-known IV techniques (section 3.3); then it discusses some elements of HCI (section 3.4) and finally it outlines the main conclusions of the chapter (section 3.5).

3.2 Overview

The goal of VA is to transform data into knowledge. Accordingly, it iteratively collects and preprocesses data, carries out statistical analysis [Peck 2011], performs data mining, and uses machine learning [Witten 2005], knowledge representation [van Harmelen 2007], user interaction [Sharp 2011], visual representations [Leung 1994a, Johnson 1991, Robertson 1991], human cognition, perception, exploration and the human abilities for decision making [Keim 2006, Llorá 2006]. Therefore, one can say that knowledge discovery is an intrinsic property of VA, as it is aimed at supporting analysts in gaining insight from large multivariate datasets [Thomas 2005].

VA is partly based on the use of IV principles and techniques. Accordingly, the definition of the VA process overlaps, partially, with that of IV: both deal with data acquisition methods, data transformation, visual representations, human computer interaction and human capacities for decision making [Card 1999b, Chi 2000, Fry 2008]. However, VA makes intensive use of automated data analysis, visualization and interaction techniques to offer more comprehensive analysis possibilities and data perspectives for aiding intelligent decision making by means of the analytical human abilities [Keim 2008a].

In this context, Coordinated and Multiple Views (CMV) [Boukhelifa 2003] is concerned with the use of several visualizations that are linked by a model or architecture that coordinates the interactions among them and the data that visualizations must represent, in accordance to the interactions performed [Roberts 2007].

The use of CMV requires a combination of different visualization types (hyperbolic trees, graphs, treemaps, radials, parallel coordinates and grids to name but a few) in order to exploit the advantages that each one has to offer [North 2000], and to provide analysts with different levels of detail. Using CMV, analysts can understand relationships among elements located in separate, but linked, visualizations. Additionally, they can explore data from many different viewpoints and have available more interaction paths that may lead to knowledge discovery. Moreover, CMV make VA tools more

scalable, compared to IV itself, in terms of data, dimensionality, information complexity and the dynamic feeding of new data [Andrienko 2007].

VA has been applied thoroughly to solve problems in several areas, as shown in table 3.1, and many research projects have also been conducted with the aim of contributing to the improvement of VA itself by the definition of frameworks, architectures and methods (see table 3.2 for some references).

Table 3.1: Application areas of Visual Analytics.

Application areas	References
Bioinformatics	[Santamaría 2009, Gribov 2010, Battke 2010] [Agrafiotis 2010, Vicente 2010, Oeltze 2011] [Cain 2012, Hasenauer 2012, Tyakht 2012] [Peterson 2012, Schatz 2013, Santamaría 2014] [Castellanos-Garzón 2013]
Biology	[Shaverdian 2012]
City traffic	[Pelekis 2012]
Construction	[Wang 2010, Danese 2010, Batty 2013]
Customer analysis	[Ko 2012]
Data center management	[Hao 2010]
Document classification and exploration	[Koch 2011, Lemieux 2011, Tomaszewski 2011] [Koch 2011, Heimerl 2012]
Education and e-learning	[Hyun 2009, Gómez-Aguilar 2009, Gómez-Aguilar 2014] [Gómez-Aguilar 2015b, Gómez-Aguilar 2015a]
e-Government, transparency and political sciences	[Rios-Berrios 2012, Kohlhammer 2012, Crouser 2012]
Emergency response	[Livnat 2012]
Graphs and graph analysis	[Chen 2010, Yang 2013]
History reconstruction	[Andrienko 2012b]
Medicine	[Agrafiotis 2010, Maciejewski 2010, Chui 2011] [Mane 2012, de Bono 2012]
Movement analysis	[Ooms 2012, Andrienko 2013c, Andrienko 2013a]
Multimedia and video analysis	[Chinchor 2010, Luo 2012]
Natural disasters and climate changes	[Therón 2006c, Chung-Wong 2009, Yuan 2010] [Kendall 2012, Kim 2012, Kasprzyka 2013, Sun 2013a]
Network and security analysis	[Pelekis 2012, Biersack 2012]
Neuroimaging	[Li 2012]
Ontology engineering	[García-Peñalvo 2012a, García 2012][García-Peñalvo 2014]

Continued on next page.

Table 3.1 Application areas of VA – continued from previous page.

Application areas	References
Pharmaceutical	[Peláez 2008, García 2009a, Agrafiotis 2010] [Barlowe 2011, Pérez 2013]
Physical sciences	[Gaither 2012]
Real state	[Sun 2013b]
Risk assessment and analysis	[Wang 2012, Migut 2012]
Simulations	[Dransch 2010, Thakur 2011, Wei 2012, Meyer 2012]
Social networks	[Chen 2010, ah Kang 2011, Elmqvist 2012] [Perer 2013, Schreck 2013]
Software understanding	[Telea 2011, Reniers 2012, González-Torres 2013a] [González-Torres 2013b]
Spatio-temporal and geospatial	[Amicis 2009, Hardisty 2010, Andrienko 2010] [Chiara 2011, Guo 2011, Maciejewski 2011] [Tomaszewski 2011, Schumann 2011]
Sports analysis	[Therón 2010, Pileggi 2012]
Time series analysis	[Wang 2011, Maciejewski 2011, Sips 2012, Dang 2013]

Table 3.2: Correlation of references with theoretical approaches.

Application area	References
Architecture	[Maciejewski 2011, Willems 2010, Omer 2010] [Tomaszewski 2011]
Collaborative Visual Analytics	[Isenberg 2009, Isenberg 2012, Mahyar 2012]
Framework	[Pelekis 2012, Andrienko 2010] [Andrienko 2013b]
Human-Computer Interaction and cognition studies	[Gotz 2008, Chung-Wong 2012b, Pohl 2012] [ah Kang 2012, Arias-Hernandez 2012, Healey 2012] [Green 2012, Basole 2012, Roth 2012, El-Nasr 2013]
Methodology	[Omer 2010, Bertini 2011, Andrienko 2012a] [Streit 2012]
Techniques	[Dinkla 2011, Shaverdian 2012, Gaither 2012] [Chung-Wong 2012b, Alsallakh 2012, Javed 2013, Chen 2013] [Nam 2013, Dang 2013]

Consequently, any VA design should be centered on the user and should intent to facilitate usability and reduce memory load on users [Hollender 2010]. Its ultimate goal should be to hide complexity details from users and provide an environment for knowledge discovery through an outstanding human experience [Takatalo 2008]. Hence, regardless of the complexity of the problem at hand, the success of any VA solution lies on the appropriate design of the visual representations and use of interaction techniques.

VA combines the advantages of machines with the strength of humans such as analysis, intuition, problem solving and visual perception. Therefore, the human is at the heart of VA [Dix 2010] and HCI is a key component for supporting knowledge discovery. It is a process whose goal is to provide insight into *Big Data* conformed by scientific, forensic, academic or business data that are stored by heterogeneous data sources such as databases, HTML and XML files, text files, metadata and source code.

The future of VA involves several challenges related to the amount of available data, algorithms, processing, user interaction and visualization design and scalability. Some of these challenges were summarized by Chung [Chung-Wong 2012a] and are listed below:

In-memory analysis: This implies the in memory processing of data when it becomes available.

Interaction and user interfaces: Technology capabilities are increasing constantly, whereas the human abilities change slowly and changes are perceived in the long term [Thomas 2005].

Large-Data visualization: The scalability of the visualizations employed is a serious limitation for representing *Big Data* [Basole 2012].

Databases and storage services: These have been moved into the cloud increasing the access latency.

Algorithms scalability: Algorithms must be scalable for dealing with *Big Data* and provide efficiency to users in terms of visualization.

Data transport and network infrastructure: Data availability is increasing in geographical dispersed locations, which requires to moved raw data or passing messages between locations if a High Performance Computing (HPC) infrastructure is being used.

Data incompleteness and uncertainty: Uncertainty quantification and the need to deal with incomplete datasets for providing real time analysis needs the use of novel data analysis techniques.

Parallelism: It requires the redesign of current VA algorithms and the need for new algorithms.

Libraries, frameworks and tools: VA requires new libraries and frameworks for dealing with the challenges previously stated and the increasing need for parallelism.

Social, community and government engagements: This refers to the need that governments and online-commerce vendors disseminate their technologies to the society.

3.3 Information Visualization

Information Visualization (IV) deals with the representation and display of a large number of data about events, and provides the visual elements to help the interpretation of a data event through its relation with other data events. It takes under consideration several techniques to support navigation, interpretation of visual elements and understanding relationships among items in their full context [Leung 1994b]. Tufte states that the visual distinctions between visual elements should be as subtle as possible, yet clear and effective [Tufte 1990] adding that information consists of differences that make the difference [Tufte 1997].

There are many IV techniques, each one with its advantages and disadvantages, so it is frequently required to use a sort of combination to provide a real solution to end users. Spence [Spence 2000] and Card [Card 1999a] provide excellent surveys on IV methods and techniques. Therefore, the following section discusses on some of the classical visualization techniques employed when designing VA tools.

3.3.1 Visualization Techniques

The principal problem of IV is how to depict a large quantity of information in a very limited space. It is thus necessary to implement interaction mechanisms that permit navigation of the data without losing sight of the context, but which also provide tools that facilitate the interpretation of particular elements [Leung 1994b].

When the design of a visualization is being carried out it is convenient to take into account the large number of interesting and useful pre-existing visualizations. This allows, when what is sought is the solution to a practical problem rather than proposing an original visualization, to use an existing visualization as the best solution for the representation of data. A combination of visualizations with minor variations can also be used. However, from a research perspective, the ideal is to design an original, interactive and intuitive visualization that must requires little effort to learn. In this sense, it needs to take into account that the design of new visualizations for managing large volumes of information requires using spatial and temporal design techniques.

The techniques of spatial design utilize the distribution of space and graphic design to present the information at one view, while the temporal strategies use transitions to distribute the information between

multiple views, where each view represents different moments in time. It is therefore frequently necessary to use a combination of both design strategies [Mackinlay 1991].

Following Leung, the techniques which are utilized to design visualizations can be grouped into two general categories: distortion-oriented and those which are not [Leung 1994b].

The distortion-oriented techniques are used in conjunction with transformation functions that define how information will be presented to the user and the interaction which will take place. They also allow users to examine in a dynamic and interactive form the data in detail at the same time that an overview of the space, in which that section is located, is provided as a location map. This type of view is known as overview + detail because it represents details in the main visualization space whereas the overview is usually depicted in a small visualization on one of the corners of the view. Thus, it contrast with the focus + context view that permits users to focus in specific details using interaction techniques, such as selection or zoom, while a complete representation is offered (the context visualization). Whilst non-distortion-oriented techniques are adequate for text based small-scale applications but do not provide an appropriate context to support browsing large-scale information. Some distortion-oriented techniques that are further discussed in detail[1] are:

Bifocal and polyfocal: Fisheye [Furnas 1986], Table Lens [Rao 1994] and Perspective Wall [Mackinlay 1991].

Timelines: Lifelines [Plaisant 1998] and Planning lines [Aigner 2005].

Hierarchies: Treemap [Johnson 1991], Cone trees [Robertson 1991] and Hyperbolic spaces [Gra 2002, Pavlo 2006].

Radials: Information slices [Andrews 1998], Radial improved with focus + context [Stasko 2000], Intering [Yang 2003], Ring Tree [Therón 2006b], Hyperbolic spaces [Pavlo 2006].

Networks and graphs: These visual representations are very common in IV and there exist many research papers that presents results regarding their use. In this context the book of Battista [Battista 1998] is a great reference to study the most common algorithms for the visualization of graphs and the survey conducted by Gibson [Gibson 2013] provides useful information on the use of networks and graphs in research works.

[1] A distorted view is created by applying a mathematical transform function to an image without distortion. Magnification functions are functions derived from the transform function and provides a profile of the factors associated to the magnification or demagnification of the area of interest in the image without distortion.

With respect to visualization techniques listed above, there are a number of implementations that use them. The following sections provide an introduction to some of them.

3.3.1.1 Bifocal and Polyfocal Techniques

Bifocal and polyfocal displays allow to review specific items of information expanding an individual spot, vertically or horizontally, or by expanding an area simultaneously in both directions (horizontally and vertically). This technique originally was proposed by Robert Spence and Mark Apperley in 1982 [Spence 1982]. Some of the most representative visualizations that make use of these techniques are the following:

Fisheye: An application of this technique was proposed by Furnas in 1986 using the name *Fisheye* [Furnas 1986]. This technique allows context to be maintained while viewing a specific area and can be applied in conjunction with any other visualization technique, whether textual, tabular, hierarchical, circular or hyperbolic.

It is important to draw a distinction between the *Fisheye* technique and ordinary geometric zoom. Ordinary geometric zoom allows the user to specify the scale of the increase each time the size of the area of interest is expanded or reduced and is usually set at a particular point and does not seek to preserve the context, while the *Fisheye* technique retains context.

The *Fisheye* technique contrasts with semantic zooming [Cockburn 2009], which changes the form or context in which information is presented. A very useful example for understanding the semantic zoom is a digital diary represented as a calendar year. When the user selects a month, this month opens and the corresponding days appear. Then when the user selects a day of that month the times for that day are shown, and if a time is selected, information is provided about scheduled appointments or tasks for that time.

The application of visualization techniques to a particular area, such as the *Fisheye* technique, can be very useful when navigating in a dataset with a large number of elements. However, when it becomes necessary to compare two or more elements the bifocal display is used together with interaction techniques to amplify two or more areas at the same time. This type of display is called polyfocal.

Table Lens: The visualization technique that is referred as *Table Lens* is a type of the polyfocal display and support focus + context functions. Therefore, it can be used very effectively with large quantities of

information arranged in tabular form. This representation can expand one or more rows and columns at the same time: column width can be extended using the mouse, subgroups of columns can be created and filtering can be performed on the data set.

Perspective Wall: Another visualization technique in this family is the *Perspective Wall* [Mackinlay 1991]. This technique is based on a detail + context view to provide details using interactive *3D* animation and linearly structured information. It uses a front panel to display a detail area and two side panels to present the context. This view was originally proposed for managing documents and files, but its use can be extended to any problem that contains a high temporal content.

3.3.1.2 Timelines and Temporal Events Representation

The discovery of the relationships between data items frequently is taken into account when designing and creating timelines. Accordingly, this section discusses some visual representations aimed in this regard.

Linear timelines: *Semtime* is a linear timeline which uses a set of stacked timelines, visualizing the same or different time ranges. Moreover, this visualization uses lines with arrows to represent the relationships between data elements in the timelines [Jensen 2003]. This visualization is of great value for comparing time periods and correlating data elements in time.

Continuum is also a linear timeline which uses a scalable histogram overview that allows the navigation through a complete hierarchical dataset [André 2007] and moreover facilitates the comparison of events in different time periods.

Some additional and interesting visualization examples are the visual representations designed by Catherine Plaisant [Plaisant 1998] and Wolfgang Aigner [Aigner 2005]. Plaisant addresses the problem of visualization of a patient's medical history with a display known as *LifeLines*. This visualization relates a group of variables such as notes, hospitalizations, tests, medications, treatments and vaccines with temporal space. It also uses labels to identify each of the instances of the variables mentioned.

An improvement of *Lifelines* is presented in [Bade 2004]. It uses three timelines; one of them is a general timeline; the other timeline is the result of the dates filtering carried out using the first timeline; and the last timeline displays the information details. This visualization uses an overview + details approach and the interaction with the

details area provides a focus + context view of the results after applying filters [Bade 2004].

Aigner proposed *Planning Lines*, a visual representation that is aimed to visualize task planning in a form similar to *Gantt* diagrams. To do so, semantics was added using colors to indicate the minimum and maximum duration of tasks, and the use of lines to indicate the premature beginning or later ending of tasks in accordance with planning.

Radial timelines: Some timeline visualizations use radials and tree ring metaphors. Therón *et al.* proposed a tree-ring metaphor, which is named after *Ring Tree*, to represent hierarchical time based structures and applied it to browse and discover relationships in the history of computer languages [Therón 2006a] or phylogenetic tree [Santamaría 2009].

Spiral Graph is another radial visual representation which uses a spiral metaphor for representing a timeline with the end of supporting the analysis and comparison of values and data sets, and the detection of periodic behaviors and trends [Weber 2001]. Similarly, the *Semantic Spiral Timeline (SPT)* [Gómez-Aguilar 2009, Gómez-Aguilar 2010] allows time periods to be compared at a glance just by looking at the appropriate region of the spiral and observing the details in the successive circumferences. The comparison of time periods in the *SPT* visualization is a similar approach to the one proposed by [Hochheiser 2004], but the time periods are stacked instead of being spread along the x axis.

Correlation of time with hierarchies: Several visualization that address the correlations of time and hierarchies. Morris [Morris 2003] worked on the visualization of temporal hierarchies plotting research documents along a horizontal track in the timeline and placing related documents according to the hierarchical structure produced by the clustering phase.

TimeTree is a relevant visual representation which was developed by Card [Card 2006] and allows exploring hierarchies that change with time. This visualization allows searches, navigation through a hierarchical representation and the filtering of results with the assistance of a time slider control.

Use of color coding: Other useful examples that show temporal events without making explicit use of a timeline are those that, by using color or

other elements, permits the addition of meaning, size and temporality to the events [Therón 2013]. In this sense, Chen [Chen 2006] shows that the use of color is very useful to depict the collaboration that has taken place over time on a single file or directory. Finally, Viégas and Wattenberg suggested a visualization to show the changes made to a document of the online encyclopedia Wikipedia [Viégas 2004]. This representation is interesting because you can see the lines of text that have been added or deleted in time from a document.

3.3.1.3 Hierarchies

The presence of hierarchical data in business and academic environments is abundant and has led to a great deal of research orientated towards its adequate treatment and representation. Accordingly, the next visualizations are some examples of the most common visual representations used for depicting hierarchical data:

Treemap: A visualization technique considered classic in this area is *Treemap*, which was suggested by Brian Johnson y Ben Shneiderman in 1991 [Johnson 1991]. This visualization permitted the representation of hierarchical information in a rectangular space in *2D*, using 100% of the available surface. It also provides interactive controls, facilitates rapid information retrieval with low perceptual and cognitive load, while providing an aesthetically pleasing presentation.

According to the authors, this visualization is suitable for hierarchies in which structure is of great importance and the information associated with the nodes is derived from their descendants. The method involves matching hierarchical information with the rectangular structure.

Voronoi Treemap: Another visualization which can represent hierarchical data and which is a variant of *Treemap* is *Voronoi Treemap* [Balzer 2005b]. This visualization uses polygons instead of rectangles used in the original version. The reason the authors use to advocate this variant is that by using polygons it is possible to cover the area corresponding to each value required to be represented because they can adapt better to the environment by having a variable number of edges.

Bubble Treemap: Another variation of this visualization is that proposed by Karl Wetzel [Wetzel 2004] for the visualization of files in a hard disk. Instead of rectangles, squares and polygons, he proposes the use of color

coded circular components. However, although this visualization turned out to be appealing, it doesn't use space to maximum advantage nor is it possible to determine at first glance the hierarchical relation between the components. An article which is useful in understanding this technique is that of Teoh and Ma [Gra 2002].

Cone Trees: The visualization of hierarchical structures with three-dimensional conical representations is visually appealing. The Cone Trees technique is an example of this [Robertson 1991]. This representation is a *3D* animated visualization of hierarchical data.

The root of the structure is at the top of the presentation and the descendants are drawn in the lower layers, always allowing interaction to expand or collapse the elements when navigating through it, as well as to select different items. In this visualization the size of elements in the lower levels of the structure is reduced to ensure that the representation conforms to the width of the display area. Also, when a node is selected, the structure rotates to show the node selected and the route towards the root of the hierarchy. An additional attribute of the *3D* visualization is that it provides a way to focus on one part of the structure without losing the context.

The authors claim the *3D* visualization is necessary because it maximizes the effective use of the display area and displays the entire structure. However, some results have shown that tracing elements and relationships in three-dimensional conical structures is slower than doing so with *2D* tree visualizations [Cockburn 2000]. These results showed that visualization diminish in utility and the performance of computer equipment is significantly reduced when tree density increases.

It is worth mentioning that this type of *3D* visualization initially aroused great enthusiasm among those who also tested the *2D* solution.

Hyperbolic: Another very striking design representation is hyperbolic visualization, which is a visual representation technique that supports focus + context, capable of handling large information hierarchies.

This visualization initially displays a tree with its root as the central element connected to a few nodes, using interaction techniques which enable additional elements to be introduced, and may also represent other elements were not initially displayed.

According to Lamping [Lamping 1995] hyperbolic visualizations may allow the presentation up to 1000 nodes simultaneously, of which

50 are located near the focus. Moreover, this visualization can also display text labels with a meaningful context in a representation of this density.

Hyperbolic visualization is a hierarchical representation technique of great utility if properly combined with appropriate interaction techniques. However, despite its visual appeal the temptation to carry out developments in *3D* must be avoided, because navigation in these structures is complex and it is difficult to obtain useful information quickly and easily.

3.3.1.4 Radials

Circular and semicircular techniques are used primarily to display hierarchical information [Andrews 1998, Stasko 2000, Gra 2002, Yang 2003, Pavlo 2006].

These techniques use algorithms that divide the area according to the weight of the elements and place these elements in concentric rings according to their position in the hierarchy which results in the location of the root node in the center of the structure. Next, the circular area is divided among its descendants on the first level of the hierarchy which places them in the first ring. The area assigned to each descendant is calculated based on its weight in a manner analogous to pie charts. Then an element is taken from first level of the hierarchy and the corresponding area is divided among its children, which are located in the next ring of the structure. This process continues with the rest of the hierarchical elements until all elements have been represented so that the number of rings in the visualization is a function of the depth of the tree. Some visualization examples that follows this kind of algorithm are the following:

Treevolution: This uses a tree-ring metaphor to represent structures based on temporal hierarchies [Therón 2006a, Santamaría 2009]. The purpose of this visualization is to facilitate viewing, navigating and describing relationships in the history of programming languages and in phylogenic tree structures.

Sunburst: Sunburst was originally proposed with the goal of determining its effectiveness in a usability study of radial visualizations which was aimed to evaluate these type of representations. The study found that the main disadvantage of such visualization was the difficulty of distinguishing circular lines when used in the representation of large hierarchies. As a result of this study, later three different types of Sunburst visualizations were proposed: angular detail, external detail and internal detail.

Further information information on radial visualizations is available in the excellent survey that has been carried out by Draper in [Draper 2009].

3.3.1.5 Networks and Graphs

Network and graph visualizations are useful in decision making as they successfully support the understanding and comprehension of complex problems where inherent relations exist among data elements [Herman 2000]. They permit cause-and-effect analysis of phenomena that are often present in data whose source is industry or academia. The graph representations are used for a variety of purposes including: dependency hierarchies, relationships between documents and genetic maps.

Graph visualization is usually carried out by means of radial or conical structures, network or hierarchical drawings. Thus, many of the visualizations that have been discussed in the previous sections represent hierarchical graphs, where the structure is displayed implicitly or explicitly with the use of nodes and lines. Some of the principal graph topologies are *Squared*, *Radial*, *Triangle*, *Cube*, *Squared mesh* and *Rectangular mesh* [Battista 1998].

Graphs can also be classified as rooted or non-rooted tree types. For example, radial and conical representations are rooted trees, while others including electronic circuits and networks are non-rooted. The latter may also be called series parallel digraphs. Important to mention also is the fact that hierarchical visualizations make use of layering to position nodes at appropriate hierarchical levels. The form in which graphs are drawn is important and there are conventions for rendering such as *Polyline*, *Straight-line*, *Orthogonal*, *Grid* and *Planar*.

A major challenge in drawing graphs is occlusion which occurs when large numbers of elements are obscured by lines crossing when large datasets are represented. Therefore, it is necessary to utilize sundry techniques and strategies, one of which is planar graphs, which employs algorithms to obviate lines crossing to make the drawings clearer.

In addition to the ways graph and topologies are drawn and utilized, it is often necessary to use additional strategies to ferret out knowledge. For example, to compare two graphs, both graphs are drawn with one on top of the other; then one is drawn using and intense color whereas the other is drawn using a less intensive color and an opacity effect. Likewise, it can be advantageous to use force directed graphs to show closeness between elements or groups of elements in the graphs [Battista 1998]. The commonest algorithms in force-directed graphs are the following:

* Springs and electrical forces.

* The barycenter method.
* Forces simulating graph theoretic distances.
* Magnetic fields.
* General energy functions.

Finally, it is important to mention that the tasks that can be performed using graphs are usually based on topology, attributes, or simply displaying an overview of the elements or by browsing the complete graph [Lee 2006] and that many research papers have been published on the improvement of algorithms and the layout of elements [Gibson 2013].

3.3.1.6 Multivariate Visualization

All visualization techniques discussed previously are capable of representing multivariate data. To accomplish this, the principal elements used are colored visual items and shapes. But the main problem facing all designs directed at effective visualization is scalability, in terms of the sheer volume of data along with the sheer number of variables which must be represented. A good example of scalable visualization of numerical data, in the terms described previously, is Parallel Coordinates [Inselberg 1985, Inselberg 2009].

Parallel Coordinates: This visualization is capable of representing a great number of variables associated with a single element. It also permits the representation of multiple elements at the same time, ranging from hundreds to even thousands of elements.

The flexibility of the visualization permits the analyst to search, highlight and group elements automatically or manually. In addition to comparing and filtering elements easily. The result is the achievement of maximum scalability and the avoidance of user information overload while encouraging user interaction with the data elements.

Finally, it is important to mention that since the inception of *Parallel Coordinates* [Gómez-Aguilar 2015b, Gómez-Aguilar 2015a], similar approaches have gained attention and a wide variety of refined approaches have arisen, including hierarchical [Fua 1999] and circular [Long 2009].

3.4 Human-Computer Interaction

Hardware and software systems are usually conceived as processing units that receive inputs from users, process such inputs (and retrieves additional information from databases or other data sources when it is applicable)

and produce outputs that are interpreted (or used) by users. However, the interaction between users and systems is not as simple as it was described when a complex problem is under consideration. Solving complex problems requires several steps to be performed, and thus the active participation of users is frequently needed to decide the course of actions to be followed.

Accordingly, the factors concerned with an effective and easy communication between humans and systems include the effective design of hardware and software systems, the appropriate design of usability elements and the psychological and cognitive aspects that intervene when humans use such systems. Therefore, HCI principles should be taken into account to create successful systems that involve users in enjoyable, engaging and productive interaction experiences.

The interactions of users with systems should be as engaging as it is for them the discussion about some topics, such as football, soccer, politics or religion. A conversation on these topics usually keeps the attention of individuals during several hours: replies and opinions are expressed in an animated interpersonal interaction. Likewise, the conversation between users and systems needs opinions to be expressed in the form of user's inputs, which should be processed in a proper manner to produce replies (outputs) according to the discussion topic and focus (user needs and expectations).

The interaction between users and systems underlies, mainly, in the easy to use of interfaces (hardware and software), the inputs provided by users and the usefulness of the outputs produced by systems. In general, the inputs from users are needed at the initial processing stage as well as in intermediate processing stages, until results are obtained in a refined or final form on the basis of user requirements.

The input provided by users to software systems is usually carried out with the use of keyboards, mice, touch screens, microphones, sensors, wired gloves, retinal lectors, face readers, thermal and infrared scanners, and fingerprint readers. Therefore, the design of visual representations may consider one or several of these input devices to offer useful interaction experiences.

In the context of how users browse and navigate through visual representations looking to solve complex problems by means of visual analysis, several pieces of research have been conducted. Typically, users form a hypothesis to solve a problem, collect and analyze data, and then accept or reject their initial hypothesis. This was explained by Wehrend *et al.*, who defined a taxonomy of eleven actions that are carried out by users in visual environments, as following: identify, locate, distinguish, categorize, cluster, distribution, rank, compare within relations, compare between relations, associate and correlate [Wehrend 1990]. Thereafter, one of the most notable research in this field is summarized by the Shneiderman's Visual Information

Seeking Mantra (overview first, zoom and filter, then details-on- demand) that outlines the tasks usually performed by users when navigating IV visualizations [Shneiderman 1996].

Furthermore, Pirolli studied visual information foraging [Pirolli 2001] from a visual attention and information foraging theory perspectives, where the latter is concerned with search, exploration, location and evaluation of information [Chen 2002].

Taking into account what has been stated so far, it is important to recall that the design of visual representations of huge datasets is often difficult because of the limited size of screens [Leung 1994b], which frequently makes browsing and navigating capabilities challenging. So, some of the main challenges on this regard were discussed by Chung [Chung-Wong 2012a], and are listed below:

In situ interactive analysis: Users require smooth interactions and rapid system responses, which usually requires in memory analysis [Basole 2012].

User-driven data reduction: Users should be capable of controlling their data and analytical needs.

Scalability and multilevel hierarchy: Keeping control of scalability and hierarchy depth is a challenge that requires fast software response times to satisfy user demand for fast answers [Basole 2012].

Representing evidence and uncertainty: The visual representation of data analysis results requires of visual representations of the level of uncertainty for informed decision making.

Heterogeneous-data fusion: This point refers to the analysis of heterogeneous data sources and their interrelationships aimed at extracting the required semantics needed in VA applications [Basole 2012].

Data summarization and triage for interactive query: This implies that I/O components must provide adequate response times for providing timely query results [Reiss 2005, Lee 2011].

Analytics of temporally evolved features: The representation of temporal data and events usually is challenging due to the time span in which the events have taken place and the large number of associated events. So, the representation of temporal data needs to consider the human abilities for exploration, creating relationships and decision making.

The human bottleneck: This challenge is related to the increase in bandwidth, memory, storage and processing capabilities, confronted with humans and their capabilities to scale their abilities in short periods of time. Therefore, awareness of these human limitations must be foremost in the minds of those whose task is to design useful and usable solutions.

Consequently, the adequate use of interaction techniques is an important element to design VA systems that are easy to use and permit fluid and engaging communications with users. Therefore, interaction mechanisms to support users in the exploration of details and the building and tracking of relationships, should be considered. Thus, the use of these mechanisms facilitate the interpretation of specific elements that could lead users to discover knowledge or facts that enable them to arrive at useful conclusions. Thereupon some of the interaction mechanisms commonly used by visual representations are the following:

* Navigation [Wilkinson 2005].
* Brushing and selection [Buja 1996, Dix 1998, Wilkinson 2005].
* Drill down [Dix 1998].
* Filtering [Shneiderman 1996, Wilkinson 2005].
* Linking multiple views [Wilkinson 2005].
* Geometric and semantic zoom [Shneiderman 1996, Dix 1998, Cockburn 2009].

Finally, it is worth to mention that interaction design patterns have been designed for building general and repeatable visual designs [Pauwels 2010, Tidwell 2011].

3.5 Conclusions

Nowadays, companies compete in a global market where successful strategies are crucial in overcoming the widespread economic crisis. And since several consulting firms have compiled large databases of business data from different market segments, more and more companies now have access to market intelligence databases and to their own historical transaction databases, which together represent a rich data source for performing analytics, using modern analytics tools and taking advantage of accumulated expertise. Additionally, automated, analytic methods and techniques have become more complex and powerful, which encourages firms to take advantage of automatic data analysis with great precision advantaged by the constant increase in performance,

processing capabilities, and reductions of costs of servers and computer equipment.

The main developers of Enterprise Resource Planning (ERP) and BI tools have taken a step forward in the current global crisis by adding VA components to their data analysis tools with the aim of improving the capabilities of managers to carry out analytics at different stages of their business processes [Institute 2011, Zhang 2012, Skytree 2013].

The main reasons for incorporating VA components into BI and management tools is to combine the capabilities of computers for performing fast and precise calculations with the human strengths of intuition, critical thinking, problem solving and visual perception. Moreover, the rise in the use of VA is reinforced by increased research and published papers on the application of VA to such diverse areas.

Consequently, this chapter has explained and described the main components of VA, as elaborated from previous definitions of this area. The purpose is to offer some guidelines that could aid software designers and architects in designing and programming of VA tools and solutions. This is particularly important in the current scenario described above, where companies require - more than ever before - the transformation of available data into knowledge to compete successfully in the global market, sometimes with very particular requirements.

Part III

Visualization and Visual Analytics for Software Systems

Chapter 4

Systematic Mapping Study

Cuando llegaron al valle multicolor de Orsi se detuvieron frente a una encrucijada. Nevo cortó una gardenia y caminó en dirección contraria, de vuelta a casa. Entonces murmuró, sin mirar a Güindy, "ahora debes seguir solo". "¿Qué camino debo seguir?", preguntó Güindy. "El que quieras, es tu camino", contestó Nevo, al tiempo que besaba la gardenia.. — El viaje de Güindy, A.González

Contents

4.1 Introduction

The aim of this chapter is to review in depth the current state of the application of visualization and VA to software systems and their evolution in facilitating the development and maintenance of software. The decision to conduct a systematic mapping study was rooted in the possibility of carrying out an analysis of greater depth and breadth; a study that would provide further details on the work carried out and the trends that mark them. This

chapter thus examines the use of IV and VA in the comprehension processes of software projects and their evolution by means of a systematic mapping study of research carried out in the last 7 years, from 2007 to 2013.

In order to do this, the tasks that this research sought to support were identified as well as the different types of visualization, data types and technologies that were used. Moreover, the types of validation used to test the applicability of the proposed methods or solutions relative to the tasks were also identified.

In order to carry out the systematic mapping study, all the papers of the principal workshops, symposiums and conferences related to our field of study for the period of analysis already mentioned were revised. Subsequently, two specialized search engines (*WorlCat* and EBSCO Discovey Service (EDS)) as well as *Google* searches were employed.

The items found were included or discarded according to their titles, a quick review of the key words and the abstract. Once the articles had been selected, the abstract and the introduction were more carefully examined in order to extract information that would permit the paper to be classified as a paper which searched for a solution to a specific problem, or one that addressed theoretical or methodological issues.

Subsequently, an overview of the development of publications and conclusions was made seeking accurate information about the task that the paper sought to resolve or the research approach employed, as well as the details of the answers given to the research questions (explicit or implicit).

The papers in both groups then went through another process of classification: the former group being sorted into 22 categories according to the tasks that the papers seek to support during the development and maintenance of software and the latter group were classified in 11 categories according to the research objetives that they address. Subsequently, several relationships were created between the categories and research focuses identified and other items which had relevance, depending on the particular group of papers under analysis. Finally, the introduction and development of the chapter was revised in detail in order to determine the visualizations used, the manner in which data was represented, the number of views and the validation techniques employed.

Accordingly, section 4.2 explains the methodology employed for conducting the systematic mapping study; section 4.3 presents the results of the study; Section 4.4 discusses some of the most relevant results of the investigation, and finally Section 4.5 makes the conclusions of this chapter.

4.2 Methodology

This section presents and explains the research questions, the criteria utilized to include or exclude research, strategies used to search for the research studies and the classification criteria applied to such research work in order to carry out the systematic mapping study described in this chapter.

4.2.1 Research Questions

The main objective of the systematic mapping study is to answer the following research question:

How have visualization and visual analytics been used in software development and maintenance tasks?

The following subsidiary research questions have emerged as a logical consequence of research into this primary question:

1. Which tasks in the process of development and maintenance of software systems are supported through the use of visualization and visual analytics?
2. Which types of visual representation are used to support each of the particular tasks identified?
3. What data derived from the analysis of software projects and their evolution are visually represented?
4. What are the technologies used in the solutions proposed by the research studied?
5. What types of validations are carried out in order to test the validity of the proposed solution or technique?
6. What types of data and visual representations are used both to support comprehension of the analysis of a revision as to support understanding of the complete evolution of the software project (or over a period of time)?

The analysis of the evolution of a software project involves the individual analysis of each of the revisions under consideration, whose results are then correlated in order to find relationships or facts which are relevant to the particular task that is being undertaken.The discovery of useful knowledge from the visual representation of the results of this analysis may require the use of navigation and interaction techniques that provide different views

and perspectives. In accordance with the above, this chapter seeks also to investigate on the following subsidiary research question:

7. How are the multiple views and multiple linked views used by research that is aimed at analyzing software evolution (several revisions or a period of time of the evolution)?

4.2.2 Inclusion and Exclusion Criteria

The inclusion and exclusion criteria that were applied to the research reviewed for this study are the following:

Time period of the study: The study included all the publications in the last 7 years (2007-2013). However, in the case of 2007 only papers from the proceedings of IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT) 2007 were included (as well as a highly cited paper that could be considered an obligatory reference, and which was published in the proceedings of the 2007 International ACM Conference on Supporting Group Work).

Papers studied: A total of 219 papers were downloaded, but once they had been filtered and revised, a final total of 149 papers were evaluated.

Relevance of papers: In this regard, the study took into account the following criteria:

1. Only full papers were considered (with the exception of two short papers included in the study because they were of special interest)
2. It was determined by the use of visualization and VA in order to understand software systems and their evolution in general terms and, more specifically, with the objective of supporting the development and maintenance of software. The following factors were taken into account in this regard:

   Type of proposal or evaluation: Visualization designs, tools, strategies, techniques, taxonomies, frameworks, validations, theoretical or philosophical discussions (see Table 4.1 for a definition), experience and survey papers have been proposed or evaluated by the papers.

   Types of data analysis supported: The research works may have used static, dynamic or a combination of both types of analysis.

Time period of data under analysis: The analysis performed may have taken into account one, several or all of the system revisions.

It is worth mentioning that this research took into account some patterns that were detected in which research groups employ the same visual approach with slight variations or change the focus of the publications using different perspectives (e.g., comprehension of systems, structures and presentation of frameworks) or making variations on how a previously proposed visual solution is validated (e.g., using a case study or a usability study). Therefore, in these cases, the study excluded those papers that are neither an extension of, nor demonstrate significant progress in relation to a previous publication. In the case of several publications related to the same research, where there is a publication in a journal concerning the same research area, the research published in the journal takes precedence over that which formed part of conference proceedings.

4.2.3 Searching for Research Studies

A complete review was performed of all full papers, and two short papers of special interest, which were presented from 2007 onwards in the ACM Symposium on Software Visualization (SOFTVIS) and VISSOFT. Later searches were carried out using the specialized search engines *WorldCat* and EDS using the following search arguments:

```
Software visualization OR Software evolution visualization
OR
Visual software analytics OR Visual analytics
AND
(Software OR System)
OR
Software evolution
```

The result of searches carried out, plus the articles accepted in VISSOFT and SOFTVIS, came to 219 papers in total, of which only 149 were included. Annex A shows the complete classification of these works by publication venue and date of publication.

4.2.4 Classification Criteria

The primary classification criterion is the research scope used by the study research paper consideration. Thus a variant of the Wieringa

and Peterson [Wieringa 2006, Petersen 2008] classification model, shown in Table 4.1 was used. The application of these criteria of classification shows that most of the research work belongs to either *Philosophical Research* (47) or *Solution Proposal* (102) categories, which are opposite fields with regards to theory/practice. This is why it was decided to group the results into 2 major groups, using those categories as a starting point for classification and the others as additional criteria subject to both criteria. In this manner the work done under the *Philosophical Research* category is associated with all the categories present in Table 4.1, including the category denominated *Solution Proposal*, whereby all work from the *Solution Proposal* category is associated with the *Evaluation Research* and *Validation Research* categories.

Table 4.1: Classification scheme of investigations according to their research scope.

Category	Description
Evaluation Research	A novel technique or solution proposal implemented and evaluated in a real life scenario. Moreover, it may also take into account an evaluation of third party tools or techniques in a controlled or real life scenario. Evaluation also includes the studies carried out to test something rather than validating an approach.
Experience Papers	The personal experiences of authors and how something has been carried out in practice are exposed.
Solution Proposal	A solution for a problem, either novel or a significant improvement over an existing technique, is proposed. The potential benefits and the applicability of the solution are validated or discussed.
Philosophical Research	This category includes taxonomies, software frameworks, conceptual frameworks, classification schemes, surveys, and contributions to the theory, case studies on theoretical foundations, and techniques or methods either novel or improved.
Validation Research	A probe of concept of a solution proposal has been implemented, but a real life implementation has not been carried out. Therefore, the authors look to validate their proposal via a case study, user evaluation, use case, examples of use or a reasoned discussion.

Overall, this study seeks to present a comprehensive survey of the current state of research into visualization and VA when used to solve problems or support tasks in the software development and maintenance process. The study also aims to show the theoretical advances in the area.

The research has thus been classified as *Solution Proposal* is that whose visual representation may be a variation of existing techniques or a combination of several existing techniques, while papers classified as

Philosophical Research are those which suggest new techniques, methods or improvements to the aforementioned techniques, but without being restricted to trying to resolve a specific problem but rather offer basic input from a methodological or theoretical perspective.

The following classification criterion that was applied to the research work was applied to the scope of the represented data and the supported analysis. Therefore, a research work was classified as Sys[1] or Evol[2] according to the number of system revisions that it has the capability to deal with. Then, after classifying the papers using the above classification criterion, an additional review of the documents was carried out to determine a secondary classification criterion based on the content of the papers under study. Accordingly, in the case of the research works in the *Philosophical Research* category, this review has sought to determine which research approaches the works are using, whereas in the case of the works in the *Solution Proposal* category it looked to identify the tasks that the research work is intended to support. In the case of work classified as *Philosophical Research*, the 11 research approaches on the following list were extracted from the papers under consideration:

1. Case study
2. Classification scheme or taxonomy
3. Evaluation
4. Framework
5. Lessons learned
6. Novel technique
7. Reflections or discussion
8. Study
9. Survey
10. Systematic mapping study
11. Technique improvement

The number of tasks which the research in the *Solution Proposal* category seek to support total 22 was determined by the aforementioned review, which also lead to the conclusion that research work sometimes supports more than one task. Therefore, research work supporting primary tasks (one task) totaled 79, whereas the ones supporting secondary tasks totaled 23 (20 of them support two tasks whereas the other 3 works support three tasks). However,

[1] A research work is classified as *Sys* if its application scope is the current system revision or a single revision.

[2] Research works classified as *Evol* are those which take into account one, several or all of the system revisions, in short are those works which study system evolution.

the classification schemes under the *Solution Proposal* category only present classification details for the primary tasks that are supported by research work.

Consequently, these schemes did not consider those tasks that only received support from the research as secondary tasks, and thus were excluded from the classification schemes. The decision to include only the primary tasks in the classification schemes was based on the need to provide information that could lead to accurate comparisons between tasks, and which could be altered if works appeared to be associated with more than one task. The secondary tasks that were not supported as primary tasks from any of the research works studied were *Source code porting*, *Source code reuse* and *Support debugging*. A complete list of tasks supported by the research works under study is shown:

1. Detect design flaws
2. Distributed systems comprehension
3. Improve software quality
4. Improve source code security
5. Memory allocation analysis
6. Multithreading execution analysis
7. Parallel execution analysis
8. Performance analysis
9. Program execution analysis
10. Software design and modeling
11. Software ecosystem comprehension
12. Source code porting
13. Source code reuse
14. Support debugging
15. Support reverse engineering
16. System analysis and understanding
17. System refactoring
18. Software testing
19. Team awareness and collaboration
20. Understand dependencies
21. Understand software changes
22. Understand system architectures

Consequently, the previously mentioned criteria allowed for a basis framework to be defined in order to permit a more detailed classification of the works under analysis. The aforementioned classification was carried out using five classification schemes, one of which is related to the investigations under the Philosophical Research category and four of which are related to the investigations from the Solution Proposal category.

The classification schemes include details on the number of works that support systems classified as *Sys* and *Evol*. An important consideration to be highlighted at this point is that supporting SE requires a greater effort than that required for supporting only the current system revision or a single revision.

4.3 Results

317 researchers were involved in the preparation of the 149 works, who mostly worked in groups composed of two and three researchers, as Figure 4.1 shows. Most of these researchers only contributed to one work (243) and the number of those who participated in two (48), three (15) or more (11) studies was very small relative to the total number of researchers (see Figure 4.2).

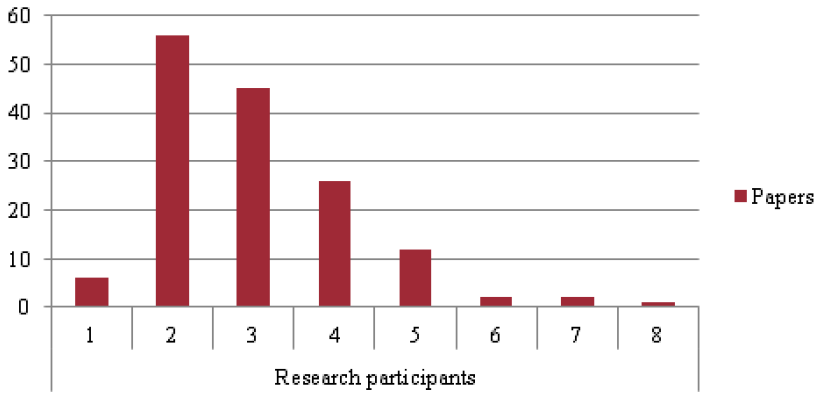


Figure 4.1: The *x axis* shows how researchers were organized in groups, in terms of the number of participants, to carry out the research works. In line with this, the *y axis* depicts the number of research papers and its correlation with the investigation groups.

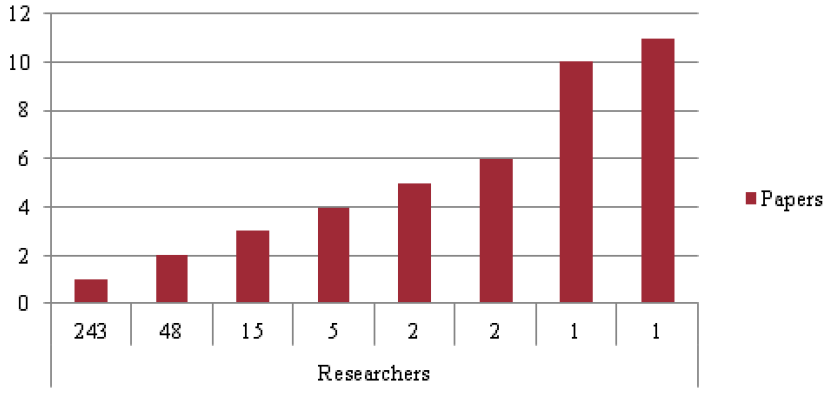


Figure 4.2: Correlation of researchers with the number of papers in which they have participated as authors.

The distribution of the authors who participated in a larger number of works is shown in Figure 4.3. It is worth highlighting that among the authors included, several authors participated in the same number of published works. The four researchers with the most publications (*Michele Lanza*, *Alexandru Telea*, *Jürgen Döllner*, and *Stephan Diehl*) appear as secondary authors in the works.

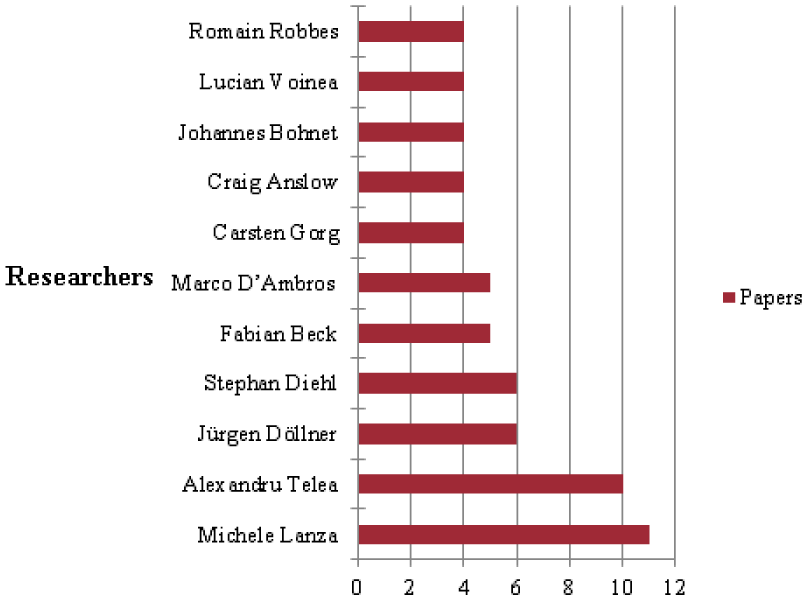


Figure 4.3: Researchers with highest participation in published papers.

Figure 4.4 presents the distribution of the research works studied by year. Accordingly with this figure the year that most research papers were published, both in the category *Evol* and *Sys*, was 2010, followed by the year 2009. It is striking that in most of the years the number of research works under the rubric Sys is highest that those classified in the category *Evol*, with the exception of the years 2009 and 2012 in which the highest number of research papers is under the rubric *Evol*.

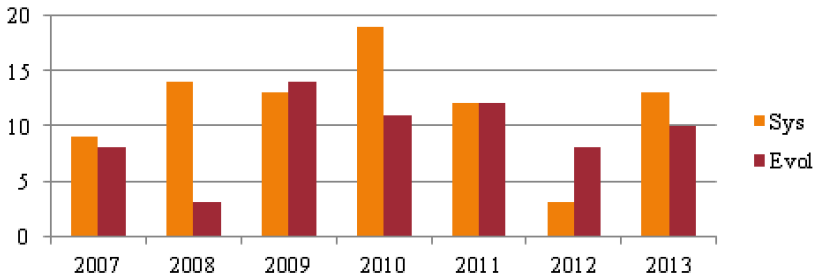


Figure 4.4: Distribution of the total number of works carried per year and category (Sys and Evol).

Based on the discussion in the previous section, table 4.2 presents some

general information about the studies analyzed. In this classification scheme, the rows denominated Solution Proposal and Philosophical Research are the primary criteria used for classification while the analysis type that has been supported (*Sys* or *Evol*) serves as a secondary criterion. Starting from the basis of these primary and secondary criteria, the research work is then classified according to the following additional criteria:

1. Education: The papers classified in this category are those that seek to support the educative processes of teaching and learning in programming courses.
2. Static analysis: This category makes reference to papers that use static analysis to obtain data to be used in visual representations.
3. Dynamic analysis: This category includes studies that use data obtained by means of dynamic analysis techniques. These techniques render the analysis of programs during their execution for a period that may include, for example, be the time it takes to perform a particular task, or a period determined by the analyst. However, when nothing more than the analysis of one revision has been carried out, the works have been classified under the rubric of Sys.
4. Use Visual Analytics: This category is used to classify research work that makes use of Visual Analytics (VA) in the analysis of one or more system revisions. Specifically, the application of VA approach to the analysis of a software system revision is denominated as Visual Analytics Software [Anslow 2009] and this research defines the application of VA to the analysis of two or more software revisions as EVSA (which will be discussed in more detail below).
5. Architecture: A paper classified in this category introduces the design of an architecture, or the process that has been defined or employed during the research. Frequently the architecture which is discussed in the work is not original and what is sought to be presented here is the interaction between the different components employed in the search for the solution to a determinate problem.
6. Web: This category is used to classify the studies that have used web technology to implement the visualizations featured.
7. 3D: Works that employ 3D visual representations are classified in this category.
8. Plugin: The visualization tools proposed by the works classified in this category can be integrated as extensions of the most well-known IDEs which are commonly used both in educational and industrial settings.

9. Animation: This category is used to classify works that use animation to facilitate teaching processes in academic settings as well as the development and maintenance of software in software development departments and the software industry.

10. Views: The works studied may be classified as Single, Multiple or Multiple Linked depending on the views that are used and the level of integration between them. In visualization, the multi-view displays can operate independently or be linked by interaction between them. The works that employ multiple views and explicitly explain the interaction between views were classified in the subcategory Multiple Linked.

According to the scheme in table 4.2 the number of papers that seek to support teaching/learning in academic settings or business is reduced, because only 10 out of 149 have that objective, and it can thus be concluded that the purpose of most studies is to support the development and maintenance of software in industrial settings.

Table 4.2: General classification scheme.

Category								**Technology**				**Views**		
		Analysis supported	Education oriented	Static analysis	Dynamic analysis	Use Visual Analytics	Architecture	Web	3D	Animation	Plugin	Single	Multiple	Multiple linked
Philosophical Research	Sys	3	15	5	2	1	1	1	1	2				
	Evol	4	27		2	3	1	7		4				
Solution Proposal	Sys	3	28	36		9	1	7	2	12	42	14	7	
	Evol		33	5	1	6	3	5	1	6	21	14	4	
	Totals	10	108	41	5	19	6	20	4	22	63	28	11	

The type of analysis used to obtain data from software systems, 103 works use static analysis and 46 dynamic analysis. The use of static analysis prevails both in the category Philosophical Research and also the category Solution Proposal, which contain 42 and 61 works respectively. Static analysis predominates in the *Evol* category, while dynamic analysis predominates in the *Sys* category.

It is striking the very small number of works that make use of VA despite

their widespread dissemination in recent years. This result is reflected in the column *Use Visual Analytics* which shows only 5 works classified in this category. One element that probably reinforces this result is that there are only a small number of papers that use multiple linked views (only 11 works in total). In this study the use of multiple linked views is considered as an essential prerequisite for a work to be classified in the VA category.

An important trend that can be observed in the results is the integration of the proposed solutions in development environments by means of plugins (22) and the use of web technology (6). The use of *3D* is also striking due to the significant number of work employing this approach (20) although its use is not recent as the case of plugins and web technology

Works which make use of multiple views form an important group (28). Some jobs that were classified in the category Views–> Multiple were placed in that category and not in the category Views–> Multiple Linked category because although they could have been classified in the former category, it could not be established with clarity whether the visualizations were linked. It should also be mentioned that the use of views was evaluated in the works classified in the category Solution Proposal but it was not evaluated in the works classified in the category Philosophical Research because of the nature of these research (*e.g.*, taxonomies, software frameworks, conceptual frameworks and classification schemes).

Section 4.3.1 presents the classification details of the category Philosophical Research, while Section 4.3.2 does the same for the category Proposal Solution.

4.3.1 Philosophical Research Studies

The number of papers classified under this rubric was 47, which were further subclassified into 11 different types of research. The resulting classification scheme, shown in table 4.3, presents details on the distribution per annum of the papers taken into account in this study (those published in a time frame from 2007 to 2013), as well as the technological elements, the type of validation and the research approach employed. Moreover, it also shows the number of papers with an educational orientation. Complementary to this scheme, Figure 4.5 shows the distribution of papers per research approach and category (*Sys* and *Evol*).

The publication of papers concentrates on the years 2008, 2009 and 2011 with no special focus on any particular research approach during those years. However, the *Novel technique* and *Reflections or discussion* approaches received special attention in general as they totaled 20 out of 47, where the former approach totaled 11 studies and the latter 9.

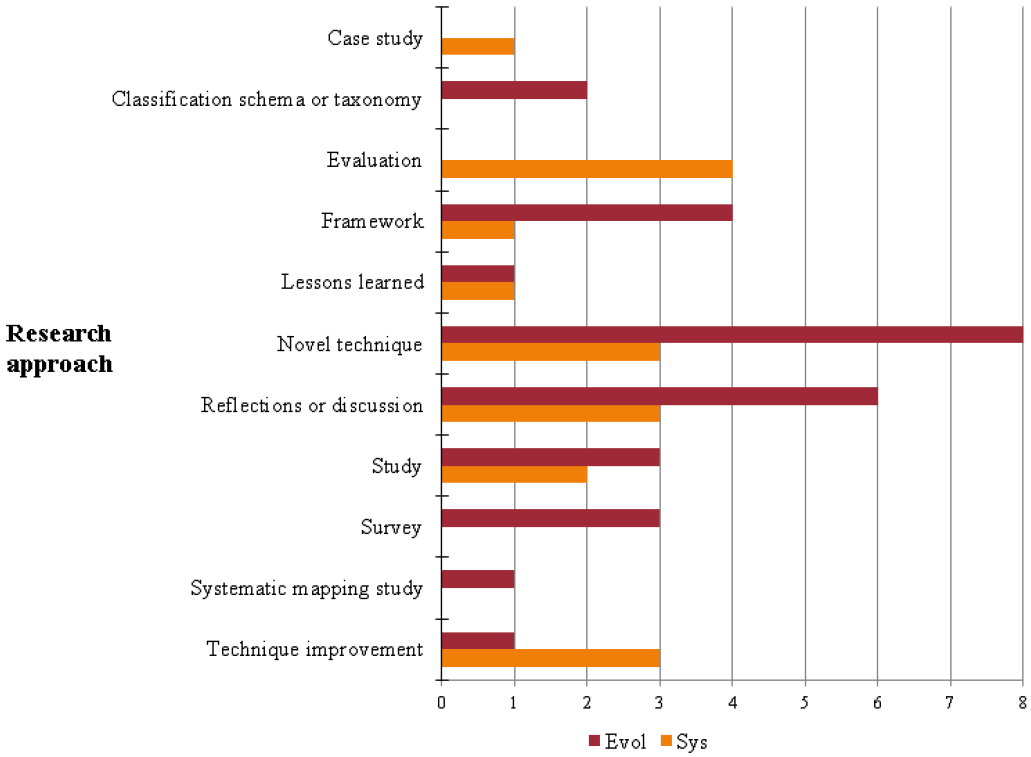


Figure 4.5: Number of papers per research approach and category (Sys and Evol).

Furthermore, the number of papers that seek to validate results is high, 22 out of 47, including the use of case studies, user studies and use cases as preferred validation methods. In this scheme the number of papers with an educational orientation is also striking (6 out of 47) in relation to the schemas presented and discussed and which are discussed in more depth later in the research. Additionally, the integration of the resulting tools into IDEs as plugins (6 out of 47) is highly relevant as is the relative importance of the use of *3D* visualizations (7 out of 47). This scheme also shows that the use of web and animation technologies is not popular with the authors of the papers classified under this category.

It is also relevant to highlight the fact that 29 out of the 47 research works were classified under the rubric *Evol* and only 18 under the rubric *Sys*. Figure 4.6 allows to observe that in all the years the number of papers published under the *Evol* category was greater than the number of works classified under the rubric *Sys*, except in the case of 2008. Figure 4.6 also allows to observe that the highest level of activity took place in the years 2008, 2009 and 2011.

Table 4.3: Research approach+papers per year+technology elements+research focus.

Research approach	Analysis type	Papers per year								Technology				Validation type							Research focus							
		2007	2008	2009	2010	2011	2012	2013	**Totals**	Web	Animation	3D	Plugin	Case study	User study	Use cases	Experiments	Tool use	Discussion	Experience report	Evaluation	Solution proposal	Validation	**Philosophical**	Architecture	Dynamic analysis	Education
Case study	Sys		1						1					1										1		1	
	Evol																										
Classification scheme or taxonomy	Sys																										
	Evol			2					2						1				1		1			2		2	1
Evaluation	Sys		2	1	1				4					1	1		1		1		1		2	2		2	1
	Evol																										
Framework	Sys		1						1									1					1	1			
	Evol		1	1		2			4			2	2		1	1			1		1		3	5	1		
Lessons learned	Sys					1			1	1								1					1	1	1		
	Evol			1					1												1						
Novel technique	Sys	1	1			1			3					1		1						3	1	3		1	
	Evol			1	2	2	2	1	8	1		2	1	5	1	3			1			7	8	8	1		
Reflections or discussion	Sys	1	1	1					3				1			1							1	3		1	1
	Evol	1		4		1			6			1	1		1						1		1	6	1		1
Study	Sys			1				1	2			1	1		1		1				1		1	1			
	Evol	1		1				1	3			1		1			1						1	1			1
Survey	Sys																										
	Evol	1				2			3		1													2			1
Systematic mapping study	Sys																										
	Evol							1	1										1					1			
Technique improvement	Sys		2				1		3						1					1			1	3			
	Evol			1					1					1									1	1			
Totals	Sys	2	8	3	1	3	1	1	18	1		1	2	3	3	2	2	2	1	1	2	3	8	15	1	5	2
	Evol	3	1	11	2	6	2	3	29	1	1	6	4	7	4	4	1		4		4	7	14	26	3	2	4
		5	9	14	3	9	3	4	47	2	1	7	6	10	7	6	3	2	5	1	6	10	22	41	4	7	6

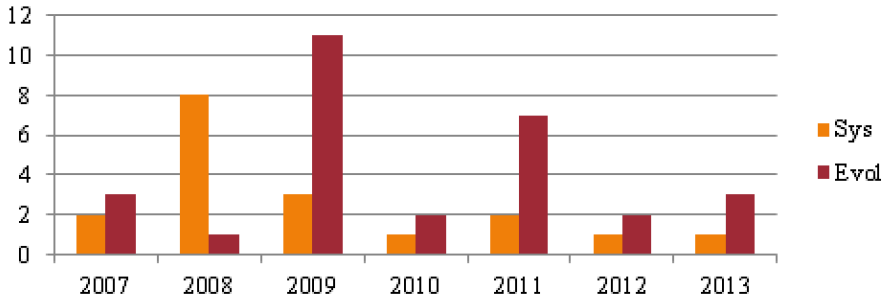


Figure 4.6: Philosophical Research: publications per year and category (Sys and Evol).

In the case of publications classified under the rubric *Evol* it is interesting to observe that publications demonstrate a wave pattern. This begins with a low level of publications in 2007, which is further diminishes in 2008 to the point where only one work was published. However, there is then considerable growth in 2009 which again falls in 2010 and then rises yet again in 2011. This pattern has continued with a fall in the number of publications in 2012 followed by a slight increase in 2013. The maximum growth in the number of publications classified under the rubric Sys, occurs in the year 2008. However, in other years the number of works does not even reach an average of two publications per year.

The works that are classified under the rubric *Novel technique* include papers which have proposed visualization techniques that have been considered innovative due to the use of new or existing elements to create a representation which possesses a high percentage of originality. Some of the visual representations presented and discussed under the rubric *Novel technique* are also referenced in the classification schemas of the next section, when other works classified under the rubric *Solution Proposal* use any of them. Finally, table B.1 in Annex B shows a correlation between research approaches and the research works studied in this section.

4.3.2 Solution Proposal Studies

The classification schemas in this section are task centric and their function is to correlate the tasks supported with the other classification criteria. The goal of these schemes is to aid the identification of data, methods and techniques used in research studies to support software development, maintenance and evolution by means of visualization and VA.

It is important to highlight that some research work considered in these classifications are aimed to support more than one task. However, the results presented in this section are focused on only the primary task that these studies aim to support (for more information see section 4.2.4). The total

number of works under consideration is 102: from which 76 support one task; 23 support two tasks; and 3 support three tasks. The correlation of the tasks supported by the research studies, their temporal focus, and the papers in which the results were published is shown in table B.2, annex B.

4.3.2.1 Distribution of papers by task addressed, year, research approach and validation type

Taking into account the tasks outlined in section 4.2.4, the scheme from table 4.4 classifies the investigations using, as a starting point, the temporal focus (*Sys* or *Evol*). It then establishes relationships between such tasks and the year of publication; the research approach (*Solution Proposal*, *Evaluation Research* and *Validation Research*); the elements of technology and the type of validation used by the research study (*Case study*, *User study*, *Use cases*, *Experiments*, *Pilot study*, *User feedback* and *Discussion*).

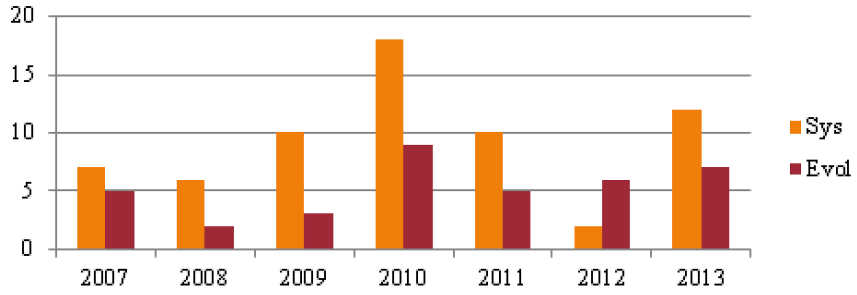


Figure 4.7: Solution Proposals: publications per year.

Furthermore, the scheme in table 4.4 indicates whether an architecture was used to implement the proposed solution and if these studies have an educational orientation.

Of the 102 publications studied, 66 were classified as *Sys* and 36 as *Evol*. As the results in Figure 4.7 demonstrate, the year in which the largest number of entries is grouped is 2010, with 2013 in second place and in third place 2011. Works classified under the rubric *Sys* follow a pattern similar to that of a Bell Curve which begins to ascend in 2009, extending through 2010 and 2011, and then begins to descend in 2012. In 2013, the curve begins to ascend again. While the publications classified under the rubric *Evol* exhibits a wave pattern which begins at an intermediate level in the *2007* and continues to oscillate in the following years. Thus, the publications in this group do not follow a stable pattern and although in the last 4 years the number of publications is greater than in the first 3 years, there is no progressive increase in this group from 2010 onwards.

Table 4.4: Task addressed+papers per year+research focus+validation type.

Task addressed		Papers per year								Focus			Tech.elements					Validation type								
	Type of paper	2007	2008	2009	2010	2011	2012	2013	Totals	Solution proposal	Evaluation research	Validation research	Web	3D	Plugin	Animation	Case study	User study	Use cases	Experiments	Pilot study	User feedback	Discussion	Architecture	Education
Detect design flaws	Sys		1		1			1	3	3		3		1				1	1	1					
	Evol				1				1	1	1	1							1						
Distributed systems comprehension	Sys	1	1	1		1		1	5	5	2	4		1					3			1	2	2	
	Evol																								
Improve software quality	Sys					1	1	1	3	3		3	1		3		2	1							
	Evol					1			1	1		1					1							1	
Improve source code security	Sys			1	1				2	2		1		1				1	2					1	
	Evol																								
Memory allocation analysis	Sys	1		1	3	1		2	8	8	1	6				1	3					1	4		
	Evol																								
Multithreading execution analysis	Sys				2			3	5	5		4				1	1		1				3		
	Evol																								
Parallel execution analysis	Sys																								
	Evol		1						1	1															
Performance analysis	Sys			1	2				3	3	1	2			2			1	2						1
	Evol							1	1			1							1				1		
Program execution analysis	Sys	1		4	4	2	1		12	9	3	8			3		4	3	1	2	1		1		2
	Evol				1				1	1		1							1						
Software design and modeling	Sys	1	1		1			1	4	4		2						2			1		1		
	Evol																								

Continued on next page.

Table 4.4 Task addressed+papers per year+research focus+validation type – continued from previous page.

Task addressed	Type of paper	Papers per year								Focus			Tech.elements				Validation type								
		2007	2008	2009	2010	2011	2012	2013	Totals	Solution proposal	Evaluation research	Validation research	Web	3D	Plugin	Animation	Case study	User study	Use cases	Experiments	Pilot study	User feedback	Discussion	Architecture	Education
Software ecosystem comprehension	Sys																								
	Evol				1	1			2	2		1	2						1				1	2	
Support reverse engineering	Sys		1			1			2	2	1	2		1				2	2					2	
	Evol			1					1	1	1												1		
System analysis and understanding	Sys	2	1	1	1	1			6	6		5		1	2		1	2	2				1		
	Evol			1		1	2		4	3		3			2		1	1	1	1					
System refactoring	Sys							1	1	1		1					1								
	Evol				1		1		2	2	1	2					1		1				1		
System testing	Sys		1			1			2	2		2					1						1	1	
	Evol				1				1	1		1					1								
Team awareness and collaboration	Sys				1				1	1		1						1							
	Evol	2	1		2	1	2	2	10	9	1	8		1	3		4	2	2				1	2	
Understand dependencies	Sys	1				1			2	2		2												1	
	Evol																								
Understand software changes	Sys					1			1	1		1			1				1				1	1	
	Evol	1		1					2	2		2					1						1		
Understand system architectures	Sys			1	1			2	4	4		4		2	1		1	1	1				2	1	
	Evol	2			3	1	1	4	11	9	1	10	1	4	1	1	4	2	3				4	1	
Totals	Sys	7	6	10	17	10	2	16	64	61	8	51	1	7	12	2	14	15	16	3	2	2	17	9	3
	Evol	5	2	3	10	5	6	3	38	34	5	31	3	5	6	1	13	5	11	1			10	6	
		12	8	13	27	15	8	19	102	95	13	82	4	12	18	3	27	20	27	4	2	2	26	15	3

Figure 4.8 shows a correlation between the tasks and the number of research works classified under the categories *Sys* and *Evol*. The aim is to provide an outlook of the interest shown by researchers in these tasks and the temporal analysis types. According to the results that can be observed in table 4.4 researchers showed continuing interest in five tasks. These are the following:

* Understand system architectures (15).
* Program execution analysis (13).
* Team awareness and collaboration (11).
* System analysis and understanding (10).
* Memory allocation analysis (8).

Moreover, *Understand system architectures* and *Team awareness and collaboration* are the tasks with the highest percentage of studies devoted to system evolution.

Similarly, a small number of works perform an evaluation research of a proposed solution that has been published in other research paper, whereas most of them contemplate some sort of evaluation, of which the most common are case studies, use cases and user studies. It can be observed, that another method that was extensively used in order to argue for the validity of the proposals was the use of reasoned discussion. It is worth noting that such discussions at times are exhaustive and in most cases they present weak and subjective arguments that are not supported by empirical evidence.

The use of web technology in implementing proof of concept implementations or implementations of finished tools is very limited, with only 4 out of 102 works making use of these techniques. However, the number of research papers which took into account the use of animation and *3D* is more significant. In the first case, 12 papers use animation; and in the second, 18 use *3D* technology. In the case of *3D* technology, its use is generally more common in displaying information and has become more widespread as software packages began offering more graphic applications. However, its use is not sufficiently widespread because of a number of limitations which include occlusion and handling. Similarly, the incorporation of the implemented visual tools as an IDE plugin is relatively high: in 18 works was the development of a plugin seen as a viable solution.

It should be highlighted that this scheme shows that 15 of the 102 works classified under the rubric of *Solution Proposal*, offer details and explanation concerning the architecture used. There are, however, a small number of works which seek to support learning and teaching of programming and debugging with the use of visualization as only 3 research studies offer some support for educational purposes.

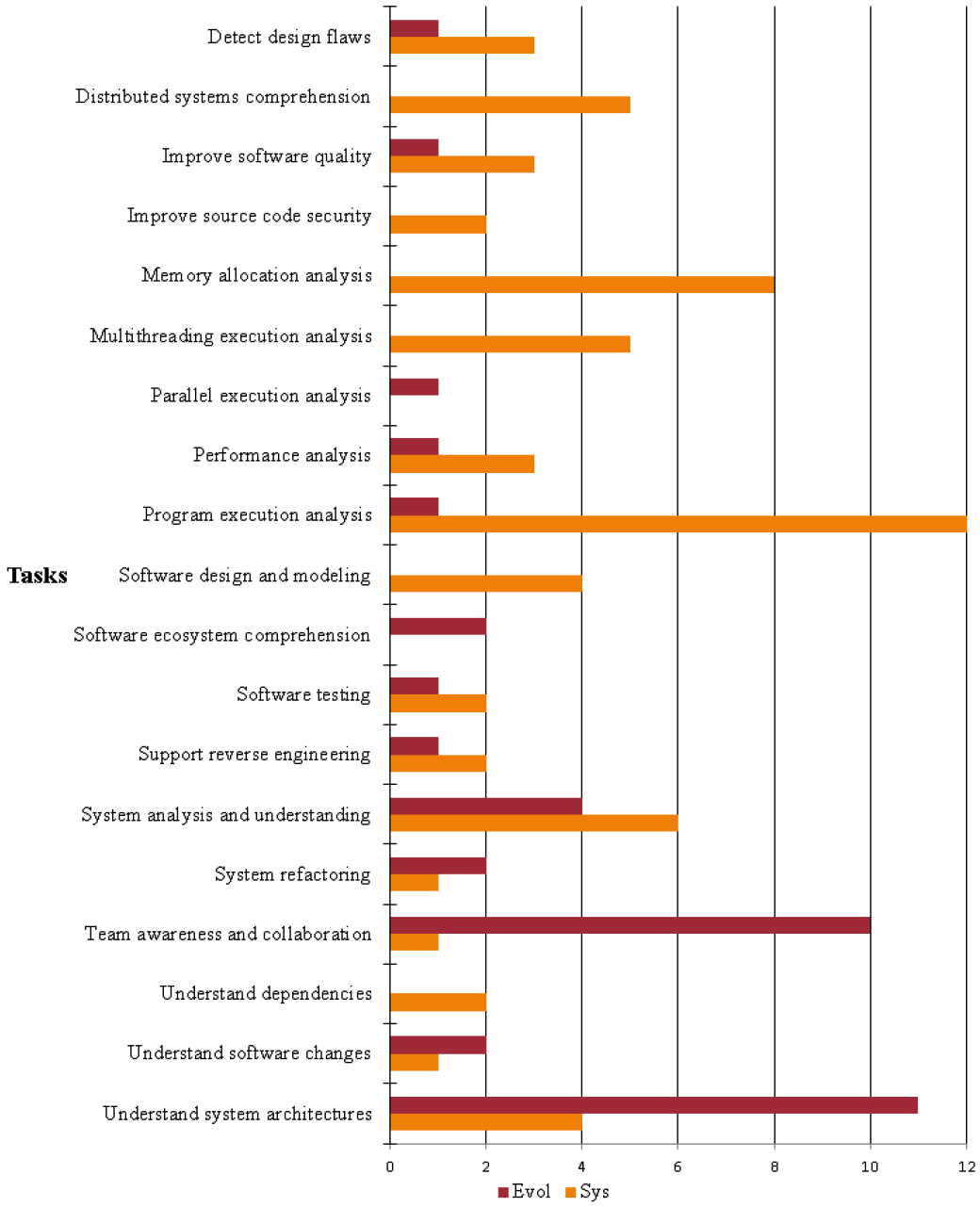


Figure 4.8: Correlation of tasks with the number of papers published per category (Sys and Evol).

4.3.2.2 Classification by task and data type

The classification scheme in table 4.5 shows in detail the relationship between the data used and the tasks identified, as well as an indicator regarding the methods employed to obtain this data (static or dynamic). The information contained in this scheme may facilitate further research as well as the design of new tools.

Table 4.5: Task addressed+data used+runtime data.

Tasks addressed			Visualizations																						Runtime data
	Analysis type	Number of works	Bug tracking issues	Class hierarchy	Code clones	Code smells	Coupling	Data flow details	Data structures	Dependencies	Execution traces	Item relationships	Logical coupling	Memory allocation	Metrics	SCM Metadata	Socio-technical relationships	Source code changes	Source code slicing	Structure	Testing data	Vocabulary	UML diagrams	Workload and performance	Others	
Detect design flaws	Sys	3				2									1					1						
	Evol	1	1			1											1									
Distributed systems comprehension	Sys	5						2				2								2				1		3
	Evol																									
Improve software quality	Sys	3										1			2					1	1					
	Evol	1													1					1						
Improve source code security	Sys	2									1				1											1
	Evol																									
Memory allocation analysis	Sys	8							3		1			8												8
	Evol																									
Multithreading execution analysis	Sys	5									5															5
	Evol																									
Parallel execution analysis	Sys																									
	Evol	1																						1		1
Performance analysis	Sys	3									3															3
	Evol	1									1															1
Program execution analysis	Sys	12						1		1	10															12
	Evol	1									1															1
Software design and modeling	Sys	4																					4			1
	Evol																									

Continued on next page.

Table 4.5 Task addressed+data used+runtime data – continued from previous page.

Tasks addressed	Analysis type	Number of works	Visualizations																								
			Bug tracking issues	Class hierarchy	Code clones	Code smells	Coupling	Data flow details	Data structures	Dependencies	Execution traces	Item relationships	Logical coupling	Memory allocation	Metrics	SCM Metadata	Socio-technical relationships	Source code changes	Source code slicing	Structure	Testing data	Vocabulary	UML diagrams	Workload and performance	Others	Runtime data	
Software ecosystem comprehension	Sys	2																									
	Evol	2								1					2	1	2					1					
Support reverse engineering	Sys	2								1	1				1			1		1						1	
	Evol	1								1					1			1		1							
System analysis and understanding	Sys	6				1				2					1					1					2		
	Evol	4		1						2					2			1	1	1					2		
System refactoring	Sys	1			1																						
	Evol	2					1						1														
System testing	Sys	2								1	1									1	1					2	
	Evol	1															1				1						
Team awareness and collaboration	Sys	1														1		1									
	Evol	10		1						1	1				2	2	6	3		2		3			1	1	
Understand dependencies	Sys	2								2					1												
	Evol																										
Understand software changes	Sys	1								1					1												
	Evol	2	1								1						1			1						1	
Understand system architectures	Sys	4		2						1					2					3							
	Evol	11		1	1		1			1			1		6			1		9		1					
Totals	Sys	64		2	1	3		3	3	9	24	4		8	10		1	2		10	2		4	1	2	37	
	Evol	38	2	3	1	1	2		3	6	4	1	2		14	3	11	6	1	16	1	5	4	2	3	4	
		102	2	5	2	4	2	3	3	15	28	5	2	8	24	3	12	8	1	26	3	5	4	1	5	41	

The similarities in the types of data used by the works classified under the rubrics *Sys* and *Evol* can enable the development of works that focus on supporting research with a greater capacity for analysis in both temporal and general terms. However, these similarities are not sufficiently well-defined to allow the design of research projects of the scope described, and so the schemes in tables 4.6 and 4.7 are thus both complementary and useful in this context. The data types that this research determined that were used by the works analyzed are:

Bug tracking issues: These include the description of the problem, registration date, the level of urgency assigned; the user who reported the incident, the person responsible for handling the incident and the change history of the incident.

Class hierarchy: Includes details about the ascendants and descendants of a particular class of or item of software.

Code clones: These are segments of source code that have been replicated (copied) in different software items. This renders the maintenance of the systems more difficult because all the copies of the segment must be changed and it is necessary to find their location within the project structure. The information about the code clones must thus include the code segment and its location in the project structure. Code clones are a particular type of code smell.

Code smells: They are symptoms of inadequate design or system design problems which are detected by means of metrics. These symptoms may cause maintenance problems. The following are typical symptoms: excessive length of classes or methods, excessive complexity and cohesion, and dependence on access to data by means of external classes [Lanza 2005b].

Coupling: They are measures of the dependencies between classes, according to the methods that are called [Yang 2007, Briand 1999].

Data flow details: This type of data offers details about the manner in which data flows through the system.

Data structures: These contain details about the dynamic behavior of data structures at runtime and how these organize and manipulate data elements.

Dependencies: They are details of the dependencies which exist between system elements. These dependencies are similar to those measured by

coupling metrics. Thus information regarding the relationships which are established is included but not information regarding the metrics.

Execution traces: This is information about what happens in the system when it executes a task. It includes details about the parts of the program which are run; the invocation of classes and methods; as well as data access and the passing of data between the different elements that are involved.

Item relationships: Dependencies and inheritance are both types of relationships between software items. However, this type of data also includes the implementation of interfaces and other types of relationships that have not been clearly defined by researchers in their papers.

Logical coupling: This type of data contains details of the dependencies of software items according to the co-change patterns that are revealed when the history of revisions is analyzed [Gall 1998, D'Ambros 2009b].

Memory allocation This consists of information about memory allocation to processes and the duration of the above assignment.

Metrics: They are the result of measuring the characteristics of a software system, such as the complexity and size of the elements of which it is composed [Laird 2006].

SCM Metadata: This stores details about revisions and changes to the system. Such details include the revision number, the name of the programmer who initiated the revision, the date and time of the revision, and the software elements that were affected by the revision.

Socio-technical relationships: In this context, this term refers to the details of the network of relationships which is formed between software items, programmers and the collaboration between programmers which takes place whilst carrying out changes in the process of software development and maintenance. [Scacchi 2004, Valetto 2007].

Source code changes: This provides specific details of the changes made in each particular software item during each revision.

Source code slicing: This is the part of the program whose behavior is worthy of study [Weiser 1981]. The works in this study used this technique to extract details of the portions of source code that had been changed during the evolution in order to study the behavior of the changes.

Structure: This contains information on how the project is structured (packages, sub-packages, classes and interfaces) and also information about how the structure undergoes changes.

Testing data: These are the results (which are stored in logs or databases) obtained when testing the system automatically with the use of specialized tools.

Vocabulary: This is the vocabulary used to denominate the elements of the software system such as; software items, methods, parameters, and identifiers, as well as the vocabulary used in the comments.

UML diagrams: These are the diagrams Unified Modeling Language (UML) extracted automatically from the software project by means of reverse engineering and the use of specialized tools. The metadata of these diagrams are used as data in order to analyze the system and provide visual elements that use additional elements and seek to provide better support for the process of development, maintenance and evolution.

Workload and performance: These data provide information about the workload generated by the execution of systems and performance measures at runtime.

Others: The data types whose use was limited by the studies analyzed are grouped. Among these types of data are included the evolution of project documentation, software features, and details of aspects-oriented systems [Kiczales 1997].

Of the 102 works studied, 61 used static analysis to obtain data whereas the other 41 used dynamic analysis. Figure 4.9 shows the correlation between the number of research works in the categories *Evol* and *Sys* and the types of data used. This figure allows to observe at a glance the temporal and general usage of each type of data. As one would expect, data obtained by means of dynamic analysis was used in works that supported tasks like:

* Multithreading execution analysis.
* Parallel execution analysis.
* Program execution analysis.
* Memory allocation analysis.
* Debugging support.
* Performance analysis.

With regard to the other tasks, most of them used data obtained using static analysis, although some tasks use data obtained with both types of analysis (static and dynamic).

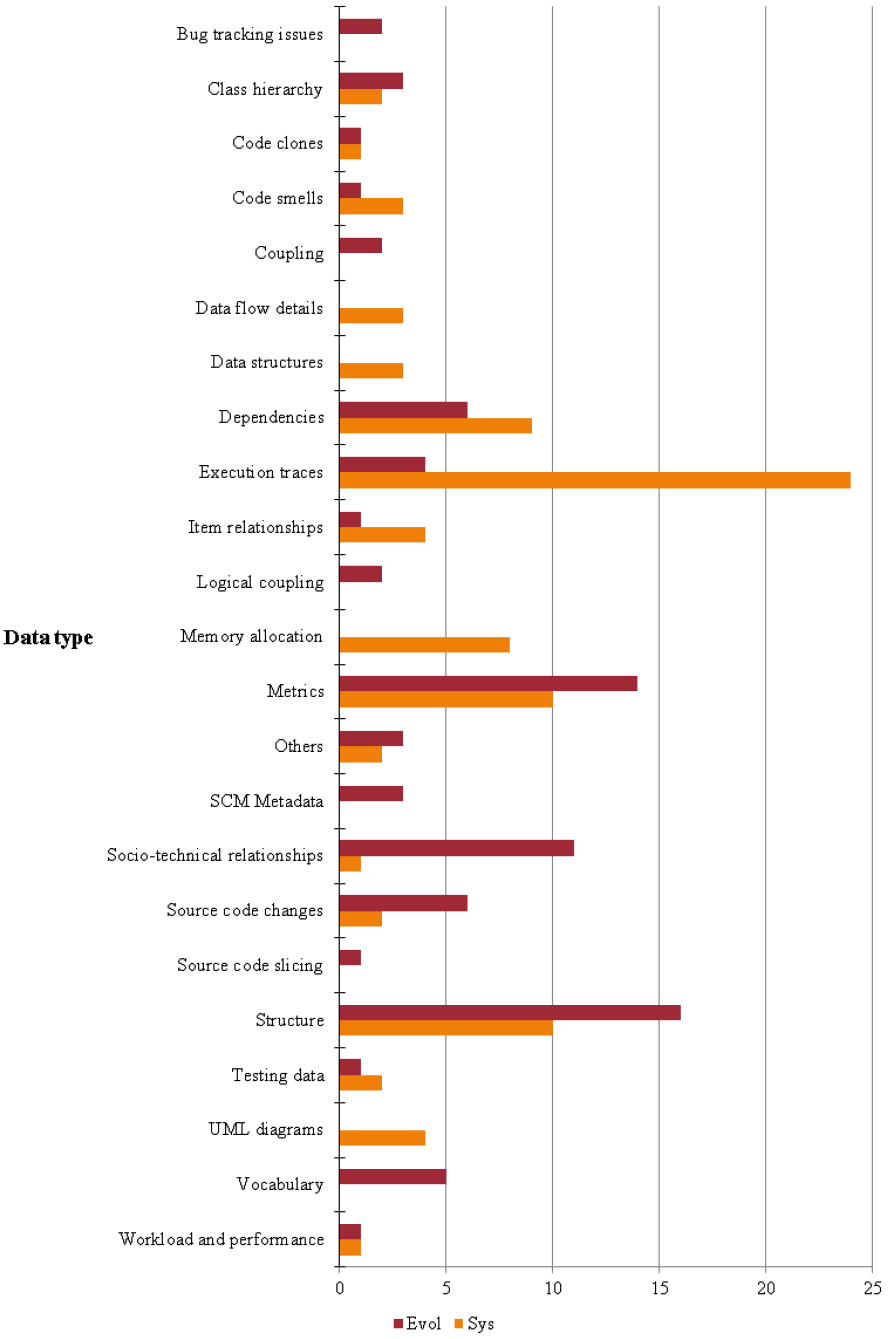


Figure 4.9: Correlation of the types of data used by the research works studied and the the temporal focus of these works (Sys and Evol).

The five types of data that, in general, received more attention in the studies analyzed were:

* Execution traces (28).
* Structure (26).
* Metrics (24).
* Dependencies (15).
* Socio-technical relationships (12).

It ought to be recalled that 64 works were classified under the rubric *Sys* while 38 were classified under the rubric *Evol*. More specifically, the three categories classified under the rubric *Sys* which aroused most interest in researchers were:

* Execution traces (24).
* Metrics (10).
* Structure (10).

With regard to the *Evol* category, the three types of data most frequently used were:

* Structure (16).
* Metrics (14).
* Socio-technical relationships (11).

4.3.2.3 Classification by task, technique and visualization

The purpose of this classification scheme (see table 4.6) is to relate the tasks with the visualizations and the type of views that were used in the research. This was done in order to provide indicators about possible visual design patterns which may be useful in developing new research and design solutions for the aforementioned tasks.

The name of visualizations in the schemes (in some cases) is a label to identify a group of the same type of visualization such as, for example, basic charts and graphs. In the category of basic charts, histograms, bar charts, pie charts and box plots were included whereas in the case of graphs, directed indirected and weighted graphs as well as trees were taken into account. Accordingly, the following visualization types were used by the research works:

Basic charts: This group of visualization types include histograms, bar charts, pie charts, box plots.

Code browser: A window that displays source code with some basic navigation elements and color.

Code city: A visual representation of project structure and metrics using the metaphor of a city.

Color lines and map: A group of visual representations that use colored lines and dots to represent and highlight patterns.

DSM[3]: A visualization that uses a matrix layout with the names of software items in rows and columns that show their dependencies with marks in the intersecting cells.

Hierarchical Edge Bundles (HEB): A radial visualization that organizes concentric rings according to the project structure and shows relevant associations with lines which interconnect ring segments.

Events lifeline: A simple visualization that shows where and event began and when it finished.

Graphs: A family of different types of common graphs used in computer science.

Heatmap: A green-red scaled color dot matrix representation that show incidence patterns.

Matrix layout: A visualization that uses a squarified layout with rows, columns and cells that show details of cells.

Parallel coordinates: A scalable multivariate visualization that shows data values in the y dimension and relationships to several other variables in the x dimension.

Parallel node-link: A visualization similar to the parallel coordinates that shows the name of software elements in the y dimension and name of data types in the x dimension.

Polymetric views: These show a large number of software items organized in a tree structure, where the software items are represented by boxes whose attributes correspond to metrics.

Radial graph: A graph that is depicted using a radial layout.

Software cartography: The vocabulary used during the project evolution is mapped into layered mountains according to its usage in time.

Stacked chart: Represents data values using a color fill strap with varying thickness as it is depicted in the x dimension. Straps are piled one on top of each other.

Sunburst: A visualization technique that depicts hierarchies and shows details on the lower levels of the hierarchy as elements are selected in the higher levels.

Tag cloud: This visualization is formed by the vocabulary used during the software project evolution, where words are sized according to the frequency of their use.

Timeline: This visualization technique is used to plot events or activities in a linear fashion according to their temporal properties.

Treemap: A squarified or circular visualization technique used to display the weights of the elements of a hierarchy or the proportions associated with variables.

UML variant: This visualizations uses visualization elements that are present in UML diagrams and additional visual features to create richer representations.

Others: Some examples are classification bins, dendrogram, *DotPlot*, organic visualization, ownership map, icicle plot and tree forest.

The correlation of the types of visualizations listed above with the number of research works published under the rubrics *Evol* and *Sys* is shown in Figure 4.10. Accordingly, the five most common type of visualizations are:

* Graphs (45).
* Basic charts (17).
* Events lifelines (14).
* Timelines (12).
* Code browser (11).

While the three most commonly used visualization types classified under the rubric *Sys* were:

* Graphs (32).
* Event lifelines (13).
* Basic charts (8).

The visualizations most commonly used for works classified under the rubric *Evol* were the following :

* Graphs (13).
* Basic charts (9).
* Timelines (7).

In computer science and software engineering, the use of graphs is common, and thus programmers are accustomed to their use and interpretation. In fact, their widespread use (indicated by the results shown in this study) was predictable.

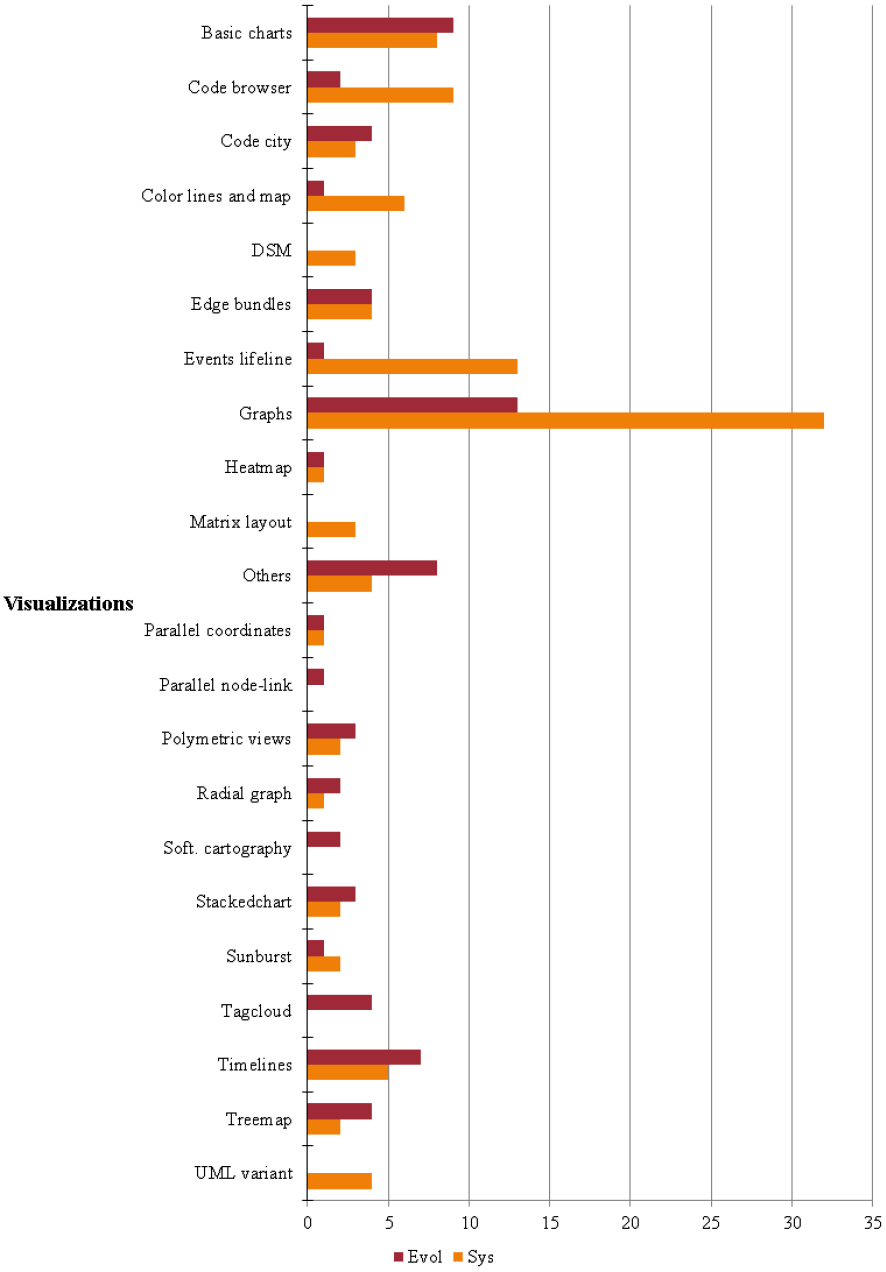


Figure 4.10: Correlation of visualization types and the number of papers published by category (*Evol* and *Sys*).

The structure of software systems is like a tree, where the elements are organized in modules or packages, but also by hierarchical relationships

in terms of inheritance. Formally, a tree structure is a kind of graph. Additionally, software elements are interrelated: they make use of the functionality and attributes of other software items. There is thus a naturally occurring graph of relationships between different software items. In addition, the same source code forms an implicit graph whose elements transition from one state to other to originate a finite state machine that is represented by a graph.

The importance of graphs in general is not restricted to the field of computational systems. In all the elements that exist in the world created by humans and nature there exist relationships that usually are depicted using the visual form of a graph. It is possible, for example, that the person making a visual representation of a graph by chance on a piece of paper does not know the technical name for such a representation, but knows how to both draw and interpret it. Thus, in the field of visualization graphs are useful, not only for their ability to represent knowledge by means of association and relationships between elements of diverse types. Moreover, graphs are easy to understand for which, as has been mentioned, makes them useful to general users. The use of graphs in the present field of study are diverse. They were used in the majority of tasks: equally in the case of studies classified under the rubric *Sys* as those classified under the rubric *Evol*.

Basic charts were used to display statistics and metrics. As these graphics are both simple and well-known, their interpretation is quick and easy thus satisfying what is expected from a good visual design. In the majority of cases, this type of chart was used and its use turned out to be appropriate. The use of more complex visual designs for the type of data which these charts typically represent may render the process of interpreting information confusing and complex. The use of this type of visualization was also very widespread, as the case of graphs, and they were used in work falling into both of the categories *Sys* and *Evol*.

The use of *Event lifelines* is dominated by works classified as *Sys* because 14 works used this representation and 13 fell into the *Sys* category. The works which used it made use of dynamic data obtained at runtime and use was mainly related to the duration of the execution of a task or memory allocation. The single work that used this visualization and was classified as *Evol* represented an isolated case related to the time associated with changing the source code of software items.

It should be noted that the works that used data obtained by dynamic analysis at runtime were classified as *Sys* because they referred to the execution of a project revision, which in many cases is the current revision. However, studies that also used data collected with the application of dynamic analysis to several project revisions were classified as *Evol*.

Table 4.6: Task addressed+visualization+view design.

Tasks addressed			Visualizations																					Views			
	Analysis type	Number of works	Basic charts	Code browser	Code city	Color lines and map	DSM	Edge bundles	Events lifeline	Graphs	Heatmap	Matrix layout	Parallel coordinates	Parallel node-link	Polymetric views	Radial graph	Soft. cartography	Stacked chart	Sunburst	Tag cloud	Timelines	Treemap	UML variant	Others	Single	Multiple	Multiple linked	Visual analytics
Detect design flaws	Sys	3		2	1																				3			
	Evol	1	1																		1					1		
Distributed systems comprehension	Sys	5	1						1	3								1			1				4	1		
	Evol																											
Improve software quality	Sys	3								2									1			1			2	1		
	Evol	1	1																		1			1		1		
Improve source code security	Sys	2	1							1												1			1		1	
	Evol																											
Memory allocation analysis	Sys	8				1			3	2						1								1	8			
	Evol																											
Multithreading execution analysis	Sys	5				1			3	1								1	1		3			1		3	2	
	Evol																											
Parallel execution analysis	Sys																											
	Evol	1	1															1			1					1		
Performance analysis	Sys	3	1						3	2															1	1	1	
	Evol	1								1					1						1					1		
Program execution analysis	Sys	12	1	3		2		2	3	7											1		1	1	4	6	2	
	Evol	1								1									1					1		1		
Software design and modeling	Sys	4								3													1		4			
	Evol																											

Continued on next page.

Table 4.6 Task addressed+visualization+view design. – continued from previous page.

Tasks addressed	Analysis type	Number of works	Visualizations																						Views			
			Basic charts	Code browser	Code city	Color lines and map	DSM	Edge bundles	Events lifeline	Graphs	Heatmap	Matrix layout	Parallel coordinates	Parallel node-link	Polymetric views	Radial graph	Soft. cartography	Stacked chart	Sunburst	Tag cloud	Timelines	Treemap	UML variant	Others	Single	Multiple	Multiple linked	Visual analytics
Software ecosystem comprehension	Sys																											
	Evol	2	2							1								2		1	2					2		
Support reverse engineering	Sys	2	1			1		1		1													1		1	1		
	Evol	1	1					1		1																	1	
System analysis and understanding	Sys	6	1	2	1					2	1	1	1										1		3	3		
	Evol	4		1				1		2			1		1							3		1	1	2	1	1
System refactoring	Sys	1																						1	1			
	Evol	2								2															2			
System testing	Sys	2								1		1													2			
	Evol	1								1															1			
Team awareness and collaboration	Sys	1		1						1																1		
	Evol	10	2	1				1		5	1				1	1	1			3	1			2	7	2	1	
Understand dependencies	Sys	2		1			1			1															1		1	
	Evol																											
Understand software changes	Sys	1	1			1		1																			1	
	Evol	2	1			1			1	1																2		
Understand system architectures	Sys	4			1					2		1			2										2	2		
	Evol	11	1		4		2	1		1				1		1	1					1		3	9	1	1	
Totals	Sys	64	8	9	3	6	3	4	13	32	1	3	1		2	1		2	2		5	2	4	4	37	21	8	
	Evol	38	9	2	4	1		4	1	13	1		1	1	3	2	2	3	1	4	7	4		8	20	12	4	1
		102	17	11	7	7	3	8	14	45	2	3	2	1	5	3	2	5	3	4	12	6	4	9	57	33	12	1

Visualizations of the type *Code Browser* were predominantly used for work falling into the category *Sys* (9 out of 11). The use of this visualization (like those mentioned above) is as diverse as the number of tasks for which it was used. However, it was generally used to directly inspect the source code with the support of colors, arrows or symbols which indicate characteristics, conflicts or problems.

Contrary to what might be expected by simple deduction, *Timelines* was used almost equally for works classified in *Sys* and *Evol*. It was used in 5 cases which fell into the former category and in 7 cases which fell into the latter. *Timelines* was used in works classified as *Sys* to represent data concerning program execution and events like memory allocation, for a single revision of a system. This visualization was intended to facilitate navigation of data obtained from multiple revisions of the system and the time interval represented often last for months or years, when it was used for works in the *Evol* category.

The usefulness of details in table 4.6 could be illustrated with an example of a simple design for the task *Distributed Systems Comprehension*. The design may include *Basic charts* to present statistics related to memory allocation or disk writing during the execution of routines. *Event lifelines* can be used to represent the duration of each one of these tasks and *Graphs* to show the relationships that exist between software elements, and *Timelines* in order to show the execution times and allow selective browsing examining execution intervals which are of interest. The earlier exercise can be performed with each of the tasks according to the type of knowledge which is sought to be obtained and the data available (or which was planned to be obtained).

In the context of this thesis, a *view* is defined as a separate visualization or a combination of several visualizations which can provide joint results as they are inter-linked between themselves[4].

Taking into consideration the results of the schema in table 4.2, the number of papers which argue for the use of VA is small, totaling only 5 out of the 149 studies analyzed. Of these 5 works, 4 were classified under the rubric *Philosophical research*, and one isolated work was classified under the rubric *Solution proposal* (see Table 4.6). This work is associated with the task *System analysis and understanding* and was classified under the *Evol* rubric. In this context, it should be recalled that a desirable element in VA tools is the use of multiple linked views.

With regard to the types of visualizations employed, 102 works were

[4]A linked view is a combination of two or more visualizations where the results of the visualizations are affected by the interaction which the user carried out with the associated visualizations. The bond or copula between visualizations can be unidirectional or bidirectional.

classified as *Solution proposal* and from these, 57 works used *Single views*, 33 *Multiple views* and 12 *Multiple linked views*. The tasks with the highest number of works that used *Multiple views*, *Multiple linked views* were:

* Program execution analysis (13).
* System analysis and understanding (10).
* Multithreading analysis (5).

It draws attention that all the works which seek to support the task *Multithreading execution analysis* make use of multiple views (2 research works of 5 also used the views in a linked fashion). It is also interesting to note that of the 6 works with two or more views classified as oriented to support the task *System analysis and understanding*, 3 works were classified under the rubric *Sys* and the other 3 as *Evol*, with one of the latter classified in the category VA.

4.3.2.4 Classification by visualization and data type

The schema in table 4.7 presents the relationship between visualizations and data types used by the research works. These relationships allow us to see the data patterns which the visualizations represent and thus serve as an orientation in the design of future research or new research tools which support the process of developing and maintaining software.

Some patterns that can be extracted by analyzing the rows in the aforementioned table are, for example, those related to *Code city*, *Edge bundles* and *Event lifelines*.

In the case of *Code city*, it can be observed that it is a useful visualization which allows the representation of data metrics and project structure for one or more of the revisions and are thus classified in both the categories *Sys* and *Evol*. The use of this visualization to represent these data types is natural because it was designed for this purpose, but the pattern may serve as guidance for a researcher in the future who is confronted with the problem of the design of a tool without detailed knowledge of the field.

HEB is useful to represent the structure and dependencies between software items, as well as execution traces, software item relationships, and the presence of code clones in software items. The use of *Edge bundles* to represent dependencies between elements may be obvious to a specialist in the area, but its use in the detection of code clones is not so obvious at first glance. In the case of code clones, the visualization is useful in indicating elements where the same fragment of code is copied. Furthermore, it displays the relationships between the software items that contain code clones, if such relations do in fact exist.

Table 4.7: Visualization+data used.

Visualizations		Data elements represented																						
	Analysis type	Number of works	Bug tracking issues	Class hierarchy	Code clones	Code smells	Coupling	Data flow details	Data structures	Dependencies	Execution traces	Item relationships	Logical coupling	Memory allocation	Metrics	SCM Metadata	Socio-technical relationships	Source code changes	Source code slicing	Structure	Testing data	Vocabulary	UML diagrams	Workload and performance	Others
Basic charts	Sys	8									2				4					1					
	Evol	9	3												5	1	1							1	1
Code browser	Sys	9				2				2	3														
	Evol	2																3	1						
Code city	Sys	3													3					3					
	Evol	4													6					5					
Color lines and map	Sys	6								1	3			1	1			1							
	Evol	1	1															1							1
DSM	Sys																								
	Evol	3								3										3					
Edge bundles	Sys	4								3	1														
	Evol	4			1					3		1								3					
Event lifelines	Sys	13									11			3											
	Evol	1									1														
Graphs	Sys	32		1			1	3	2	6	11	1			3		1			2				3	1
	Evol	13		2						4	1		1			2	6			4					2
Heatmap	Sys	1								1															
	Evol	1									1						1	1		1					
Matrix layout	Sys	3										1			1					1					1
	Evol																								
Parallel coordinates	Sys	1													1										
	Evol	1													1										
Parallel node-link	Sys																								
	Evol	2		1	2	1	1	1					1	1											

Continued on next page.

Table 4.7 Visualization+data used – continued from previous page.

Visualizations	Data elements represented																							
	Analysis type	Number of works	Bug tracking issues	Class hierarchy	Code clones	Code smells	Coupling	Data flow details	Data structures	Dependencies	Execution traces	Item relationships	Logical coupling	Memory allocation	Metrics	SCM Metadata	Socio-technical relationships	Source code changes	Source code slicing	Structure	Testing data	Vocabulary	UML diagrams	Workload and performance	Others
Polymetric views	Sys	2													2										
	Evol	3									1				2										
Radial graph	Sys	1							1																
	Evol	2								1										1					
Software cartography	Sys																								
	Evol	2													2								2		
Stacked chart	Sys	2									2														
	Evol	3													2		1								2
Sunburst	Sys	2																		1	1				
	Evol	1									1														
Tag cloud	Sys																								
	Evol	4																1					3		
Timelines	Sys	5									5														
	Evol	7	1								1				2		3		1				1		2
Treemap	Sys	2								1										1	1				1
	Evol	4													3			1		3					
UML diagrams	Sys	4													1					1		4			1
	Evol																								
Other																									
Cartesian and radial icicle plot	Sys																								
	Evol	1									1														
Class blueprint	Sys																								
	Evol	1		1						1															
Data flow chart	Sys	1						1																	
	Evol																								

Continued on next page.

Table 4.7 Visualization+data used – continued from previous page.

Visualizations	Analysis type	Data elements represented: Number of works	Bug tracking issues	Class hierarchy	Code clones	Code smells	Coupling	Data flow details	Data structures	Dependencies	Execution traces	Item relationships	Logical coupling	Memory allocation	Metrics	SCM Metadata	Socio-technical relationships	Source code changes	Source code slicing	Structure	Testing data	Vocabulary	UML diagrams	Workload and performance	Others
Dendrogram	Sys																								
	Evol	1									1														
Dotplot	Sys																								
	Evol	1									1														
Evolution radar	Sys																								
	Evol	1											1												
Organic visualization	Sys																								
	Evol	1												1											
Ownership map	Sys																								1
	Evol	1															1								
Smells view	Sys	1				1																			
	Evol																								
Software visualization story	Sys																								
	Evol	1															1								
Tree forest	Sys														1	1				1					
	Evol	2																							

Event lifelines is mainly used in the context of analysis of the execution of a program and the allocation of memory to tasks. This type of visualization is thus associated with execution traces and memory allocation, and most research papers were classified in the category *Sys*. However, this kind of visualization can also be used to indicate the duration of any event or task, in a form similar to that of *Gantt charts*.

In this scheme it can be noted that the use of *Graphs* has special relevance. As had been mentioned in Section 4.3.2.3 a large number of works make use of *Graphs* because of their ability to represent knowledge in terms of relationships and partnerships. This makes it inevitable that they are used to represent various data types about whose elements it is sought to extract details about their relationships and interactions. In the corresponding row for *Graphs* in the scheme (see table 4.7), it can observed that they were used in the representation of most of the data types which were used in the works that were studied.

Therefore, when analyzing the row corresponding to *Graphs*, it is not possible to observe a clear pattern about their use with certain types of data. However, their flexibility with regard to the representation of different data types must be emphasized. A similar case is that of *Basic charts*, which are used to represent various classes of statistical and quantitative data, which makes it difficult to discern a clear usage pattern in the representation of specific data types.

The approach used in the preceding discussion permits the extraction of patterns with regard to the visualizations and the data types that they represent, using as a reference the rows of she schema. However, if the columns and data types are used as a reference it is possible to identify the visualizations that are useful to represent a particular data type. An example is the representation of the data related to the project structure, which can be represented by *Code city*, *DSM*, *Edge bundles*, *Graphs*, *Heatmaps*, *Matrix layout*, *Radial graph*, *Sunburst* and *Treemap*. The use of any of these representations will depend on the variables related to the structure that the tool designer seeks to represent. To represent the structure and metrics *City code* can be used. To represent the structure and dependency between the elements *DSM* and *Edge bundles*, for example, are useful.

It is thus possible to extract a large number of patterns from this scheme (in table 4.7) about the types of data that can be represented by a single visualization and the visualizations that can represent a particular data type.

4.4 Discussion

It is important to note that the identification of tasks on the basis of the contents of the work was not a straightforward process because a large number of the research works did not clearly state the problem they sought to resolve, nor the goals and objectives they sought to pursue. Similarly, many woks did not describe the data used nor how the data were represented by the visualizations they used.

This occurs not only with tasks, but also with visualizations. A large number of papers do not describe the characteristics of displays adequately or provide clear information about the possibilities of interaction or the opportunities they offer to facilitate the discovery of knowledge. These problems extend to the description of the interaction between the visualizations (when multiple visualizations are used) and only in a few cases do the papers explain whether the visualizations are linked. Given these limitations, it was not possible to determine with clarity in the case of a significant number of research works whether the visualizations operated separately or together.

This lack of information led, in many cases, to the deduction of some important elements of the work as objectives, goals and the tasks that it sought to support, because these were not enunciated in an explicit manner. This could have led, eventually, (in the more ambiguous cases) to a situation in which the clear identification of what a research work seeks to support or resolve could have been carried out with little precision.

According to the analysis conducted, 23 studies support two or more tasks and some of the tasks identified were designated as only secondary tasks. As a result, these tasks were not included in any of the schemas that have been discussed with the aim of making more precise comparisons and quantifications although the complete list of tasks is shown in section 4.2.4.

The majority of the primary tasks were undertaken by works classified in both the category *Sys* and *Evol*. However, some tasks only support works in one of these categories. Thus tasks have been divided into two groups in order to analyze them in more detail. These groups were made up with those tasks that support research works under the rubric *Sys* and those which support research under the *Evol* category. Consequently, the first group is comprised by 6 tasks and the second group by only one task (*Software ecosystem comprehension*). The tasks in the former group are the following:

1. Distributed systems comprehension.
2. Improve source code security.
3. Memory allocation analysis.

4. Multithreading execution analysis.
5. Software design and modeling.
6. Understand dependencies.

It is important to remark that tasks 1, 3, and 4 from the above list used data obtained at runtime. The aim of these tasks is to find defects and facilitate the realization of improvements based on the data produced when a given revision is used as the base for running the software project. Moreover, the classification of these tasks in this group seems natural because of the characteristics involved in the real-time execution of a system (the execution of the current version), but the classification of other tasks in the category *Sys* is not so obvious. Finally, this may be an indication that these tasks need to be studied using an evolutionary approach to determine if different results may be obtained to improve their contribution to the process of developing and maintaining software.

Most of the works that are sought to support *Performance analysis* and *Program execution analysis* felt into the category *Sys*, although each one of these tasks is supported by a work which falls into the category *Evol*; this implied the analysis of data obtained during the execution of the system for a given number of revisions in a limited period of time. These exceptions are striking because are inline with the above regarding the use of dynamic data for studying the evolution of the system.

With regard to the second group, *Software ecosystem comprehension* is supported by works in the category *Evol* and is aimed to facilitate understanding of the relationships between projects that are located in a software repository, as well as the interactions of programmers with projects. Other tasks which mostly are supported by research works under the *Evol* category are *Team awareness and collaboration*, and *Understand system architectures*, although a few number of works in the rubric Sys also supported them. The task *Team awareness and collaboration* has as its aim the study of awareness levels of team members and the intensity of collaboration between them, ideally over a significant time period. Meanwhile, the task *Understand system architectures* aims to understand the changes that have taken place in an architecture during the several revisions which have taken place over a period of time.

The results obtained demonstrate that the use of VA to support the process of software development and maintenance is very limited. Of the 149 studies analyzed, only 5 made use of VA while 144 works make use of SV and Software Evolution Visualization (SEV). Of these 5 papers, 4 papers were sub-classified in the category Philosophical research and were in turn then classified in the categories *Sys* and *Evol*, with 2 works classified in each of those categories.

These results suggest that the application of VA is still undergoing a process of evolution at both the theoretical and methodological levels.

With regard to the visualizations, work classified in the *Sys* and *Evol* categories did not demonstrate any particular preference for a visualization, except in very specific cases where the visualization was originally designed to support one of these categories and then began to be used to support the other. Among the specific cases which may be mentioned are, in the case of the category *Sys*: *DSM*, *Matrix Layout* and some variants of *UML*. There were special cases whose visualization design was intended for works classified in the *Evol* category including: *Code city*, *Software cartography* and *Timelines*.

The pattern that has been analyzed up to now is similar to that which the data used for the works follows. Some data types are used both for works in the category *Sys* and works in the category *Evol*, but there are also types of data which are exclusively used for one category or the other. However, it is interesting to note that when the number of data types for each category is accounted the difference in the use between one category and the other is minimal, even though the number of research works in the category *Sys* is greater than the number of those classified in the category *Evol*. This pattern can be explained by the fact that visual representations which facilitate the understanding of the evolution of software projects, seek to provide a greater number of details in order to permit comparisons between revisions of the system or show relationships. Therefore, research works in the *Evol* category represent a higher number of data types.

According to the results obtained there are tasks that are supported by works classified in both in the categories *Sys* and *Evol*, but there are also tasks that are supported by works from just one of these categories. Similarly, some visualizations and data are used by research in both categories, but in some cases they are only used by works falling into one or the other category (*Sys* and *Evol*).

This opens many possibilities which could lead to the exploration of better results thus supporting the process of software development and maintenance by extrapolating the use of elements that support one of both approaches (*Sys* and *Evol*). As a concrete example, the first group of tasks (listed above) could be employed more broadly to analyze the behavior of the project at runtime for a given number of revisions or a reasonable period of time. This could be used to calculate metrics which permit more precise knowledge to be applied in order to make specific improvements as the project evolves. It is worth recalling that one of the principal objectives of software project evolution is to ensure maintainability.

4.5 Conclusions

This research conducted the analysis of 149 research papers. 47 of which were classified in the category *Philosophical Research* (18 under *Sys* and 29 under *Evol*) and 102 were classified in the category *Solution Proposal* (64 under *Sys* and *Evol*). The results obtained were exhaustively discussed and widely disseminated. The questions posed at the beginning of the research were answered satisfactorily.

In general, the study allowed to become more fully aware of how visualization and VA are used to support the development and maintenance of software systems by taking into account the following:

1. Details of the tasks that were supported by the research.
2. The visualizations that were used to support these tasks.
3. The data types that were represented visually.
4. The mapping between the types of data and the visualizations.
5. The technologies employed by these works in the solutions proposed.
6. The different criteria of validation employed
7. The use of *Single*, *Multiple* and *Multiple linked* views.

It is worth to mention that the results allowed to answer progressively to the first 6 research questions formulated in section 4.2.1. Regarding research question 7, the results showed that the use of *Single views*, *Multiple views* and *Multiple linked views* is proportionally distributed for works falling into the *Sys* and *Evol* categories. Although from the results it was possible to ascertain that the visualizations used by the works in the *Evol* category represent a greater number of elements, in accordance with the number of data types used by the scheme in Table 4.5. This could allow to further argue that research in the *Evol* category requires the representation of data and relationships of greater complexity and therefore the use of more sophisticated techniques, as *Multiple linked views*. However, the results are clear and lead to the conclusion that the research works studied do not showed notable differences in the pattern of use of the views to indicate whether either category (*Sys* or *Evol*) makes a predominant usage of a particular type of view. Thus, this results could be showing that there is margin to improve the results of the research works in the *Evol* category with the use of Multiple linked views and further, with the use of VA principles.

This research has yielded a large number of interesting results, as can be observed in the schemes that have been presented. Some of these results may be details which were expected to be found, but in many others cases they are not, and they reveal patterns which can be used for the design of new research and tools. A critical analysis of the schemes, allows questions to be

raised about the areas that still have not been the subject of research and also into modifying the focus of research in order to obtain additional results that may eventually lead to better solutions. Furthermore, by studying the classification schemes, it can be observed areas that could be extrapolated from one category to another, speaking in terms of the classification of works as *Sys* or *Evol*, in order to find new ways of using established techniques and methods that have been tested on a closely related field.

It is worth noting that not all the information presented in these schemes has been analyzed in this work; for reasons of time, space and the author's own biases, leaving, therefore, many details that have to be analyzed by the readers themselves.

In this context, reference must be made to the work of *Lima et al.* [Novais 2013]. This work consists of a systematic mapping study of SEV and suggests that the number of published papers decreased in recent years. Those results contrast with the ones obtained in this research with regard to the application of SEV to software systems. This is because in this research the number of publications published per year does not show a pattern which allows to arrive to the same conclusions of that earlier study. In this regard, it is appropriate to emphasize that the methodology used in the study was carried out by *Lima et al.* differs from the methodology employed in this study. Furthermore, the work done by *Lima et al.* examined studies were published up to 2011 and the sources of the research works studied are different from those used in this research.

Based on the results of this study and those presented by the work done by *Lima et al.*, concern arises as to the manner in which the results of research of visualization and VA to support software development and maintenance are being used in practice by internal software development departments and the software industry. In this context, it deserves special attention the dissemination and transfer of research results to industry in order to improve their processes and thereafter provide feedback to help improve the quality of the research being carried out. Accordingly, the following question, which will be addressed in the next chapter, is posed:

How are software companies and software development departments using visual tools to facilitate software development and maintenance?

Chapter 5

Understanding system architectures

Güindy se quedó desconcertado. Al cabo de un rato se dio cuenta que Cucho no estaba, se había marchado por uno de los caminos y lo siguió. Más adelante se encontró con un hombre llamado Rastin, al cual preguntó "¿a dónde conduce este camino?". "He acompañado a varios hasta al final de este camino y cada uno ha llegado a un lugar distinto", respondió.. — El viaje de Güindy, A.González

Contents

5.1 Introduction

The architecture of a software system is designed to display an abstract view of the system and its design can be perceived as a number of layers with different levels of detail. The level of detail of each layer depends on the purpose of the system architecture, its requirements and the environment in which the system will function.

A high-level architecture is useful in communicating an overview to managers, project managers and users, while a low-level architecture is used to guide the detailed design of the system and programmers when they have not yet been familiarized with the system [Kazman 1996], to show, for example, information about the relationships between the software items [Balzer 2005a] and data structures used.

Given this, it is possible to make different architecture designs for the system according to its purpose. This research adopts the definition provided by Bass *et al.* [Bass 2003] with regard to the architecture of a software system:

> The software architecture of a program or computing system is the structure or structures of the system, which is comprised of software elements, the externally visible properties of those elements, and the relationships among them.

It is noteworthy that on the basis of the architecture it is possible to understand how a system is organized in terms of: software modularity; components; relationships between components; data structures; access to data; physical distribution of the system components on servers and user computers.

Modern software systems are developed using object-oriented languages and consist of thousands of entities [Balzer 2005a]. Therefore, the architectures of such systems are organized using hierarchical structures (packages, classes, methods and attributes), which sometimes exceeds 20 levels, to adequately reflect their organization. Thus, this research is based on the analysis of software systems developed within the object-oriented programming paradigm and takes into account that for their comprehension information at different abstraction levels is required.

Another consideration regarding the system architecture is that the software quality assurance process uses metrics to measure the architecture elements (*e.g.*, size, complexity, dependencies and relationships), and that the particular objective of evolution metrics is to allow the comparison and evolution of the quality taking into account several revisions or time periods.

Ball and Eick assert that the three properties of software systems that can be visualized are the structure, runtime behavior and source code [Ball 1996].

However, in order to understand the processes of SDME surrounding a system it is necessary to be aware of other relevant aspects such as the socio-technical relationships that arise from the interaction of programmers with system elements (*e.g.*, to be aware which elements have been modified by each particular programmer), the volume of contributions made by programmers, the relationships that have been established between them, reported errors and their relationship to system elements.

In accordance with the above, it should be remembered that the tasks which received more attention in the results of the systematic mapping study presented in chapter 4 were *Understand system architectures* and *Team awareness and collaboration*. The research works which sought to support these tasks gave attention to the types of data and visualizations presented in tables 5.1 and 5.2. Research works that support both tasks have made use of most data types which are shown in those tables.

Table 5.1: Data elements of software systems used in the tasks (a) *Understand system architectures* and (b) *Team awareness and collaboration*.

Software architecture elements	**a**	**b**
Coupling and logical coupling	x	
Code clones	x	
Dependencies	x	x
Execution traces		x
Metrics	x	x
SCM metadata (contributions and collaboration)		x
Software item relationships (inheritance and interface implementation)	x	x
Structure	x	x
Socio-technical relationships		x
Source code changes	x	x
Vocabulary	x	x

The results shown in table 5.1 and the obtained by Khan *et al.* [Khan 2012] outline that the architectural elements that are most commonly visualized, taking into account one or more revisions of the system, are the following:

* Coupling and logical coupling.
* Dependencies and relations among software items (*e.g.*, inheritance and interface implementation).
* Metrics and evolution metrics.
* Structure and changes in the structure.
* Vocabulary.

Table 5.2: Visualization techniques used for the tasks supported by research works: (a) *Understand system architectures* and (b) *Team awareness and collaboration.*

Visualizations	**a**	**b**
Basic charts	x	x
Code browsers		x
Code city metaphors	x	
Edge bundles	x	x
Graphs (including radial graphs, hyperbolic tree and cone tree layouts)	x	x
Heatmaps		x
Iciplots	x	
Matrix layouts (including Dependency Structure Matrix (DSM)	x	
Parallel node-link	x	
Polymetric views		x
Software cartography	x	x
Sunburst	x	
Tag clouds		x
Timelines		x
Treemaps (including Voronoi and circular treemaps)	x	

Consequently, this chapter and the next discuss the most relevant research works which seek to support the tasks *Understand system architectures* and *Team awareness and collaboration.* However, additionally other important publications which are related to the above tasks are also studied in order to provide a better picture of the state of the art of the research focused on supporting these tasks. Therefore, the following sections are devoted to review in detail the visual representation of the architecture of software systems. Accordingly, section 5.2 is focused in analyzing research works devoted to the visualization of the architecture of software systems for a single revision, whereas section 5.3 is committed to study research conducted on the representation of the evolution of the architecture of software systems.

5.2 Architecture Visualization

5.2.1 City Metaphors

The use of city metaphors has become popular in the visualization of software in recent years, after Panas *et al.* proposed their use to represent the architecture of software systems [Panas 2003, Panas 2005]. It should be noted that this research refers to this type of visualizations using the term *software cities.*

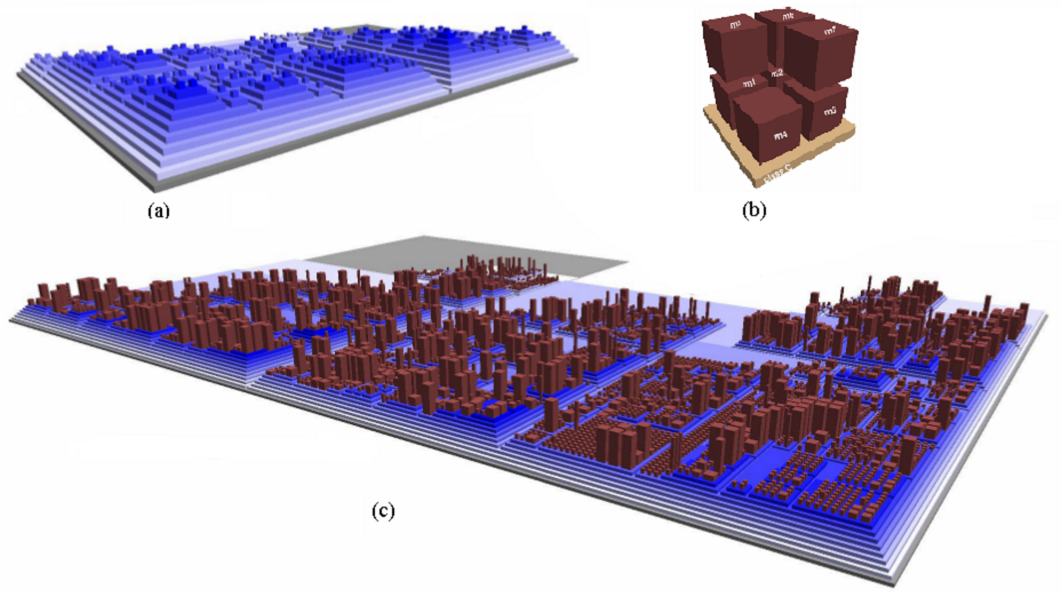


Figure 5.1: Visualization using a city metaphor [Wettel 2007, Wettel 2008a]. (a) Use of levels to represent the elements contained by other elements. (b) Visual representation of the methods in a class using brick figures. (c) Visualization of a complete software project.

The way how the *software city* representation is implemented differ between researchers and research groups. For example, Panas *et al.* represent a package making use of the full visualization and they use the districts to represent the classes, and the buildings to depict methods. While the research conducted by Wettel and Lanza use the districts and sub-districts to represent packages and sub-packages, the buildings to depict software items (classes and interfaces) and bricks to represent the methods [Wettel 2007, Wettel 2008a].

In general, these approaches use the height, width, and color of the buildings to represent different types of metrics associated to software items. Regarding the structure, the approach used by Panas *et al.* differs from the one used by Wettel and Lanza only in that the former represents a package while the latter represents the entire system. So that the visual structure of the approach used by Panas *et al.* begins with the representation of the sub-packages, while the approach used by Wettel and Lanza begins with the visualization of the packages in the first level of the system. Thus in the first approach, districts (sub-packages) and blocks (software items) are branches of the tree, while the buildings (methods) are the leaves; while in the latter approach the districts (packages), sub-districts (sub-packages), blocks (sub-packages of the last level) and buildings (software items) are branches of the tree, and the bricks (methods) are the leaves.

The implementation carried out by Wettel and Lanza of the *software city*

metaphor highlights the hierarchical order of the districts, sub-districts and blocks using elevations, where the highest elevation represents the elements that are found closest to the leaves on the tree structure. Figure 5.1 (a) shows the use of levels to represent the hierarchical order of the software items in a system (sub-packages that are part of other packages or sub-packages); Figure 5.1 (b) illustrates the depiction of class methods and Figure 5.1 (c) shows the complete representation of a software system.

The main advantages of the *software city* are the scalability and ability to visualize the structure and metrics of large-scale software systems. This type of visualization also provides valuable information at a glance and permits the exploration and acquisition of additional details by using interaction techniques.

It is relevant to note that a large number of research papers have introduced additional variants of the *software city* metaphor which have been standalone implementations mostly developed in *Java*. The research work done by Limberger *et al.* [Limberger 2013] used web technologies to develop a tool that implements this visual metaphor. The use of web technology has the advantage that the resulting tool is hardware independent and only requires a supported Internet browser.

It is also worth noting that *software city* visualizations can represent any kind of hierarchical structure. So, it is a representation that can be used to visualize any software system developed with a language that has this kind of structure.

A research work carried out by Bentrand and Melasti [Bentrad 2013] shows the potential of this type of visualization to represent the elements of a software system developed with AspectJ, an aspect oriented language which also is based on hierarchical structure. Such research proposes a tool that is integrated into Eclipse as a plugin and uses the *software city* metaphor to visualize the structure of a system as follows: the packages are represented as districts, and the aspects and software items are represented as buildings. Like other implementations of *software city* metaphors, the width, height and color of the buildings are used for the representation of metrics.

5.2.2 Treemaps

Modern programming languages impose a hierarchical structure to the architecture of software systems, which could successfully be represented using treemaps, taking into account the hierarchical nature of such visualizations [Johnson 1991]. Accordingly, Baker and Eick [Baker 1995] implemented SeeSys, a tool that is based on a treemap representation to depict metrics associated to software items.

In this visualization the size of nodes is proportional to the value of the metric associated to the software elements in the system structure. A similar visualization was designed by Balzar and Deussen [Balzer 2005b] to represent, making a better use of the visualization, the structure and metrics of software systems using Voronoi tessellations (see Figure 5.2).

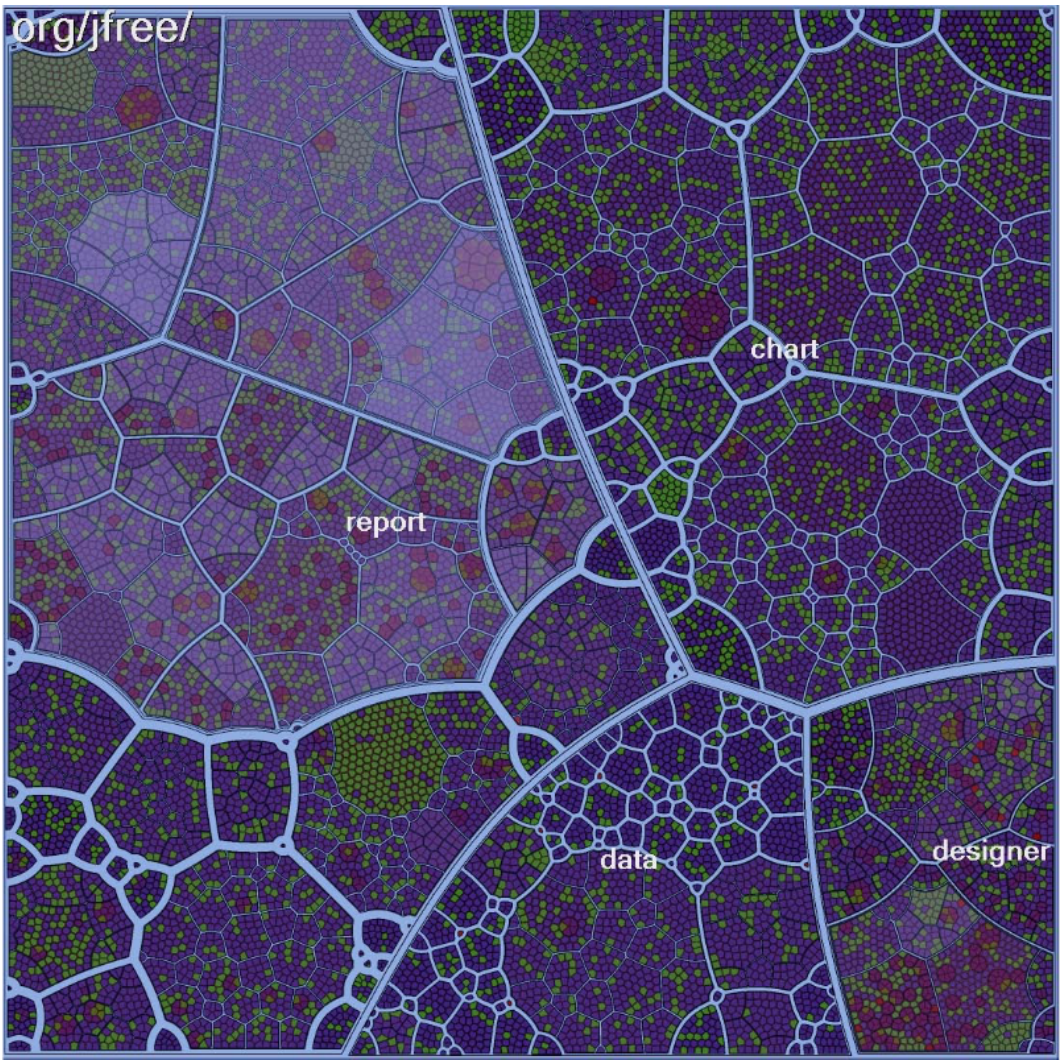


Figure 5.2: System structure and metrics representation using a Treemap based on Voronoi tessellations [Balzer 2005b].

An interesting design that makes use of a treemap representation [García 2009b] was carried out as part of this research (see Figure 5.3). The aim of such design is to disclose system details such as structure, class relationships, class coupling, class level metrics and source code.

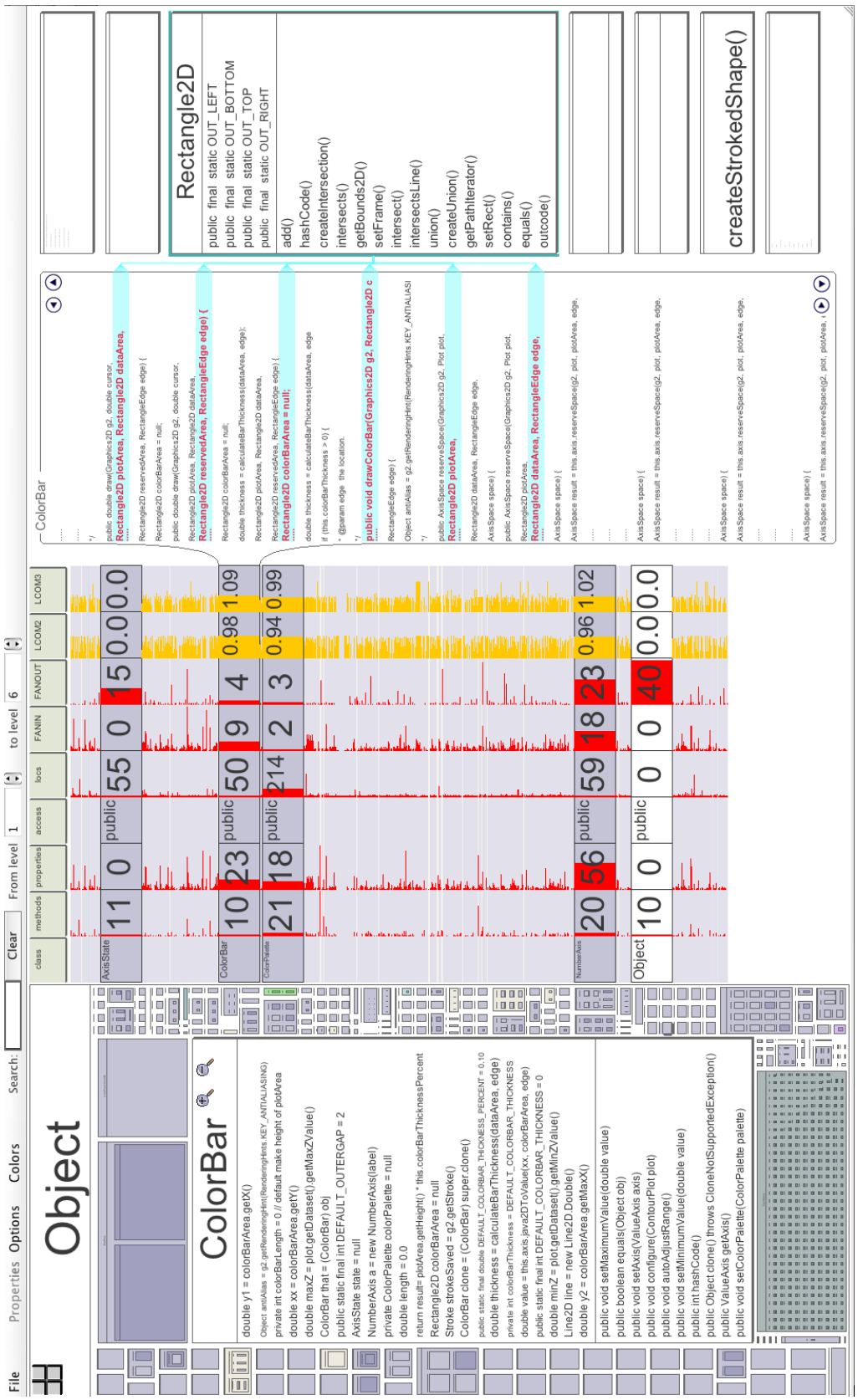


Figure 5.3: Visual representation of the system structure, and software item details (including metrics, methods and attributes) [García 2009b].

The visualization proposed by García *et al.* makes use of interaction techniques to support navigation, interpretation of visual elements and understanding relationships among data elements in their full context [Leung 1994a]. The user, by means of interaction techniques, can filter, transform, browse and discover relationships, as well as inspect relevant source code fragments and obtain insight of their relationships and coupling. The data used was extracted from SCM tool repositories.

This visualization represents the hierarchy of classes, so the classes and their parents are shown for inspection and analysis on the integrated representation view (the treemap) as well as class level metrics and source code. The visualization is conformed by four visual representations that are layout from left to right in Figure 5.3 and are used to carry out analysis tasks according to the following:

1. The system is loaded using information from a particular package or the complete system, according to user selection, and displays a treemap with the inheritance structure of the system or package.

2. The user selects a package from the treemap and the classes it contains are displayed on a table lens that shows the values of the metrics for each class.

3. Then, the user selects a class from the table lens and its content (methods and attributes) is displayed by a browser of objects.

4. Finally, the user selects methods or attributes from the browser of objects to review their content in an auxiliary view.

Moreover, this visualization tool offers the possibility to locate a specific class using the search engine included or browsing the visual elements on the treemap and the table lens representations. The selections made by the user can be done using as reference the inheritance relationships or the values of the metrics that are displayed in the table lens.

A common problem that is faced when designing a visualization tool is the small space to represent a large amount of data and details, which is accentuated when it is required to represent the structure and relationships between component of large systems. Zhao *et al.* designed a hybrid visualization that combines the use of trees and treemaps (see Figure 5.4) with interaction techniques for browsing and knowledge discovery in hierarchical structures [Zhao 2005]. The concept presented in this paper coincided with the research made by Balzer and Deussen concerned with the use of a hybrid structure that consists of a graph and treemaps [Balzer 2005a] (see Figure 5.5).

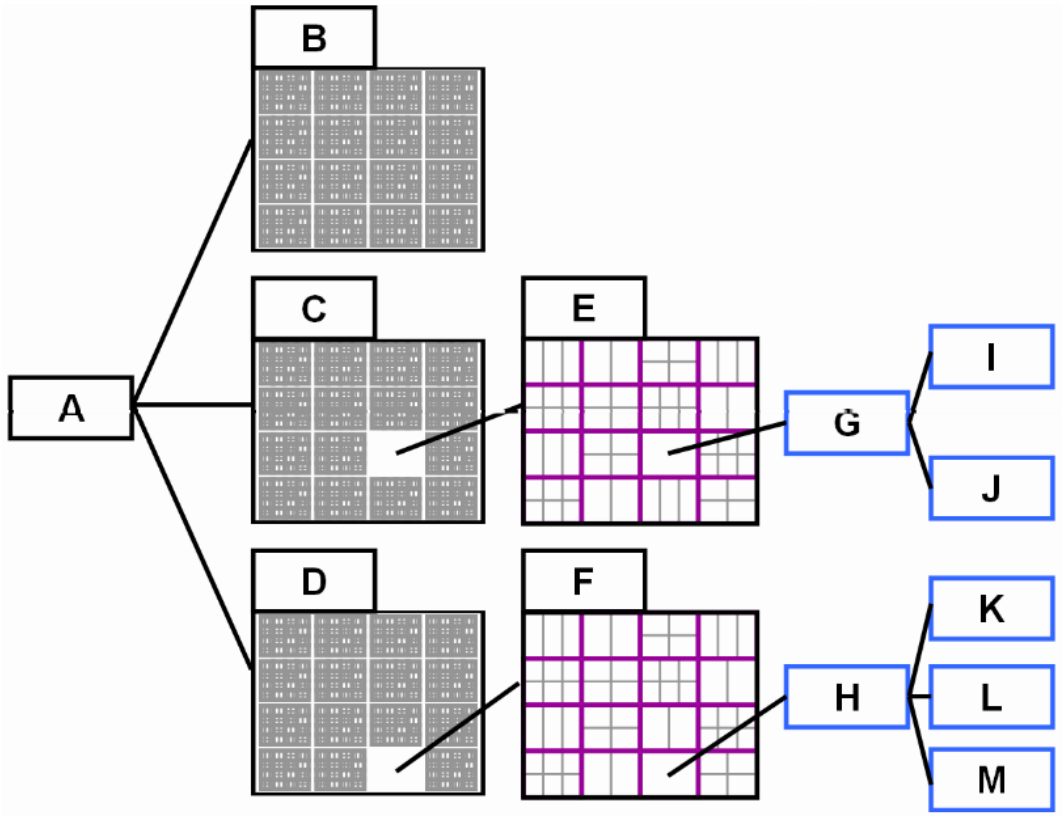


Figure 5.4: Hybrid visualization that combines the use of a conventional tree and treemaps [Zhao 2005].

The visualization proposed by Zhao *et al.* uses graphs to represent the relationships (*e.g.*, inheritance) between the software items that make up the system. In this visualization nodes represent packages whereas edges depict the relationships between packages. The size of nodes indicates the degree of connectivity of the element with other elements, while the width of edges indicate the number of relationships between the software items of the involved packages.

The paper published by Balzer and Deussen [Balzer 2005a] does not discuss details of the interaction possibilities that are offered by the visualization, but this representation could be further improve to obtain greater detail of the relationship between software elements in the lower levels of the hierarchy if the concepts discussed in the research presented by Zhao *et al.* are applied.

5.2.3 Grid Based Designs

The visualizations based on grid layouts have been used broadly to represent the structure of systems because of their scalability capabilities and ease of

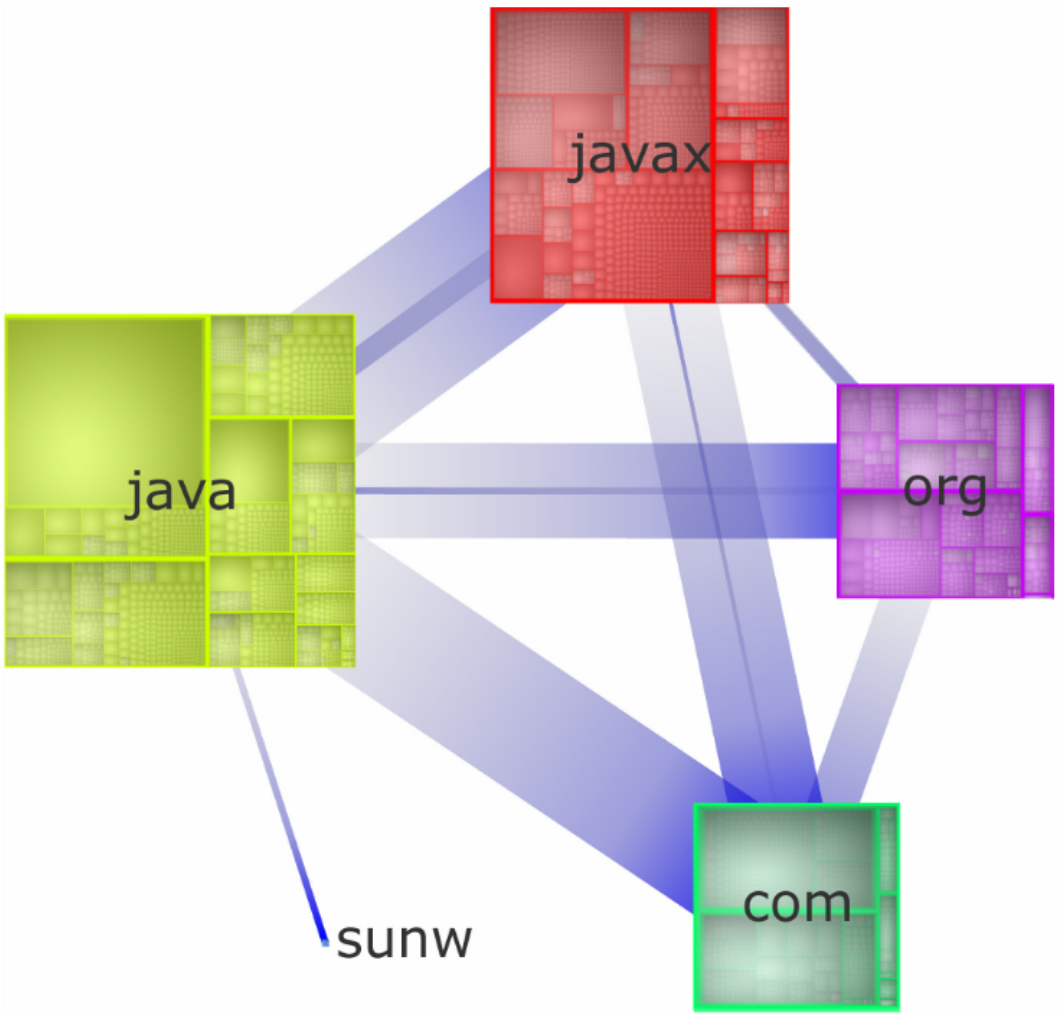
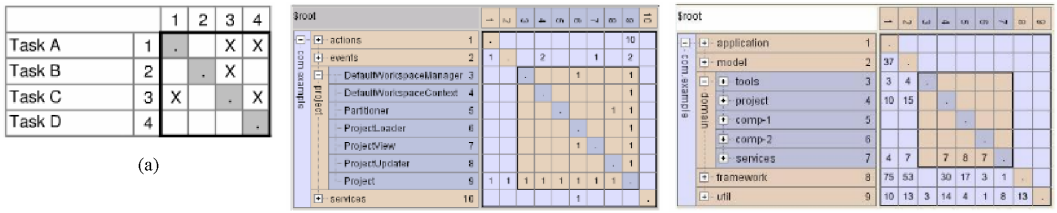


Figure 5.5: Top level view of the relationships among packages in *JDK* 1.4.2 using a hybrid visualization that combines a graph and treemaps [Balzer 2005a].

understanding. Sangal *et al.* [Sangal 2005] propose Lattix, a tool that extracts the dependencies between software items from the source code using static analysis and a DSM representation as a visualization method.

The method used by Lattix consists of placing the same software items (*e.g.*, packages and classes) in the rows as well as in the columns of the grid. It creates a numbered list using as a reference the placement of the software items in the rows and then takes the resulting list of numbers to enumerate the column headings of the corresponding software items. This correlation creates the cells of the grid, as elements in the rows and columns intersect. Therefore, the dependency between two software items is shown in the cell corresponding to the intersection between those items. Figure 5.6 (a) shows the dependency between software marking a **X** in the appropriate cell.

The hierarchy of system structure (packages and software items) is represented using a variable number of columns which depends on the number of levels of the depth of the tree structure: each column is used to locate elements corresponding to the hierarchy level as shown in Figure 5.6 (b). The tree structure can expand and contract according to user needs and according to screen space. When the visualization is representing the structure at package-level, this indicates the number of dependencies between packages, as can be seen in Figure 5.6 (b). However, when the structure has been expanded up to the last level of the hierarchy, the values of the dependencies between software items will be equal to 1, as can be observed in Figure 5.6 (c), because the dependency relation has been formed between atomic elements of the system.



(a) (b) (c)

Figure 5.6: Characteristics of Lattix [Sangal 2005]. (a) Correlation of dependencies between tasks (software items). (b) Expandable features of the visualization and the number of dependencies between software items contained in software packages. (c) The package *project* is expanded and to depict the software items that contain and the dependencies in which the items are involved.

Lattix provides the possibility of specifying rules to describe the dependencies considered acceptable in accordance with the system design. The visualization makes use of these rules to show possible breaches of the design, indicating this using colored marks that are placed in the cells of the grid: the green marks indicate that a relationship of dependency can be established, black marks show that it is prohibited to establish a dependency between the involved software items and red marks indicate the violation of the rules that govern valid dependencies (see Figure 5.7).

The visualization of multiple types of relationships and dependencies between software items is a difficult challenge to address. Graphs are limited in their capacity to signal several relationships at the same time and furthermore they suffer from problems to scale properly when trying to represent several hundreds of relationships.

An approach based on an adjacency matrix that permits the representation of various types of dependency is proposed by Abuthawabeh *et al.* [Abuthawabeh 2013], and was denominated IMMV. This visualization uses

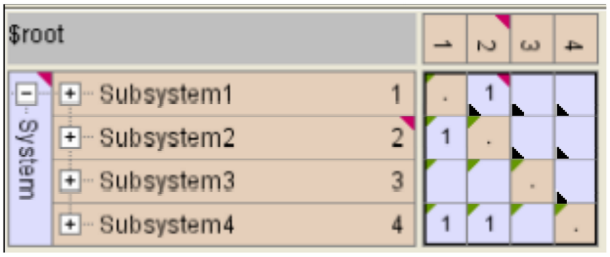


Figure 5.7: Design Rules: dependencies permitted, not permitted and violations to the design of the system [Sangal 2005].

an *icicle-plot* representation for the structure of the system and divides the cells of the matrix into sub-cells that are filled with different colors in order to represent different types of dependency. Therefore, when a type of dependency relationship exists, a sub-cell is filled with the color which corresponds to the type of dependency, as shown in Figure 5.8.

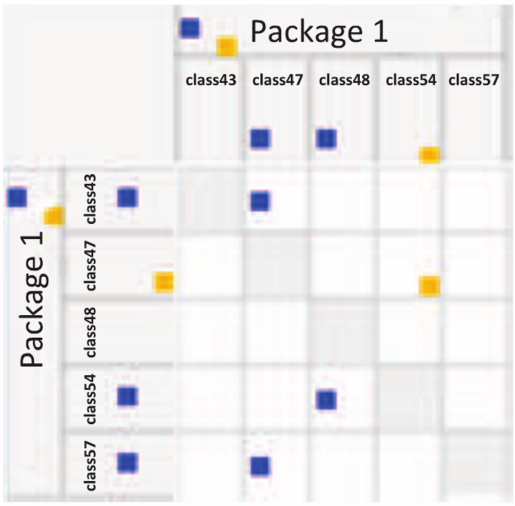


Figure 5.8: Dependency relationships with IMMV [Beck 2013].

5.2.4 Node-link Diagrams

Another alternative for the representation of dependencies is the visualization proposed by Abuthawabeh *et al.* [Abuthawabeh 2013], that was named PNL, which permits the representation of n dependency types. The number of dependencies that PNL is capable to represent is constrained by the display space of the screen.

The structure of a software system is depicted by PNL using an *icicle-plot* representation, which is positioned on the left side of the visualization, similar to the manner in which this is done by DSM. This representation allows to

expand software items to show their inner sub-structures. The expansion of software items could be performed up to the deepest level of the system structure.

PNL shows the dependency relationships between software items at the level at which the structure has been expanded, and uses parallel representations, one for each type of dependency. The dependencies are shown by lines connecting the software items in the structure represented by the *icicle-plot* and the software items in the parallel structures, as shown in Figure 5.9.

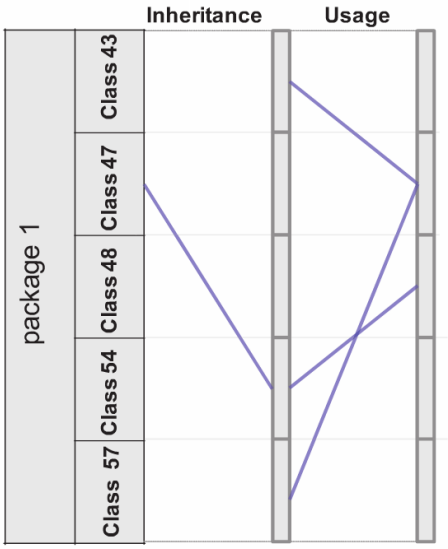


Figure 5.9: Visualization of dependency relations using PNL [Beck 2013].

5.2.5 3D Visualization

Some software libraries are large, which makes difficult their comprehension and the manner in which their development has been organized in terms of work and team organization. Ali [Ali 2009] pointed out that currently a large number of open source libraries are developed by programmers who contribute voluntarily with the programming of modules and software items. This type of development is performed using uncoupled coordination mechanisms, which in some cases affects the organization of libraries and contrasts with the better structure organization of libraries that are programmed using more rigorous coordination mechanisms [Ali 2009].

Mudrik is a 3D system developed in Java and *OpenGL* in order to support programmers in understanding external open source libraries (written in Java) that are used by software projects, and with which software developers have not been familiarized [Ali 2009].

The structure of libraries is represented by *Mudrik* using 3 visualizations to provide browsing and searching mechanisms with the aim of offering information about the structure and functionality of the classes within the library. The visualizations used by this tool are a complete tree representation of the structure of the library (*Class Browser*), a *Cone Tree* visualization and a DSM representation (both in 3D).

Class Browser is a simple visual representation of a tree (similar to the Java *JTree*) showing the structure of the library (packages, classes and interfaces). This visualization allows the user to select the items she wants to explore in the *Cone Tree* representation. When a single software item is selected in *Class Browser* (depending on the visualization option that has been chosen) the system displays a view with its details, subclasses or relationships it has with other elements. If the selected item is the root of the library (the system permits the classes to be filtered out by means of a filter control), the system displays the inheritance hierarchy of all classes (using the *Cone Tree* visualization) or the relationships among all classes (using a DSM representation with histograms in the cells to represent the number of references in the dependencies).

5.2.6 Polymetric Views

CodeCrawler is a software visualization tool developed by Lanza *et al.* which is language independent visualization that relies on the *FAMIX* metamodel and was implemented on the top of Moose [Lanza 2005a]. One of the visualizations that was designed as a component of this tool is Evolution Matrix. This visualization uses simple rectangular shapes [Lanza 2001a] to depict software items and 3 associated metrics (similarly to polymetric views), which can be selected by users according to their analysis needs (see Figure 5.10). In general terms, this visualization represents the evolution over time of the metrics of software items, as it is shown in Figure 5.11.

Figure 5.10: Representation of metrics in the Evolution Matrix visualization [Lanza 2001a].

Polymetric Views is part of *CodeCrawler* and uses a similar approach to

the one used by Evolution Matrix. However, it is able to represent the 5 metrics using the height, width, color and X and Y positions of rectangular shapes [Lanza 2003], as it is shown in Figure 5.12. This visualization, unlike the Evolution Matrix visualization, does not represent the evolution of metrics, but does allow to visualize the relationship between software items (*e.g.*, inheritance relationships or coupling between software items). Figure 5.13 illustrates a display of *Polymetric Views* in which metrics are depicted using the size, color and position of the shapes and the relationships between software items are also represented.

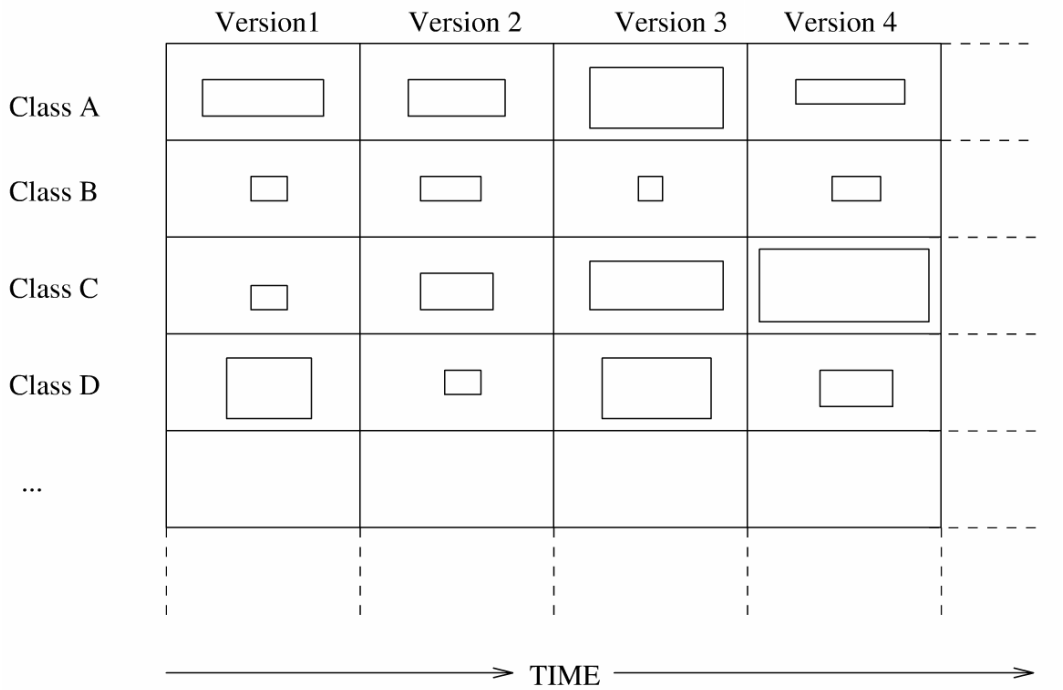


Figure 5.11: Overview of the Evolution Matrix visualization [Lanza 2001a].

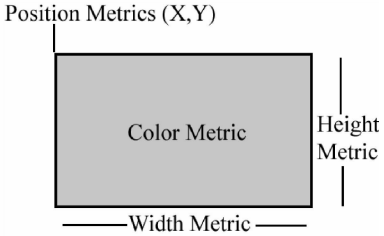


Figure 5.12: Metrics representation in Polymetric Views [Lanza 2003].

In line with the visualizations previously discussed, *Class Blueprint* is a visualization which shows the map of the internal structure of a class that also depicts its inheritance relationships with other classes [Lanza 2001b,

Ducasse 2005]. This visualization represents attributes, methods, and method access and invocation from left to right using 5 layers (Initialization, Interface, Implementation, Accessor and Attributes). Moreover, it uses a call graph to represent access to attributes and the method calls, where the elements of the left of the visualization are those that invoke or access the ones located on the right, as shown in Figure 5.14. This visualization uses a similar approach to the one used by the Evolution Matrix: the width, height and color of rectangular shapes are used to represent metrics.

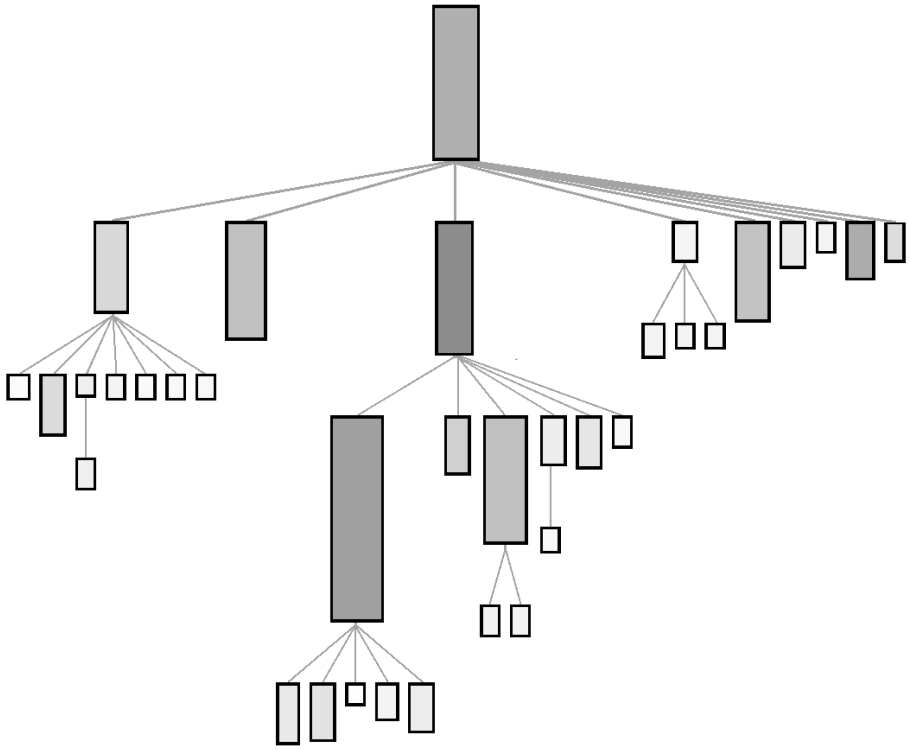


Figure 5.13: Overview of Polymetric Views [Lanza 2003].

The Initialization layer of the *Class Blueprint* visualization contains the methods that are responsible for creating and initializing the values of new objects, while the interface layer represents the public methods that the class renders accessible to other classes or which are invoked by the methods in the Initialization layer. Meanwhile, the Implementation layer depicts the private methods that implement the functionality of the class and which are invoked only by methods in the same class. With regard to the Accessor layer, this layer is composed of those methods that are in charge of establishing or obtaining the values of attributes, and the Attribute layer represents all the attributes that are accessed by the other layers of the visualization.

Inheritance is represented in this visualization by means of node-link

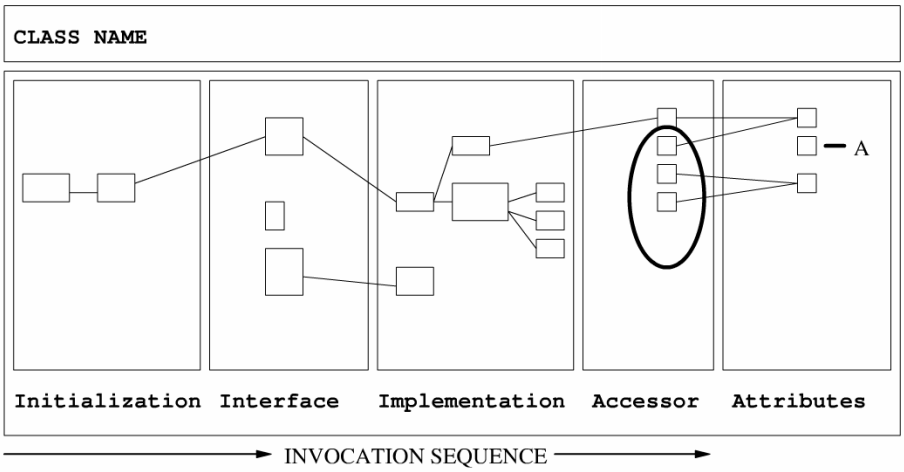


Figure 5.14: Overview of the design of the Class Blueprint [Lanza 2001b].

diagrams, where each class (represented by a map like the one that was described above) finds itself related to the other classes using a tree structure. Figure 5.15 shows the representation of inheritance and allows to observe, using the *UML* notation [Booch 2005], the inheritance relationship between classes (black colored lines) and the access to attributes between classes and subclasses (represented by cyan lines).

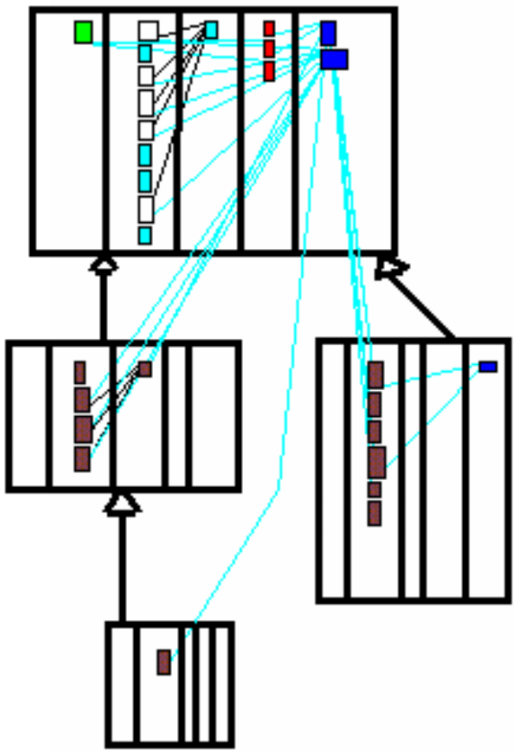


Figure 5.15: Inheritance view of the Class Blueprint visualization [Lanza 2001b].

5.2.7 Circular Visualizations

Holten *et al.* designed EXTRAVIS [Holten 2007], a visualization tool that consists of two linked visual representations, HEB and Massive Sequence View (MSV), and a time control (see Figure 5.16).

HEB uses a series of rings to represent the hierarchy of software items, so that the outer ring depicts the first level of the hierarchy and the following rings render the next levels. To show the relationship between software items it uses green and red coded lines (the green color indicates the caller and the red color characterizes the callee) with an arrow to describe the direction of the relationships. Whereas MSV uses an icicle plot to show the hierarchy structure and green and red coded bars to display the relationships between software items, similarly to HEB. With regard to the time control, this allows to setup a time window for which the user wants to review the relationships between software items.

HEB and MSV are linked together, so that when an item is selected in one of the representations, the selection is reflected in the other representation and viceversa.

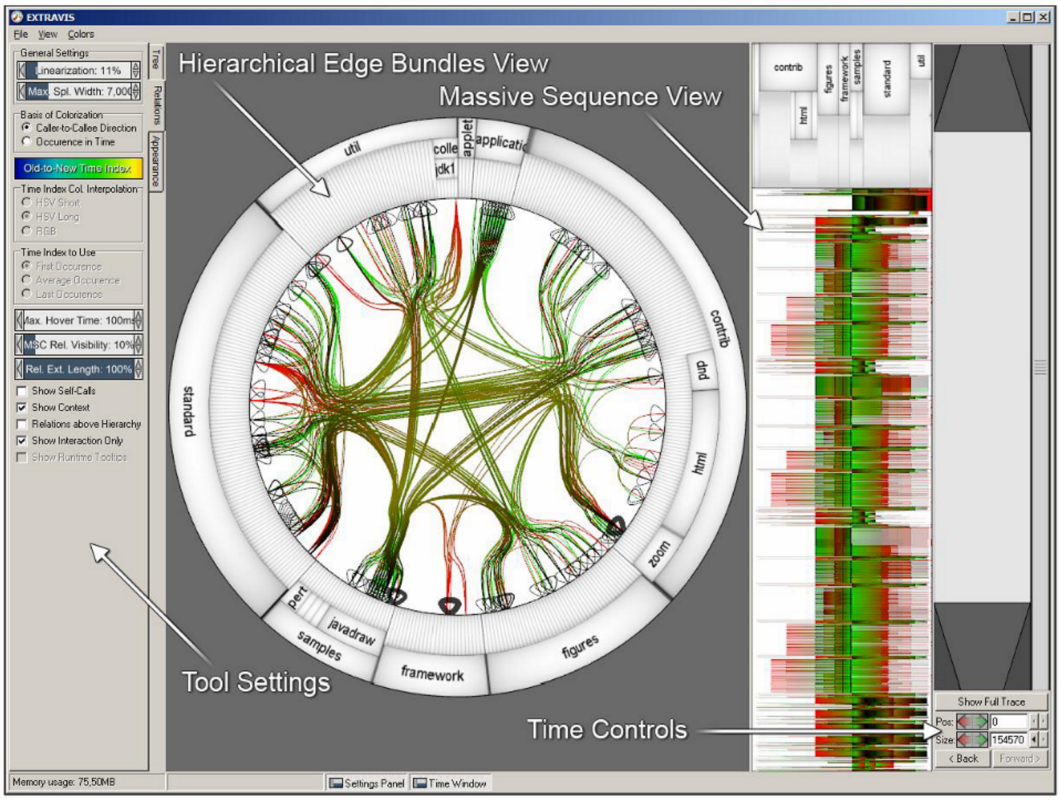


Figure 5.16: Overview of EXTRAVIS [Holten 2007].

5.3 Architecture Evolution Visualization

5.3.1 City Metaphors

Understand a software system requires the comprehension of both the evolution of the architecture and metrics associated to the elements of its structure. In line with this, Wettel and Lanza [Wettel 2008a] support the understanding of software evolution using *software cities* depictions with two levels of representation: the system in general and software items in particular. In order to represent the evolution of the system and the software items, a succession of visualizations is used to highlight the changes that have occurred between one revision and another. Figure 5.17 shows that between each revision of the system the elements indicated by the arrows and the black color circle have changed. In the case of specific software items, for each revision the size and age of methods are shown by the height of buildings and the use of colors (the darker color represents the oldest method) as illustrated in Figure 5.18. Overall, these methods have scalability limitations to represent a large number of revisions, even for small and medium scale systems.

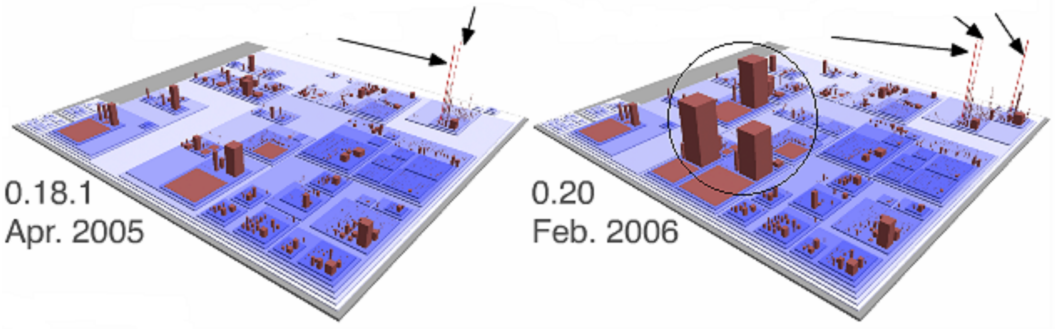


Figure 5.17: Visualization of two revisions of a software system [Wettel 2008a].

The implementation of the *software city* metaphor that was carried out by Wettel and Lanza [Wettel 2008a] calculates the system structure for all revisions of the system from its creation up to a determinate point in time, and then depicts the map according to the calculation performed. So, the visualization is used as an exploration space to evaluate the changes that have occurred between revisions or time periods, but does not contemplate the possibility to incorporate changes to the structure after the map has been calculated. Thus, the elements of the visualization and their positions may require a rearrangement for the incorporation of changes to the system structure. This implies that the mental model of programmers about the system structure would be altered and could involve difficulties to carry out comparisons between revisions and time periods.

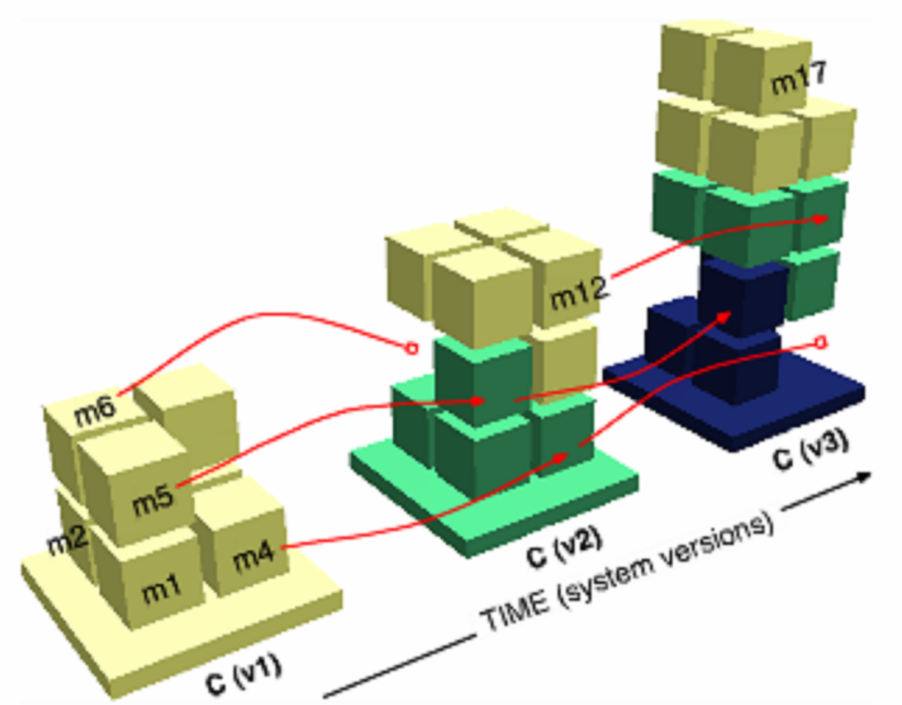


Figure 5.18: Visual representation of the evolution of a software item and its methods [Wettel 2008a].

The difficulties mentioned earlier were described by Steinbrückner and Lewerentz [Steinbrückner 2013], suggesting that the visualization of *software cities* as metaphors intended to represent the evolution of systems suffer from problems in dealing with the representation of changes in the structure of systems. For this reason, they proposed a visualization called *EvoStreets*, which is based also on the metaphor of a city that is built as the system evolves.

The concept employed by *EvoStreets* revolves around the concept of the streets of a city. In this visualization, the main street represents the entire system, the secondary streets represent packages or sub-packages whereas the buildings depict software elements. The width of the streets is inversely proportional to the depth that the package or sub-package has in the hierarchy of the structure of the system: streets located in deeper levels of the structure have a narrower width that those located in the top levels. The size of buildings is used, as in the other implementations of the *software city* metaphor to represent properties or metrics of the software items. The main elements that are represented by *EvoStreets* are packages, classes, inheritance, dependencies, type and size of software items.

The visual representation employed by *EvoStreets* is built upon the structure that is obtained from the accumulated analyzes of revisions that

have been made over the time period under analysis. The construction of the visualization takes as baseline the structure of the system that existed when the first revision was created. Accordingly, the coordinates of the visual elements in the initial representation of *EvoStreets* are calculated and fixed. Thus, the new elements which are added in the subsequent revisions are fitted into the existing structure as part of an existing street or added to a new street.

The layout of this visualization grows from the center outward. The main street is located at the top of the system structure and in the next level it is split into secondary streets. This process is repeated successively in all subsequent levels of the structure. The streets get longer and extends to the periphery of the visualization when new elements are added on both sides of them (see Figure 5.19). Moreover, the addition of secondary streets is carried out when new sub-packages are created.

The layout of this visualization is immutable, once software items are added they can not be moved to another position; when an item is removed, it is highlighted as such but it is not eliminated from the visual representation; and when an item is changed to a new position it is highlighted as a removed element and then it is added to its new position as a novel item. It is relevant to mention that a similar approach to the one used by *EvoStreets* was previously proposed by González *et al.* [González-Torres 2009] using a 2D layout. Figure 5.20 shows a sequence of the evolution of a system structure using this visualization design.

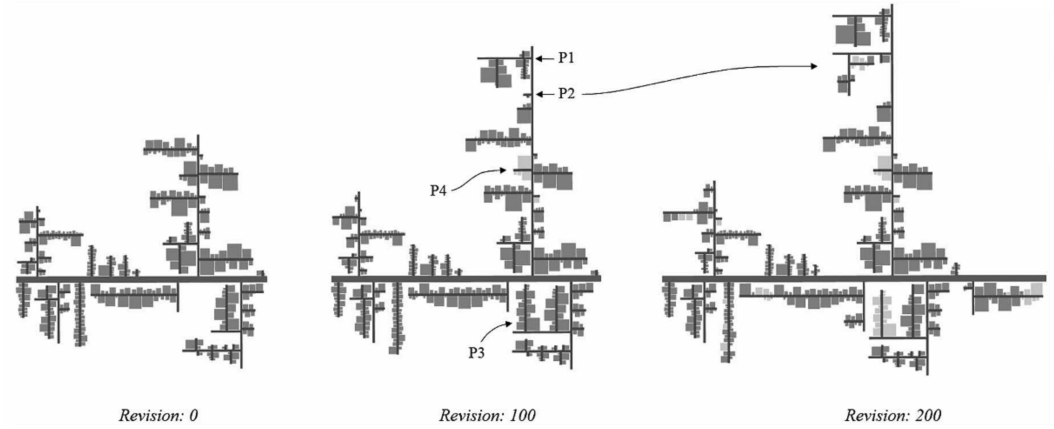


Figure 5.19: EvoStreets: Evolution of the structure of a software system [Steinbrückner 2013].

The evolution of software systems is represented by *EvoStreets* using levels and contours: the oldest items are located in the upper levels and the newest at the lower levels. Figure 5.21 illustrates this feature: when package P is added to the structure (Figure 5.21 (a)) a new level is created to place the

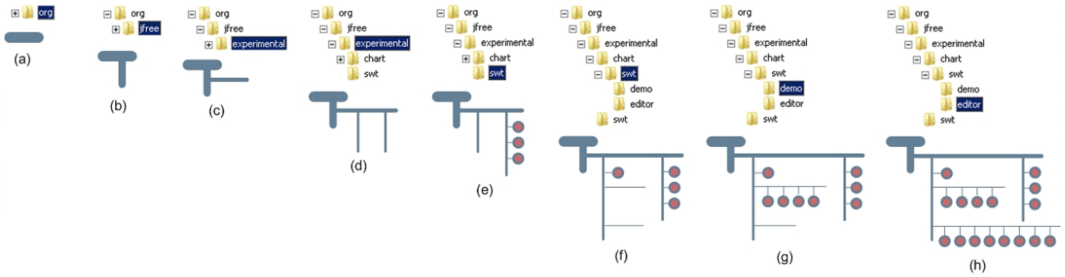


Figure 5.20: H-V tree layout for the visualization of the structure of software systems [González-Torres 2009].

element (Figure 5.21 (b)).

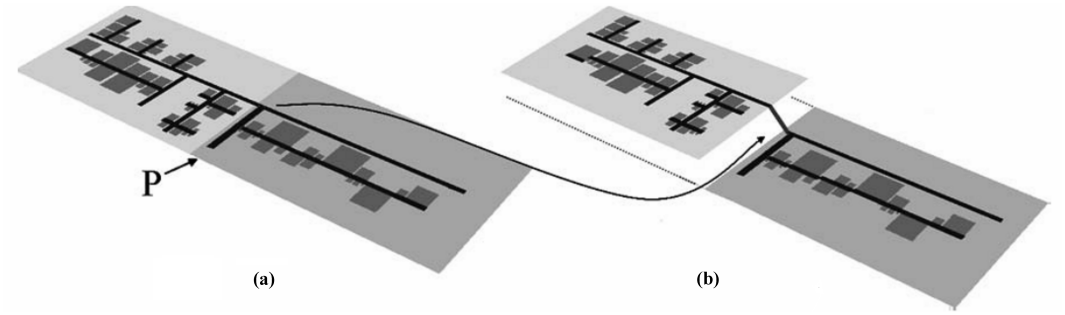


Figure 5.21: EvoStreets: Use of the levels in the visualization to show when a new package is added. [Steinbrückner 2013].

The height, width and color of the buildings are used in *EvoStreets* to indicate properties (see Figure 5.22) such as the names of programmers as well as the number of changes that have been carried out, the coverage of testing cases and metrics, for example.

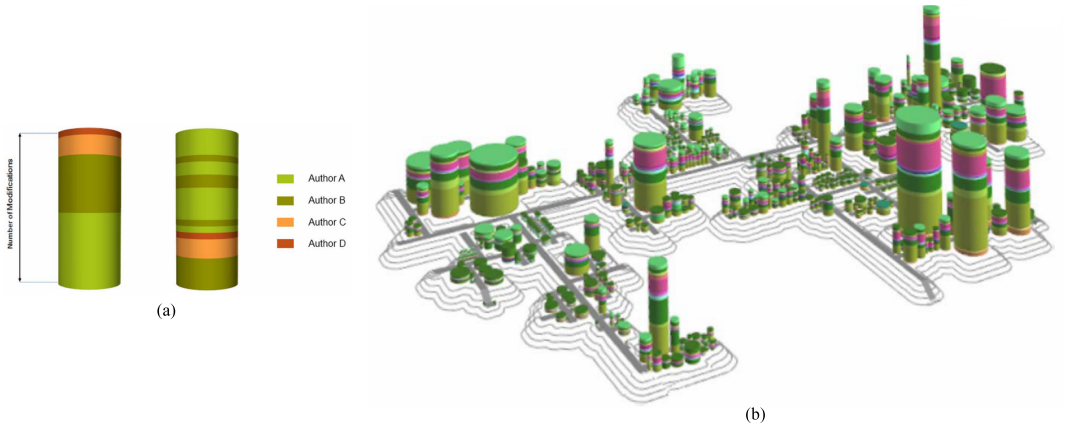


Figure 5.22: EvoStreets: The properties of the software items are represented by the width, height and color of the buildings [Steinbrückner 2013].

5.3.2 Grid Based Designs

Beck and Diehl [Beck 2013] proposed a visualization based on DSM to identify the differences in structure, as well as dependencies between software items, when comparing two revisions. This visualization displays the system structure of the revisions under analysis in the left and top sections of the grid: the structure of one revision is depicted in the left section whereas the structure of the other revision is represented in the top section. The possible differences between the structures are managed using an algorithm which sorts the elements taking into account the parents and relatives that are not common between one structure and another [Beck 2013].

Once the structures have been represented, a comparison between revisions is performed using a color code: whenever there is a dependency relationship between two elements, the cell representing this relationship is highlighted by a particular color. The color code used by this visualization is composed by the blue, purple and red colors. The blue color is used to indicate the dependencies that exist in the first revision, while the purple color points out existing dependencies in the second revision, and red is used to show the dependencies that are common to both revisions (see Figure 5.23).

5.3.3 Animation

Yarn is a visualization that represents the evolution of the architecture of software systems on the basis of changes in the source code and the use of animation [Hindle 2007]. This visualization consists of a circular graph whose nodes represent software items and whose edges use weights to indicate the number of dependencies between the items. The evolution of the system is presented as an animated graph that retains the position of its elements and gradually shows the changes occurring in the dependencies.

The animation used by Yarn can emphasize the accumulation of changes or only those changes that have occurred recently. To do this, it makes use of colors and the thickness of the edges. Yarn allows the dependencies of the complete system evolution to be known by means of a representation that shows the accumulation of dependencies, which takes into account the interval of time between the instant at which the dependencies were created until the moment in which the last revision of the system has been committed. It also allows information to be obtained about the dependencies that have recently changed using colors to highlight them while darkening the oldest dependencies. Additionally, the animation displays information about the revision number and the date on which it was created (see Figure 5.24) when it is played.

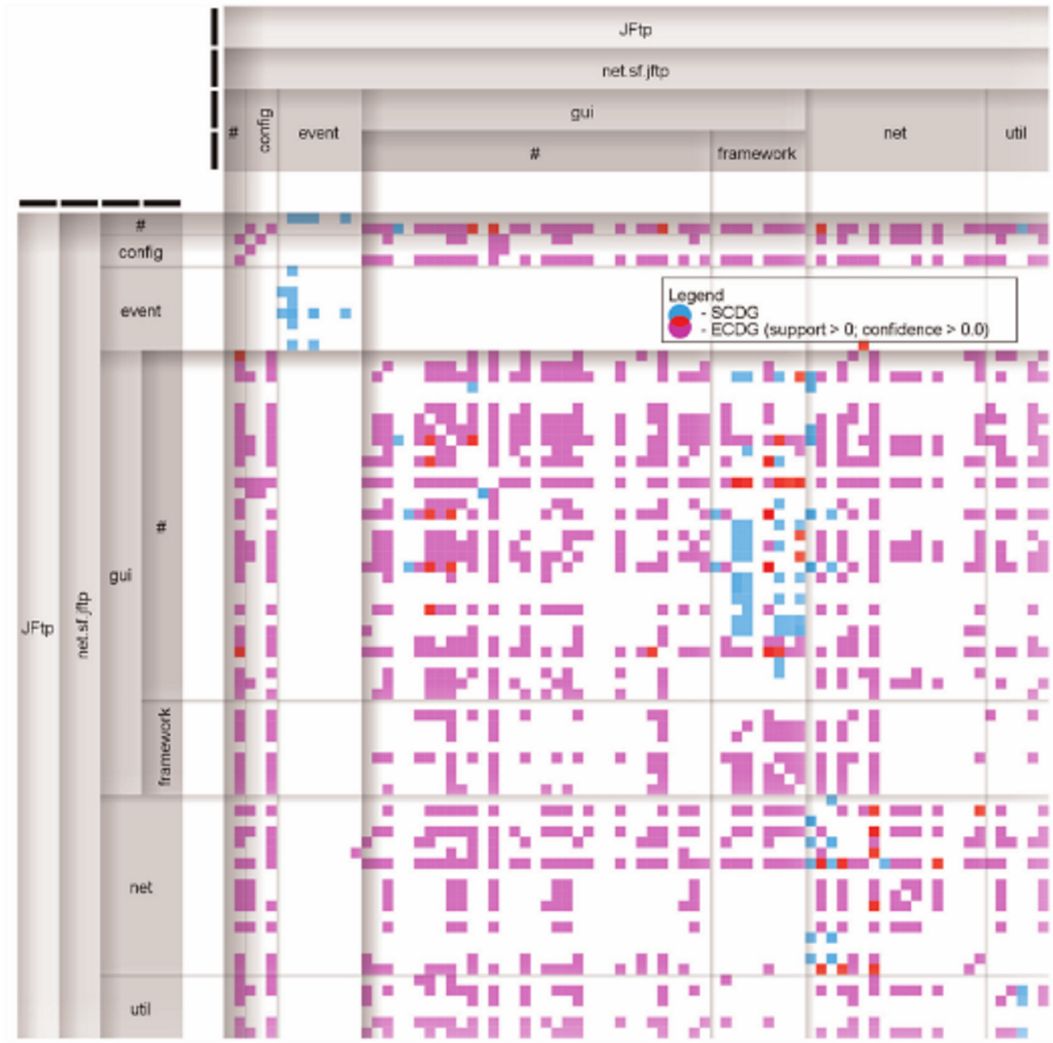


Figure 5.23: Comparison of dependencies between two revisions of a software system [Beck 2013].

Among the limitations of Yarn are its lack of capability to represent the structure and relationships of large systems, due to the occlusion that is caused by a large number of connections between software items. Additionally, it should be taken into account the limited interaction capabilities of this visualization as a result of providing information in its final form and thus not allowing the possibility of interactive navigation and knowledge discovery.

Evolution Storyboard is a tool that shows the dependencies between software items through a series of animated visualizations which is obtained by combining the visual representations (e.g., frames in a movie) produced when analyzing the changes which have been made to the system in a determinate number of revisions over a period of time [Beyer 2006]. The tool displays a strip formed by the animated visualizations, where each animated

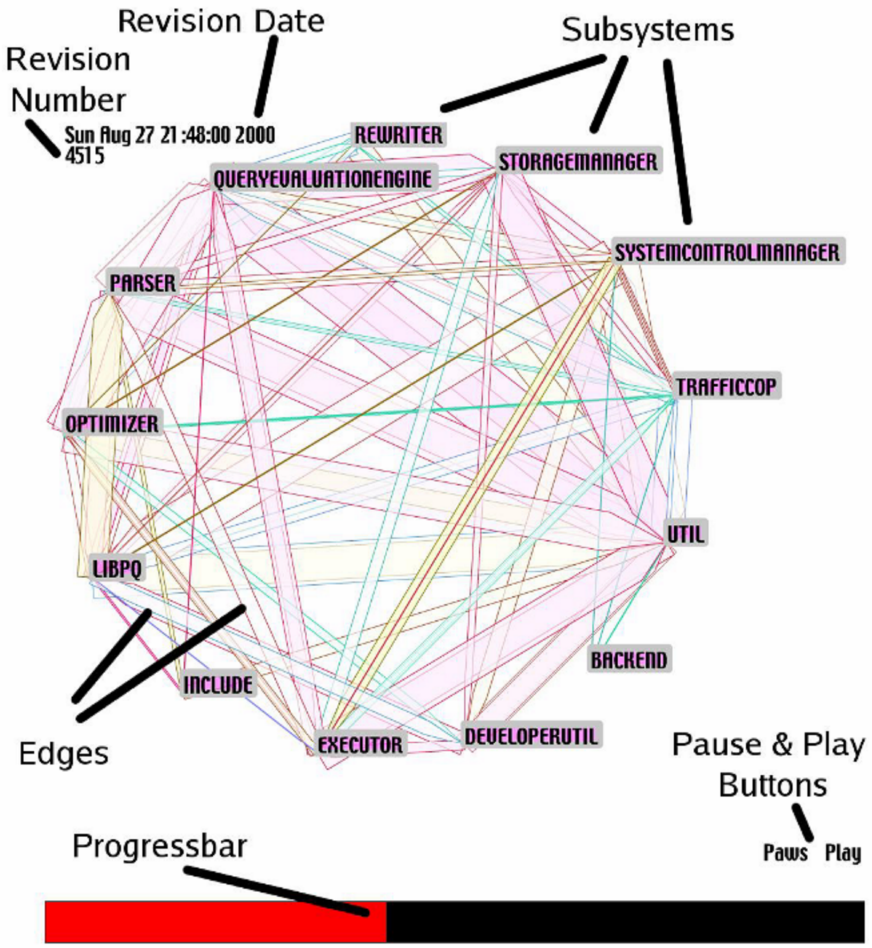


Figure 5.24: Animated visualization of the evolution of the architecture of a software system using Yarn [Hindle 2007].

visualization represents an interval in time (the time period of study is divided into time slots), and allows the programmers to focus on the observation and comparison of consecutive periods of time of the evolution of part or the whole of the system.

The dependencies between software items are represented by a force-directed graph whose nodes are colored depending on the package they belong to and whose the distance is determined by the dependencies between the software items. The relationship between the software items is not shown explicitly by means of the edges, but rather by their location: the proximity between software items represents the dependency ratio, the more closely two nodes relate to each other, the greater the degree of dependence. The size of the nodes indicates the number of changes that a software item has undergone and a red colored ring shows that this element was changed during the time

interval being studied. The thickness of the red colored ring represents the number of changes made to the software item in the time interval.

Each one of the animations allows to observe the changes in the dependencies of software items by using a line and a gray colored arrow which indicates the previous position of a software item (gray colored node) and its new position. Figure 5.25 illustrates a strip of animated visualizations whereas Figure 5.26 shows the use of arrows to indicate the position change of the nodes in a scenario that depicts a set of logical coupling dependencies (co-change).

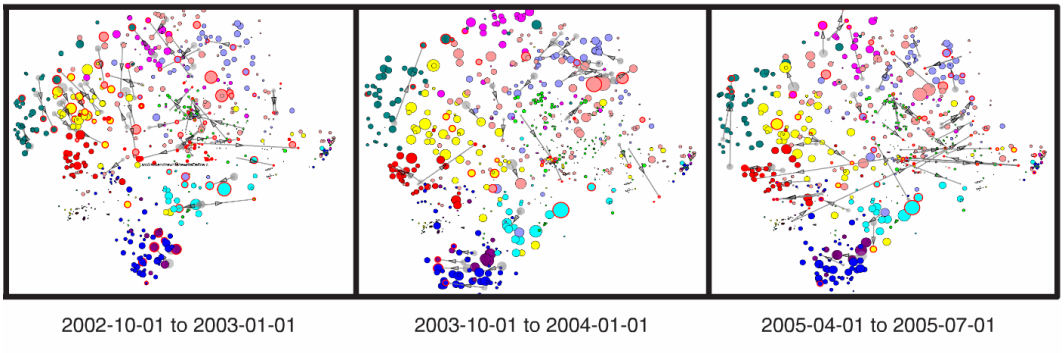


Figure 5.25: Strip of animated visualizations [Beyer 2006].

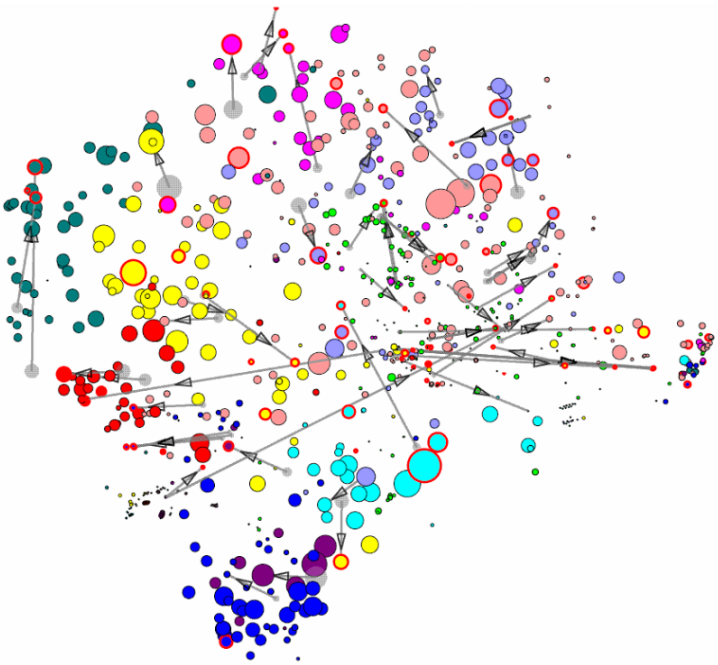


Figure 5.26: Use of lines and arrows to depict new node positions because of changes in the dependencies [Beyer 2006].

5.3.4 Software Cartography

Khun *et al.* [Kuhn 2010a] like other researchers point to the need of preserving the location of visual elements in time to render possible the comparison of information. Bearing this in mind, *Software Cartography* was proposed. Accordingly, the position in the space of visual elements in this visualization is calculated using as a base the similarities of the vocabulary used when naming software items. According to Khun *et al.*, the lexicon used by software systems tends to grow over time, but does not change as dynamically as its structure. This permits a robust and consistent visualization layout to be created for the representation of different aspects of software systems, including their evolution. The aforementioned approach facilitates users the understanding of the system to the users as they retain the same mental model over time.

The construction of the map of *Software Cartography* can be carried out by processing all the revisions available or using an incremental approach that adds individual revisions as required. The former method calculates the map for the whole evolution while the latter calculates the map cumulatively as each revision is processed. In either case, the result is a consistent map which conserves the position of the elements throughout the evolution of the system.

The processing of the data to create the visualization takes into account the terms that appear in system source archives, which are placed in a matrix of occurrence of terms to be indexed and ranked in accordance with their frequency of use. The terms of the lexicon include the names of classes, methods, parameters, variables, invoked methods, words in comments and literal values. Then, a distance is calculated among software items using as a base the similarity of the terms.

Figure 5.27 allows to observe that during the process of building a *Software Cartography* map:

1. Software items are placed in the plane in accordance with the distance between the terms.
2. The area of influence of a software item is determined according to its size and proximity with other software items.
3. The height of the mound is calculated based on the file size or class. Mound height changes as the element size is reduced or increased, but in any case the configuration of elements in the plane is not changed.

Figure 5.28 shows the succession of the representation of four system revisions in which the consistency of the visualization as well as the variations in the areas of influence and the height of the software items can be observed.

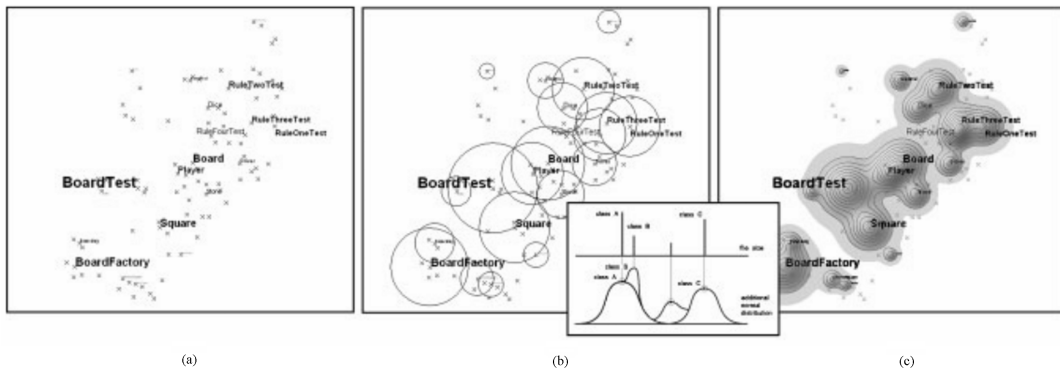


Figure 5.27: The process of building a map of a software system with emphSoftware Cartography [Kuhn 2010a]. (a) Placement of software items in the plane in accordance with the distance of the terms. (b) Area of influence of the software items according to their proximity and size measured by the number of lines. (c) Height of the mounds calculated with reference to the size of the system.

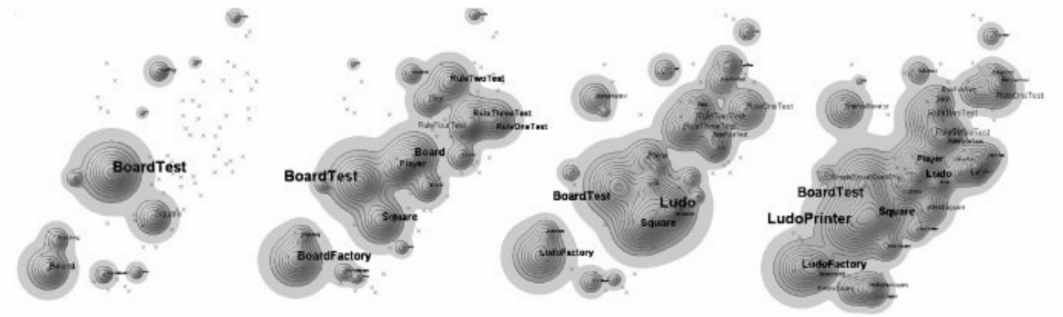


Figure 5.28: A series of four visual representations for the same number of system revisions using *Software Cartography* [Kuhn 2010a].

Software Cartography is a visualization that is integrated into Eclipse as a plugin to provide the programmer with a tool easy to access within his programming environment [Kuhn 2010b], and has the following objectives:

* Allow a quick exploration and understanding of the system.
* Facilitate the comparison of metrics.
* Support the construction of a framework that facilitates a common understanding of the system and the collaboration between the team members.
* Allow the connection between two or more programming environments (IDEs) to provide each other with information on the activities carried out in the software items.

It is important to highlight that a common characteristic of these visualizations is their scalability in representing structures, dependencies and metrics.

5.3.5 Graphs

As it was mentioned in section 3.3.1.5, graphs are useful for showing relationships between elements, and in accordance with Gansner *et al.* these are an ubiquitous structure in software engineering structure that is used for the representation of the structure of systems and the relationships between their components. In line with this, Gansner *et al.* proposed a library toolkit, based on the specification of a common language, that have been used in several areas of software engineering to create, filter, represent, animate and interact with graphs [Gansner 2000].

One notable visualization for the representation of graphs over time is *GEVOL* [Collberg 2003] from data extracted from *CVS* repositories. The objectives of this visualization are to show details of the changes made to systems, the inheritance of software items, method calls, the flow control of programs, who made the changes, when these were carried out and how the complexity of the system has changed over time(*e.g.*, days).

GEVOL creates several graphs to represent changes for the points over an evolution period of time. This representation layouts graphs into strips and placed them next to each other. So, it creates a succession of graphs that are based on the layout of their predecessors to maintain the mental map of users (elements maintain their positionover time), in spite of changes such as the addition or deletion of elements.

This visualization uses color coding to indicate the age of changes and so, initially, all nodes of the graph are of the same color (*e.g.*, red, yellow, green.). However, the color of nodes that have not changed over time undergo a color transition to become completely blue. But a node recovers its original color when a change is made to the software item it represents. To depict this feature, Figure 5.29 shows six snapshots of a call graph, which allows to observe the color transition of nodes from red to blue. In quadrant $A2$ of this figure, a purple box that shows a group of elements that have not changed recently can be observed, as well as an area that portrays red elements that have been recently modified. Thereupon, Figure 5.29 (B1) shows some graph areas that have transitioned to blue, whereas other areas of the graph have reappeared as red color to depict that elements in this area have recently been changed (see the center of the graph).

5.3.6 Radial Visualizations

Evolution Radar [D'Ambros 2006a, D'Ambros 2006c, D'Ambros 2009b] is a visualization that represents the logical coupling between packages and system elements. Information on the logical coupling is obtained by identifying

software items that have changed together, according to the revision history and changes made to the system [Gall 2003]. This permits to obtain information of the architecture of systems and may eventually support the implementation of structural changes to improve the susceptibility of systems to maintenance, and to facilitate the prediction of the evolution of the system.

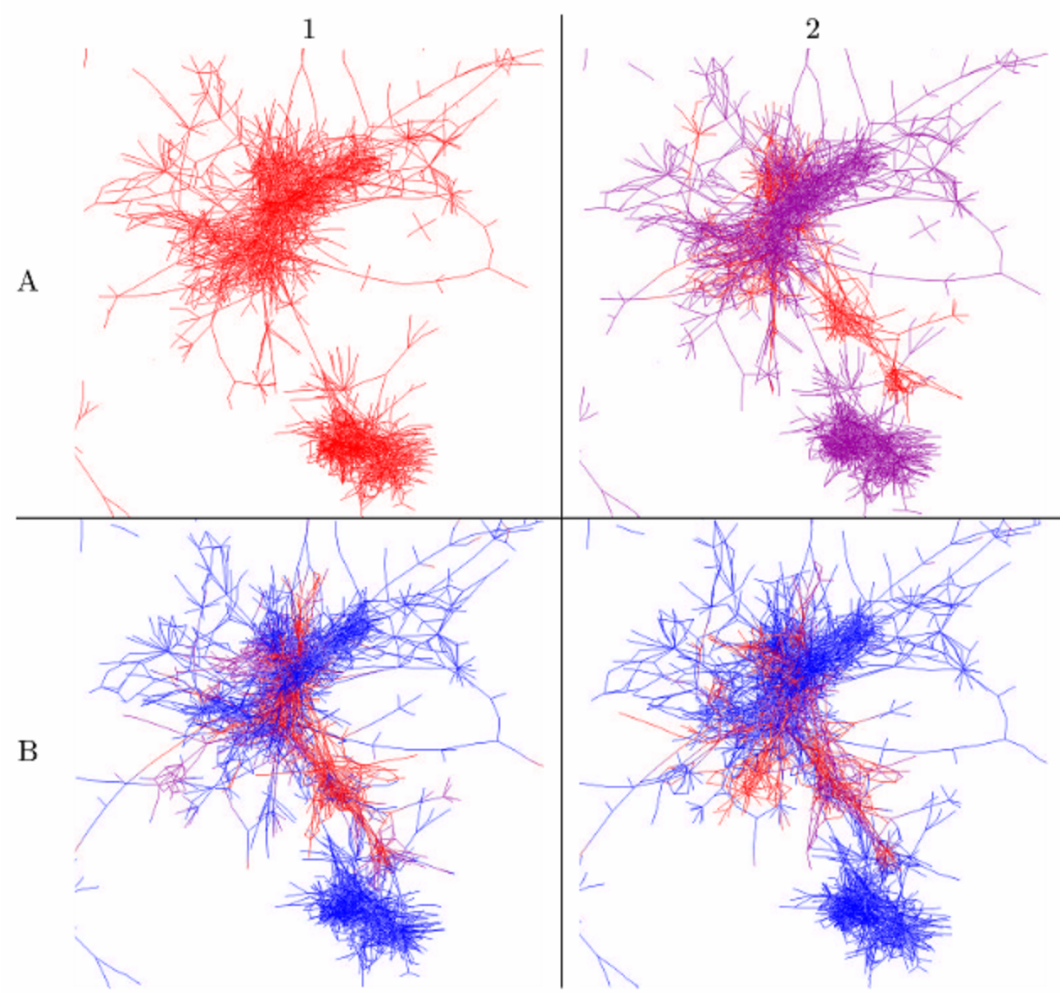


Figure 5.29: Succession of call-graph visualizations using GEVOL [Collberg 2003].

This visualization consists of a radial representation that is divided into a number of sectors that contain colored circles. Sectors in the representation depict system packages, whereas circles represent to files. So, the number of sectors in the visualization is proportional to the number of packages in the system and the size of each sector is related to the number of files it contains.

The coupling relationships are represented by the relative distance between a selected package and the files in other packages.

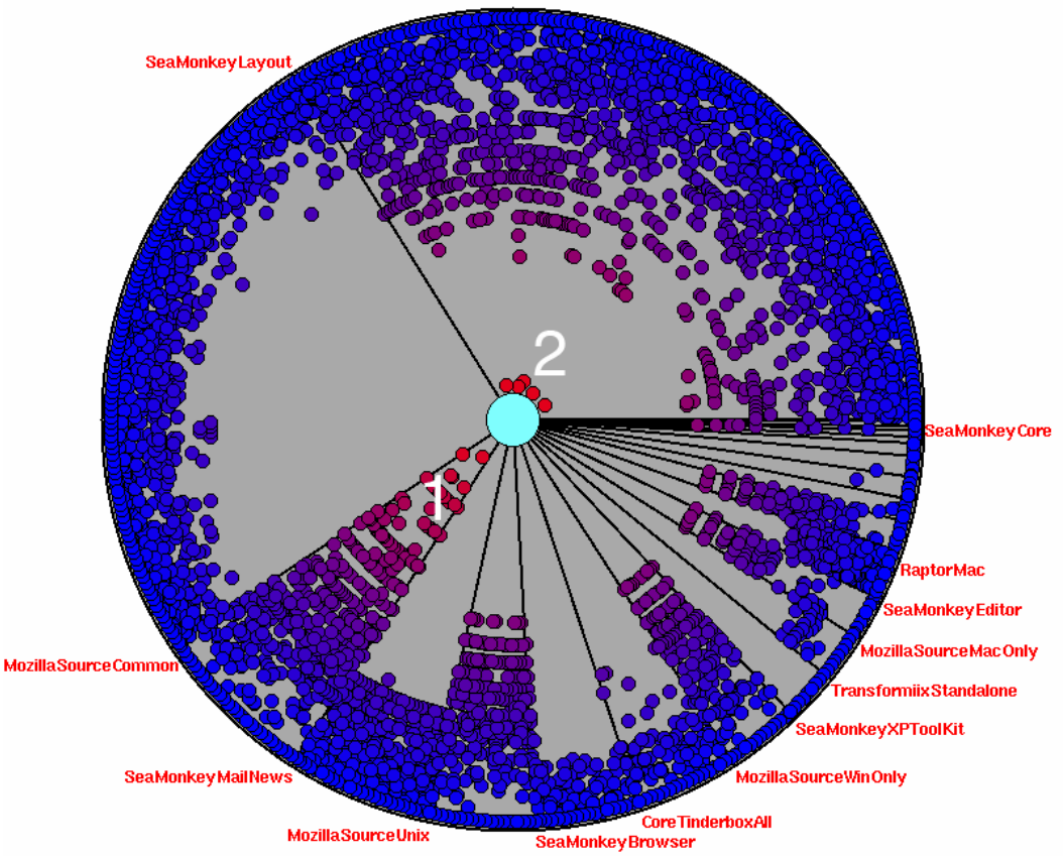


Figure 5.30: Visual representation of the logical coupling between packages and files [D'Ambros 2006a].

Figure 5.30 allows to observe a package selected by the user (highlighted by a light green circular shape in the center of the visualization) and that the files of the other packages (whose colors range from blue to red) are located at different distances of it, according to their coupling relationships (the files close to the center have a stronger coupling relationship with the package). Additionally, the use of colors reinforces the representation of coupling: the deep red color indicates a stronger coupling with the selected package, while the blue color indicates less coupling. Finally, the size and color of files (circles) can also be used to represent different type of metrics

5.4 Discussion and Conclusions

The dynamic nature of software systems makes it difficult to precisely anticipate the size and growth of the underlying architecture and structures. This implies that the design of their visual representation must be scalable and capable of accommodating both growth as well as changes over time

(*e.g.*, aggregation, deletion and relocation of elements). This means that the visual representation must be consistent over time in order to maintain the mental map of users: the elements must appear in the same position on the visualization in all revisions of the system which are represented. This entails that the removal or relocation of an element must be represented by visual notations indicating the action taken and allow users to maintain the mental map of the system.

Another aspect that must be taken into account is the representation of the metrics and the relationships between software items. It is common that the number of relationships between elements increases in accordance with the size of the system, which adds an additional aspect to the problem of the design of scalable visualizations and greatly increases the difficulty of achieving an adequate design. This may have major implications when the full or partial visualization of the evolution of a system is carried out. The temporal aspect is linked to the identification of patterns related to differences or similarities between revisions or periods of time and requires the representation of a larger number of visual elements.

Additionally, it should be considered that the visual representation of a system (or part thereof) should include elements that allow the rapid identification of patterns of interest according to the problem to be solved. As a result, it is necessary to make the representation on a single screen thus permitting that, once a pattern has been identified, its review is conducted on an additional visualization.

The analysis in this chapter shows that visualizations based on city metaphors can allow the representation of architecture and metrics of large software systems [Panas 2003, Panas 2005, Wettel 2007, Wettel 2008a, Bentrad 2013]. However, the majority of studies that make use of this metaphor are oriented towards representing only a revision and not the evolution of a system. Some exceptions to this trend are the works carried out by Wettel *et al.* [Wettel 2008a] and Steinbrückner *et al.* [Steinbrückner 2013]. Wettel *et al.* visualize several revisions using side by side representations, while Steinbrückner *et al.* utilize only a representation in order to visualize the evolution of system architecture, aggregation, deletion, the relocation of elements and metrics.

Meanwhile, treemaps are scalable representations that can represent the system structure [Baker 1995, Balzer 2005b], inheritance relationships and metrics of large software systems [García 2009b], although these visualizations suffer from limitations in the representation of other types of relationships such as the coupling between software elements as well as the evolution of systems. The visualization of relationships between software items can be overcome by the combined use of treemaps with graph structures [Balzer 2005a], and

the representation of the evolution of systems can be achieved with the use of side by side visualizations. However, the latter is impractical when the representation of the evolution of a system for an extended period of time or a large number of revisions is required.

The strength of *Polymetric views* is that they allow the efficient representation of the metrics of software items for one [Lanza 2001b, Lanza 2003, Ducasse 2005] or several revisions [Lanza 2001a]. The representation of the relationships between the software items is carried out in an acceptable fashion for these visualizations for a single revision, although by using graphs and trees they may suffer from scalability problems. In comparison, the representation of the structure of systems and the relationships between software elements is accomplished quite efficiently by PNL [Abuthawabeh 2013], *Lattix* [Sangal 2005], IMMV [Abuthawabeh 2013] and *EXTRAVIS* [Holten 2007]. These visualizations manage to represent a large number of elements and relationships, but are unable to represent the addition, deletion AND relocation of elements and metrics.

The animated visualizations that were studied in this chapter [Hindle 2007, Beyer 2006] showed weaknesses in aspects of scalability and representation of the evolution of software systems. Both Evolution Storyboard and Yarn are visualizations that may be impractical to compare a large number of time periods for systems that have evolved over long periods, but may be useful when it is necessary to obtain knowledge about the changes that have been recently made to the system. One of the main weaknesses that *Storyboad Evolution* presents is the use of side by side representations in order to compare time periods. Similarly, *GEVOL* [Collberg 2003] uses a succession of graphs in order to visualize the relationships between software items and its main limitation is its incapacity to represent large software systems.

Radial visualizations that were analyzed are capable of representing a large amount of data in a very attractive and intuitive way, such as *EXTRAVIS* and Evolution Radar [D'Ambros 2006a, D'Ambros 2006c, D'Ambros 2009b]. The scalability of these visualizations is one of their principal advantages, provided that adequate and appropriate representation and interaction techniques are used to select and filter elements. It may also be worthwhile considering that the use of these representations in the comparison of revisions or time periods may be complex because of their circular configuration, to which it must be added that the representation of a large number of aggregation, deletion and relocation operations of elements may also present scalability issues and may require the additional use of novel techniques and methods of visualization.

The above allow to conclude that the challenges that are implied in representing the relationships between the software items in a scalable,

compact (visual representations represent the system or part of it and its evolution on a single screen), easy to understand (and use), and consistent form over time (maintaining the mental map of the user about the position of the elements over time) is a difficult challenge that none of the works studied fully overcame. It is thus advisable to take into account the main characteristics of each work studied, as well as others related to it, to carry out the design of tools that manage to solve the challenges mentioned.

Chapter 6

Team awareness and collaboration

Rastin caminó junto a ellos y contó anécdota tras anécdota. Por su parte Güindy reía mientras seguía a Cucho, quien al parecer tenía más claro el camino. Al cabo de un rato, el hambre los asaltó, llamaron al perro y se desviaron del camino en busca de aves silvestres. Después de varios intentos sin atrapar ave alguna, decidieron acorralar juntos a un cerdo salvaje que comía al lado de un árbol gigante, el cual agitaba sus ramas y silbaba al ritmo del viento. — El viaje de Güindy, A.González

Contents

6.1 Introduction

The development of software using GSD models has become a common practice among companies which makes necessary the use of tools to support collaboration and teamwork. The results of the study that was carried out in chapter 4 showed that *Team awareness and collaboration* is a research subject

in which researchers have focused their attention with the goal of supporting to the SDM processes.

This chapter deepens in the study of factors such as teamwork, cognition, communication, coordination, control, Team Situation Awareness (TSA) and Distributed Situation Awareness (DSA) (see section 6.2). Furthermore, it also exposes the factors involved in the design of tools to support team awareness and collaboration tasks (see section 6.3) and presents the analysis of several visualization proposals that have been designed for supporting those tasks during the SDM processes (see section 6.4).

6.2 Factors Involved in Global Software Development

A large number of companies has been motivated to carry out the development of software systems using GSD models motivated by the assumed benefits that could be obtained [Carmel 1999, Herbsleb 2001b, Saldaña-Ramos 2014]. So, it is advisable to consider the analysis made by Conchúir on the achievement, in practice, of some of the benefits that are often considered as the most important when these models are used [Conchúir 2009]. Conchúir found in his research that some of the assumed benefits were partially met, whereas others were myths that could not be verified as benefits in reality (see table 6.1).

Table 6.1: Assumed benefits of adopting a GSD approach.

Assumed benefit	**Reality**
Reduced development costs	Partial benefit
Leveraging time-zone effectiveness	Mythical benefit
Cross-site modularization of development work	Partial benefit
Access to large skilled labor pool	Partial benefit
Innovation and shared best practice	Mythical benefit
Closer proximity to market and customer	Partial benefit

To better understand the potential impact that GSD models can have in SDME processes, the study of Ågerfalk about the opportunities and threats of these models must be considered [Agerfalk 2006]. His work makes a correlation of the temporal, geographical and sociocultural factors with the communication, coordination and control factors with the aim of showing the opportunities and threats that are associated to these. In summary, the main challenge that is faced by the software industry when using GSD models consists in finding how to overcome the involved distances in the development

of global projects, according to the variables and correlations described in the research mentioned.

6.2.1 Teamwork

As it was mentioned above, the development of a software system is a complex process that usually is divided into several stages and tasks, that are interconnected and can be performed using different approaches. The development process may require several years and the involvement of a large number of people, which are usually organized into teams of two or more people (in some cases highly specialized), that frequently are distributed globally [Kiekel 2011]. Therefore, it is necessary to keep in mind the difference between a group of people and a team: a group of people is a collection of people with functions that can vary considerably and whose work does not necessarily depend of the other members of the group, while a team is formed by a group of differentiated and interdependent individuals [Klimoski 1994].

In this context, it is also important to note that teamwork requires the communication, interaction and coordination between individuals, and even the mutual control of the work that is performed by others (with independence of the role that is fulfilled by each team member). This differs from the work focused on tasks, in which individuals carry out tasks with independence of what is performed by other team members. However, the skills for teamwork and work focused on tasks are complementary to achieve the objectives assigned to the team [Salmon 2013].

Organizations have adopted teamwork for software development because they believe that the effective functioning of teams can provide good results [Fiore 2004a] due to the diversity of its members in terms of experience and expertise. However, it must be pointed out that the differences of nationality, culture and geographical location of individuals, when working under GSD models, could become advantages or disadvantages according to the management practices that are put in place [He 2007, Kiekel 2011].

Several arguments in favor of teams is that the work that is assigned to them can be performed in a more effective, efficient and quick manner [Fiore 2004a, Kiekel 2011] that a single individual, because:

* Can detect, recognize and solve problems faster.
* Can plan, acquire knowledge and design solutions or products in less time.
* These can adapt quicker to changes.
* Are able to assess a situation, make better decisions, and consider the combination of knowledge and experience of its members.

* Have the ability to better manage stress and tasks during periods of heavy work loads.
* The coordinated action of teams, as one of their intrinsic properties, makes them to act as a block when they face situations, and have a greater reaction capacity to solve complex problems.

It is noteworthy to mention that for a team to work properly, its members should be able to work together effectively [He 2007]. This is extensible to the general context of software development, considering that the joint working between teams is also needed, particularly when a GSD model is used. Thus, teams and team members, need access to the information required to perform their tasks [Kiekel 2011]. The bottom line is that at the end of the day the modules and system elements must be integrated and work together, according to the architecture of the system [Mockus 2001].

6.2.2 Cognition, Communication, Coordination and Control

The factors of communication, coordination and control are closely interrelated. It is important to add the variable *cognition*, taking as a reference point the work done by Comfort [Comfort 2007]. Crisis management [Comfort 2007] has several similarities with the (sometimes unpredictable) changing and dynamic nature of the development of software systems: software development necessitates that the actors be rapidly adaptable reacting appropriately when new events occur and when new scenarios arise, particularly when development is distributed. It should be borne in mind that these activities can be carried out using a vertical or horizontal approach. This implies that in the first case the project manager carries out the appropriate action following a hierarchical approach; while in the second case the action can be carried out by any member of the team, an approach which takes into account the fact workers are professionals who possess the skills and the capacity to react and take action whenever they consider this to be appropriate [Carmel 2001]. These four factors are discussed below as well as in the next section, their relationship with the concepts of team cognition and team awareness is discussed and used to define a framework for team awareness and collaboration.

Cognition: Using the definitions provided by Comfort [Comfort 2007] with some slight modifications, cognition can be defined as a process that depends upon a clear mental model of how the system under observation should work and therefore it activates the processes of communication,

coordination, and control when discrepancies are detected between what individuals view as normal performance and the change in status of key indicators that alerts about potential process deviations.

Communication: The aim of communication is to *communicate* or to make others participate in something by means of a common language [Luhmann 1992, Dainton 2015].

The types of communication that can occur in the context of GSD between individuals are diverse and include the following:

* Face-to-face and distance enabled (e.g., email, telephone, video-conference).
* Synchronous and asynchronous.
* Formal and informal.
* Centralized and decentralized controlled communications.

The main problem during the development of software projects is the lack of communication [Agerfalk 2006] among members of development teams, regardless of the model used (collocated or GSD). David Parnas (see sidebar in [Agerfalk 2006]) notes that poor communication is a problem at several different levels between:

* Users and developers.
* Architects and programmers.
* Programmers and other programmers.

With regard to the lack of communication among members of the development team, Ramesh [Ramesh 2006, Colomo-Palacios 2013, Colomo-Palacios 2014] summarizes the following points:

* Difficulties in initiating communication.
* Lack of understanding
* Dramatic reductions in the frequency of communication between members of the team.
* Increases in the cost of communication in terms of time, staff and money.
* Time zone differences.

It is important to highlight that in a GSD environment the aspects that have already been mentioned [MacMillan 2004, Mockus 2001] add delay times to the tasks of the SDME process. So, additional measures to mitigate the negative impact of such aspects are required [Mockus 2001, Herbsleb 2001a, Herbsleb 2003], among which are the following:

1. Divide the tasks up optimally between different sites.

2. Increase communication between team members using appropriate tools for that purpose.
3. Contract or identify staff internally with the skills and experience necessary to carry out tasks efficiently.
4. Tools that maintain awareness of activities carried out between team members.

Coordination: Coordination can be implicit or explicit. Implicit coordination depends on the knowledge of the team and their ability to make decisions in critical situations with a reduced level of communication, so team members must adjust their behavior dynamically to anticipate actions and address proactively the tasks that require it [Khan 2010, MacMillan 2004]. This type of coordination is associated with high performance teams which members clearly understand the needs and responsibilities of its tasks, and provides advantages when workload is high, because less communication is required [MacMillan 2004]. Whereas explicit coordination is the process of organizing things, people or groups to work together properly [Godart 2001].

In order to carry out tasks related to coordination, it is necessary to use mechanisms that permit the processes of effective exchange of information and understanding in order to align priorities and the actions of different actors in order to achieve a shared goal [Comfort 2007, Kotlarsky 2008, MacMillan 2004].

The advantages and disadvantages of using either type of coordination depend on the circumstances and the tasks that are performed [MacMillan 2004]. It should be added that coordination mechanisms acquire particular importance when a GSD approach is used with regard to the distances (be they geographical, cultural and distances of time) among the different sites involved in the development of the project [Herbsleb 2003].

During software development, the process of coordination necessarily implies that the people who are working on a project have agreed on a number of elements, starting with the detailed design specifications that will permit the construction and organization of the components of a system. So, participants can work together based on the user needs and the requirements of the organization. According to Kraut [Kraut 1995] the following are factors that affect the coordination of systems development:

Scale: Large-scale projects require the involvement of a large number of

people and it is not possible for one person or a small group to be aware of all the details [Cataldo 2007]. The coordination of a project becomes more difficult when its size and complexity increase [Kraut 1995], leading to a necessary division and specialization of labor.

Time: The development of large software projects usually spans several years and their maintenance may last for many more years, and consequently the involvement of a large number of people will continue to be necessary even when the development stage has been completed.

Uncertainty: The development of many software systems becomes an unpredictable process for one or more of the following reasons:

* The system specifications are incomplete due to the loss of information that occurs when translating the requirements of users and business into specifications. This loss of information occurs because, in many cases, analysts, architects and designers are not specialists in the field to which the problem is related and thus cannot understand all the details (or because even if they have understood them they have not included them in the specifications).
* A prototype has not been developed that allows the basic functionality of the system to be captured so that subsequent modifications can be carried out based on the specifications and user feedback.
* The environment of organizations in general is changeable, so system functionality requires change over time. This produces changes in specifications during the process of development, or subsequently when the system has already begun to operate [Lai 2003, Cataldo 2007].
* Lack of a clear definition of the methodology, lack of quality control, detection of problems, errors and failures (sometimes belatedly) during the development of the project [Rook 1986].
* Problems with team members due to inadequate performance or the number of participants (either too many or too few) [Rook 1986].
* Differing points of views between different actors who intervene in the development of the system.
* The complexity of a large-scale system which has to be developed over a long period of time [Lai 2003, Cataldo 2007].

Interdependence: Systems are built on the basis of components (in some cases thousands of components), which then have to be integrated

precisely and accurately.

Formal and informal communication: Efficient communication is the key for achieving a good level of coordination between team members and among the sub-groups that are scattered throughout several different geographic locations. Given this, it is necessary to use both formal and informal communication [Lai 2003] according to the type of problem that is being addressed.

Kraut [Kraut 1995] enumerates a list of coordination techniques, among which the following can be found:

* Formal impersonal procedures (project documents and memos, modifications requests, error tracking procedures and data dictionaries).
* Formal interpersonal procedures (status and design review meetings, and code inspections).
* Informal interpersonal procedures (group meetings, collocation of requirements and development staff).
* Electronic communication (email and electronic bulletin boards).

Control: Taking into account the diverse factors which cause uncertainty during the software development process, control can be defined as the ability to maintain actors and their actions focused on achieving the goals and objectives which have been set [Comfort 2007] in order to produce an appropriate software product; on time and within budget [Rook 1986]. All this is in accordance with the requirements, specifications, terms, costs, standards, policies, standards of quality and other factors inherent in the process of software that if they are addressed in a timely fashion, will make a successful resolution of the project much more likely. Control is usually classified into two categories: formal and informal.

Formal control consists of the monitoring and evaluation of behaviors and outputs; whereas behavioral control consists of controlling how people behave, and output control consists in measuring the effect of people behavior on output of the process. In order to utilize behavioral control appropriately, it is necessary to know precisely what actions have to be performed during the process of project development; transforming a set of inputs (e.g., requirements and designs) into outputs (e.g., a system that functions adequately). It is thus necessary to determine the actions which need to performed as well as evaluate the actions carried out by individuals in order to determine whether their actuation has been appropriate. While output control can be

used when it is feasible to measure the performance of individuals according to the results produced and the results that were expected to be produced, in a form independent of the behavior of these individuals [Olchi 1978, Kirsch 1996].

Concerning informal control, the two best-known types are clan control and self-control. A clan is a group of people with mutual dependencies and shared goals, objectives, values, philosophy and common beliefs; as well as a strong sense of identity and group belonging, whose behaviors that are not known a priori, and whose results may change over time. In this type of control, a group is a self-monitored unit: supervision is carried out by each member of the group, and peer pressure enforces the accomplishment of tasks according to the goals of the group. Consequently, the individuals who participate in these groups should be carefully selected and it is essential that they have appropriate training. With regard to self-control, in this type of control individuals set out their own goals, monitor their own performance, evaluate their own progress and are motivated to carry out their work; so this type of control is useful in tasks that require autonomy, creativity and intellectual work [Kirsch 1996].

Drawing a parallel between the uncertainty that is present during the development of software projects with crisis management, it is useful to consider the approach employed by Comfort in relation to the individuals and group actuation of the members of teams involved in control tasks. Comfort [Comfort 2007] considers that control can be maintained in highly complex, changeable situations if the following factors are present:

* Shared knowledge.
* Commonly acquired skills.
* Reciprocal adjustment of actions to fit the requirements of the evolving situation.

The goal of control activity is to enable and facilitate decision-making by comparing the information obtained (in the form of status and progress reports) of the activities of project development, verification, validation and testing, Software Quality Assurance (SQA) [Fischer 1978, Kitchenham 1989, Kan 2002], SCM [Rook 1986], bug tracking, incident and change control systems [Barbara 1987] with the results that are expected to be obtained from the project in accordance with:

* System planning.
* Project procedures and standards.

* Models and risk analysis.
* Configuration management plans and procedures.
* System requirements.
* Top level and detail design.
* SQA requirements and plan.
* General and detailed test plans.

Control, in its ideal form, is a process of continuous feedback that seeks to identify and eliminate potential hazards from:

Detection and control of unanticipated changes: This is devoted to the detection of changes that must be made because of variations in system requirements, in such a way that they have to be controlled in order to maintain the integrity of the design and thus should be incorporated in an orderly way.

Detection and error correction: One of the main objectives of the control process is to detect and correct deviations or errors in order to align the development in accordance with the requirements, specifications, goals and objectives of the project.

Monitoring: It aims to measure the progress of the project by means of meetings, specialized tools and the use of SQA such as metrics, technical reviews, and walkthroughs.

Evaluation: It consists of analyzing the control information which is available, carrying out an assessment of the consequences of possible alternatives actions, choosing one of these alternatives (decision- making) and plotting the course which should be followed.

6.2.3 Team Situation Awareness

Teamwork allows members to familiarize themselves with the other members of the team and learn about the knowledge, skills, experience, background, personalities and habits of each other. That mutual knowledge varies with time and increases as it passes, which enables better planning of work [Fiore 2004a].

From a cognitive standpoint, teams build a mental model[1]. from the shared understanding of tasks and involves knowing the procedures, actions and strategies to implement them. Hence team members should have common expectations and understanding of tasks [He 2007].

[1]The mental model of an individual is how knowledge and information are represented by her mind and reflects the tendency to categorize what she knows [Klimoski 1994].

A shared mental model is the representation of knowledge in an organized manner with respect to tasks, situations, response patterns, goals, strategies and working relationships. So, one can say that the mental model of the team is the way it thinks collectively and characterizes situations according to beliefs, assumptions and common perceptions [Klimoski 1994].

However, a team mental model is not the sum of the mental models of individuals. However, it takes the relevant knowledge of team members and transform it into the team's knowledge with the aim of guiding decision-making and actions to achieve the goals and objectives of the tasks entrusted to the team [Fiore 2004a]. In this context, Cooke noted that the elements which may be included by the cognitive processes of teams [Cooke 2013] are the following:

* Learning.
* Planning.
* Reasoning.
* Decision making.
* Problem solving.
* Remembering.
* Designing.
* Assessing situations.

Accordingly, team cognition is build up from the interactions between team members [MacMillan 2004, He 2007] when they:

* Work together.
* Clarify their individual roles for each task.
* Distribute the sub-task to be carry out.
* Communicate by different means to build and maintain a shared mental model of the team situation [MacMillan 2004].
* Meet in person or virtually to share view points and concerns.
* Coordinate project activities.
* Observe the work of others and learn from it.
* Monitor the progress of activities.

Team cognition is the ability of the team to acquire, process, store and use knowledge when they perform tasks, especially when the collaboration of a large number of people is required in realtime to resolve some urgent situation [Kiekel 2011]. Some important features to consider regarding team cognition [Fiore 2004a, Fiore 2004b, He 2007] are listed below:

* It is not the sum of the individual cognition of the team members, but it does use of the individual cognition of members.

- It is the sum of the behaviors of the team during the communication, coordination and control activities.
- The team makes use of metal models concerned with project and task related aspects.
- Teams are aware of the goals and objectives of the work they perform as well as the status of activities and tasks (Situation Awareness (SA)) [MacMillan 2004].
- Team members exhibit behaviors and attitudes that offer evidence of the coordinated action that takes place among them [Fiore 2004b].

Endsley defines SA as the perception of elements in the environment during a given period of time, as well as the understanding and forecasting of these elements in the near future [Endsley 1995, Stanton 2001, Salmon 2013]. In this research SA is defined as the degree of knowledge about the status of tasks, activities, changes and the resolution of problems related to SDME processes. Furthermore, it also considers that situational awareness is important during software processes at both the individual and team levels. Then, the following points deserve special consideration:

Individual perspective: From an individual perspective, situational awareness is the consciousness of the state of things of a particular team member. The consciousness and knowledge of individuals with regard to tasks, and the factors that affect their development, permit the successful completion of these.

Team perspective: From a team perspective, SA is the consciousness and knowledge shared by team members about the status of the project. Each team member has specific awareness of the factors related to their tasks but also has an overall perspective of the project, in terms of team awareness, that allows its collaboration with other team members to contribute to the project in general.

This kind of awareness is known as TSA. It is important to highlight that TSA is closely related to mental models and team cognition, as it is explained later. The construction of TSA, like the construction of shared mental models and team cognition, is not the sum of the individual SA of team members, but it is the composition that originates from each individual perspective and the points of coincidence of team members about the state of things of the overall project.

The way individuals process information and create mental models of situational awareness depends on their goals, skills, experience, training and the role they play in the project. So, project managers are interested in aspects

of higher level [Leinonen 2005] and programmers in specific details, as it was discussed in chapter 2.

The experience of teams and individuals in similar projects is invaluable in the construction of SA, including knowledge regarding to which are the abilities of the other team members and team operation [He 2007]. However, each project is unique because of the problem that seeks to solve, the difference between the approaches that are used to solve specific issues and the interdependence of internal system elements. So, while a software system has common factors in relation to another, the problems that teams and individuals must face and the variables to be considered, are different. Thus, the construction of situational awareness requires time and the accumulation of experience of the team and individuals during the development of a project.

6.2.4 Distributed Situation Awareness

A recent approach, that is complementary to the above, is known as DSA. This approach is based on systems in which individuals and technological elements are considered as agents that interact, have different purposes (for the tasks and activities they carried out) and are in possession of their own SA of the tasks they perform and the project in general. The idea behind is that team members do not need to know every detail about the project on which they work, but only those details that allow them to perform their tasks. However, this implies that team members must be aware of the state of the project, the tasks of which they are responsible, and the information that others need to know to make it available [Stanton 2006].

It should be mentioned that access to the same information does not produce an identical situational awareness in team members, because of their particular goals, tasks, roles and experience, that leads them to use and interpret information in different forms. Knowledge is distributed in the environment and SA of an agent may be different but compatible with SA of other agent. Therefore, the performance of certain tasks that are interrelated requires the collaboration among agents, so the compatibility of the SA of agentes is useful.

According to the DSA principles enunciated by Stanton *et al.* [Stanton 2006], the following points are important:

* Both human and non-human agents have their own SA.
* Each agent has its own point of view about a given situation, but it could be different to the point of view of other agents.
* Agents have leading roles in the development and maintenance of the SA of other agents through the interaction that takes place among them.

* Agents compensate the lack of SA of one or more agents in certain situations.
* The points of view of agents about the system could change over time, according to the tasks they perform.
* The knowledge that makes up DSA is activated at different times in accordance to the goals, objectives and requirements of situations, tasks and activities that arise and are carried out in time.
* SA coincidences between agents depends on the goals these have.
* Communication between agents can be non-verbal and use electronic or other custom mechanisms.
* DSA helps to the cohesion of loosely coupled systems.
* The perspective of agents may be redundant, but it is always complementary in the context of collaborative environments.
* SA is an emergent property of the system and its parts, and not something that exists in the minds of individuals (from a DSA perspective).
* DSA can be seen in collaborative environments as the result that is obtained from the interaction between agents and their behavior in the environment.

6.3 Considerations in Designing Awareness Workspaces

The design of a Situation Awareness Workspace (SAW) requires the identification of the elements that an individual or team should be aware of, in accordance with the goals and objectives of the project, and the tasks or activities that have been assigned to them. Thus, to support software maintenance, for example, it is useful that the SAW provides information to help understanding the changes that are made to the system architecture and which can help improve the performance of team members and the team in general [Salas 1995, Fiore 2004a].

It is important to consider the points listed below when designing a SAW:

* It is common that the development of software systems is carried out using a GSD model.
* The goal of a SAW is to facilitate the collaboration among team members when carrying out tasks and activities which are their responsibility.
* SDME is a dynamic process that makes it difficult to keep up to date information on the activities, tasks and patterns of interest that could be considered during the design of SAW.

* Decisions must be taken immediately or in short periods of time when dealing with complex and changing environments, such as those of SDME, so having at hand updated information is always required.
* To have up to date information available on the current state of things is important even when the members of the team perform trivial tasks.
* A SAW should provide information on the current state of things, but it is necessary to consider those elements which can reveal future changes about the state of the SDME processes with the aim of deciding the best course of action to be carried out.
* Erroneous information about the state of the process can lead to mistaken decisions.
* Team members need to be aware of what happens in relation to their tasks and the project in general, and they also must be able to interpret different situations according to the goals of the project so that they can take decisions based on those goals and carry out the pertinent actions in a timely manner (understand the situation -> take decisions -> perform actions).
* The lack of training and skills of team members can lead to a misinterpretation of SA [Salas 1995, Endsley 1995] and can lead to take erroneous decisions.
* The formation of teams with cohesive structures in terms of knowledge, skills and control of activities is a difficult objective to achieve.

The visualization of software systems and their evolution arises, then, as a viable alternative to develop a SAW because of its ability to convey information and provide mechanisms for interaction with users. The information provided by visualizations about what is happening on the system could lead to the activation of the cognitive processes of team members and thus, initiate the communication, coordination and control tasks that are required according to the particular situation and circumstances.

According to the focus of this research, the appropriate design of visualizations should consider the perception, cognition and sensorial abilities of users [Card 1999b]. Therefore, the use of visual elements to allow the immediate comprehension of information (preemptive processing), without previous training and independently of the culture or origin of PMs and programmers should be taken into account. In order to do this, the use of visual representations, patterns and colors based on international conventions and the principles of Gestalt laws could also be considered [Ware 2004].

It is relevant to highlight that a large number of scientific papers make reference to the elements that must be considered when designing visualization tools to support the tasks involved in the SDME processes. Following

that line of research, Young and Munro [Young 1998] identified 6 factors to be considered when designing tools using *3D* technologies (representation, abstraction, navigation, interaction, correlation and automation), which due to its relevance could also be applicable to visualizations that have been developed using *2D* technologies. Maletic *et al.* [Maletic 2002] enumerated 5 additional aspects that is convenient to keep in mind (tasks, audience, objective data, representation and method). Thus, taking as a reference the factors identified by both research works, the design of a tool for visualizing software systems requires to consider the items listed below:

Audience: During the design of a visualization tool it is convenient to take into account the user characteristics, their needs and objectives with the aim of determining the most appropriate visual representations and interaction techniques to be used.

Tasks: The precise identification of the tasks that will be supported by the visualization tool is a critical factor during its design. This identification allows to define the use cases of the tool and determine the most adequate visual representations for such tasks.

Target data: The identification of data is also a critical factor because based on their characteristics, the audience and tasks, the design of visualizations is carried out.

Correlation: This consists in the correlation of information from different data sources, as well as the possibility of on linking visual elements with documents and source code.

Representation: A fundamental problem in visualizing changes of software systems is the appropriate selection of the graphic elements, metaphors and colors as well as make an effective combination of these elements to show different perspectives of data [Eick 2002]. This is of great importance for designing an intuitive visualization which allows to transfer information effectively while demanding a little efforts from users in terms of cognitive complexity.

Abstraction: In order to convey and allow the effective interpretation of information it is necessary to determine the level of information detail that will be presented (the user could choose the level of detail by means of interaction) as well as the representations and visualizations that will be used.

Navigation and interaction: SDME processes generate large amounts of information. The use of several visualizations linked together and also that these visual representations are implemented using navigation and interaction techniques (*e.g.*, focus + context, overview + detail, landmarks, zoom, search history and filtering).

The goal is that the user has the possibility to explore details while these are properly placed in the context of the visualization and the analysis that is carried out. As part of the interaction, elements can be incorporated that allow users some flexibility in customizing the visualization such as the possibility of choosing colors.

Automation: Ideally, the visualization of large software systems should be done automatically including the extraction of information from the selected data sources, and is desirable to incorporate new information as it is created. Young and Munro [Young 1998] remark that the possibility of allowing users to create the visualizations by means of interaction must be taken into account. Using this approach the users could make decisions regarding the system elements that will be represented, thus by means of this practical exercise a better understanding of the system under study could be achieved.

6.4 Visualization for Team Awareness

SCM tools are widely used by software development departments and, consequently, by development teams and programmers to record source code modifications. Thus, the interactions of programmers with software items are reflected in the data that is collected with the use of these tools, which then are accessed and analyzed using automatic mechanisms.

The aspects that visualization tools seek to support with the creation of shared knowledge spaces (to facilitate communication, collaboration and control) during the development and maintenance of software systems are diverse. While some of these tools try to provide information to other programmers, others are designed to assist project managers in decision-making. It is thus necessary to take into account factors such as the status of projects in terms of quality (measured using metrics) system changes (including the source code, dependencies, relationships and structure), understanding aspects of system design, socio-technical relationships and collaboration between team members.

6.4.1 Teamwork

In a GSD environment the existence of small teams located in different geographical locations is common. This makes necessary to design tools to enable team members to work together effectively. In this scenario Anslow *et*

al. proposed two visualization tools, which were called System Hotspots View (SHV) and SourceVis to support collaborative work [Anslow 2010].

Figure 6.1: System Hotspots View: visualization of system structure and metrics for supporting the collaboration between programmers [Anslow 2010].

Some visualization tools such as System Hotspots View (SHV) have a dual purpose: first to provide information on technical aspects of the systems in order to facilitate their evolution (development and maintenance) and secondly support the collaboration among team members. This visualization tool is based on *Polymetric Views* and it is aimed to support the understanding of the structure of a software system, as well as for the detection of structure and quality problems through the use of metrics applied to packages, classes and dependencies between software items. Its main contribution is to deploy a large wall screen that allows sharing knowledge about the system with the aim of facilitating the discussion, coordination and collaboration among members of the development team (see Figure 6.1). This tool, therefore, can be used by both programmers and project managers.

The effectiveness of SHV was tested by means of a user study. The results of the study showed that participants enjoyed the visualization with the use of large visual panel and the ability to observe a large number of details at once,

though they felt that the lack of tactile interaction with the representations was a drawback. Some participants made comments about the height of the screens; in some cases because the height of the users was very low and thus they were not able to properly observe the top of the visualizations; and in other cases because the participants were tall and they found it difficult to see the information at the bottom of the screens. Another observation made by the participants was related to the interruption of the continuity of the visual representation by the edges of the screens.

SourceVis, meanwhile, is a visualization tool that allows the interaction and discussion among collaborators by means of a large multi-touch table [Anslow 2013]. This tool allows the simultaneous interaction of several users and supports multiple visualization types such as SHV, *Class Blueprint* and vocabulary views using word cloud representations. Figure 6.2 shows to several users that are exploring dependencies among software items with *SourceVis*.

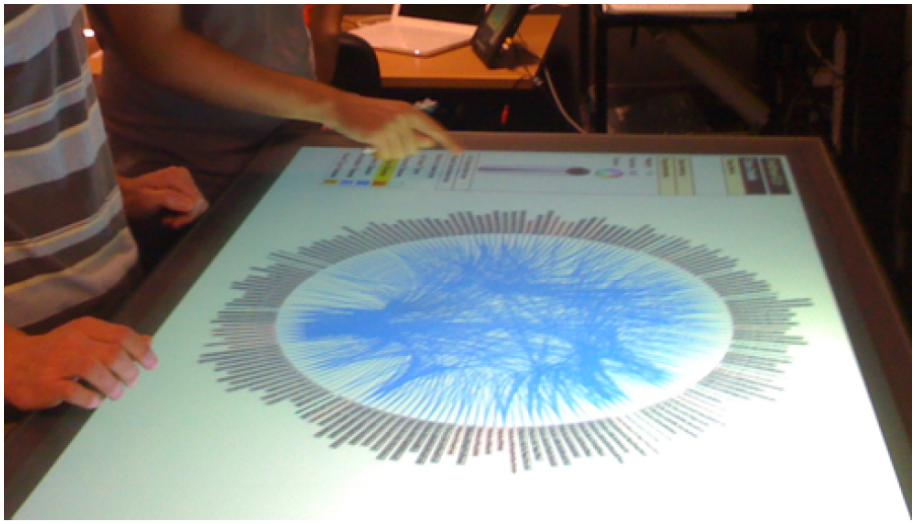


Figure 6.2: SourceVis: a large interactive multi-touch table for the interaction and collaboration between team members [Anslow 2013].

6.4.2 Situational Awareness

Ownership Map [Gîrba 2005, Hattori 2012] is a visualization that provides information to project managers about the collaboration that has taken place during system development. The purpose of this representation is to support decision-making and provide details on:

* The number of programmers that have participated in the development of the system.

* Modifications or parts of the system that have been developed by each programmer.
* The behavior of programmers during development and maintenance of the system.

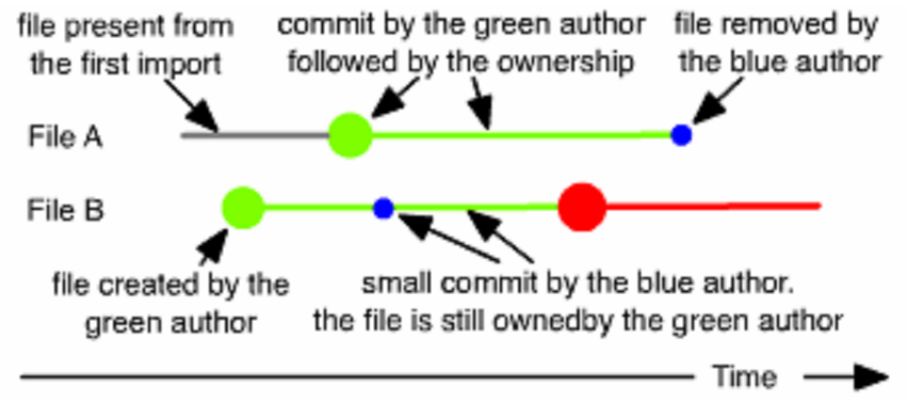


Figure 6.3: Representation of changes and ownership of the software elements in *Ownership Map* [Gîrba 2005].

Data used by *Ownership Map* were extracted from the logs of Concurrent Versioning System (CVS) and took into account details of the relationship between changes, at the level of source code lines, and programmers. Using this data as a base, it can be determined which programmer is the owner of a software item for a period of time or during the complete evolution of that item. The visual representation of programmers and software items is carried out using lines, circles and colors. So, lines depict software items whereas circles represent the magnitude of the change that was made and colors are associated with the programmers. Moreover, it alternates the color of lines to show the time intervals and programmers who have been responsible for a particular software item. Additionally, gray lines represent an unknown programmers or the initial import of software items into the software repository; a circle, that is painted using the color of a programmer, at the end of a line indicates that the item has been deleted (see Figure 6.3).

The use of this visualization allows to identify different patterns that are derived from the activities and collaboration between programmers, in accordance with Gîrba *et al.*. The following list explains these patterns and relates them, when it is applicable, with Figure 6.4:

Monologue: This pattern consists of the activity undertaken by a single programmer in most of the files over a period of time. It can be seen on the left side of the Figure 6.4, where the changes made by the programmer associated to the green color (indicated by $R5$).

Familiarization: It shows how a programmer carries out changes progressively in software items, until possession is taken of virtually all software items, as it can be observed in inset $R5$ of Figure 6.4 (note the pattern of the programmer depicted by the blue color).

Expansion: This pattern is associated with the addition of new files to the system by a programmer, as it is the case of the programmer identified by the blue color in accordance with the boxes $R8$ and $R12$ of Figure 6.4.

Edit: It is associated to changes related the rename of identifiers, the cleaning of comments or other necessary changes that add functionality to the system. This type of pattern can be observed as a vertical column of changes carried out by the same programmer, as is indicated in the Figure 6.4 by insets $R7$, $R11$ y $R15$.

Taking possession: The name of this pattern is derived from the fact that the programmer takes possession of most software items in a very short period of time, as is indicated by insets $R13$ y $R14$.

Teamwork: This pattern is identified where a group of programmers who take ownership successively of multiple software items in a short period of time. Figure 6.4 shows the involvement of multiple programmers in multiple periods of time by means of the highlighting of insets $R1$, $R3-4$ and $R12$.

Correction of errors: It consists of a specific intervention of a programmer to correct a mistake, and because of the few changes that this implies, the programmer takes possession of the software item for a very short period of time, (sometimes difficult to perceive) as depicted by the three points shown in the representation where a yellow point is indicated by a circle (see inset $R10$).

Cleaning: This pattern is the opposite to the **Expansion** pattern, and involves the removal of a significant number of software items from the system, as shown in inset $R2$.

Silence: It represents a period of time in which little or no change is shown. It is identified as a rectangle where the software items do not change of color.

CodeTimeline [Kuhn 2012] is a visualization tool that uses two visual representations. One of these visualizations evolves the concept employed by *Ownership Map* allowing developers to add notes and photographs with comments about the visual representation, in those points of the evolution

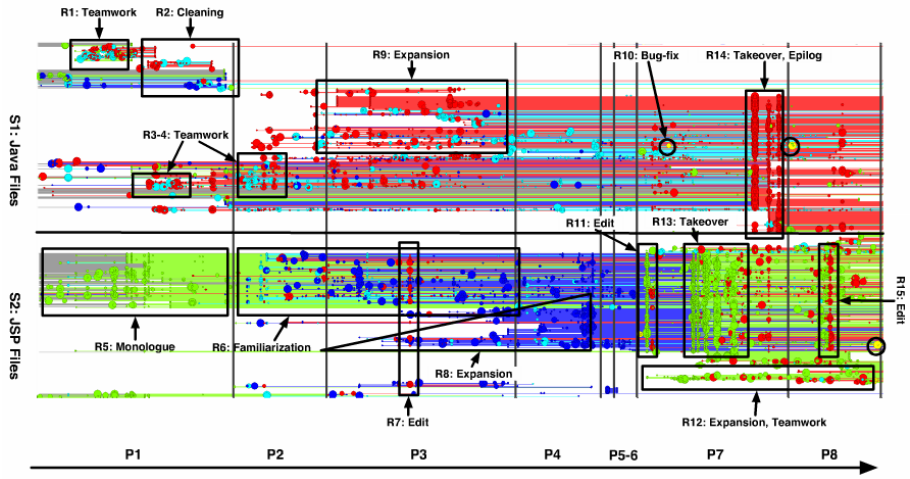
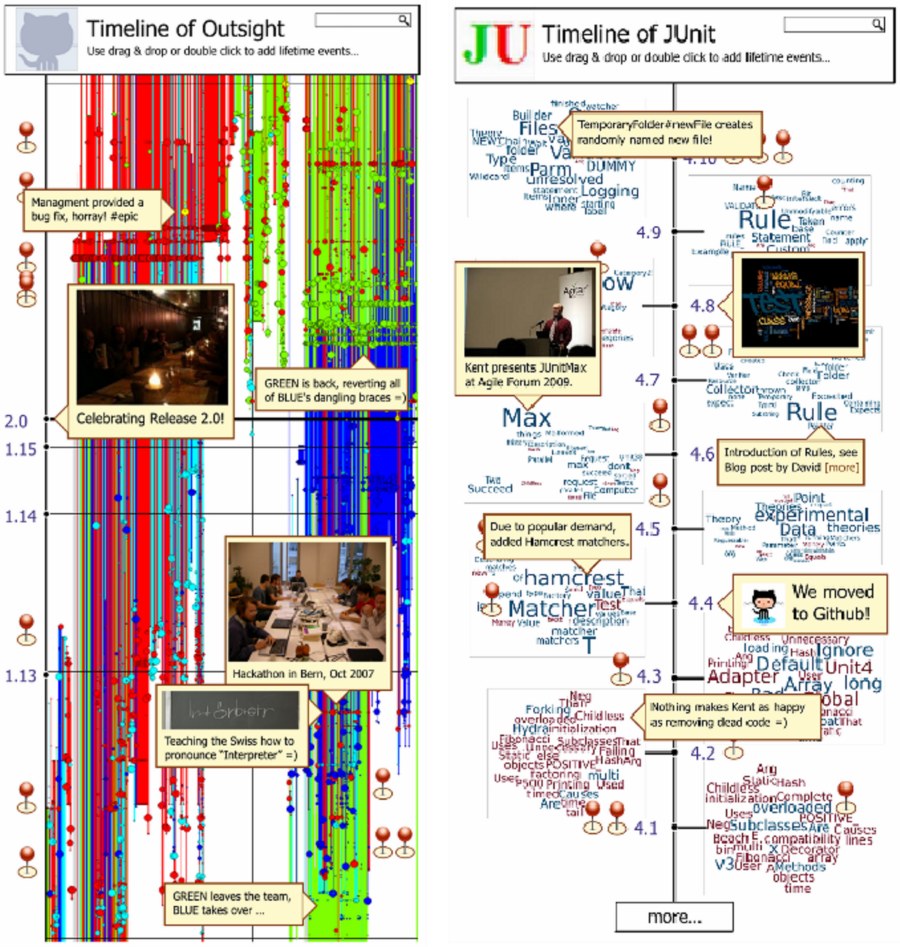


Figure 6.4: Visualization of the patterns of behavior of programmers using *Ownership Map* [Gîrba 2005].

where it is considered relevant to maintain the memory of events that have occurred during the development process (see Figure 6.5). Whereas the other visualization is a timeline that represents the terms and their frequency of use in the source code. This is achieved by using word clouds for each revision of the system, where the size of the words and the color represents the frequency in which the term is used. The blue color, in this representation, represents an increase in the frequency of use of the term and the red color a reduction in its use. The size of the term reflects the extent of their use in relation to the other terms in the cloud. Additionally, this visualization also allows to use annotations (see Figure 6.5).

Another visualization which permits the display of activities carried out by programmers is the activity viewer that Ripley *et al.* [Ripley 2007] built on the basis of Palantír. This viewer was programmed using *3D* technology, the logs stored by SCM tools and the information obtained from the local workspaces of each programmer as data sources. The activities about which information is obtained and represented includes *check-in* and *check-out* operations; the synchronization of the local workspaces with the software repository, as well as the editing and deleting of software items of the local workspace of each programmer.

This visualization displays information about the workspaces and the active software items by means of two visual representations which use cylinder graphs as their main graphical element. These representations focus on visualizing the activities of programmers and show details about the changes carried out in software items.



(a) (b)

Figure 6.5: *CodeTimeline*: (a) Notes of the development team concerning the *Ownership Map* visualization. (b) Visualization of the frequency of terms for each revision by using word clouds and notes from the development team [Kuhn 2012].

The local workspace of each programmer is visualized using cylinder stacks, where each cylinder represents a software item. The height of cylinder stacks reflects the number of activities carried out by programmers (see Figure 6.6).Cylinder stacks move forward or backwards in the visualization depending on the time elapsed since the last changes that were made in the workspaces that they represent. Then, cylinder stacks with the most recent changes are placed at the front of the visualization, while those with the earliest changes are located at the back of the visual representation.

Moreover, cylinder stacks are used to depict software items and the changes made by each programmer. In this context, each cylinder corresponds to a particular programmer and its size reflects the magnitude of the changes that have been made, as shown in Figure 6.7. It is noteworthy that (generally

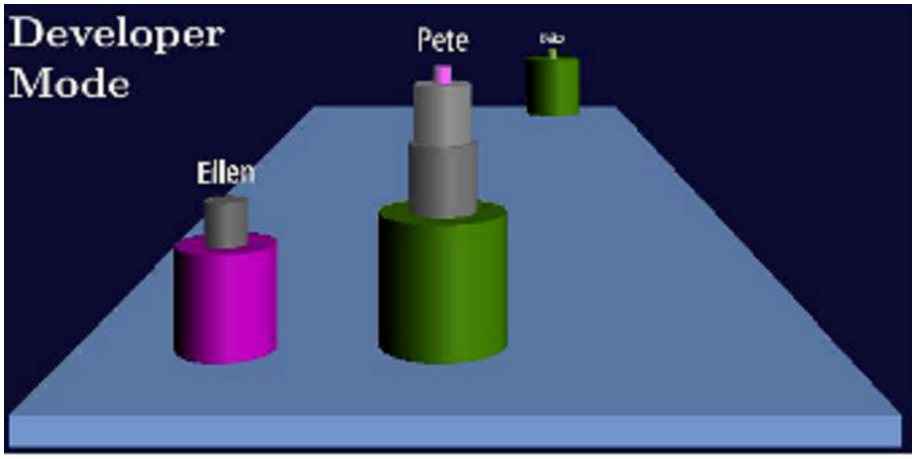


Figure 6.6: Visualization of the activities carried out by programmers [Ripley 2007].

speaking) when an element of Palantír is selected, it is possible to obtain information about the magnitude of the changes carried out, as well as the names of the corresponding programmers and the value of metrics.

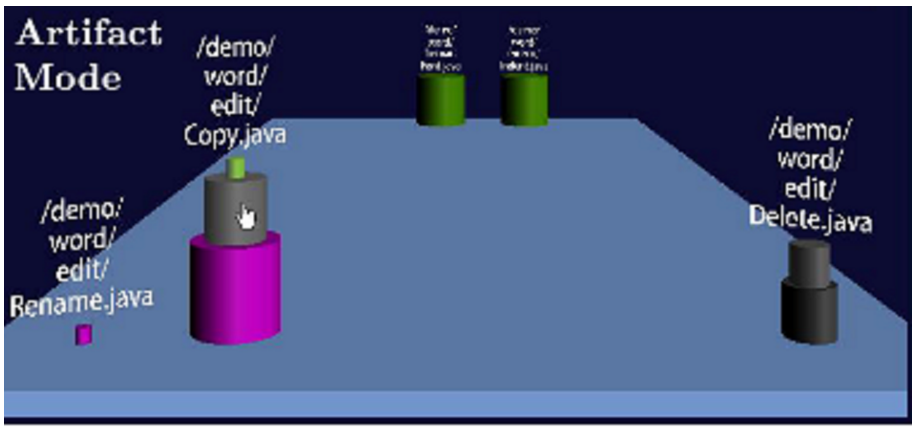


Figure 6.7: Representation of software items and changes that were made by each developer [Ripley 2007].

6.4.3 Collaboration and Socio-technical Relationships

In most of the cases, the objective of the tools that support the construction of common knowledge spaces is to improve cooperation among team members. This cooperation is usually necessary because collaborators work together on a common project and the work of each one is related to the work of other contributors. However, it is advisable to consider that cooperation among individuals can also be present when several people contribute to the solution of determinate problems, working of independent form and without having as

aim the achievement of common objectives or goals. A remarkable example are the web sites used by programmers to ask for cooperation from other programmers in solving particular problems [Assogba 2010]. Programmers who participate in these type of sites share membership in a community of individuals working on subjects over which they have knowledge, but which are not necessarily part of their daily work.

Assogba and Donah [Assogba 2010] denominated this type of cooperation as *loosely bound cooperation* and they defined it as "a form of cooperation, sometimes indirect, between members of a community that gives them the freedom to pursue their individual goals while allowing them to help each other". As a summary of the foregoing and the features that the researchers stand out, the following are the main points of this type of cooperation:

* Individuals are not under any obligation to help others.
* Each participant has their own goals and most of them do not share goals.
* The cooperation can range from casual to continuous and committed, and involves the solution or development of a particular software item.
* The participants in this model of cooperation are part of a community of individuals who actively practice their profession.

Assogba and Donah carried out the development of a visualization tool which they denominated Share [Assogba 2010]. The aim of this tool was to support the sharing of source code among members of a community of programmers. In order to implement this tool, they used a client/server architecture, which provides server-side authentication and data storage, while the client side is a desktop application where the user carries out programming tasks. Each programmer who uses this tool is assigned the same color in all the projects in which is involved.

The client side of Share provides a file browser, a program editor, a reference manager, a search engine, the visualization of the network of relationships (relationships browser) and mechanisms for synchronization with the server. The program editor and the visualization of the network of relationships are of interest to this research and are thus further explained below.

The program editor provided by Share uses the color assigned to each developer to indicate who is the author of each piece of reused code that is part of a program, but does not use any color for the new code that has been developed within the program (see Figure 6.8).

The browser of relations of Share is aimed to facilitate the tracing of reused source code, for which it uses a graph that depicts the correlation

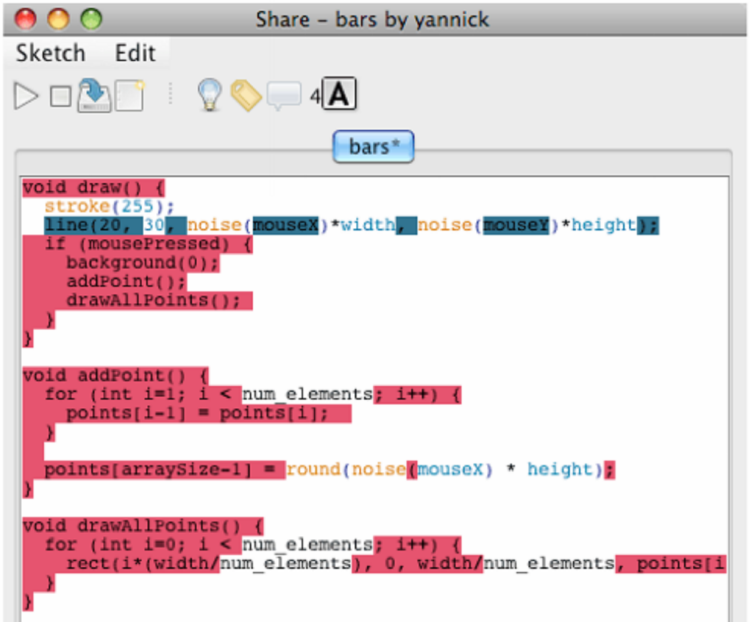


Figure 6.8: Share: Text Editor showing the pieces of source code that have been reused and which represent by means of colors, the person who has made the original contribution [Assogba 2010].

user – user, user - software item and software item – software item. This visual representation has two variants which allow to easily determine who has contributed source code to a project and those who have reused that code.

The first variant of the browser of relations utilizes a radial layout whose central item is the one that have been selected by the user. This visualization provides an overview of all contributions that has been made. Figure 6.9 depicts the contributions made by each developer and the reuse of source code between software items by means of arrows, in which an arrow indicates the item from which the source code has been reused. The second variant of this browser is sought to provide information about which software items are lending or borrowing a given software item, as illustrated by Figure 6.10.

The visualization of collaboration between programmers allows a programmer to learn about those from whom they can expect collaboration based on the items that have changed and the relationships that exist between those software items. This type of visualization provides information to project managers and assists them in making decisions about which programmer can replace another programmer in case of sickness, accident, resignation or dismissal. Moreover, it can also help in forming teams according to past and current collaborative relationships between programmers [Jermakovics 2011].

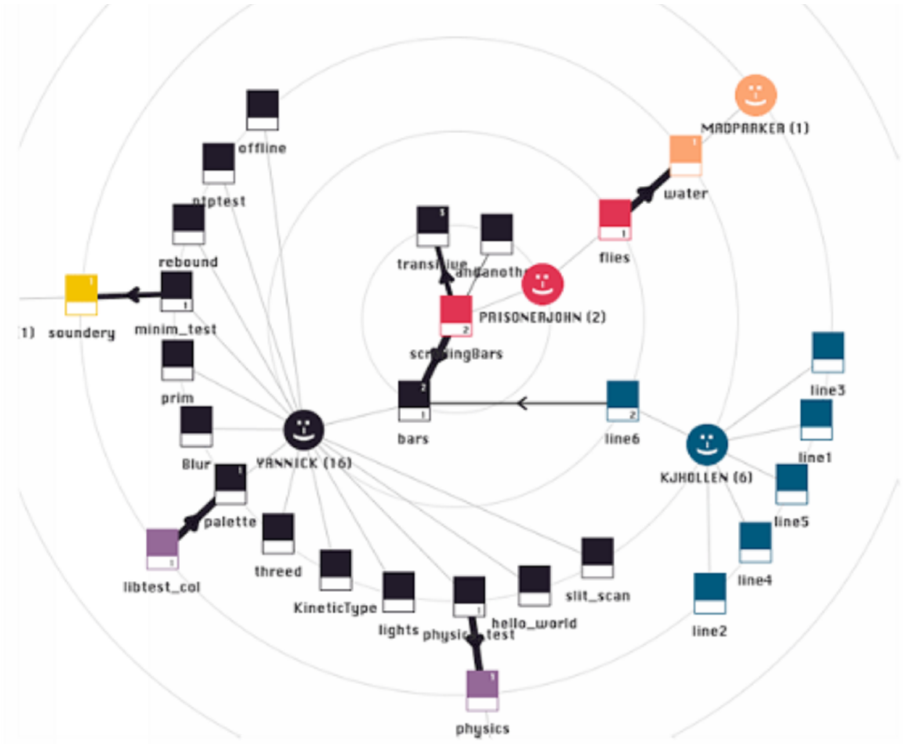


Figure 6.9: Share: Browser of relationships using a radial layout to show the relationships between software items according to the reuse of source code [Assogba 2010].

Jermakovics *et al.* [Jermakovics 2011] construct a network of the collaboration that takes place among developers on the basis of the changes that have been made to software items. This collaborative network emerges as a product of the analysis of the similarities among developers based on the software elements that they have changed in common. The network that results from the analysis of changes is represented using a force directed graph, where the nodes represent the programmers and the edges reflect the relationships between them, on the basis of similarity. The size of the nodes is used to represent the number of *commits* made by programmers, while the forces of the graph are calculated in accordance with the similarity among programmers and the number of connections between them.

The similarity measure is also used as a filtering criterion, which allows the user to choose a threshold to filter edges that do not meet the criteria selected. Another interesting aspect of this visualization is that in addition to providing information about the relationship between programmers, it also provides details about the membership of a working group of programmers and also the relationship between these working groups, as illustrated in Figure 6.11.

A complementary approach to the above is that of Heller *et al.* [Heller 2011]

Figure 6.10: Share: Basic browser to show the relationships of a particular software item [Assogba 2010].

concerning the representation of the collaboration among programmers, but taking into account their geographical location. This visualization represents data that is obtained from *GitHub* by means of a graph which is drawn over a map to show the relationships among programmers, and this also allows the density of programmers by country or region to be shown.

The development of strategies to increase the level of knowledge about the activities that programmers can contribute to coordination in GSD environments. Taking this into consideration, it should be recalled that SCM tools have been widely disseminated and used to assist in the coordination of parallel software development and that in distributed environments they have been of great utility. In addition, due to the richness and the large quantity of information that they manage, a large number of visualization tools make use of this information. However, the disadvantage of SCM tools is that developers do not realize of the changes made by other programmers until they have been sent to the software repository by a *check-in* operation, and not at the time at which the changes are made [Lanza 2010].

Considering this problem, Lanza *et al.* propose an architecture that uses an Eclipse plugin to record and to transmit the source code changes made by a programmer to the other programmers. The aim of such architecture is to support the understanding of changes and to provide information for programmers to react in time to changes that are made to the system.

Additionally they developed a tool, as well as an Eclipse plugin, which

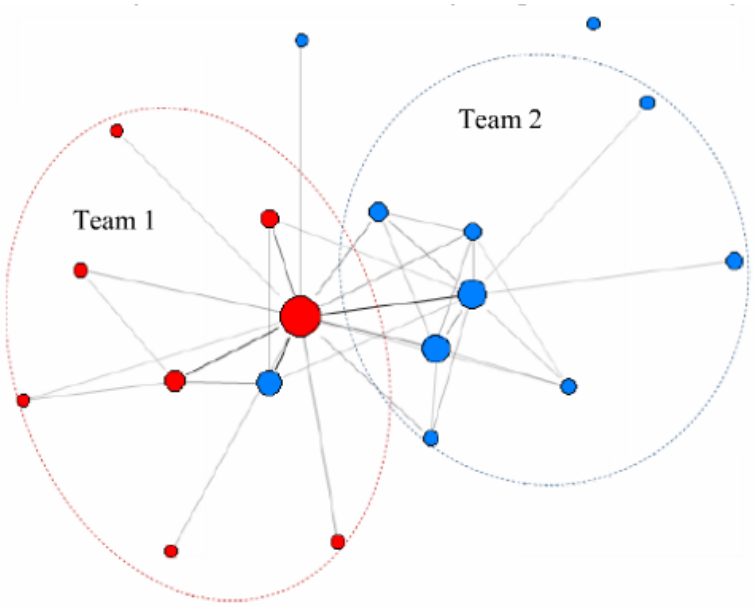


Figure 6.11: Visualization of a collaborative network between programmers based on the software items that have been changed in common [Jermakovics 2011].

consists of three simple visualizations that are updated in real time on each IDE as changes are made [Lanza 2010].

Bucket View is a visualization that is part of this tool and uses a metaphor of buckets: each software item is represented by a bucket that contains small colored squared shapes. Every change that is made to the system is represented by a colored square shape, where the color denotes the programmer that carried out the change. Changes are sorted chronologically, so the oldest updates are located at the bottom of the bucket and the most recent ones are located on top.

This visualization uses color to identify the programmer that is considered as the owner of a piece of software. Figure 6.12 shows that the software item associated to bucket D has been changed by a single developer (associated to the red color) which, therefore, is the owner of that item. While the item associated to bucket A has been changed concurrently by two programmers, where the programmer that is associated to the blue color is the owner of such software item.

6.5 Discussion and Conclusions

Software development under GSD models requires the use of tools to support teamwork and collaboration. Therefore, the role of software visualization was discussed in this chapter as a central element of such tools, and in helping

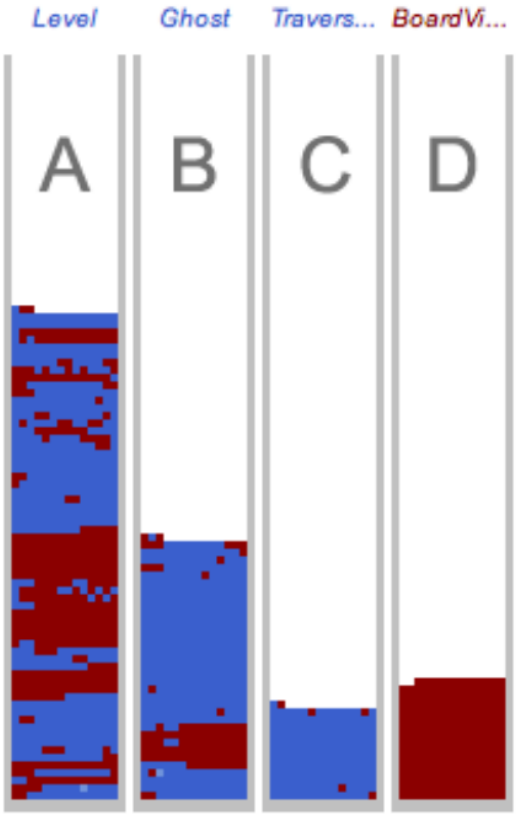


Figure 6.12: Buckets View: Visualization of changes made to software items and the collaboration between programmers [Lanza 2010].

transform the huge amounts of information that are derived from the SDME processes into knowledge. The importance of the previous discussion is rooted in that the correct design of the architecture of these kind of tools should pay special consideration to the geographical, temporal and cultural distances involved. Thus, in order to facilitate collaboration among team members, the design of these tools requires the use of mechanisms that allow reviewing in real time the changes made to the system as well as details of who has carried them out.

In this context, the vocabulary and terms that are used in the SDME processes are commonly used globally (in different geographical areas and countries), which facilitates the communication and collaboration. However, the design of visualizations requires a careful selection of symbols, representations and conventions.

The research discussed in section 6.4 looks for facilitating the obtention of knowledge on the state of things during SDME processes, and in general their goal is to support the collaboration among team members. In summary, the research works that were studied focus on:

1. Sharing information about the architecture of systems and the metrics associated with the software items which make it up.
2. Supporting the interaction and collaboration among team members.
3. Providing details on the patterns and collaboration networks that are formed among programmers and which are derived from the software items that have changed in common.
4. Supporting the identification of the relationships that are formed between teams based on their interaction and collaboration.
5. Providing details on the collaboration of programmers in virtual communities of voluntary cooperation.
6. Providing information on the ownership of software items on the basis of the changes that have been made and the programmers who made them.

At this point it is important to analyze the relationship between chapter 5 and this chapter. Chapter 5 focuses on the visualization of the architecture of software systems with the aim of facilitating the understanding of the system structure, the metrics associated to its software items and the changes that are carried out. But it is desirable to take into consideration that this type of visualizations also serves to support collaboration among team members and to provide details on the state of things. So, the design of tools to facilitate collaboration should not only consider the elements where the interaction among team members is reflected, as most of the research works described in this chapter do, but all those factors that provide insight of the activities that occur around the system: through that knowledge is that different actors can take action and initiate communication, coordination and control processes.

Finally, it should be highlighted that all the factors that could facilitate the performance of teams though the appropriate support to individuals should be considered. This is in line with the discussion presented in Section 6.2 and it is aimed to improve the performance of teams in working environments that operate under GSD models.

Chapter 7

Survey on the Use of Visual Tools in Software Development and Maintenance

Después de haber saciado el hambre, Rastin emprendió la vuelta a casa, mientras Güindy y Cucho continuaron su camino. Esta vez caminaron juntos, Güindy no seguía a Cucho, y después de varias horas vislumbraron a lo lejos una luz intensa. Apresuraron el paso, y al acercarse se dieron cuenta que habían llegado a una pequeña ciudad con fábricas y chimeneas. Entraron a la ciudad de forma tímida y silenciosa, y comprobaron que aunque la ciudad tenía visos de modernidad, en realidad seguía siendo rústica para los tiempos modernos que corrían. — El viaje de Güindy, A.González

Contents

7.1 Introduction

In the last chapter, the results of a study were presented. The aim of the study was to conduct an in-depth review of the current state of the application of

visualization and VA to software systems and their evolution in facilitating the development and maintenance of software. It thus examined the use of IV and VA in the comprehension processes of software projects and their evolution by means of a systematic mapping study of research carried out in the last 7 years, from 2007 to 2013. Consequently, it identified the tasks that this research sought to support as well as the different types of visualization, data types and technologies that were used.

The research which has been carried out is limited to assessing the works that have been published in the aforementioned time period and the results do not provide insights regarding technology transfer between the research community and industry. There has been concern about the spread, impact and transfer of scientific results to industry. Consequently, this chapter introduces and discusses the results of a survey that was answered by 113 participants working for 65 companies and who were living in 6 different countries.

It would seem natural that visualization tools should have been adopted by the software industry to support the development and maintenance process taking into account the benefits and the increased use of visual tools in other industries for knowledge discovery (see more details in chapter 3). However, the research conducted in this chapter supports the opposing hypothesis:

How are software companies and software development departments using visual tools to facilitate software development and maintenance?

Consequently, this chapter seeks to answer the above question by means of a survey of the use of tools which support the software development process. The outcome of this study provided important details on the availability of data and blockage points that may be helpful for the design, implementation and adoption of tools.

7.2 Survey Description

The survey was prepared using a powerful commercial web application specialized in online surveys [Qualtrics, Inc. 2013] and distributed by email. It was sent to the email list of the Computer Science graduates of a large university (35,000 students) who are currently working in the software industry. In addition, the survey was also sent to the email lists of professional groups in the field of software development and maintenance.

This survey was aimed at programmers, team leaders, project managers, architects, analysts, and SQA professionals. The survey questions were branched according to the job position occupied by the person who was

answering. Additionally, the questions were divided into four groups, as the following list shows:

Data collection: These questions were aimed at surveying the use of tools for collecting and storing data generated during software development and maintenance. The goal of these questions was to determine the availability of data that could be used by visualization and visual analytic tools aimed at analyzing software systems.

Product tools: These questions were targeted at obtaining perspectives on the use of visualization and visual analytic tools aimed at analyzing software systems in *product*-related tasks, (*e.g.*, debugging, understanding the code structure (dependencies, inheritance, coupling, cohesion)), and understanding code changes (refactoring).

Process tools: These questions were directed towards the use of visualization and visual analytic tools aimed at analyzing software systems in *process*-related tasks, *e.g.* project management and software quality assurance (analysis and monitoring of team activity and quality metrics).

Blocking points for adopting tools: These questions were aimed at identifying obstacles for adopting visualization and visual analytic tools aimed at analyzing software systems for supporting software development and maintenance tasks.

Table 7.1: Number of answers per role type.

Position	Answers
Programmer	41
Team leader	18
Project manager	8
SQA Specialist	3
Architect	2
Analyst	1
Totals	69

7.3 Questions and Results

The survey was completed by 113 participants. The answers from participants which did not match the roles in Table 7.1, contradictory answers, and answers which came from respondents whose job includes systems support (servers and

network) and carry out help desk and development tasks simultaneously[1] (as they do not work full time in software development and the survey seeks for answers of full time software practitioners), were discarded.

After the filtering process, 69 answers remained. Table 7.1 shows the professional roles of the survey participants considered. The participants came from over 65 companies in 5 Spanish-speaking countries and one Portuguese-speaking country, and were distributed into 11 market segments, with the majority of companies coming from the software industry[2] as shown in Table 7.2. The respondents worked for companies whose headquarters were based in 8 different countries. Of the 65 companies, 24 companies were multinational and, among these, 19 were based in the United States, 2 in Mexico, one in Germany, one in Nicaragua and one in Puerto Rico.

Table 7.2: Number of answers per company type.

Market segment	Companies	Answers
Software industry	43	46
Finance and banking	7	8
Government	3	3
Services	3	3
Telecommunications	2	2
Education and research	2	2
Energy	1	1
Agriculture	1	1
Manufacturing	1	1
Healthcare	1	1
Transportation	1	1
Total	65	69

The survey results are presented in the following sections, classified by means of the groups mentioned in earlier sections of this work.

7.3.1 Data Collection

The first group of questions (as shown in table 7.3) was aimed at assessing the availability of software-related *Big data*, and data-collection tools, within the participants' companies. Since data collection is the first step in the analytic process, it may represent the first bottleneck on the road to implementing

[1]The answers revealed that most of these respondents work for small companies with 1 to 3 professionals in the IT department.

[2]Companies which main business is the development and commercialization of software products.

Table 7.3: Question group: Data collection.

	Question	Answer
Q1	Does your company use a SCM tool? If Yes, specify which SCM tools.	Yes/No Plain text
Q2	Does your company use a bug tracking tool? If Yes, specify which bug tracking tools.	Yes/No Plain text
Q3	Are the SCM and bug-tracking tools linked to connect bugs with changes?	Yes/No
Q4	Do you collect data for metrics calculation? If so, how do you collect metrics data?	Yes/No Plain text

IV and VA for the process of comprehension of software projects and their evolution.

Question Q1: SCM tools are key componentes for data collection as they record and manage source code versions and the metadata associated with changes in software repositories. Figure 7.1 shows the answers to $Q1$. The 65 companies surveyed (100%) use at least one SCM tool; 24 companies (37%) use at least two SCM tools; and 8 companies (12%) use up to 3 SCM tools. Hence, the raw data required by the visual representations should be readily available in all the cases surveyed.

Questions Q2 and Q3: Most bug tracking tools permit the creation of relationships between bugs and changes recorded by SCM tools, integrated as part of the tool or as a plugin. Such relationships are essential for corrective maintenance [D'Ambros 2006b, D'Ambros 2007a, Sensalire 2008].

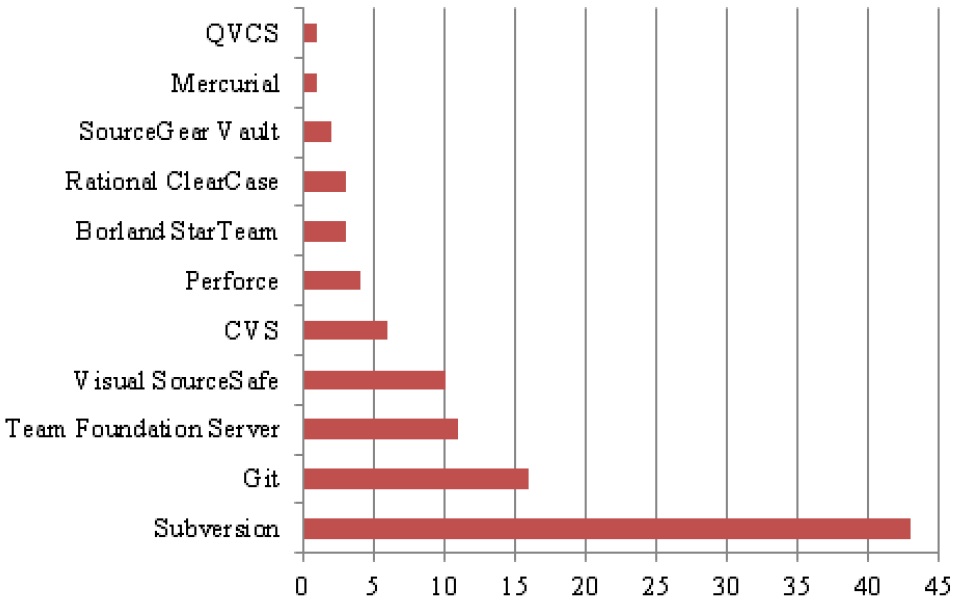


Figure 7.1: $Q1$: Use of SCM tools.

The answers to $Q2$, presented in Figure 7.2, show that 16 companies (24.6%) do not use any bug tracking tool; 51 companies (78%) use one bug tracking tool; and 13 companies (20%) use more than one tool. However, the answers to $Q3$, shown in Figure 7.3 are not as encouraging as the ones in $Q2$ as only 16 companies (24%) have linked the bug tracking to the SCM tools. This may limit the amount of insight that visualization tools can provide because collected data does not reflect all the activity carried out during the development and maintenance process.

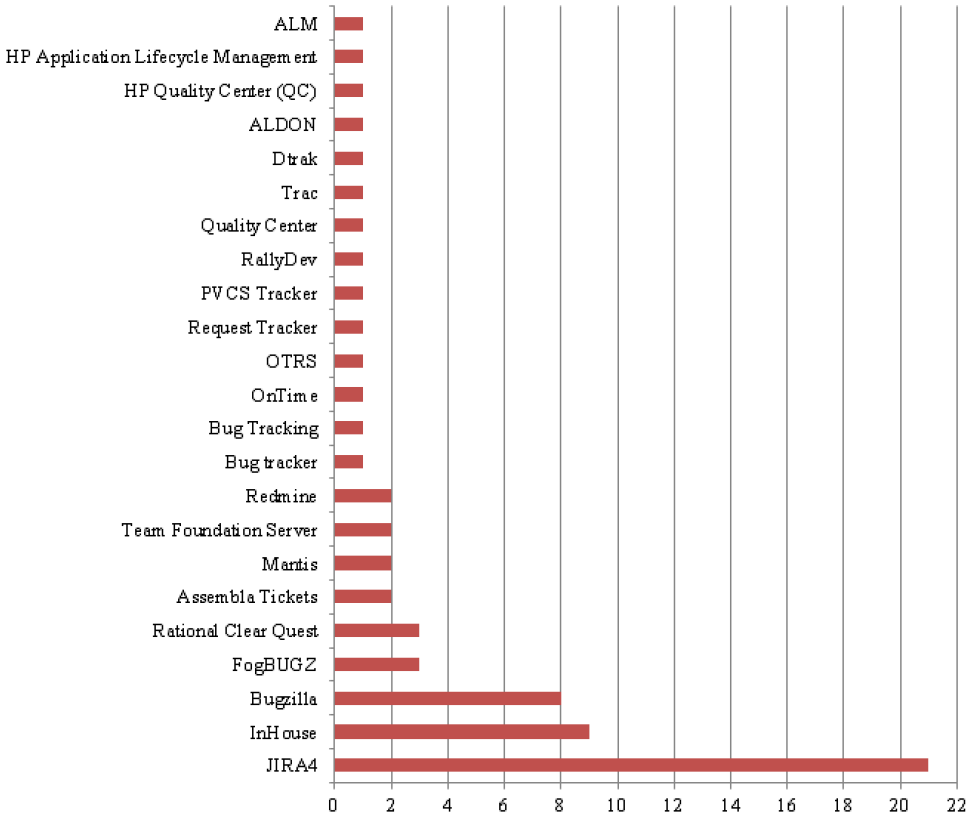


Figure 7.2: $Q2$: Use of bug tracking tools.

Question Q4: This question was aimed at project managers, team leaders and SQA specialists, as these are the typical stakeholders interested in quality metric analysis during software development [Pinzger 2005, Lanza 2005b, Telea 2009c]. These professional roles accounted for 29 respondents from 29 companies. The answers were divided up as follows (see Figure 7.4): 17 respondents (59%) collected data for calculating metrics; but the other 12 respondents (41%) did not. Of the 17 positive responses, 10 users (59%) collected the metric data manually; 4 users (23%) used custom metric tools (developed internally); and 3 users (18%) collected metric data using commercial metric tools.

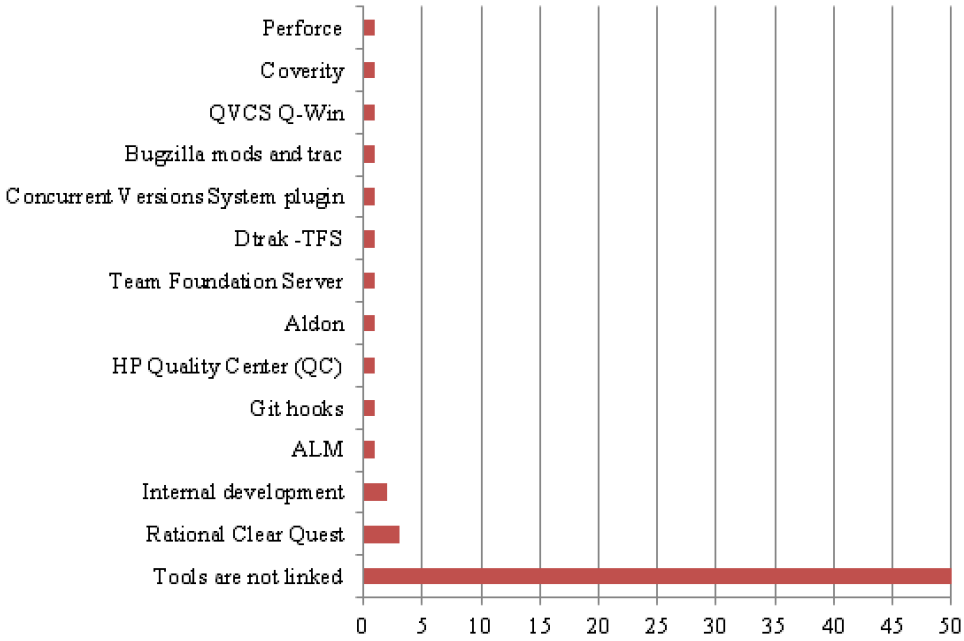


Figure 7.3: $Q3$: Correlation of SCM and bug-tracking tools to make relationships between bugs and changes.

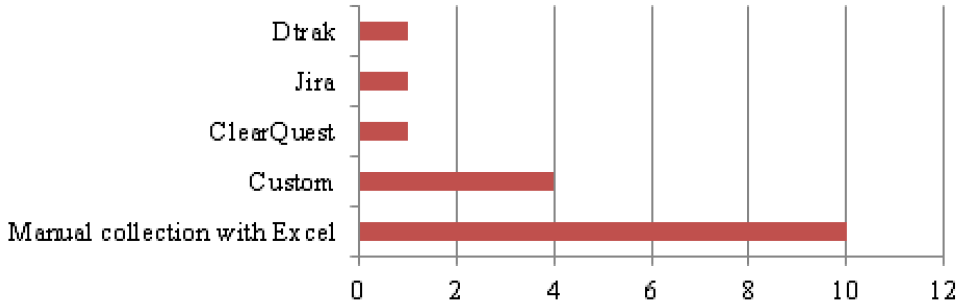


Figure 7.4: $Q4$: Tools for metrics data collection.

7.3.2 Product Tools

The questions posed in this group focus on the use of visualization tools for product-related tasks such as debugging, understanding the structure and dependencies of a software project, and assessing cohesion and change during development, and perfective and adaptive maintenance. The selected questions are not exhaustive with regard to all tasks related to program comprehension, but carry out and adequate sampling of most frequently encountered activities of this type [Storey 1998, Maletic 2002, Sensalire 2008]. As such, their answers are a good indicator of the penetration of visualization tools aimed at general-purpose program comprehension tasks. The questions in this group are listed in table 7.4.

Question 5: Debugging is estimated to cover about 25% of software maintenance costs [Storey 1998, Koschke 2003]. As such, it is worthwhile evaluating whether visualization can effectively support debugging activities. The answers to $Q5$ displayed in Figure 7.5 show that the debugging tools included in IDEs are the favored option (59 answers, 91%). Visualization tools, either third-party (4 answers, 6%) or in-house developed (2 answers, 3%) are a minority option.

Table 7.4: Question group: Product tools.

	Question	Answer
Q5	Which of the following tools do you use for debugging?	IDE debugging functions / Visualization tool or plugin (3^{rd} party) / In-house development) / Other
Q6	How do you navigate class hierarchies? (*e.g.*, find ancestors or descendants of a given class)	Built-in IDE visualizations / Visualization tool or plugin (3^{rd} party)/ In-house development / Manual text search / Manual review of the class diagram / Other
Q7	How do you navigate dependencies?(*e.g.*, find callers or callees of a given function)	Built-in IDE visualizations / Visualization tool or plugin (3^{rd} party)/ In-house development / No specific tool is used / Other
Q8	How do you find and examine code clones?	Built-in IDE visualizations / Visualization tool or plugin (3^{rd} party) / IDE search functions / In-house development / Manual search
Q9	Upon code refactoring, how do you how do you find where old code has been moved?	Built-in IDE visualizations / Visualization tool or plugin (3^{rd} party) / IDE search functions / SCM logs review / Manual search / Other

Questions 6 and 7: These questions relate to the most frequent types of relationships in program understanding – examining class hierarchies ($Q6$) and review dependencies ($Q7$). For $Q6$ (see Figure 7.6), 43 respondents (66%) answered that they use the basic visualizations included into IDE tools, 25 companies (38.5%) rely on manual searches and a single respondent (1.5%) used a specialized visualization tool. For $Q7$ (see Figure 7.7), the answers are similar: 49 companies (75.5%) used manual search; 14 companies (21.5%) use built-in IDE visual functions; and only two companies (3%) used a specialized visualization tool.

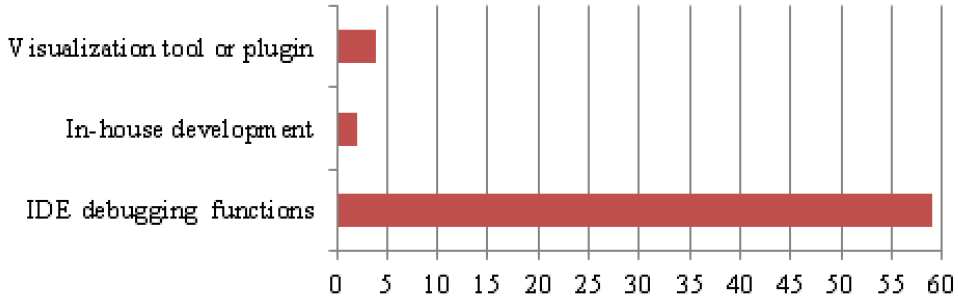


Figure 7.5: $Q5$: Use of visualization tools for software debugging.

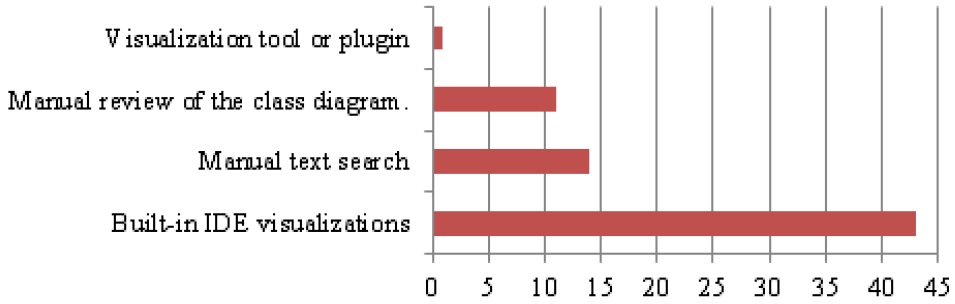


Figure 7.6: $Q6$: Use of visualization tools to navigate class hierarchies.

Question 8: Another important topic that respondents were asked about was the detection of source code clones (see Figure 7.8). The results showed that 27 companies (41.5%) carry out a manual search (some indicated that they manually maintain records of the location of clones), 23 companies (35.5%) use the IDE capabilities for searching clones, 14 companies (21.5%) use the basic visualizations provided by recent versions of IDEs and only one answered (1.5%) that they used a specialized visualization tool.

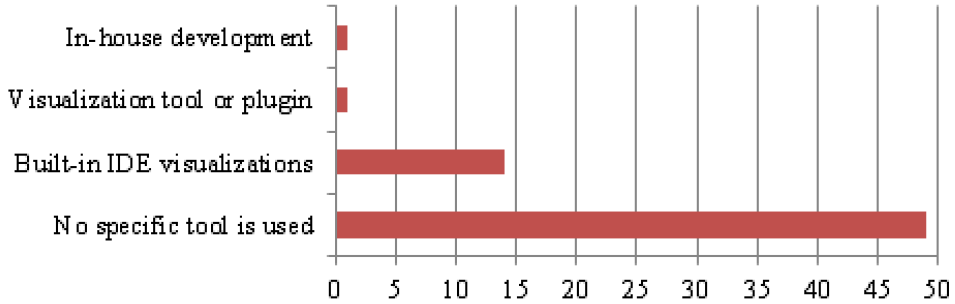


Figure 7.7: $Q7$: Use of visualization tools to navigate dependencies.

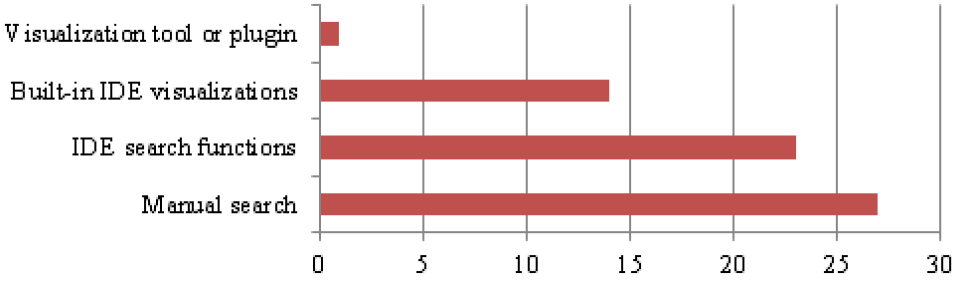


Figure 7.8: $Q8$: Use of visualization tools to find and analyze code clones.

Question 9: This question captures a typical task in software understanding during perfective or adaptive maintenance: the relocation of software items. When a software project undergoes refactoring, various software items change location. An essential task is to find the new location of such software items. The answers to $Q9$ (Figure 7.9) show that 38 companies (58.5%) use the IDE capabilities to search for the new location of software items; 14 companies (21.5%) find such locations manually; 8 companies (12.5%) use the log of SCM tools for this; 3 companies (4.5%) use other tools; and only 2 companies (3%) use a visual tool for this task.

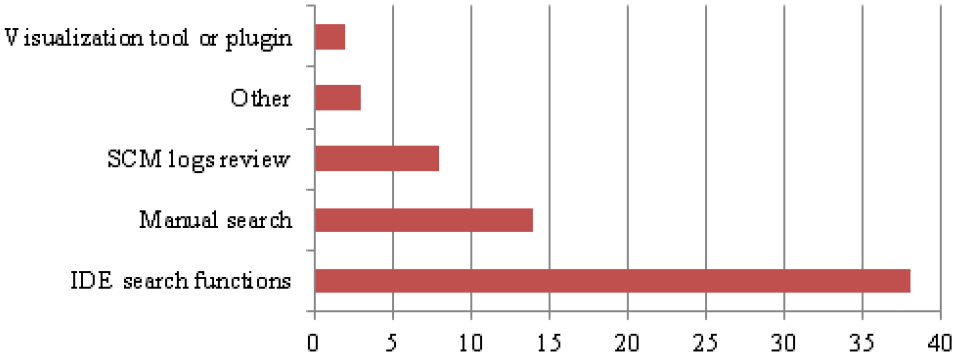


Figure 7.9: $Q9$: Use of visualization tools to find source code fragments after refactoring.

7.3.3 Process Tools

This question group covers the use of visualization tools for understanding metrics related to project and process quality ($Q10$ and $Q12$) and collaboration $Q11$. As such, $Q10$ and $Q12$ are aimed at project managers, team leaders, and the SQA team (29 respondents from the same number of companies), and $Q11$ is aimed at all respondents. The questions in this group are shown in table 7.6.

Table 7.5: Question group: Process tools.

	Question	Answer
Q10	Do you use a tool to measure and visualize individual programmer contributions?	Yes/No
Q11	Do you use a visualization tool to track which users changed which software items?	SCM visualization tools / Visualization tool or plugin (3^{rd} party) / In-house development / No specific tool / Other
Q12	Do you use a tool to visualize metrics? If Yes, specify which tool.	Yes/No Plain text

Question 10: Only 4 out of 29 respondents (14%) responded to this question. Those who answered use a tool for monitoring programmer contributions, and none of them used a visualization tool (see Figure 7.10). This may indicate that in practice the use of metrics is limited. However an analysis of this falls out the scope of the present work.

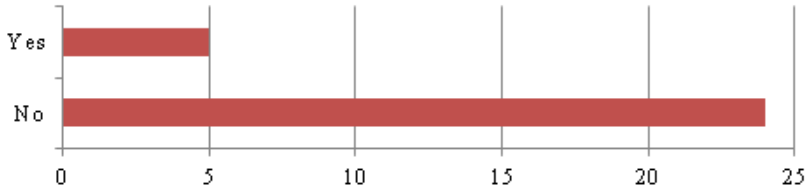


Figure 7.10: $Q10$: Use of a tool to measure and visualize individual programmer contributions.

Question 11: This question was aimed at all participants. Therefore, 29 of the 65 companies (44.5%) used the basic version-tree visualization of SCM tools (see Figure 7.11); a single user (1.5%) used a visualization tool developed internally whereas the other 35 companies (54%) do not use any specific tool for this task.

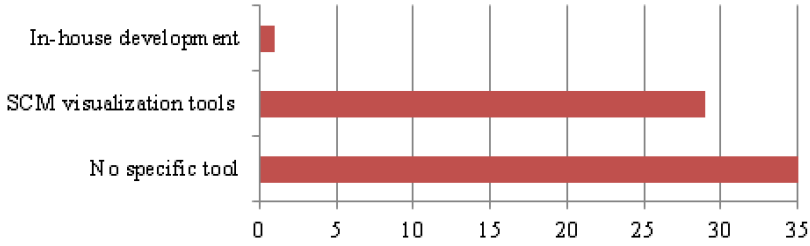


Figure 7.11: $Q11$: Use of visualization tools to show which developers change which software items.

Question 12: Although 59% of the respondents (see section 7.3.1) collect software metrics data, only 2 out of 29 companies (7%) use a visual tool for software metrics (specifically, Excel); and also only 2 companies use the same visual tool for tracking the evolution of such metrics through different revisions or product releases (see Figure 7.12).

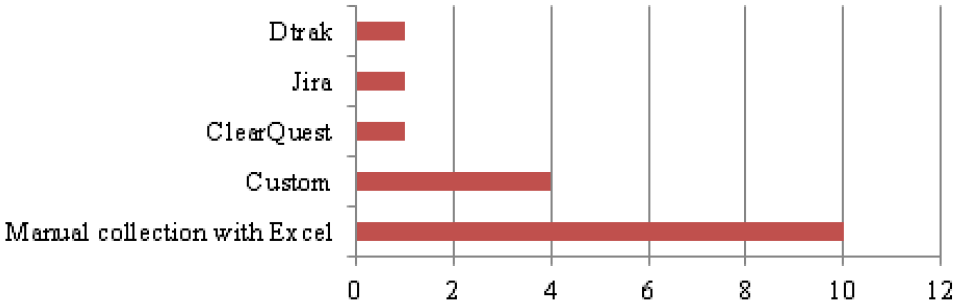


Figure 7.12: *Q*12: Use of visualization tool for metrics.

7.3.4 Impediments to Adopting Tools

The questions in this group are focused on determining the impediments to the adoption of visualization tools that support software development and maintenance tasks. The questions in this group are listed in table 7.6.

Table 7.6: Question group: Blocking factors.

	Question	Answer
Q13	Do you think that visualization tools help to reduce software development and maintenance time?	Yes/No
Q14	What do you think is the main reason for not using visualization tools during software development?	Don't know such tools / No suitable tools found / Tools found not suitable
Q15	Which are your perceived adoption blockers for visualization tools?	(see option list in Figure 7.15)
Q16	Do you consider that software engineering courses should include topics on the use of existing visualization tools?	Yes/No

Question 13: This question was aimed at assessing the perception about the experienced effectiveness of visualization tools in general, and not only for software development and maintenance. As such, *only* those users who already had used a visualization tool (44 out of 69) in any task were asked this question. The majority (37 respondents, 84%) answered that they

consider that tools like the ones they have used could help to reduce software development time (see Figure 7.13). Negative answers were given by only 7 respondents (16.3%).

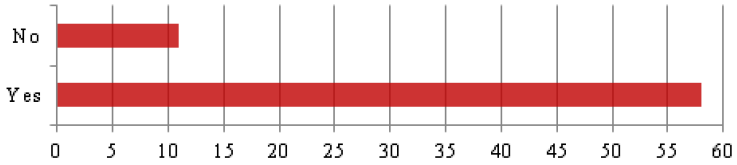


Figure 7.13: $Q13$: Use of visualization tools to help reduce software development and maintenance time.

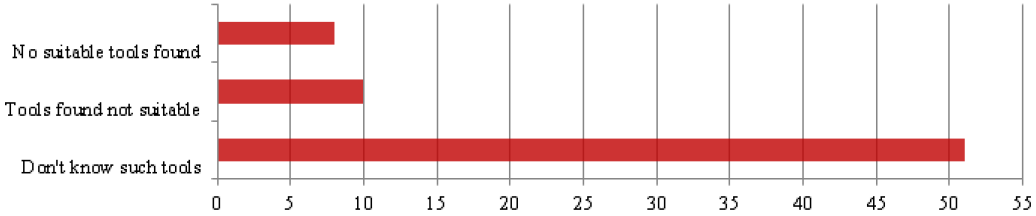


Figure 7.14: $Q14$: Reason for not using visualization tools during software development.

Question 14: In contrast to $Q13$, this question was asked to *all* users (69 respondents), as shown by Figure 7.14. Interestingly, 73.9% of the responses indicated that the subjects did not have information regarding the existence of visualization tools. 14.5% of respondents indicated that they had tried out visualization tools but decided not to use them as they did not fit in the required tasks (table 7.6, *Tools found not suitable*). The remainder of 11.6% respondents indicated that they had searched for suitable visualization tools for their tasks, but these tools did not meet their expectations (table 7.6, *Tools found not suitable*).

Question 15: This question was posed to those who answered $Q14$ and enquired about the perceived blockage points for the adoption of visualization tools. The options offered to respondents are taken in line with [Bresciani 2009]. Figure 7.15 shows the answers given to $Q15$.

Question 16: Finally, $Q16$ was posed to all respondents with regard to teaching and learning about the existence and use of visualization tools in software engineering courses and 71% answered this question positively, as illustrated by Figure 7.16.

7.4 Discussion

The analysis of this survey has produced a wealth of information that cannot be easily summarized in a few pages. The survey results provided details

about the availability of data from SCM and bug tracking tools in most of the companies surveyed. The availability of data may thus facilitate the design, development and implementation of visualization and visual analytic tools to support software development and maintenance.

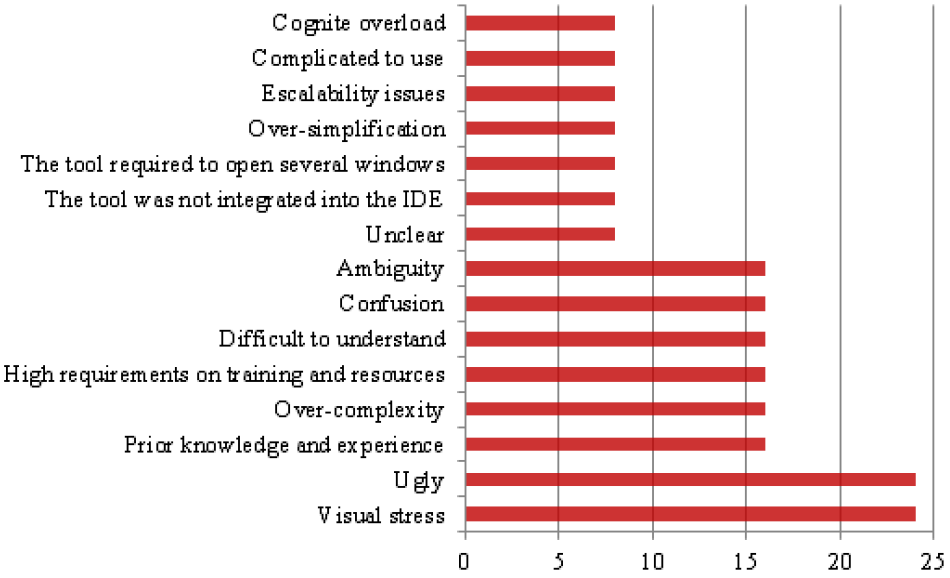


Figure 7.15: $Q15$: Perceived adoption blockers for visualization tools.

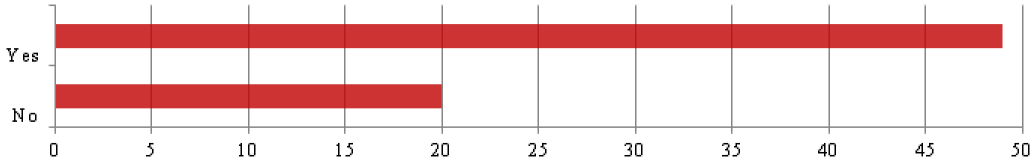


Figure 7.16: $Q16$: Do you consider that software engineering courses should include topics on the use of visualization tools?

However, the survey results reveal that few companies have linked the tools mentioned (16 out of 65) in order to obtain details of the correlation between bugs and changes, limiting the utility of the data generated by the same process of development and maintenance. Additionally, the collection of metrics by means of specialized tools is only carried out by 7 companies while 10 companies collect metrics manually using Excel. The consequence of this is that quality control of software products in most companies is not carried out in a systematic manner during the software development process, as much as during development and maintenance.

Moreover, the survey revealed that tasks such as debugging, navigation of dependencies, detection of source code clones, refactoring, tracking changes and contributions, and SQA metrics monitoring are carried out without the

support of visualization tools. 66% of the respondents answered that they use the basic visualizations included in IDE tools to examine class hierarchies and navigating dependencies and 44.5% make use of the basic version-tree visualization included in SCM tools.

This demonstrates that the majority of users use the tools that are integrated in their working environment and are directly accessible by means of SCM tools and IDEs. This is congruent with the arguments made by Lintern *et al.* [Lintern 2003] regarding the design of visualization tools to support the process of software development and maintenance. It is argued that this is carried out without regard to the user environment in which tools are used and without seeking their integration with existing tools.

This in turn leads us to consider that the integration of visualization tools and VA to support the process of SE is a key aspect and that visual tools must be integrated in the tools that developers use in their development environments, such as IDEs and SCM, in the form of plugins. Therefore, there exists a potential market for developing visualization tools that are integrated into well known existing tools such as IDEs and SCM tools without burdening users with unnecessary additional tools.

It is important to recall that the perception of respondents who had used a visualization tool, in general context (and not only in a software systems context) is positively high. However, when they were asked about the factors which may block the adoption of these type of tools indicated the following:

* Visual stress factors caused by the visualizations.
* Inadequate design.
* The complexity of the visual representations.
* The time required to learn the tools.
* Requirement of previous knowledge.
* Aspects related to the lack of clarity and ambiguity of designs.

It is thus important to consider the requirements of users with regard to the tasks which they seek to support and thus make them active elements of the design process by means of brainstorming meetings and usability studies to evaluate and improve the design of visualizations and of the tools in general.

Other points which are relevant in considering the factors which impede the adoption of visualization tools are related to the awareness of tools available to support the process of software development and maintenance. 75% of the respondents (69 in total) indicated that they did not have information available concerning the existence of visualization tools to support tasks of this process. When respondents were asked if they considered it appropriate to include the teaching of these type of tools in software engineering courses,71% answered positively.

Furthermore, some comments made by respondents highlighted factors such as, the absence of a culture of using visualization tools in software development and maintenance, the cost of tools, difficulties in calculating the return of the investment, information cluttering and the scalability of tools.

As a result, software companies and software development departments make limited use of visualization tools to facilitate processes as they only use the basic visualizations included in the IDEs and SCM tools. It is thus clear that more complex visualization tools are rarely used by software companies and software development departments. Finally, clues offered by the survey results indicate the integration of more complex visualization tools into well known and accepted tools currently used in software development and maintenance.

7.5 Conclusions

The survey results provide important details concerning the availability of data, the use of visualization tools in product and process related tasks, and the identification of impediments which could be used positively for the design, implementation, and adoption of tools. These results show the need to integrate bug tracking and SCM tools and other tools to capture and store information from the process of software development and maintenance. This should be done in order to obtain better data which permit better tracking of the progress of the project in order to allow it to be supported more effectively.

The results obtained also show that the visualizations which are being used by programmers are those which are integrated in the tools they use in their daily tasks, such as SCM tools and IDEs. It is thus desirable that new versions of these tools allow for the use of visualizations of greater complexity and utility that the ones that currently these tools include; but also that the more specialized visualization tools which are developed are integrated in the tools mentioned in the form of plugins. With regard to this last proposition, it is noteworthy that a large number of recent studies are considering the use of plugins in their proposals, according to the results of chapter 4.

Two points that deserve special consideration are related to the improvement of awareness of the existence of these tools and the promotion of their development. It would thus be appropriate to include their use in software engineering courses in order to raise awareness of their benefits. It may also be advisable to incorporate courses or seminars in graduate programs that address the design and development of tools applied to SV systems, taking into account not only the technical aspects but also aesthetic aspects, ease to learn, interaction and integration in programmers working environments.

Accordingly, an important step in facilitating the processes of learning and teaching, the design and development of these tools is the clear definition of a process which describes in detail the major components, methods and techniques involved in the process of transforming system data into useful knowledge that will facilitate a better understanding of the dynamics of systems development, their maintenance and evolution.

Part IV

Process Design and Validation

Chapter 8

A Visual Analytics Process for Software Evolution

Cuando algunos habitantes notaron la presencia de Güindy y Cucho, se mostraron amables, algunos incluso ofrecieron alojamiento y comida. Al sentirse tan bien acogidos decidieron quedarse algunos días. Pero no transcurrieron muchas horas para que un lugareño ofreciera trabajo a Güindy, el cual aceptó sin hacer muchas preguntas. El primer día de labores Güindy fue sorprendido, su principal tarea sería diseñar relojes de arena.
— El viaje de Güindy, A.González

Contents

8.1 Introduction

This research has so far discussed the fact that *Software engineering* is concerned with a set of processes that cover the entire life-cycle of software systems: ranging from requirement analysis and design up to development, testing, release, and maintenance. The results of the previous chapters showed that a large number of research works have been conducted with the aim of supporting software development and maintenance related tasks. It is evident from these results that pertinent data is collected by most companies during SDM processes and that the visualizations which are most frequently

used in these processes are those that are built into the tools that developers use in their daily activities, such as IDEs and SCM tools.

Furthermore, according to the results in chapter 7 and what was previously discussed, the data collected during software development and maintenance share many commonalities with typical *Big Data*:

* Large amounts of *data* with *missing* values (*e.g.*, millions of lines of source code [Baker 1995, Kagdi 2007a, Petre 1998] and thousands of software components [D'Ambros 2008]).
* *Complex* and *hybrid* datasets (*e.g.*, large databases of program metrics, design documents, test results, execution logs, and bug reports [Hassan 2005, Lanza 2005b], attributes of numerical, categorical, and textual types, interconnected by a wide variety of types of relationship such as inheritance, hierarchy, and call, control, and dataflow dependencies).
* *Evolving* datasets with thousands of versions of the software system stored in the software repository [Mens 2008, Mahoney 2009].

Therefore, to provide methods that facilitate the comprehension of software projects it is necessary to carry out a detailed analysis of the data generated during software development and maintenance over a specific period of time and (in exceptional cases) for the entire evolution of the project. This form of analysis is known as SEA, and its principal objectives are to provide information that contributes to the understanding of the SE process, and thus supports the improvement of the development process (including project management). However, as has already been discussed in chapter 2, SEA is sought to reduce the size of the *Big Data* produced SDM processes but it also produces large and complex datasets, due to the number of variables involved in the process of source code change and the complexity of their relationships which makes it difficult for users to carry out an adequate analysis. Hence, although the result provides useful information, it does not provide the information necessary to carry out the tasks of understanding changes and project evolution in a satisfactory fashion and thus provide adequate support for decision-making that will lead to future changes and system improvement.

Given this situation, research efforts have focused on the use of visual representations combined with interaction techniques in order to gain insight into using such large and complex datasets (see chapter 3 for a reference on information visualization). These research efforts have concentrated on SV [Diehl 2007] and SEV [González-Torres 2009, Voinea 2007]; although more recently some research has been carried out into the application of VA to software systems [Telea 2011] with the aim of providing better results.

The goal of this research is to support the process of understanding SE and improve the design and implementation strategies of tools designed to satisfy the analysis needs of both programmers and managers (see chapters 5 and 6 for a focused discussion on visualization in SE).

At this point, it is worth mentioning that VA combines the advantages of machines with human strengths such as analysis, intuition, problem solving and visual perception. Therefore, human beings are at the heart of VA [Dix 2010] and HCI is a key component for supporting knowledge discovery.

The results shown by chapter 4 portrayed that the number of research projects that use VA to support the process of software development and maintenance is quite low. Additionally, the analysis carried out in chapter 4 showed the absence of a detailed definition of the process involved in the application of VA to software systems and their evolution. It is thus appropriate to define this process with the aim of providing guidance to both new researchers and software tool engineers. The definition of this process may also be used in accordance with the recommendations of chapter 7, with regard to the inclusion of instructional content about these tools in software engineering courses.

Consequently, the principal objective of this chapter is to define the process of applying VA to SE and attempts to answer the following research question:

How should the process of applying visual analytics to the evolution of software systems be defined?

Accordingly, the definition of such a process requires, on the one hand a description of the process of applying VA to SE; and on the other hand, the identification of the roles, borders, interactions and relationships between modules, components, methods and techniques involved in this process. Therefore, the arguments presented in chapter 2, the discussion carried out in chapter 3 as well as the results in chapter 4 and 7 will be taken into account.

Furthermore, in order to obtain the results that will meet and answer the research question formulated in line with this objective, it is necessary to design and implement an architecture that demonstrates the usefulness of the application of VA to SE. The definition of the architecture will be carried out taking into consideration the results of chapter 7 that show that visualization tools that are used by those involved in the process of developing and maintaining software are those that are integrated into tools VA and IDEs. The implementation of this architecture will thus be carried out using an Eclipse plugin.

The rest of this chapter proposes the definition of the VA process (see Section 8.2), explains the relationship between VA and SE and defines

the EVSA concept, specifies an architecture for applying VA to SE and finally defines a framework for situational awareness and collaboration on the base of using EVSA for that purpose.

8.2 Visual Analytics Process

The functions and responsibilities of the modules that comprise the VA process are explained in Table 8.1. It takes into account the analytical process proposed by van *Wijk* [van Wijk 2005], the adaptation for VA of the *Visual Information - Seeking Mantra* [Shneiderman 1996] that was introduced by *Keim* [Keim 2006, Keim 2008b] (analyze first, show the important, zoom, filter and analyze further, details on demand), the IV model proposed by *Card* [Card 1999b], the visualization process proposed by *Chi* [Chi 2000] and the seven visualization stages identified by *Fry* for visualizing data [Fry 2008].

Table 8.1: Responsibilities and functions of the modules that make up the Visual Analytics process.

Module	Description
Extraction, Transformation and Load (ETL)	This module has the function of performing the connection to data sources and data retrieval using predefined criteria. Then it cleans and merges data and loads it into a data warehouse.
Advanced Data Analysis (ADA)	The function of this module is to make use of one or more analysis techniques in order to extract *Knowledge Facts* and store them into a database.
Visual Knowledge Explorer (VKE)	This module is made up of three components: the IV, the Views Linker and Facts Analyzer (VLFA), and the Visualization Abstractions and Coordination Support (VACS) components. This module has the responsibility of the visual representation of *Knowledge Facts* and conforms to the fundamentals of CMV [North 2000, Card 1999b], and must provide the visualization, interaction and coordination mechanisms for knowledge discovery.

The VA process has been defined using a modular-based approach, where each module is a collection of components. The use of such approach allows for greater process comprehension, flexibility, ease of change and specialization through the development of specific components that can be tested individually. Consequently, the process is constituted by three modules: Extraction, Transformation and Load (ETL) [Vassiliadis 2002, Vassiliadis 2009, El-Sappagh 2011], the Advanced Data Analysis (ADA)

and the Visual Knowledge Explorer (VKE) modules. Accordingly, this modular-based approach facilitates to extend architectures based on the VA process through the addition of new components such as data analyzers and visualization components.

The VA is a data transformation process that could be thought like a funnel, where raw data are analyzed and filtered in several steps, until these are converted into knowledge. Therefore the output of the process is a reduction, in terms of volume, of the original input, that contains all the required elements to inform decision making.

ETL is the first module that intervenes in this process and is comprised by several components aimed at retrieving, cleansing and integrating data from data sources such as spreadsheets, legacy systems, databases, text, XML and HTML files, logs, email communications, data streams, sensors and any other data source.

The aim of the ADA module is to produce *Knowledge Facts* using, for example, data mining, genetic algorithms, neural networks, statistical analysis and support vector machines. The ADA module carries out intermediate steps in the process of transforming data into knowledge. Its results provide very important information that could lead to decision making, but the results are still unmanageable, because of the large volume, when dealing with *Big Data*. So, the presentation of thousands or even millions of *Knowledge Facts*, in the large, is not feasible and still requires additional steps for providing usable knowledge that could be successfully employed in informed decisions. This is achieved with the use of IV and HCI.

The other component of the VA process is VKE, which is integrated by three components: the IV, the VLFA, and the VACS components (see table 8.2).

The IV component plays a central role for this module and consists of a set of visualizations. This component makes use of the VLFA component to define the associations between *Knowledge Facts* and the visualizations, and to define how the visualizations are linked together. In addition, the VLFA component carries out an automatic selection of the *Knowledge Facts* that will be visually represented in accordance with the requirements of the visualizations. VKE also makes use of the VACS component for the creation and management of data models, data structures and visual mappings, besides to keeping track of the interaction and coordination between visualizations for deciding on the data to be visualized according to the interactions and the linking between visualizations.

IV makes use of several theories, methods and techniques such as usability principles, multidimensional and multivariate visualization, HCI and information design theories among others.

The VA process (see Figure 8.1) starts with the retrieval of data, relevant to the problem under study, from heterogeneous data sources such as logs, email communications, text files or databases(Extract, Arrow 1). Following this, the data is cleaned and integrated, and then stored in the *Data Warehouse* (Load, Arrow 2). Thereafter, the ADA module reads data from the *Data Warehouse* (Read data, Arrow 3) and uses knowledge extraction techniques for discovering, representing and managing knowledge. In turn the derived results are *Knowledge Facts* that are stored in the *Knowledge Facts Database* (Produces Facts, Arrow 4).

Table 8.2: Components of the Visual Knowledge Explorer module.

Components	Description
Information Visualization (IV)	This component is the most important element of the VKE module and the VA process: it provides the visual representations and interaction mechanisms for supporting users in the knowledge discovery process. Basically, one can say that the aim of any other module or component of the process is to support the aims and tasks of the IV component.
Views Linker and Facts Analyzer (VLFA)	This should provide an interface for allowing users to create associations between visualizations and *Knowledge Facts*. Moreover, it must support the definition of the linking and coordination of visualizations based on common attributes of facts (e.g., an approach based on a relational data model [North 2000].)
Visualization Abstractions and Coordination Support (VACS)	The responsibility assigned to this component is to create the required data models, data structures and visual mappings for supporting the creation and operation of the visualizations. It also has the function of coordinating the data to be displayed by the visualizations.

The VKE module makes use of the graphical user interface component of VLFA for creating the relationships between facts and visualizations and for defining the linking relationships between visualizations. This is carried out by the tool designer before handing out the tool to the analysts that will be the final users.

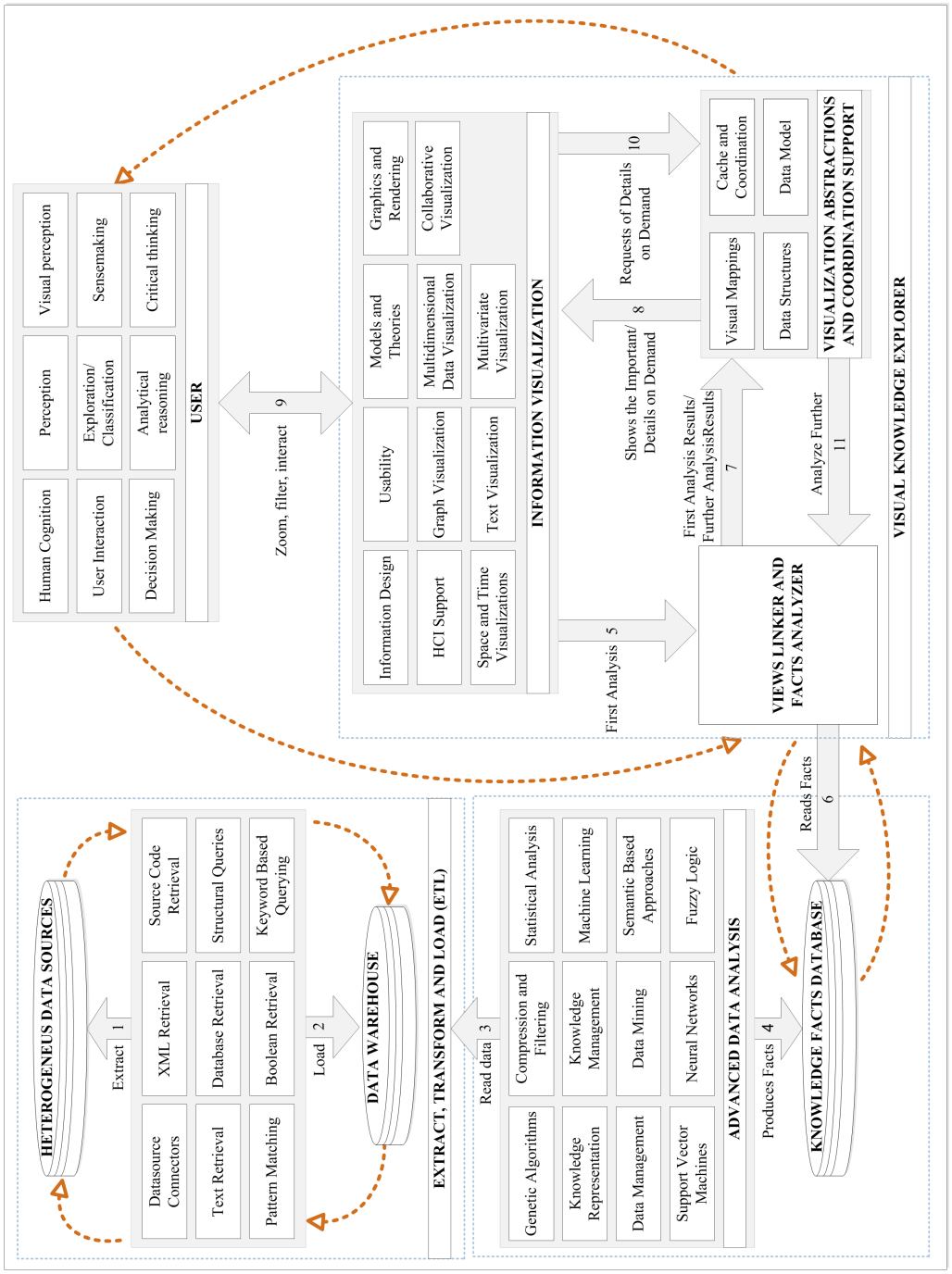


Figure 8.1: Visual Analytics Process.

When the analyst starts using the VA tool, the IV component asks to perform the initial analysis of facts to the VLFA component (First Analysis, Arrow 5), according to the linking relationships, visualization features and requirements. Next, the VLFA component reads facts from the *Knowledge Facts Database* (Reads Facts, Arrow 6) and process them appropriately, and optionally performs clustering or summarization, depending on the visualization requirements.

After this, the VACS component takes the analysis results from the VLFA component (First Analysis Results, Arrow 7) and creates the appropriate data model, data structures and visual mappings required by the IV component to display the most important results (Shows the Important, Arrow 8). Then, the IV component permits users to explore relationships and discover knowledge using a combination of visualization and interaction techniques (Zoom, Filter, Interact, Arrow 9).

It is important to take into account the fact that the processes performed by the ETL and ADA modules usually deal with large and complex datasets, and require the use of complex algorithms, thus demanding considerable processing capacity and many hours or even days for task completion. Therefore, the results produced by these modules are usually visualized once their execution has been successfully completed. The processes performed by the ETL and ADA modules should run automatically, when new data is added to the data sources, to generate new *Knowledge Facts* that are stored into the facts database in order to automatically update the visualizations.

The aforementioned process is iterative and requires additional details as the user interacts, explores and discovers knowledge through the creation of associations and relationships among visual elements in the IV component (Request of Details on Demand, see Arrow 10). Thus, the IV component automatically requests additional information for providing more details to users, and if the requested data is unavailable from the VACS component it requests a further analysis from the VLFA component (Analyze Further, Arrow 9). Consequently, the results of the further analysis process are passed to the VACS component (Further Analysis Results, Arrow 7) and added to the data model, data structures and visual mappings and cache elements. Finally, the corresponding details are visually represented in the IV component (Details on Demand, Arrow 8) and the user continues working on the knowledge discovery process until the decision to stop is taken once the proposed goals have been reached.

In this context, the fact that analysts are strongly influenced by factors such as their experience, education, cultural values [Heuer 1999] and cognition [Drigas 2011] has to be taken into consideration as a central element of the VA process because these allow them to gradually acquire

strategies for remembering, understanding, decision making, and solving problems [Academies 2000]. Thus, it can be reasonable supposed that users form a hypothesis to solve a problem, then collect data, analyze such data and then accept or reject the initial hypothesis.

8.3 Visual Analytics and Software Systems

The application of VA principles to software systems [Telea 2010, Reniers 2012] is known as Visual Software Analytics (VSA) [Anslow 2009, Telea 2011]. The use of VA in this context is an improvement relative to SV, considering VA as a comprehensive process which includes advanced data analysis and the use of multiple linked views.

Furthermore, the application of VA principles to SE shares common elements with VSA, but the principal difference between the two is that the former takes into account two or more revisions, while the latter only takes into account the analysis of only one revision of the software project. The application of VA to SE entails carrying out an individual analysis of each revision and then requires an additional analysis in order to compare and correlate the results in an endeavor to discover relationships, similarities and dissimilarities between these relationships, as well as taking account additional factors such as:

* Visualize different types of data in different time scales (years, months, hours days) and also correlate these different scales.
* Although much work has been done regarding graph animation and graph evolution, the visual representation of software structural changes is still a difficult endeavor.
* Developers and managers must be much more skillful and cautious in noting relationships and differences when more than one software project revision is required to reach a solution.

Therefore, the application of VA to SE is a specialization of VSA. A practical analogy is that VSA is like a movie frame while the application of VA to SE is a movie that is composed of multiple movie frames temporally ordered and interrelated. Consequently, this research defines the process of applying VA to SE as EVSA. The conceptual definition of this process is the following:

Evolutionary Visual Software Analytics is the process of applying Visual Analytics to software evolution to enhance understanding of software system changes with the active participation of users by means of Human-Computer Interaction.

8.3.1 Evolutionary Visual Software Analytics

The EVSA process is described in Figure 8.2 and, in general terms, it is shared by the VSA and the VA processes (see Figure 8.1). Therefore, the process uses a modular-based approach, where each module is a collection of components that are in turn formed by methods and techniques. Accordingly, the main modules of the EVSA process are: ETL, Advanced Software Evolution Analysis Engine (ASEA) and Visual Knowledge Explorer for Software Evolution (VKESE) (see Table 8.3), whose functionality is similar to their counterparts in the VA process.

The overall functionality of the module VKESE is similar to that of its analogue in the VA process, which was described in section 8.2. The main difference between the two is rooted in the components of visualization and data types that they represent. It is thus recommended that Figure 8.1 and Table 8.2 be revised if greater details are sought. It is worth mentioning that the visualization components of the SEV sub-module (see Figure 8.2) are the visualizations which were identified in chapter 4.

Table 8.3: Responsibilities and functions of the modules that make up the EVSA process.

Module	Description
Extraction, Transformation and Load (ETL)	This module has the function of performing the connection and data retrieval from software repositories, defect-tracking systems, emails, source code revisions, testing systems, logs and any other available data source. When the data is retrieved, it is cleaned, merged and loaded into a data warehouse.
Advanced Software Evolution Analysis Engine (ASEA)	This module is comprised of analysis techniques [Hassan 2005, Hassan 2006, Kagdi 2007a] that could be used in a individual basis or combined in order to extract knowledge facts. For further details on these techniques see chapter 4 and Figure 8.2.
Visual Knowledge Explorer for Software Evolution (VKESE)	This module is made up of three components: SEV, VLFA and VACS.

The steps followed by the EVSA process were organized into phases and listed as follow:

Phase I: Data Retrieval and Loading It retrieves and carries out an initial data processing, after which it stores them into a data warehouse.

Data retrieval: According to the type of task that the researcher or designer seeks to support the retrieval process can be performed in software repositories, defect-tracking system logs, emails, source code and testing system logs. Techniques used in recovering data may include source code retrieval, structural queries, pattern matching and text retrieval (Extract, Arrow 1).

Data warehouse: Once the data has been recovered, it is then cleaned, integrated and correlated and then stored in a data warehouse (Load, Arrow 2).

Phase II: Data Analysis This phase analyzes and extracts SE facts and then proceeds to store the results in a database.

Analysis and facts extraction: When new data is available in the data warehouse, ETL reads the data (Read data, Arrow 3) and then ASEA proceeds with the analysis, using one or more analytical techniques, depending on the task being undertaking. The analysis techniques include origin and contribution analysis, frequent patterns mining, defect classification and refactoring analysis.

Storage of evolution facts: Once the analysis has been carried out, the evolution facts are then stored in the Software Evolution Facts database (Produce facts, Arrow 4).

Phase III: Structure Loading and Visualization Mapping The tasks of this phase include loading SE facts, creating the data structures and visual mappings, and loading the visualizations.

Visualization loading: The user launches the SEV component that uses linked visualizations. Some of the visualizations that can be used are shown in Figure 8.1.

Data fact structures request: When the SEV component is loaded, the data fact structures required by the visualizations are requested by the VLFA component (First analysis, Arrow 5 and Read Facts, Arrow 6).

Facts loading: The VLFA component read facts from the Software evolution Facts Database and pass them to the VACS component (First analysis results, Arrow 7).

Structures and visual mappings: The VACS component creates and passes the appropriate data model, data structures and visual mappings to SEV (Show what is important, Arrow 8).

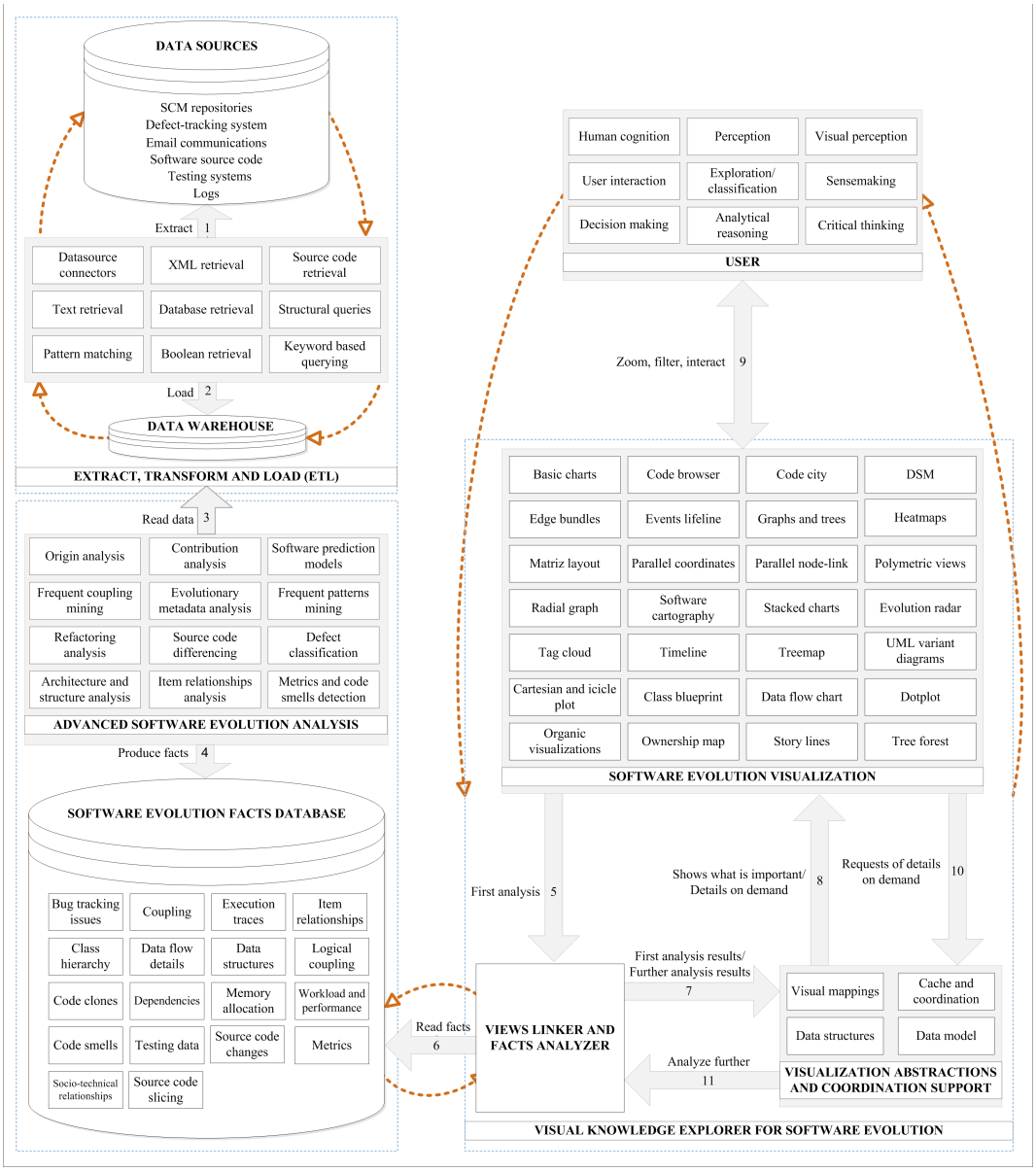


Figure 8.2: Overview of the Evolutionary Visual Software Analytics process.

Phase IV: User Interaction and Details on Demand This phase is the final stage of the process of transforming data into knowledge. After the retrieval, analysis and visual mapping of information, this phase makes possible a feedback loop between the user and the system: the user requests additional data to the system by means of the available interaction possibilities, and the system provides the requested data. According to user interactions the knowledge discovery process is refined and progresses towards the finding of useful knowledge and answers.

User interaction: During the process of knowledge discovery, the user

browses, filters and explores different perspectives on the data, selecting elements from one or more of the visualizations (Zoom, filter, interact, Arrow 9).

Requesting details: According to the needs and the interactions of the user, the visualization requests new data fact structures and visual mappings to provide additional information to the user in accordance with the selected options (Request of details on demand, Arrow 10).

Additional details: If the additional details that have been requested are available in the form of data fact structures and visual mappings are passed to the SEV component (Details on demand, Arrow 8). However, if these details are not available, a request is passed to the VLFA component (Further analysis, Arrow 11) which reads the additional facts, (Read facts, Arrow 6), transforms the details and then (Further analysis results, Arrow 7) passes them to the VACS component so it can proceed to create data fact structures and visual mappings.

Discovery of knowledge: The user continues to interact with the system until the necessary knowledge is obtained or it is considered that it is impossible to reach a determinate conclusion using the available data and representations.

8.3.2 Architecture Specification

Defining the architecture of software tools is a very complex task that requires careful analysis. It is a challenge to determine which techniques to use and how these will be interrelated. This section seeks to contribute to the specified objectives, answer the research question formulated in section 8.1 as well as support tool design in situations where VA is applied to SE. Accordingly, using as a reference the EVSA process, an architecture for a tool denominated *Maleku* was defined [González-Torres 2011, González-Torres 2013b, González-Torres 2013a].

Maleku seeks to support both programmers and software project managers when correlating metrics, project structure, inheritance, interface implementation and socio-technical relationships. Such architecture has been implemented in Java and tested on open source software projects, and the test results are presented in chapters 9 and 10.

The modules of the architecture (see Figure 8.3) are similar to those of the process described in the previous section and have been given the same name. The operation of the modules ETL and ASEA is synchronous while the operation of VKESE is asynchronous, in relation to the other two

modules. The architecture is based in the client/server model, in which the modules ETL and ASEA are executed by the server and VKESE is an Eclipse plugin executed by the client. The different modules and components of the architecture are described in the following order: data retrieval, data analysis and visual representation.

The ETL module comprises a sub-module (SM) and two components (C), as shown in the following list:

Data Source (C):[1] The data sources used by the implemented architecture consist of the SCM software repositories of software projects. The information that is extracted from these repositories include the metadata associated with changes to source code, programmers activities, project structure and source code.

Sensor of New Revisions (C): The *Sensor of New Revisions* is a process that continuously monitors the addition of new revisions to software projects and notifies to the Data Extractor.

Data Extractor (SM):[2] The function of this sub-module is to extract the data required in order to carry out an analysis, whose results are used to feed the visualizations of the VA tool. It is made up of the *Architecture and structure retrieval*, *Source code retrieval*, and the *Metadata retrieval* components, described as follows:

Architecture and structure retrieval (C): This component is responsible for extracting details of the project structure for each revision, with particular interest on the packages of the system and their organization.

Source code retrieval (C): It is responsible for recovering the source code for each of the system revisions and for storing classes with basic information about their location in the system architecture.

Metadata retrieval (C): The data that this component is responsible for retrieving, includes the logs of each revision and its associated details: the date on which the revision was carried out, which programmer carried it out and which elements were affected.

The sub-modules that conforms ASEA are *Source Code Analyzer* and *Metadata, Software Evolution Analysis and Correlation Engine*, whose components and descriptions are explained next.

[1] C makes reference to component.
[2] SM refers to a sub-module.

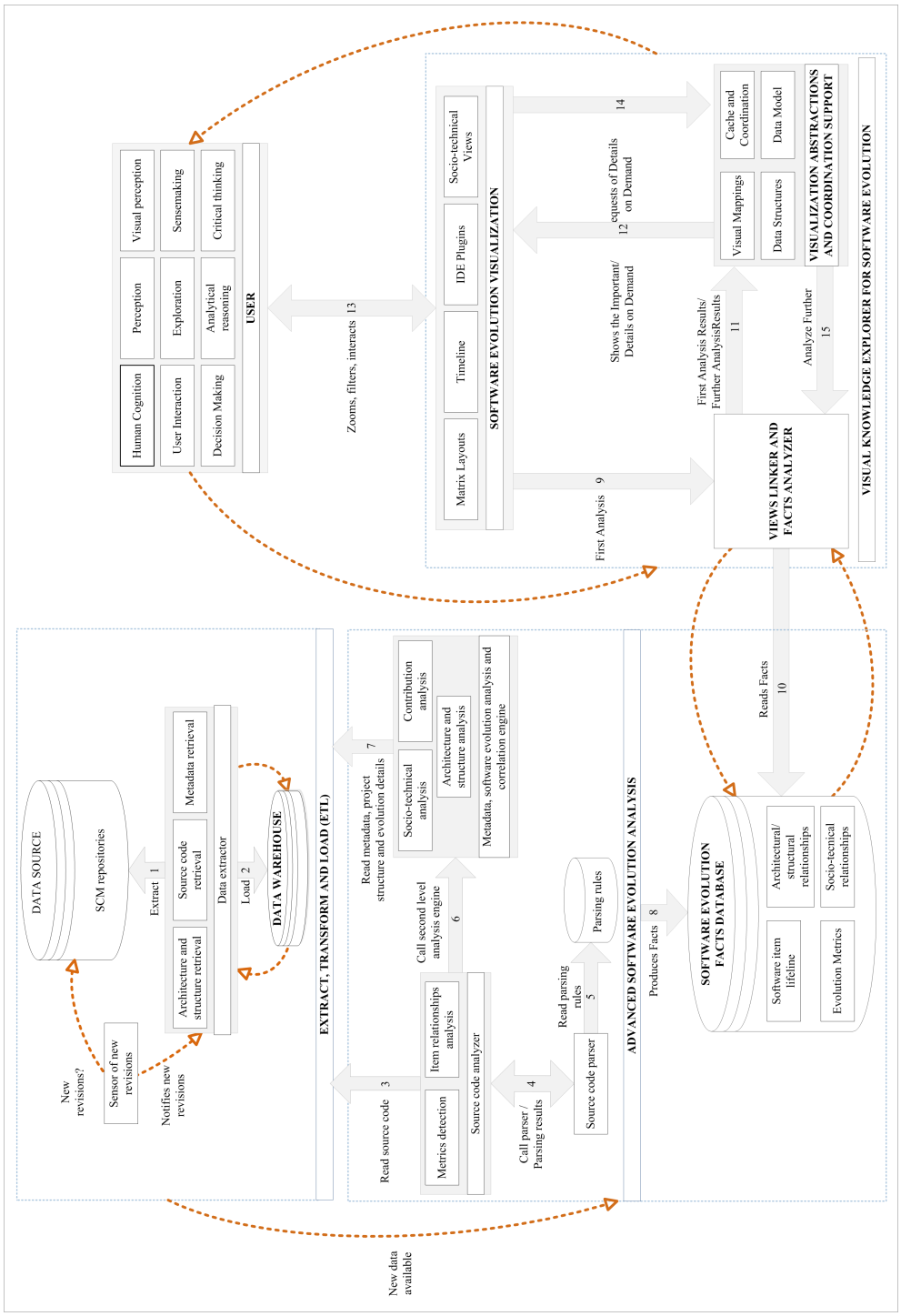


Figure 8.3: Overview of the architecture for *Maleku* .

Source Code Analyzer (SM): This sub-module is responsible for carrying out analysis of the revisions of the source code of the project using the following components:

Metrics detection (C): This component is responsible for detecting and calculating metrics using details from the parsed source code. Some of the metrics that can be calculated by this component include LOC, Number of Methods (NOM) and Cyclomatic Complexity.

Item relationships analysis (C): The functions of this component include the detection of inheritance (parent-child and child-parent) and interface implementation (implementing and implemented by) relationships.

Source Code Parser (C): It reads each source code file, line by line, in order to identify classes, interfaces, methods and declarations, and applies parsing rules. It allows to calculate metrics and to identify the relationships between software items.

Parsing rules database (C): *Source Code Parser* applies parsing rules, which are stored in text archives. Some of these rules are generated automatically while others are created manually.

Metadata, software evolution analysis and correlation engine (SM): This is invoked by the sub-module *Source Code Analyzer* when it terminates the segment of analysis apportioned to it. Its function is to identify socio-technical relationships and determine the contributions made by individual programmers, as well as analyzing the architecture and structure of the project for each revision under analysis. The components of this sub-module are:

Contribution analysis (C): Based on the metadata of SCM repositories, a cumulative calculation of the elements changed for each revision and programmer is carried out.

Socio-technical analysis (C): Using the metadata of SCM repositories the relationships between programmers and software items are examined as well as the relationships which are created between programmers using as a basis the elements which have been changed in common.

Architecture and structure (C): The results produced by *Source Code Analyzer* and the information obtained from the metadata of SCM repositories are used to correlate software project structure, metrics and relationships between software items. It further

gathers information about the creation of software items and their life-line during the project.

Software evolution facts database (C): This database stores the analysis results produced by other sub-modules and components of ASEA. In order to do this, it uses a database design that emulates the structure of software projects: project –> revision –> package –> file –> software item.

The sequence of steps that follows the process of data retrieval and analysis (made up of ETL and ASEA modules) are:

1. The user enters the connection parameters of the SCM repository and database where data of the particular project that needs to be analyzed will be stored.
2. When the process has been initiated, the retrieval components included in the sub-module *Data extractor* (*Architecture and structure retrieval*, *Source code retrieval* and *Metadata retrieval*) carry out the task of data retrieval (Extract, Arrow 1)
3. Once data has been retrieved, it is loaded into the *Data Warehouse* (Load, Arrow 2). Data loaded into the Data Warehouse are used as benchmarks for later retrieval processes.
4. When project data have been retrieved, the *Sensor of new revisions* will be responsible for monitoring the availability of new revisions in the SCM repository and to notify retrieval modules in a timely manner.
5. As data is retrieved, ETL informs ASEA that new data is available to perform the respective analysis concordant with the analysis components available.
6. The sub-module *Source Code Analyzer* reads the *Data Warehouse* in the ETL module (Read source code, Arrow 3) in order to detect and calculate metrics for classes and methods, and analyze the relationships between software items such as the hierarchy of classes and interface implementation.
   6.1. To carry out its tasks, *Source Code Analyzer* requires to parse the source code and then the *Source Code Parser* component is notified. (Call parser, Arrow 4).
   6.2. The component *Source Code Parser* reads the parsing rules from its own database (Read parsing rules, Arrow 5) and perform the parsing of the source code.

7. When the sub-module *Source Code Analyzer* has finished carrying out the analysis, it stores the results in the *Software Evolution Facts Database* and notifies the *Metadata, Software Evolution Analysis and Correlation Engine* (Call second level analysis engine, Arrow 6).
8. The sub-module *Metadata, Software Evolution Analysis and Correlation Engine* reads evolution facts from the *Software Evolution Facts Database*, metadata, project structure and evolution details from the database of the ETL module (Read metadata, project structure and evolution details, Arrow 7). Using this information the module then carries out a more profound analysis regarding socio-technical relationships, analysis of the contribution of programmers and the architecture and structure of the software project under consideration.
9. The process which carries out the ETL and ASEA modules runs indefinitely for each of the projects configured until the analysis for one or more projects is stopped by the user.

The design of the architecture permits the addition of new components to modules and sub-modules to allow connections to be made to new data sources, perform other types of data analysis and visualize the results of the analysis with new visual representations. The steps followed by ASEA are the same as those that were described in section 8.3.1 with regard to this module, so that the explanation of these steps is omitted.

8.4 Conclusions

The process explained builds on the visual analytics process which is described in section 8.2 and deepened by means of the design and implementation of an architecture which follows this process, referred to as Evolutionary Visual Software Analytics (EVSA). The detailed design of the architecture identifies and explains the roles, border and interactions between the modules, components, methods and techniques used.

Therefore, the feasibility of the implementation of the architecture presented in this chapter, and thus the applicability of the EVSA process is discussed in the next chapters.

Chapter 9

Visual Analytics Explorer for Software Evolution

Sorprendido por la tarea que le había sido encomendada, Güindy preguntó a Azul, el dueño de la fábrica, "¿por qué quieres que diseñe nuevos relojes, si los que fabricas son famosos por su precisión?". "Son famosos porque son los mejores que se conocen, pero tu trabajo será hacer otros mejores", respondió Azul. Güindy se quedó desorientado y ansioso, aunque antes de adentrarse en esta aventura vivía en una gran ciudad, no conocía de relojes y nunca había trabajado en una fábrica. — El viaje de Güindy, A.González

Contents

9.1 Introduction

The aim of this chapter and chapter 10 is to explain the features of the visualizations that conforms the VKESE module (see section 8.3.2 and Figure 8.3 for further details), their use in supporting situational awareness and collaboration and in the analysis of patterns in software projects.

Accordingly, the following section presents a framework for explaining how situational awareness supports GSD and collaborative work, as well as the relationships of this framework with the VKESE module of *Maleku*. Thereafter, the next sections explain the decisions that were taken for designing the visual representations that comprises VKESE and how those visualizations contribute to knowledge discovery and decision making processes in the context of software project management. Furthermore, the sections in chapter 10 are committed to explain the design decisions that were taken concerning the design and implementation of Revision Tree (RT) (a visualization that is integrated into VKESE), a case study linked to the software industry on the use of this visualization and its use in analyzing source code files of open source software projects.

9.2 Framework: situational awareness and collaboration

SDME processes are complex, dynamic, unpredictable and it is common that they are carried out in different geographical locations and for several years, as it was mentioned in chapter 6. This makes it essential that team members establish collaborative relationships, be they long term or one-off, to accomplish tasks and solve specific problems. Thus, when a task has been completed or a problem has been solved the individuals will go on to work on the next task on the agenda which is based on project planning or on the list of pending problems. This implies the collaboration is not always conducted among the same members of the team, but rather in terms of expertise and the tasks being undertaken at any particular moment.

Collaboration among members of a team can begin in a number of different situations, which implies that there is no relation between aspects of teamwork (see Section 6.2.1). Accordingly, the framework illustrated in Figure 9.1 aims to provide guidance on a possible configuration of the relationship between these aspects as well as define the role they can play in the process of collaboration. In Figure 9.1, the composite labels (*e.g.*, activates / supports) on lines with bidirectional arrows are read following a descending / ascending order. The descending order is initiated with the concept at the top end of the line and ends with the concept at the lower end (*e.g.*, Distributed Team Cognition –> Activate –> Collaboration), while in the case of ascending order, the reverse order is followed (*e.g.*, Collaboration –> supports –> Distributed Team Cognition).

The framework of Figure 9.1 has been defined taking into account a distributed cognition approach in which the members of the teams use their

individual cognition, according to their specialization and experience, and which is complemented with the cognition of the other individuals to act in the performance of tasks and problem solving. On this basis, the collaborative process can be triggered by one or several members of the team using their cognitive abilities when required to perform a task or when a situation is detected about which it is necessary to act.

To make collaboration possible among team members, communication, coordination and control among team members may be required. It is important to note that collaboration among individuals requires information to facilitate the distribution of tasks and to determine the actions that should be followed according to the status of the project. So, the capacities and information provided by SAWs on the evolution and status of the processes, tasks, activities and changes that have been made to the software items can be very useful. The relationship between the process of collaboration and the communication, coordination and control activities is reciprocal, as shown in Figure 9.1.

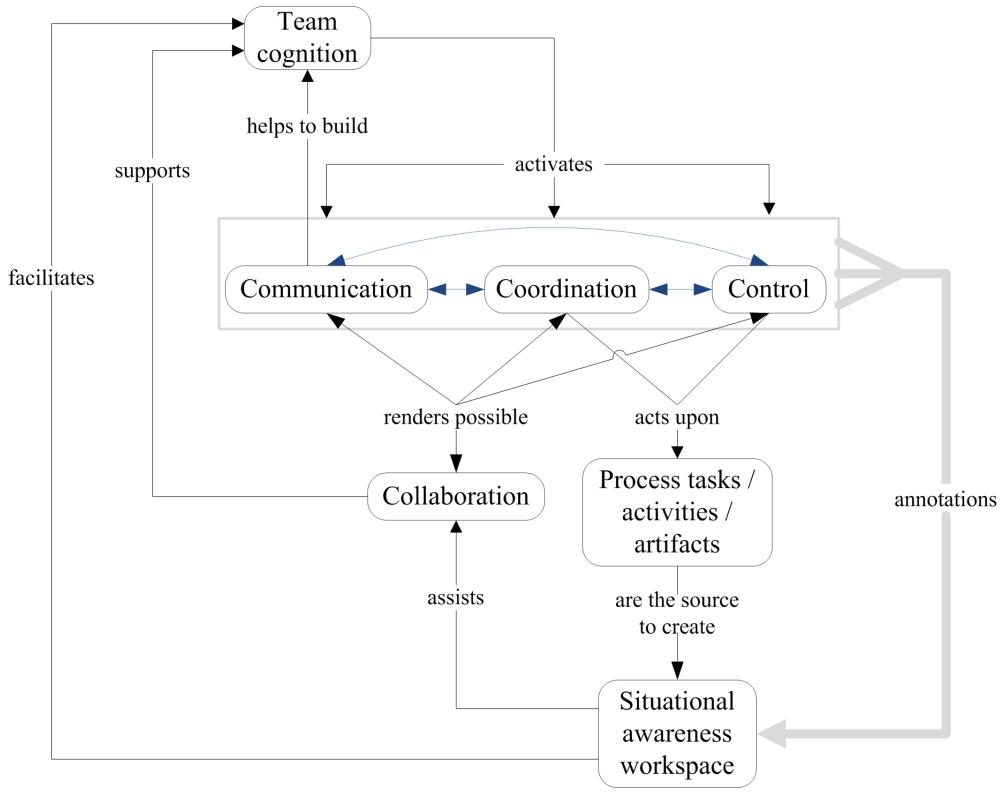


Figure 9.1: Framework for collaborative work in SDME processes [González-Torres 2014].

In an analogous manner one or more members of the team may activate the communication, coordination and control activities, so in turn the process

of collaboration between the appropriate individuals could be initiated, and if necessary the capabilities of SAWs can be used.

With regard to this point it is worth mentioning that the goal of a SAW is to provide information about the processes, tasks, activities and artifacts of software systems, but also provide the possibility of interaction for users who can annotate visualizations during the communication, coordination and control activities (see Figure 9.1). The purpose of the SAWs is to support the collaboration and decision making processes, so its design can consider some of the following objectives:

* Provide information on the state of the system to facilitate the completion of a task or to solve a maintenance problem.
* Facilitate the construction of individual and group cognition based around the processes of SDME of a software project to provide tracking information on the status of the system.
* Show patterns of unexpected behaviors derived from system changes to trigger collaboration mechanisms among team members.
* Support project managers in decision-making processes that may be related to the allocation of tasks to team members as well as to activities and tasks in progress or very specific technical details of software system elements.
* Provide details about the progress of work being done by other team members and the general state of the project to assist programmers in the fulfillment of their tasks, goals and objectives.

It should be noted that coordination and control activities have direct effects on the processes, activities and software items, which are the data source used to feed the visualizations of SAWs tools.

Finally, it is worth to mention that the visualizations that comprise the VKESE module of *Maleku* are targeted to serve as a SAW for facilitating the comprehension of the evolution of software systems and hence to support the SDME processes.

9.3 Visualization Designs and Use Case Scenarios

The main view of VKESE and the visualizations that it includes are shown in Figure 9.2: GT (displayed at the left bottom corner), *Gridmaster* (located at the right top panel), STG and RT[1] (the last two are located at the right bottom panel).

[1]The design and features of the RT are explained in detail in chapter 10.

The visualization outcomes that are presented in the remaining sections of this chapter are the result of applying GT, *Gridmaster* and STG to data from the evolution of *jEdit*, *JabRef* and *JFreeChart*, three open source projects written in *Java* which are described below:

1. *jEdit* is an open source text editor for programmers that is available at http://sourceforge.net/projects/jedit and whose development started on December 1999. This study takes into account nearly 1212 classes and 5801 revisions that were carried out between September 2001 and April 2014.
2. *JabRef* is a bibliography reference manager, released on November 2003 and available in http://jabref.sourceforge.net. The evolutionary data under consideration includes 3719 revisions and 1236 software items that were created during 9 years (from October 2003 to November 2011).
3. *JFreeChart* is a chart drawing library (http://www.jfree.org/jfreechart), under the responsibility of a single programmer, that started being developed in February 2000. This study takes into account 916 revisions of this project and 1130 software items distributed from 2007 to 2010.

GT provides an overview of project activities; *Gridmaster*, STG, and RT depict specific details regarding the correlation of project structure, associations, collaboration and time at different granularity levels of the project structure. Thus, after the visualizations are launched from the contextual menu of the *Eclipse's Package Explorer* view (located at the left top panel and pointed out by the number 1), the knowledge discovery workflow (as indicated by the numbers and arrows in Figure 9.2) begins with the analysis of the programmers' contribution patterns in GT, and the selection, for further analysis in the *Gridmaster*, of one or more time units from the radial view. Afterwards, the user can select either one of these two options from *Gridmaster*:

1. A software item in the tree located in the left hand side of *Gridmaster* to obtain details regarding the collaboration during the evolution of such item (using RT).

2. A time unit in *Gridmaster* for presenting the associated socio-technical relationships in STG.

The SE facts that have been taken into account for the visualizations presented later are the following: software item lifelines, evolution metrics, socio-technical relationships and some architectural/structural relationships such as, for example, inheritance, interface implementation and the correlation of structural data with metrics.

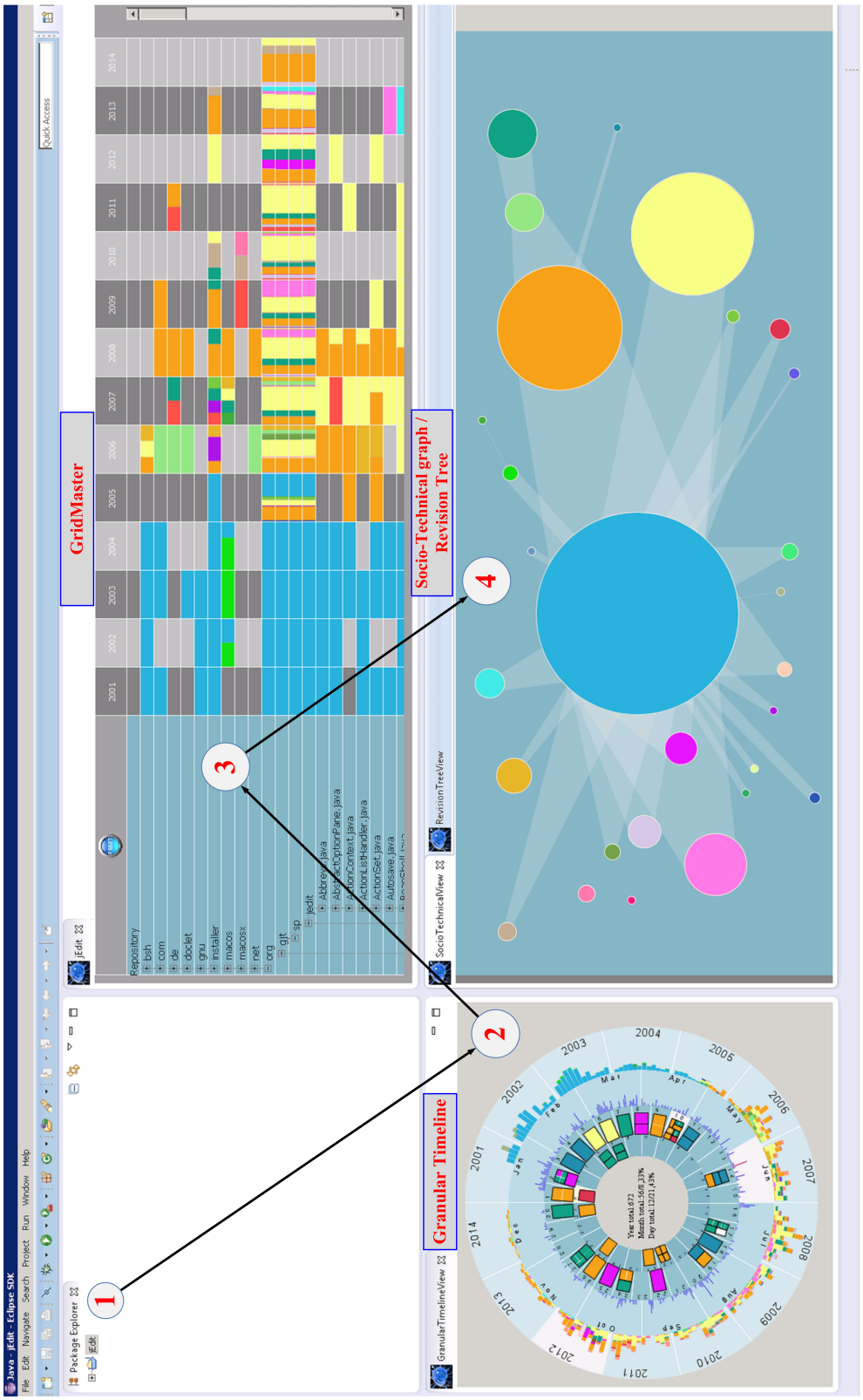


Figure 9.2: Knowledge discovery workflow in the Software Evolution Visualization module of *Maleku*.

9.3.1 Granular Timeline: Analysis of Statistics on Revisions and Contributions

GT uses a modified circular ring chart layout to show an entire overview of the temporal dimension of a project (Figure 9.3). Concentric rings convey the different time scales that record change events, from coarse (years, outer ring) to fine grain (hours or finer, innermost ring). A related layout was used by Holten *et al.* to depict software project hierarchies [Holten 2007, Cornelissen 2007]. This type of layout compactly presents large quantities of data and provides an overview + detail strategy.

GT can be used to represent any kind of quantitative data produced over time, given its nature to depict numerical values and statistic results. However, it is utilized in this research to represent statistics based on the commits of revisions and programmer contributions, as it is explained below.

Statistics on revisions: The space within each chart cell is used to embed different types of visualizations for representing the number of revisions at the level of detail of a particular cell. The *years* ring cells insert bar charts that show the number of revisions for each month for each year. The *months* ring embeds a height plot chart presenting the number of committed revisions per day of month. Next, the *days* ring shows revisions in each day, so this ring has 28, 29, 30 and 31 cells (according to the number of days in the month that is selected), and the revisions at this level are depicted by bar charts or treemaps according to user selection (see Figure 9.4 for the treemap representation). In each cell, a matrix dot plot is drawn in polar (ρ, ϕ) coordinates, where ρ maps the hour at which a revision was created. The innermost *hours* ring shows revisions made each hour, so it has 24 cells, and represents the revisions by means of bar charts or treemaps.

The bar chart representation provides details about the number of contributions at each granularity level. However, the representation of data at the *days* and *hours* granularity levels does not take full advantage of the small graphic area available. Thus, treemaps were applied as they employ a space filling algorithm, which makes better use of space, and additionally permits better highlight of individual revisions.

Statistics are displayed in the center of the visualization as the time units are selected. Note that time unit selections begin in the outer ring that corresponds to 'years' and follows the sequence months –> days –> hours (e.g., 2008 –> June 2008 –> June 3th, 2008 –> June 3th,2008 at 22), as highlighted in Figure 9.3.

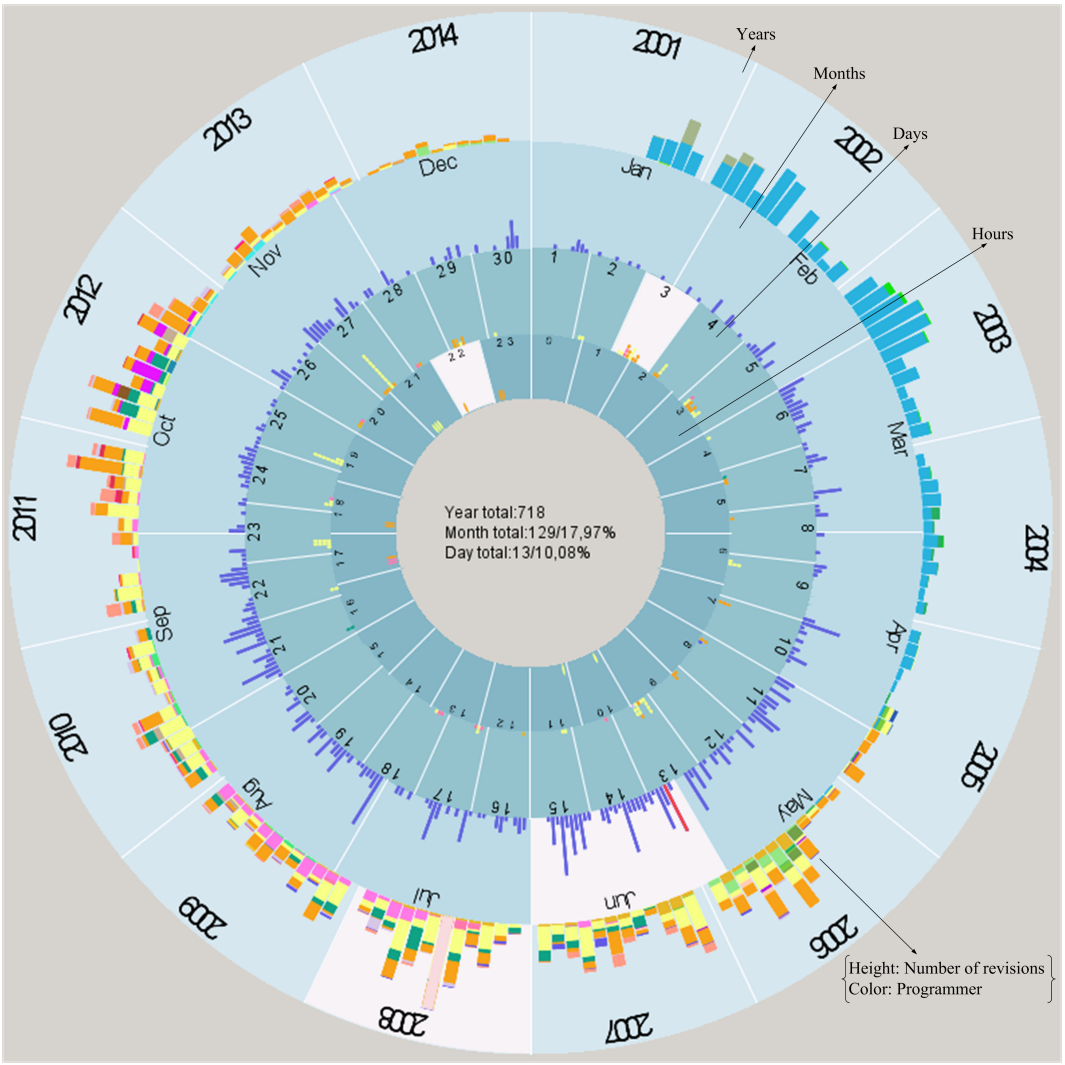


Figure 9.3: GT showing statistics for revisions committed for the *jEdit* source open software project spanning 14 years.

Programmer contributions: The contributions of programmers are color coded, where each color is matched to a particular programmer. In general the use of color can permit the acquisition of information related to statistics of committed revisions and those who have carried out such revisions, information that can be observed at first glance.

It is thus easy to extract patterns concerning the programmers who intervened in the development of a project. This visualization is simple and intuitive and can be used with little change in the representation of any type of statistics, as it was previously remarked.

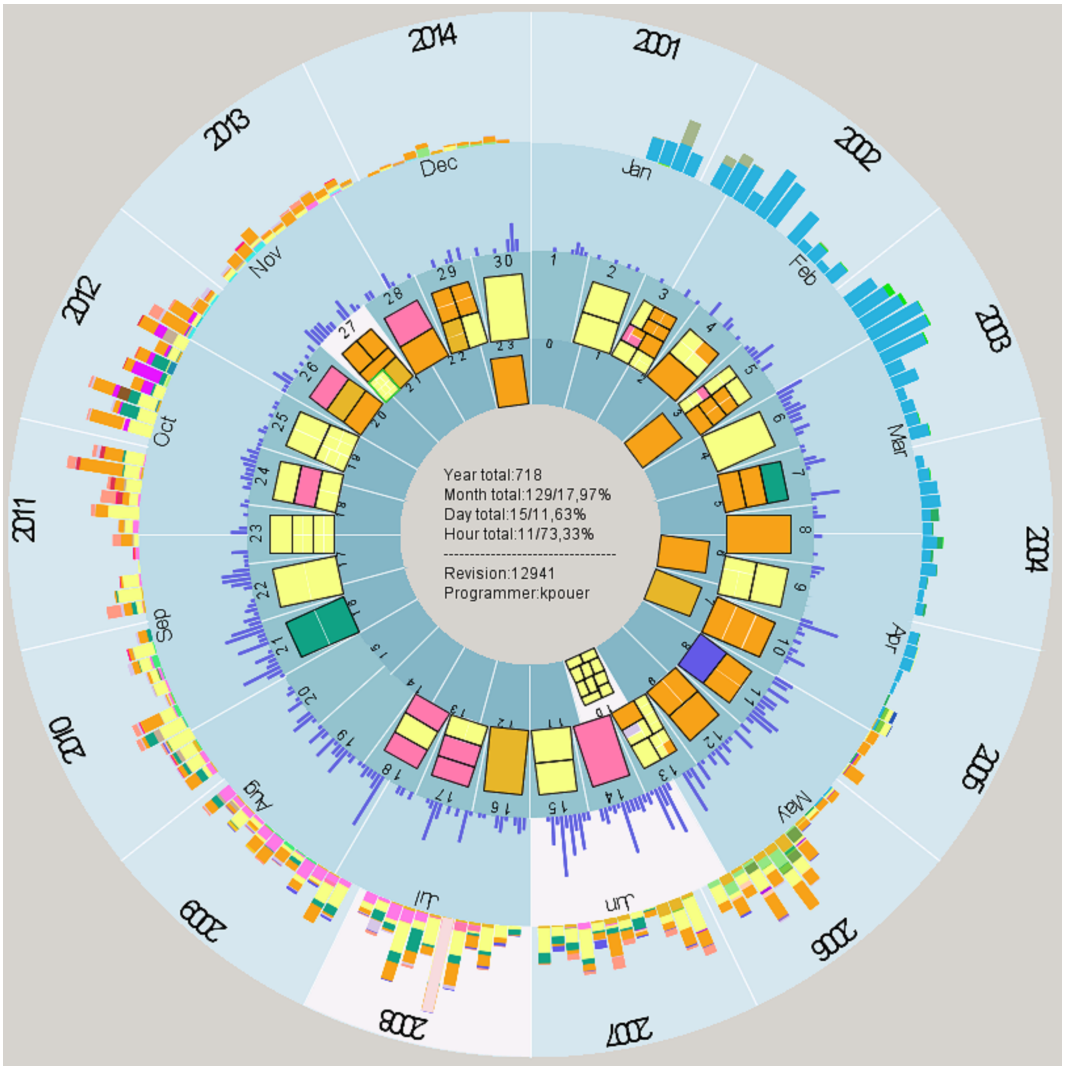


Figure 9.4: GT showing statistics for revisions using treemap representations for *jEdit*.

Figure 9.3 allows to observe at first glance that the evolution of *jEdit* has been carried out since 2001 to 2014 and accounts 14 years. Moreover, it easily conveys details concerning the years with more revisions (2008 and 2012) and the collaboration patterns of programmers. Thus, *spestov* (turquoise) is the developer that contributed the most to the project during the first 5 years (2001, 2002, 2003, 2004 and 2005), and later other programmers such as *ezust* (orange) and *kpouer* (yellow) led the programming of the system, according to the information derived from the number of revisions.

Although some software projects follow a similar pattern to the one of *jEdit* (*e.g.*, a lead programmer contributes to the project during its first years of life), the arrangement of programmer contributions varies from one project to other. This is the case when comparing the output of GT for *jEdit*, *JabRef*

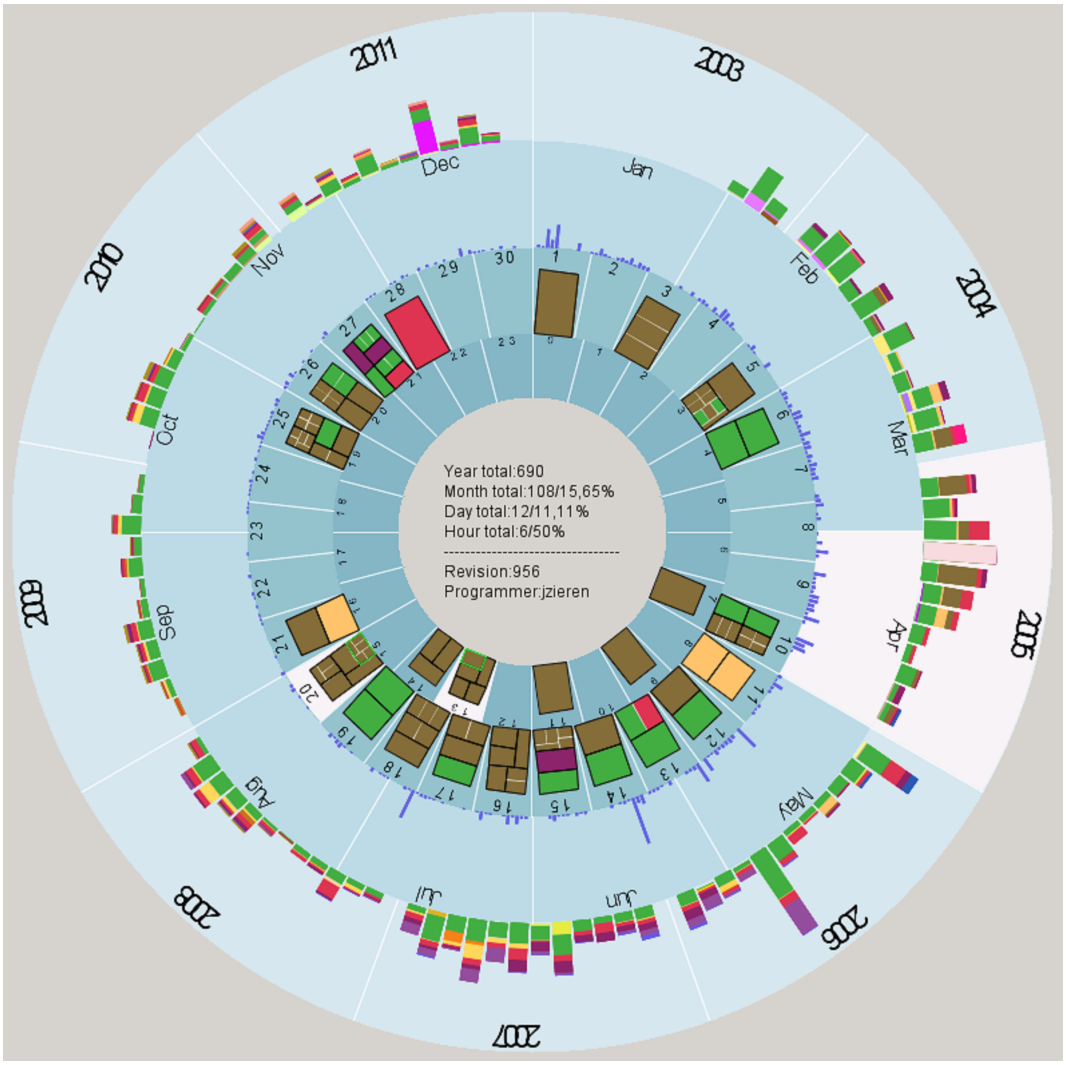


Figure 9.5: GT showing statistics for *JabRef*.

and *JFreeChart* (see Figures 9.5 and 9.6).

The contributions of *mortenalver* (green) in *JabRef* (Figure 9.5) are distributed throughout the evolution of the project and are not concentrated at the beginning of the development. Furthermore, the contributions of other programmers such as *coezbek* (dark purple), *jzieren* (brown) and *kiar* (yellow) are interleaved with those made by *mortenalver*. But in contrast, *JFreeChart* (Figure 9.6) is a system that has been developed, mostly, by a single programmer with some small collaborations of other two developers. So, most of the contributions depicted by GT (see Figure 9.6) are colored blue: the color used to represent to *mungady*, the username of the project leader. Thus, the pattern of contributions of *JFreeChart* differs from the one of *jEdit* and *JabRef*, which have been developed by several programmers.

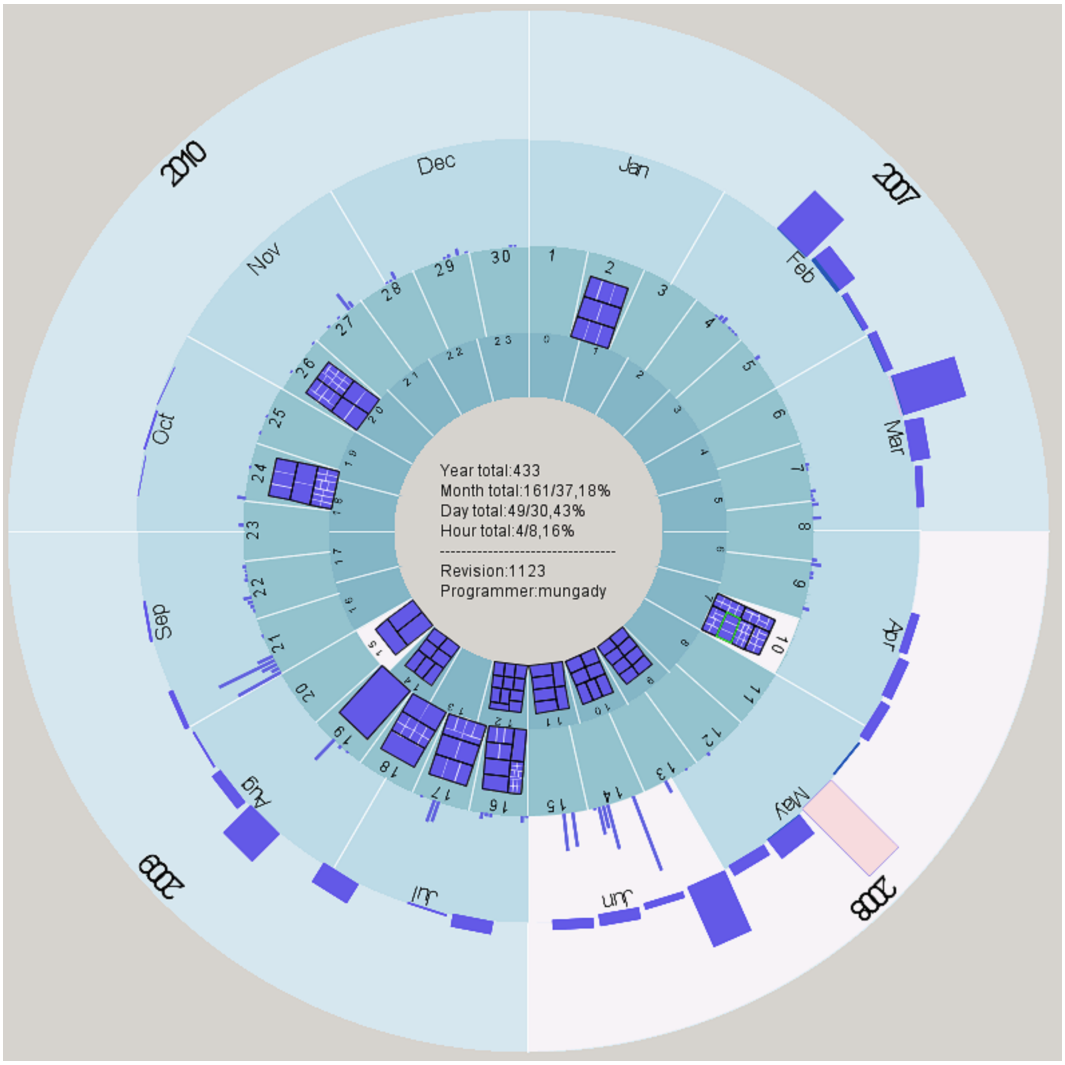


Figure 9.6: *JFreeChart*: GT visualization depicting statistics on revisions.

Another relevant feature of this visualization is its capability for uncovering contributions patterns. Figure 9.5 makes evident that programmers made revisions several times per hour, as can be noted in the innermost ring (the *hours* ring). This can lead to many questions from the project manager, among which are: Why has this programmer made so many commits? Has this programmer corrected errors on previous commits, then fixed them and commit again? Is this programmer accustomed to make commits as an analogy to saving changes to the project? Moreover, this visualization permits project managers to obtain insight on contribution patterns associated to the working hours of programmers: one can select a given day and review the hours in which programmers have made commits, revealing the working pattern of developers.

The answer to the previous questions could help the project manager to make decisions regarding the compliance of human resources to rules and training. A relevant decision for project managers, after this analysis, is the one concerned with the programmers that require training and the contents of such training.

GT is a visualization that conveys details on the evolution of software projects for long time periods and which offers interaction mechanisms to follow a path that starts at coarse grained levels and allows to move into finer grain levels of details. Similar to the outcomes shown for *jEdit* in Figure 9.4, the results displayed in Figures 9.5 and 9.6 allow to quickly grasp general information about the years of the evolution of *JabRef* and *JFreeChart*, as well as details concerning the time period that is under study and the months with more contributions (in terms of the number of revisions).

9.3.2 Gridmaster: Correlation of Project Structure, Software Item Relationships and Metrics

Gridmaster is based on a tree and a grid representation, two structures widely known by programmers as these are common in computer programming (see Figure 9.7). The tree structure is made up of all the packages and software items that have been added to the project during its evolution, whereas the grid layout is created by the intersection of rows and columns: the packages and software items are placed in the tree structure and associated to rows, and the time units are linked to columns. The use of the tree and the grid permits the correlation of all the software items involved in the evolution process with programmer contributions, the creation of software items, architectural relationships changes such as the addition or removal of inheritance and interface implementation, and metrics.

Accordingly, some of the features of *Gridmaster* permit to extract collaboration details at different granularity levels similar to GT. However, *Gridmaster* was designed to be a complementary view of GT. In this sense, GT was sought to offer an overall picture as well as top-down statistics views concerning the revisions of a software system, while *Gridmaster* was designed to correlate programmerś contributions with software items. Thus, *Gridmaster* can be used to depict the socio-technical relationships between software items and programmers as well as the lifeline of software items for the entire evolution of a system or a particular period of time that is selected from GT. The representation of these features is carried out by means of colors, as it is explained below.

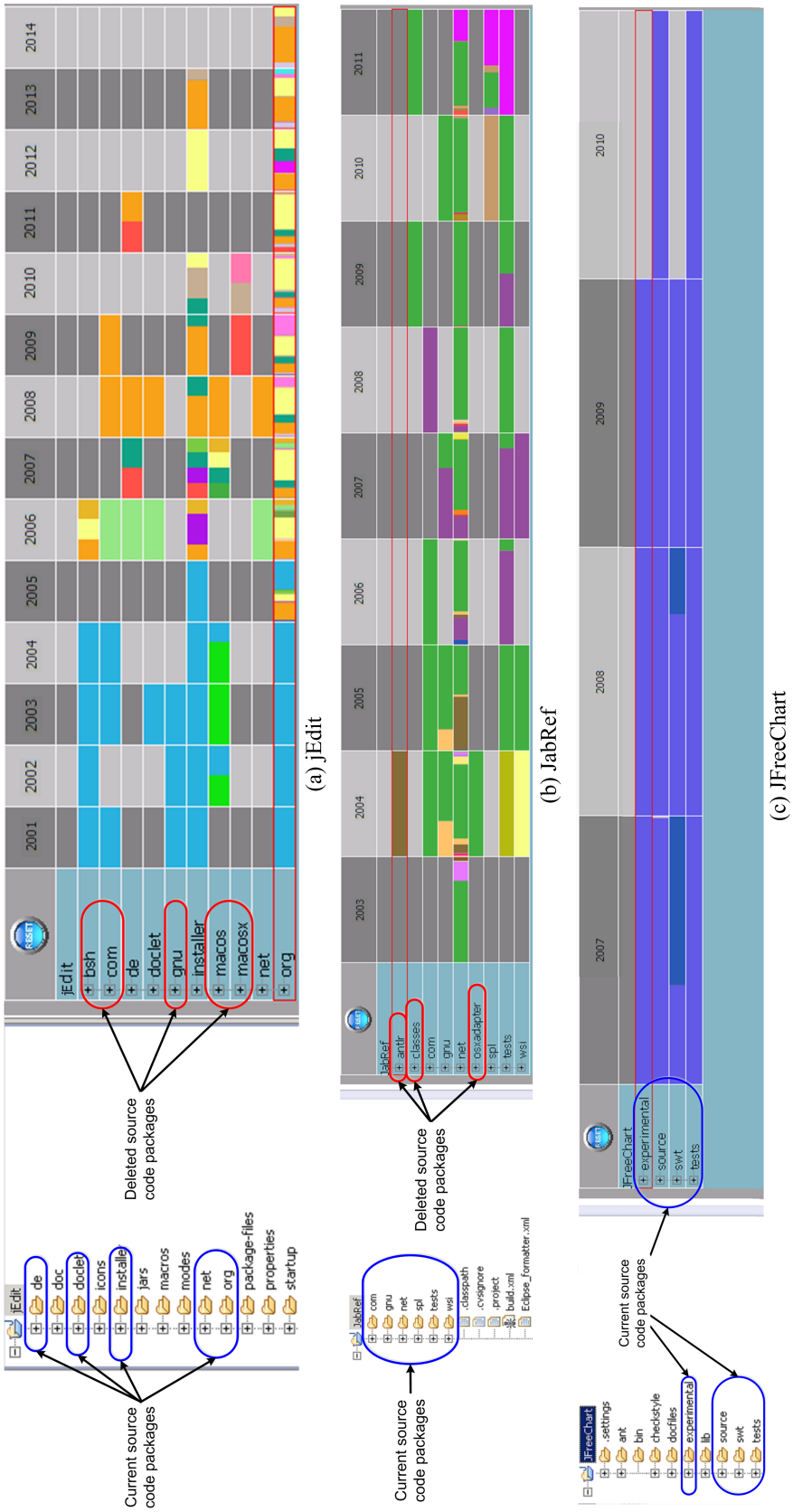


Figure 9.7: *Absolute representation* of programmer's contributions, project structure and lifelines for *jEdit*, *JabRef* and *JFreeChart*.

In this visualization colors are assigned to programmers and the area associated to each programmer depends on the number of contributions made (in terms of committed revisions) as well as on the representation that has been chosen from the contextual menu: whether relative or absolute (see Figure 9.7 for screenshots on these characteristics). In this context, $\varepsilon = \Delta/\eta$ is the area assigned to any time unit in the visual representation, where Δ is the size of the graphic area in pixels and η is the number of time units (years, months or days) in the visual representation. Thereafter, for the *relative representation*, $\alpha = \varepsilon/\tau$ is the area assigned to a programmer contribution, where τ is the number of contributions for the time unit with the most contributions. Therefore, the area assigned to a given programmer for a given time unit is $\Lambda = \alpha * \beta$, where β is the number of contributions made by the programmer during that particular time unit. In the case of *absolute representation*, the area assigned to a programmer contribution is different for each time unit. Thus, $\gamma = \varepsilon/\omega$, where ω is the number of contributions for the time unit under consideration.

The purpose of the *absolute representation* is to depict the lifeline of software items and packages using an intuitive approach. Figure 9.7 (a) shows that the activities performed on the package *bsh* of *jEdit* were carried out between 2001 and 2006. It also displays that this package currently is not part of the latest version of the project, which is corroborated when reviewing the structure of *jEdit* on the *Eclipse* workbench. Moreover, Figure 9.7 (a) allows to note that other packages such as *com*, *gnu*, *macos* y *macosx* either are part of the first level of the project structure (although *macos* and *macosx* were moved into other packages located at a lower hierarchy level, and hence, they continue being part of the project).

The pattern described in the previous paragraph is similar to the one illustrated in Figure 9.7 (b), where the packages *antlr* and *osxadapter* from *JabRef* are not part of the system structure in the latest revision that is under analysis. This contrasts with the structure of *JFreeChart* that is shown in Figure 9.7 (c) which does not exhibit changes after some years of evolution.

In summary, the lifelines of software items are depicted by colors and the relationship between programmer's activities and the structures of the project are established.

The aim of the *relative representation* is to permit the comparison of the activity carried out in packages and the volume of programmer's contribution, as the area assigned to a contribution is the same for all time units in the visualization. It can also be viewed as a *linear* representation of the statistics on contributions and collaboration shown by GT.

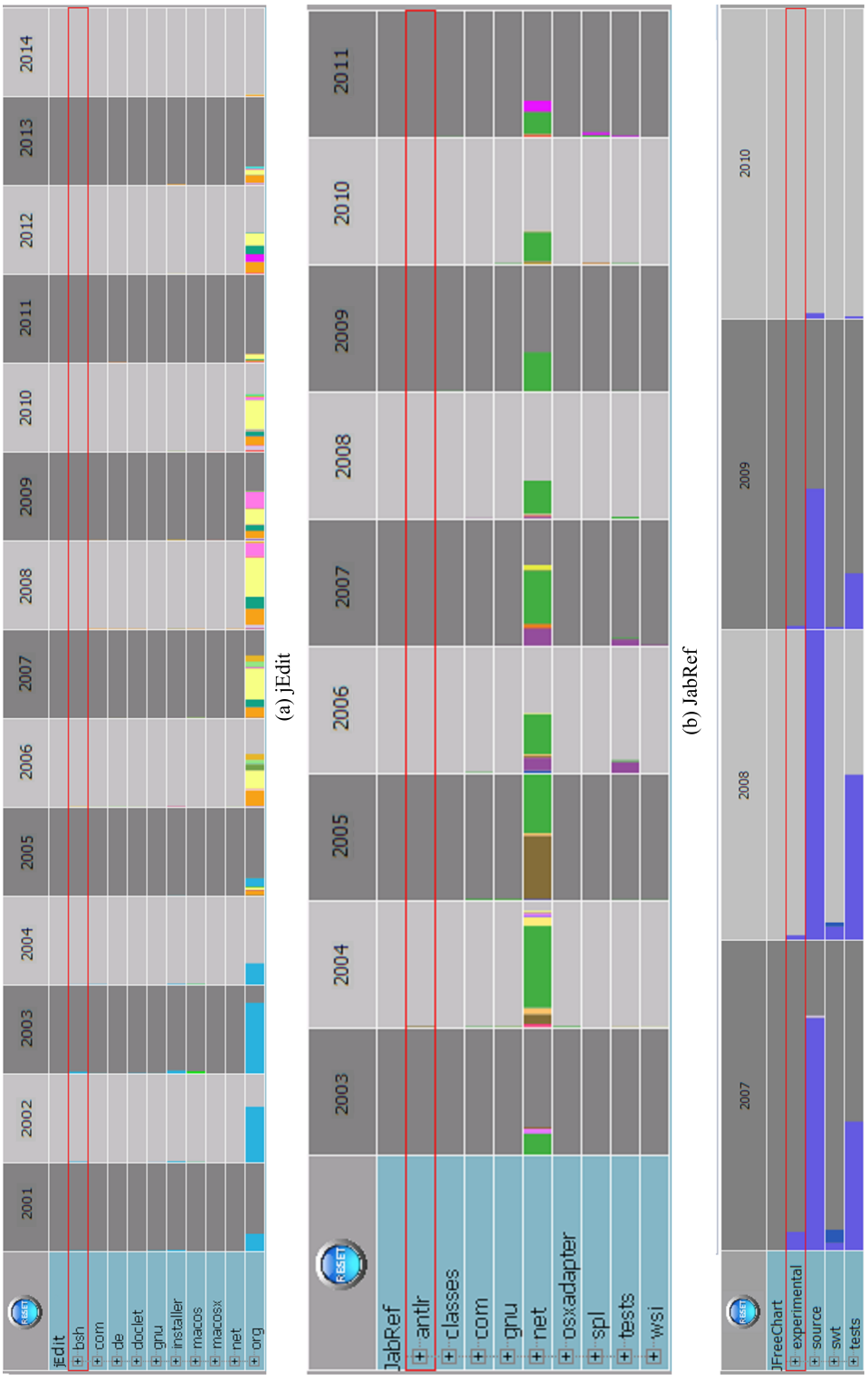
The *relative representation* supports to correlate with precision the time periods with programmer's contributions and aids to the identification of

packages that play a key role in the system, using as basis the concentration of high volumes of activity in their evolution. Therefore, the characterization of programmers contributions shows that years 2008 in *jEdit*, 2005 in *JabRef* and 2008 in *JFreeChart* are the years with a higher volume of activity in the corresponding project (see Figures 9.7 (b), 9.7 (b) and 9.7 (c)), and hence are used as the reference year for calculating the area of each individual revision. Moreover, figures 9.8 (a), 9.8 (b) and 9.8 (c) portray a pattern in which the activities carried out during the evolution of *jEdit*, *JabRef* and *JFreeChart* concentrate in one package: *org*, *net* and *source*, respectively. Accordingly, it should be highlighted that these packages contain several sub-packages and most of the classes in the system they are part of (this can be concluded after expanding and examining the content of the package and its sub-packages).

Contribution patterns could be observed in GT as well as in the *Gridmaster*, where colors are also used to show the volume of contributions made by programmers for one or more time units. Furthermore, the *Gridmaster* correlates contributions with the project structure down to the file level, providing details of the software items that have been changed by each programmer. The aim of this feature is to assist software project managers in the assignment of programming tasks to programmers, according to their previous experience based on the changes made to software items.

Accordingly, Figure 9.9 (a) shows a partial screenshot of *Gridmaster* for *jEdit* which depicts that *spestov* has made changes to all packages displayed in the figure through the year 2001 to 2005, which practically made him indispensable for error correction and maintenance during that period of time, being the only exception the year 2005 where other programmers (*e.g.*, *ezust* (orange) and *kpouer* (yellow)) started contributing to some of the packages shown. Thereafter, the contributions to packages have been split between two or more programmers, with a predominance of *ezust* during the years 2006 and 2008 and *kpouer* on 2007, and from 2008 tol 2012. It is relevant to highlight that after the year 2006 the packages in the figure that concentrate the activity of a higher number of programmers are *browser*, *bsh* and *buffer*, which could lead to the conclusion that this packages are of great interest to the community of collaborators.

In line with the previous analysis, Figure 9.9 (b) shows a screenshot of *JabRef* to depict that *mortenalver* (green) has made changes to all packages in the project, except *antlr*, which was created and changed only by *jzieren* (brown). Moreover, it shows that *coezbek* (dark purple) also contributed to most packages in the project during years 2006, 2007, 2008 and 2009, hence he could have substituted to *mortenalver* in case of an eventuality.



(a) jEdit (b) JabRef (c) JFreeChart

Figure 9.8: *Relative representation* of programmers' contributions for the projects *jEdit*, *JabRef* and *JFreeChart*.

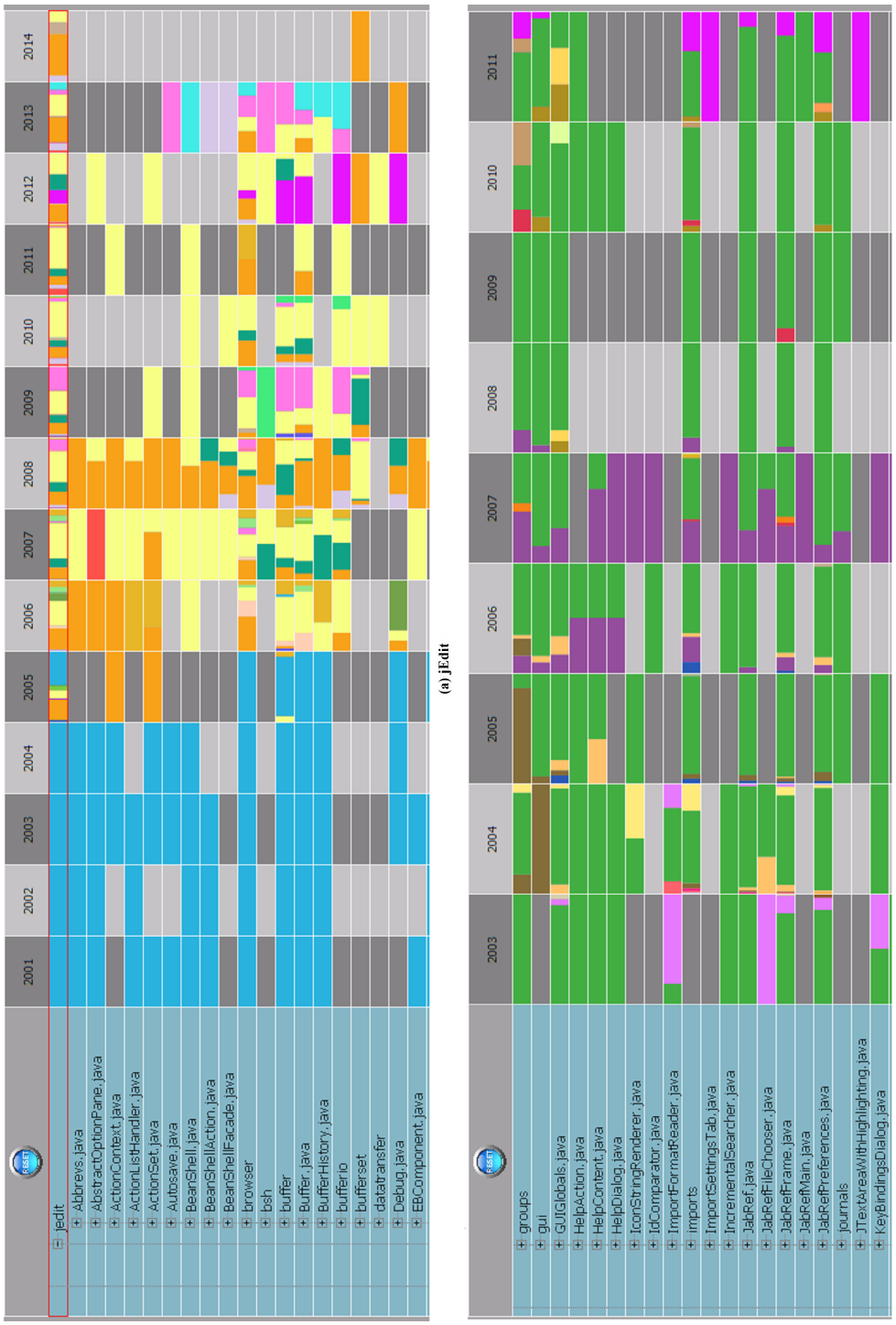


Figure 9.9: Correlation of programmers with packages for the projects *jEdit* and *JabRef*.

Figure 9.9 (b) also shows a similar pattern for the year 2011, in which *olly98* (light purple) has contributed significantly to some important packages and therefore, he could take over some tasks assigned to *mortenalver*. The packages modified by *olly98* are *net*, *spl* and *tests*, from which net is the core package of the system and tests is the package associated to system's tests cases. Hence, the assignment of tasks to programmers can be carried out taking into account not only the functionality of the first level elements of the project structure but also considering the lower level elements.

The GT also uses a *relative representation* to show the contributions of programmers precisely, so *Gridmaster* is an analogous visualization in that sense under this context. It allows details which are at different granularities of time to be examined and this means, in practice, that it is possible to select a particular year in order to analyze the distribution of monthly activities, or amplify a particular month in order to revise the levels of daily activity.

Figure 9.10 shows years 2006 and 2008 in expanded form so that changes in inheritance relationships and modifications of metrics can be observed, and their monthly distributions examined. Furthermore, the use of GT and *Gridmaster* can lead to similar conclusions when carrying out an analysis of the collaboration of programmers, as it can be grasped from the contributions of *spestov* (turquoise), who is the programmer with most activity in the period from 2001 to 2005.

The tree structure used by *Gridmaster* is a scalable representation based on a foldable tree to depict packages, within which are the archives, which in their turn contain classes, interfaces, enumerations and annotations (the four basic elements of Java). In this context classes are denoted by a small square symbol, while interfaces are depicted by a small circle, enumerations are represented by a triangle and annotations are drawn using a small star symbol. When the foldable structure expands it is possible to see the contents of the files and details on the establishment and termination of inheritance and interface implementation relationships, as well as metrics.

Figure 9.10 points out the associated software items that intervene in inheritance and interface implementation relationships by means of red colored legends that indicate their locations (*Java* Application Programming Interface (API), current project or external API library). Furthermore, the establishment of these relationships is depicted by a green oval while its termination is represented by a red oval. The algorithm used to determine whether an associated software item is included in the current project, *Java* or an external API library is shown by Algorithm 9.3.1.

Concerning metrics, these are represented in the rows that corresponds to software items (the rows associated to packages, inheritance and interface relationships are not used for this purpose) using bar charts with the aim of

highlighting changes (see Figure 9.10). The values of metrics are displayed when a commit of a software item has been carried out, regardless of whether the metric value has changed or remained the same. Similar to the representation of programmer contributions, metrics are represented using absolute and relative areas. *Absolute representation* takes into account the software item with the highest metric value to calculate the chart height. However, the *relative representation* only takes into consideration the highest metric value associated with the software item. The design of the visualization only permits the representation of one metric at a time so the user must select the metric which is sought to be visualized.

Algorithm 9.3.1: EXTRACTION OF SOFTWARE ITEM DEPENDENCY.(*repos*)

```
repository ← repos
while n < repository.lastRevision()
do { revisionClasses ← DataExtractor.getRevisionItems(n)
     for each item ∈ revisionItem
     do { if item.hasParent()
          then { parent ← item.getParent()
                 self ← Analyzer.isParentOnProj(parent)
                 if project
                   then location ← Project
                   else java ← Analyzer.isParentOnJava(parent)
                 if java
                   then location ← Java
                   else location ← Library
                 package ← Analyzer.getClassPackage(parent)
                 extends(x) = ( parent  location  package )
                 vector.add(extends)
          do { if item.hasImplements()
               then { interfaces ← item.getInterfaces()
                      for each interface ∈ interfaces
                      do { self ← Analyzer.isIntOnProj(interface)
                           if project
                             then location ← Project
                             else java ← Analyzer.isIntOnJava(interface)
                           if java
                             then location ← Java
                             else location ← Library
                           package ← Analyzer.getIntPackage(interface)
                           implements(x) = ( parent  location  package )
                           vector.add(implements)
return (vector)
```

Gridmaster represents the inheritance relationship both for parents to children as children to parents. Thus when selecting an item, the software elements which it has inherited over time can be observed as well as those software items which have been inherited from the item selected.

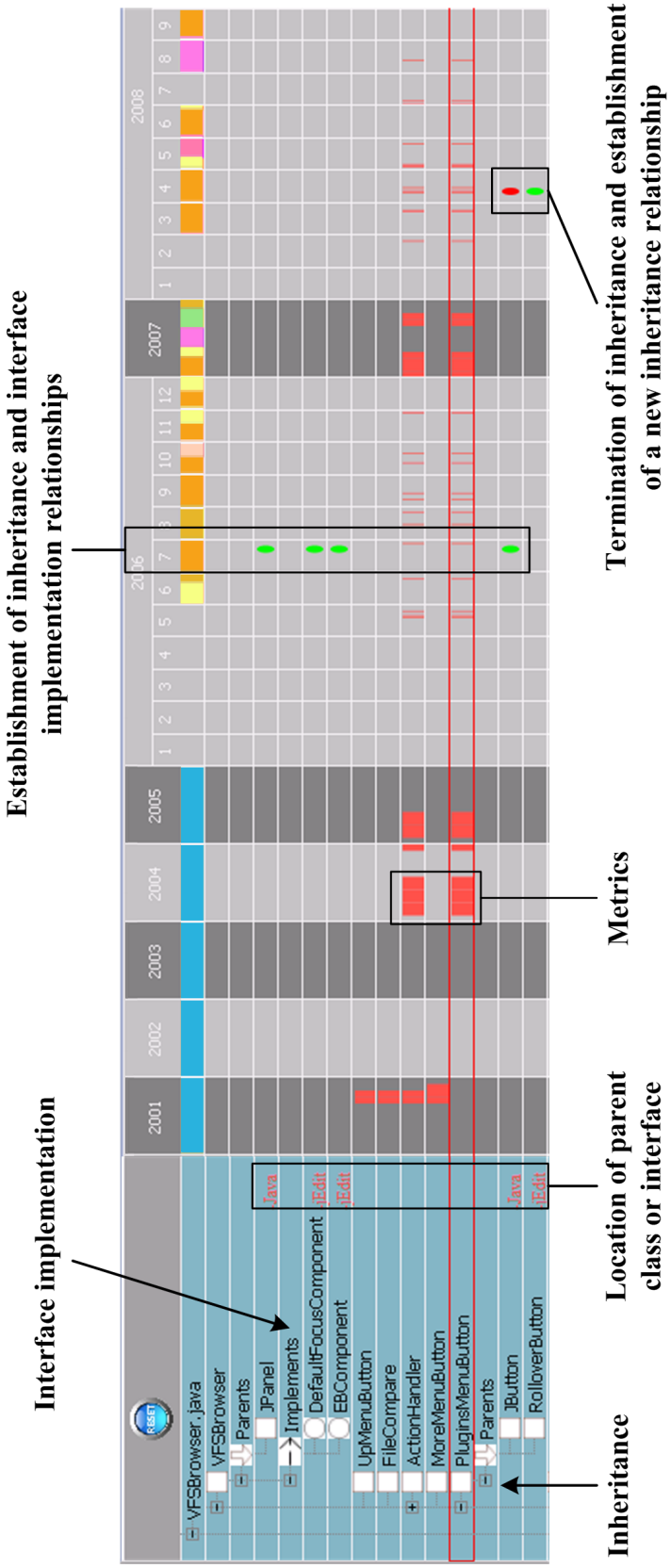


Figure 9.10: Inheritance and interface implementation relationships, including expanded years and metric values, for the file *VFSBrowser.java* in *jEdit*.

Accordingly, inheritance is represented by *Gridmaster* in the following form : Parents <– Selected software item –> Children. Figure 9.11 depicts the relationships that *AbstractOptionPane* has established with other software items: it inherits from *JPanel*, implements the interface *OptionsPane* and has 8 subclasses.

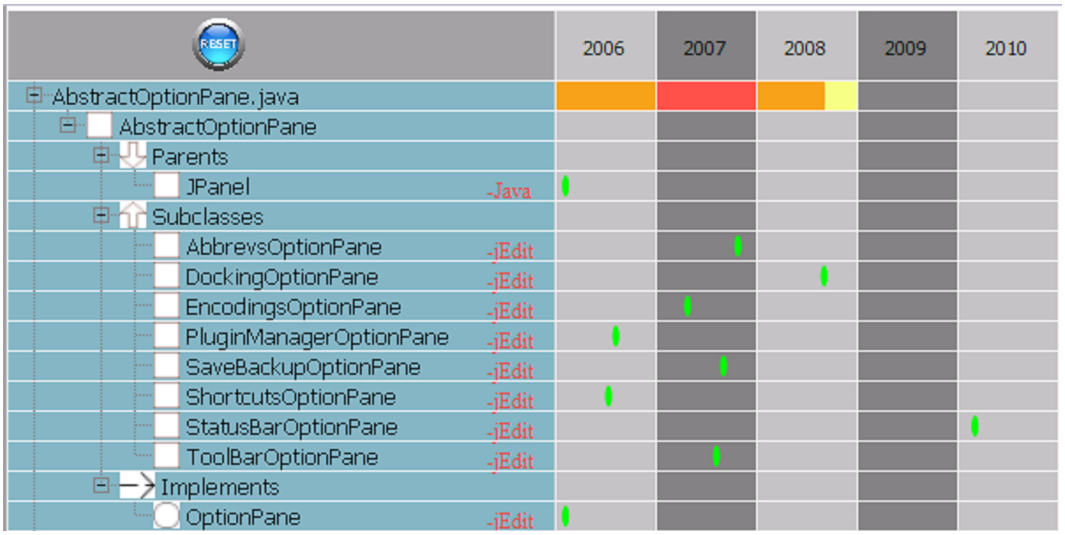


Figure 9.11: Software item inheritance and interface implementation relationships for the file *AbstractOptionPane.java* in *jEdit*.

The representation of interface implementation is carried out in a similar manner to the representation of inheritance relationships and depicts the interfaces that are implemented by a particular class (from a class perspective) and which classes implement a specific interface (from an interface perspective). This feature is shown in Figure 9.12, which highlights with a red oval the interface *Compare* and depicts that this interface inherits from the interface *Comparator* and is implemented by the classes *MenuItemCompare* and *StringCompare* (highlighted by blue circles).

The use of *Gridmaster* during the analysis of *jEdit* and *JabRef* permitted to unveil that a common pattern in these projects is that a large number of files contain several software items. Moreover, it also showed that the intensive use of inheritance and interface implementation is also a common pattern. However, these patterns are not common in *JFreeChart* as most files in this project do not contain more than one software item (*e.g.*, JFreeChart.java is one of the few files that contains two classes) and inheritance is rarely used in comparison with the other two projects (*jEdit* and *JabRef*).

Figure 9.13 shows that the file *OperatingSystem.java* (located in the package *installer* of *jEdit*) contains 10 classes and several inheritance

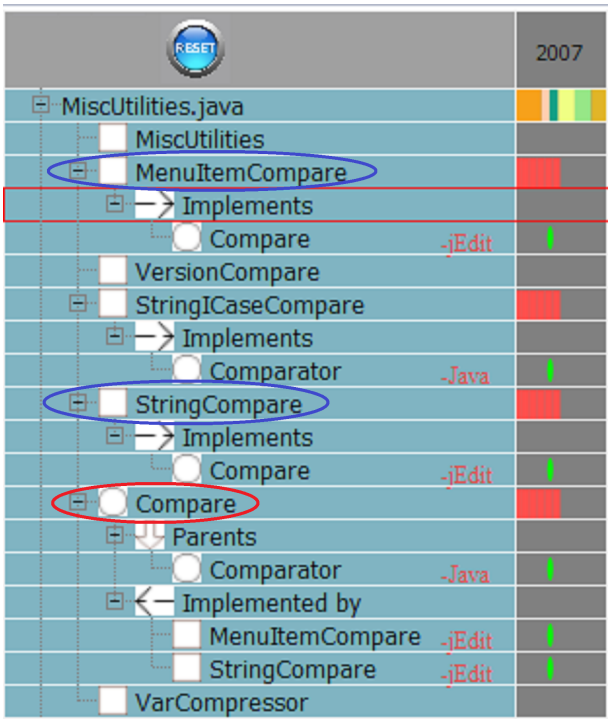


Figure 9.12: Implementation relationships for the interface *Comparator* of the project *jEdit*.

relationships (none of the classes is implementing an interface). Moreover, it depicts a hierarchy tree in which *OperatingSystem* is a class with four sub-classes (*HalfAnOs*, *Unix*, *VMS* and *Windows*) from which other classes can inherit from, such as the case of the class *MacOS* that inherits from the class *Unix*. Furthermore, Figure 9.13 shows that most classes in the file *OperatingSystem.java* established inheritance relationships during the year 2003, and also depicts that metrics for the classes *Unix*, *Windows* and *OSTask* had some slight variations during the same year.

Figures 9.14 and 9.15 demonstrate similar patterns on the evolution of the structure of the files *HelpViewer.java* (*jEdit*) and *BasePanel.java* (*JabRef*), in terms of the inheritance and interface implementation relationships of the software items they contain (the analysis of the evolution of these files with the use of RT is discussed in chapter 10).

HelpViewer.java has evolved since the year 2002 up to *April*, 2014 (a partial view of this lifeline is shown in Figure 9.14) and contains 6 classes (*HelpViewer, LinkHandler, KeyHandler, ActionHandler, PropertyChangeHandler* and *AsyncHTMLEditorKit*) that have established inheritance or interface implementation relationships. Accordingly, 3 of these classes have established inheritance relationships with other classes whereas 4 of them have implemented interfaces. In addition, the value of the NOM

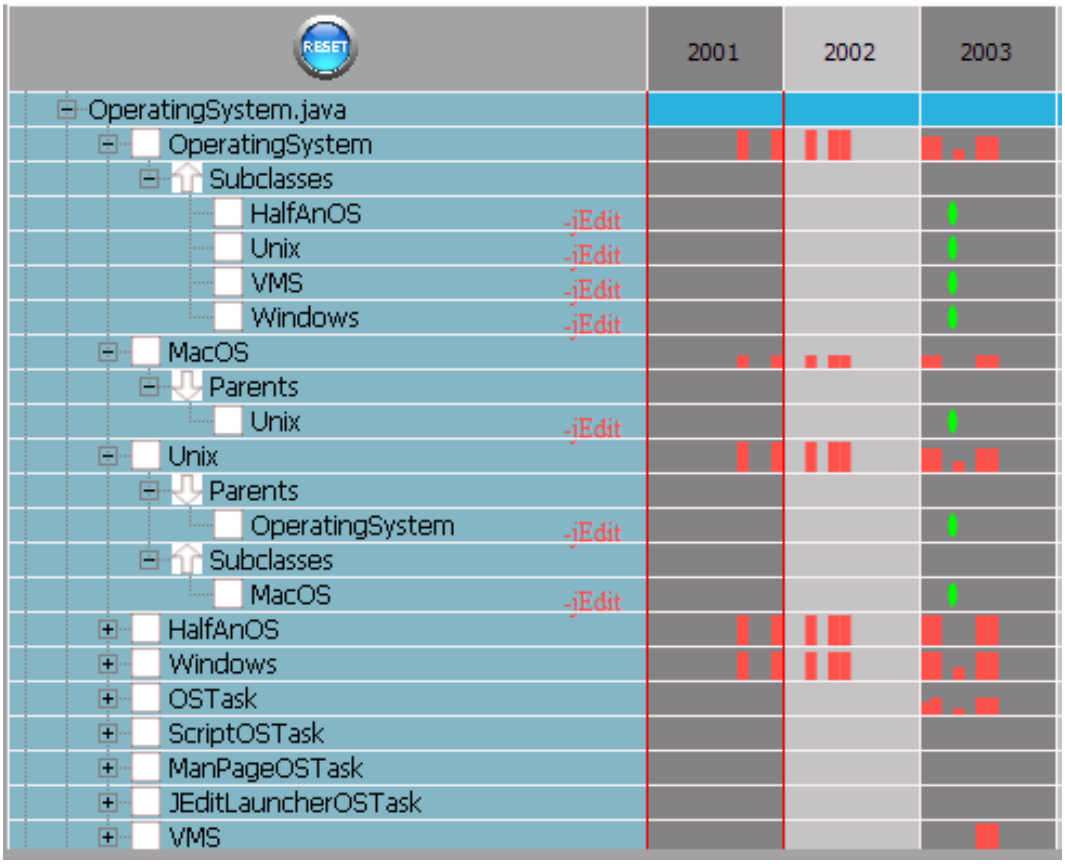


Figure 9.13: *OperatingSystem.java*: file that contains 10 classes, which makes intensive use of inheritance.

metric associated to 5 out of the 6 classes have not changed with time, with the exception of the values for the class *HelpViewer* (see years 2003, 2005 and 2007 in Figure 9.14).

An important detail to be noted is that there were an intensive activity concerning the establishment or change of inheritance an interface relationships during the years 2006, 2007 and 2008 in *jEdit*, as it can be observed in Figures 9.10, 9.11, 9.12, 9.13 and 9.14. Therefore, this could indicate that a restructuring had undergone during those years, and particularly during 2007 that is the year that shows more activity on this regard.

Figure 9.15 shows that the file *BasePanel.java* (located in the package *net.sf.jabref*) contains 6 classes, from which 5 of them (*BasePanel, UndoAction, BaseAction, RedoAction and LocalEditsListener*) have an inheritance or interface implementation relationship with other classes.

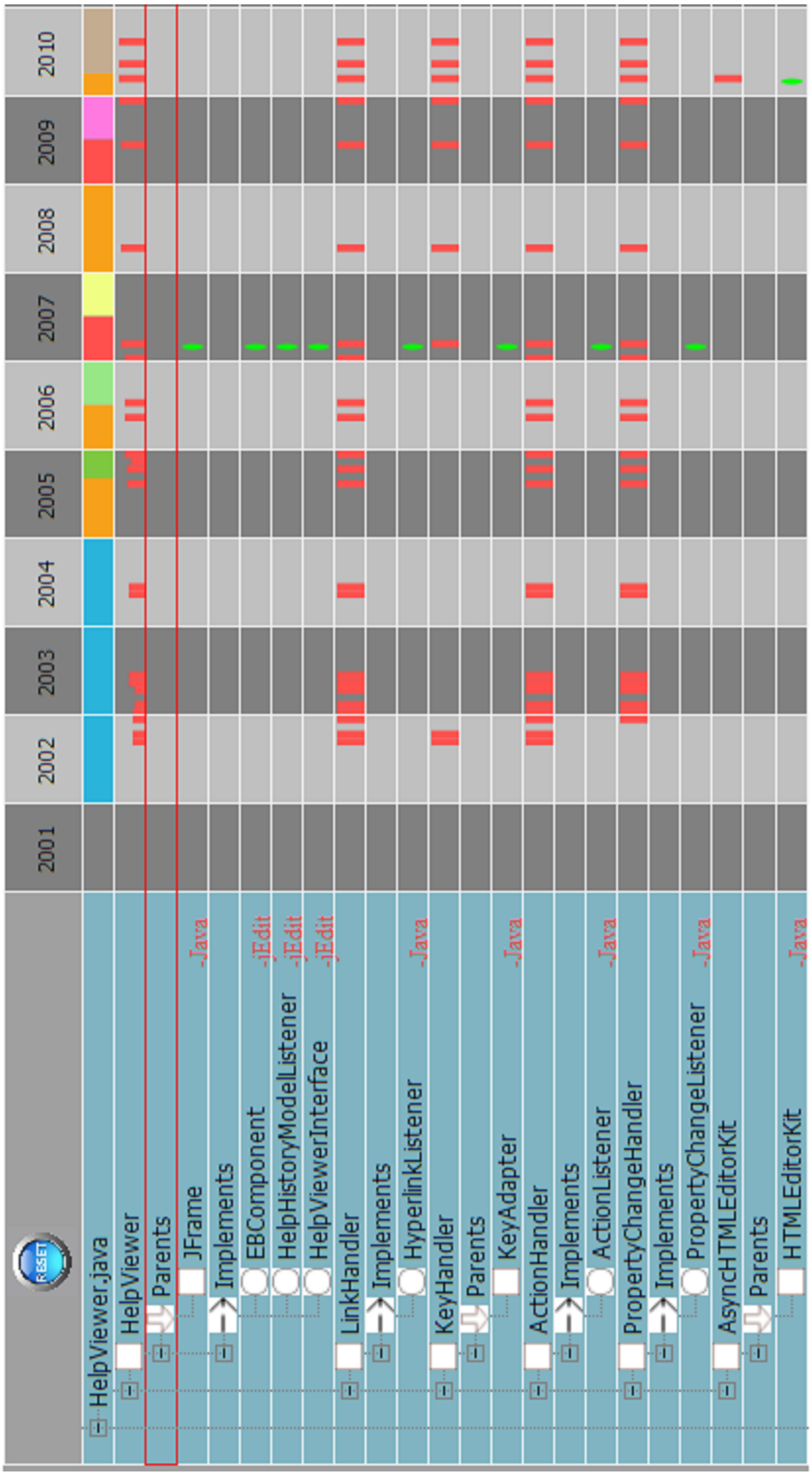


Figure 9.14: Relationships between software items of *HelpViewer.java* (*jEdit*).

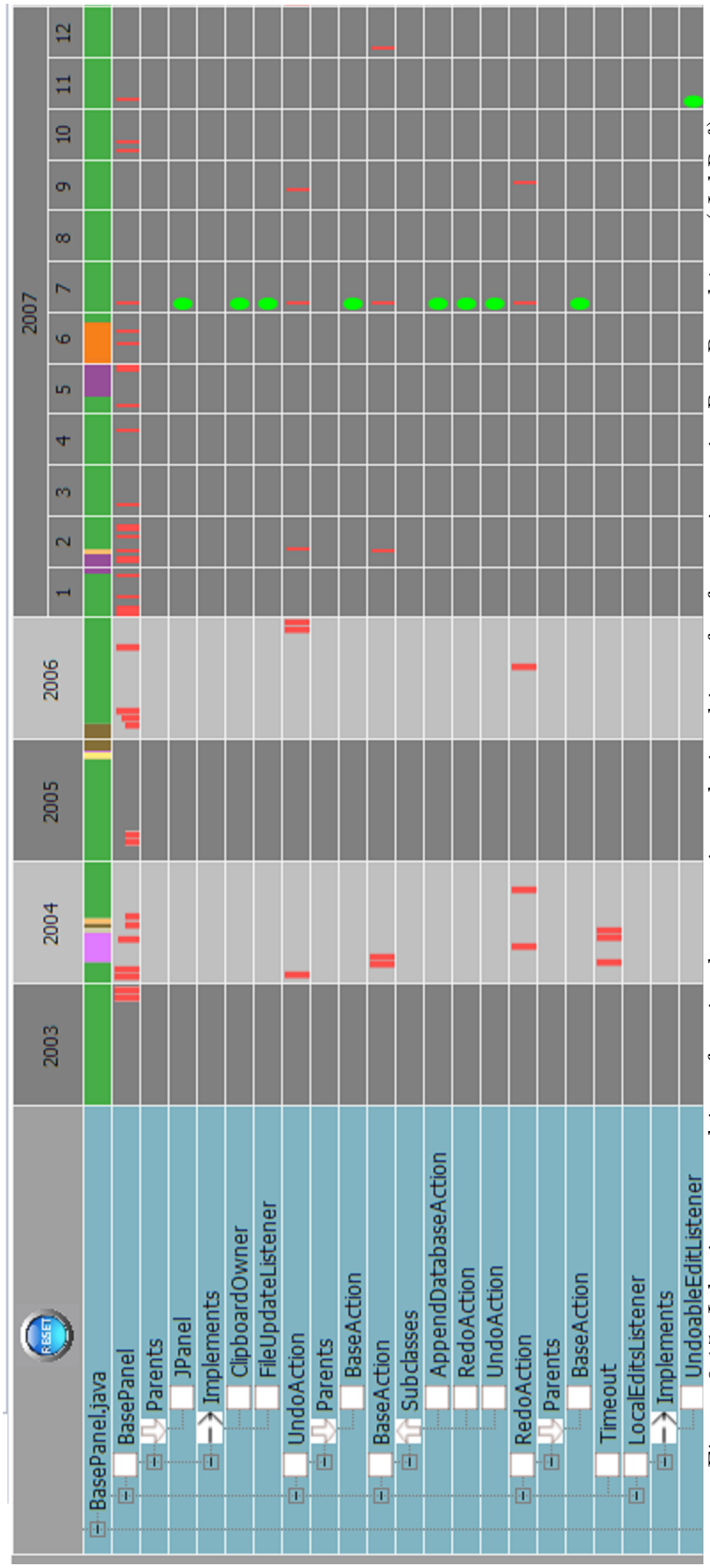


Figure 9.15: Inheritance and interface implementation relationships of software items in *BasePanel.java* (*JabRef*).

The evolution of *BasePanel.java* has extended from 2003 to 2011 (the last year taken into account in this study). Accordingly, the classes *RedoAction* and *UndoAction* inherit from *BaseAction*, whereas the class *BasePanel* inherits from *JPanel* and implements the interfaces *ClipboardOwner* and *FileUpdateListener*; and in addition, the class *LocalEditListener* implements the interface *UndoableEditListener*. Other important detail depicted in Figure 9.15 is that most relationships established by the software items mentioned above were performed in *July*, 2007.

Finally, it is worth to highlight that *Gridmaster* makes use of several interaction techniques that include the possibilities of zoom-in and zoom-out, fisheye distortion, and the capacity to filter out nodes from the structure. In addition, it supports year selection from the timeline for depicting data according to associated months and the user has the possibility to choose how the metrics and the programmers are represented by selecting between relative and absolute value representations.

9.3.3 Socio-Technical Graph: Visual Representation of the Collaboration and Relationships between Programmers

STG is a complementary view (see Figure 9.16) that is based on a graph representation and displayed when a time unit (*e.g.*, a year or month) is selected from *Gridmaster*. This visualization is aimed at depicting the contributions of programmers (in terms of number of files and revisions committed) and the relationship among them, and built upon the software items they have changed in common. Accordingly, nodes depict the contributions of programmers and colors are associated to the username of programmers, whereas edges represent the collaboration relationships that have been established between programmers from the changes they have made to software items.

The size of nodes in STG conveys the number of contributions made by programmers which are determined by the number of files that have been modified in each commit (the computation of node weights differs from the one made by Jermakovics *et al.* [Jermakovics 2011] that only takes into account the number of commits). Therefore, the contributions weight, w, is the sum of the number of files committed per programmer and per commit, and is calculated as following:

$\sum_{i=1}^{c} F_i$, where c is the number of commits made by the programmer and F_i is the number of files for the commit i.

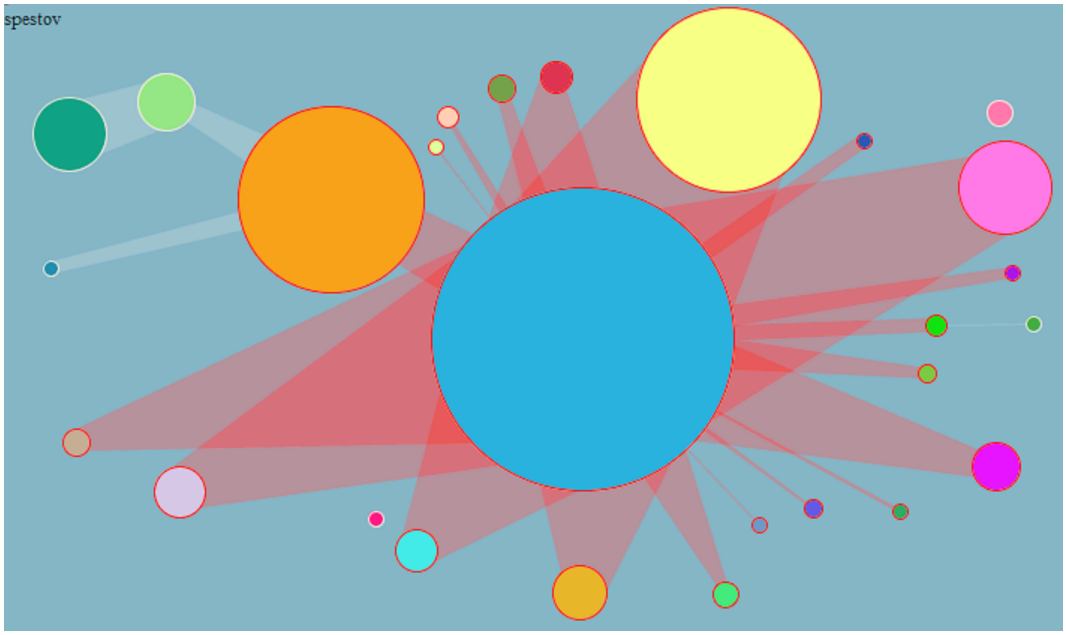


Figure 9.16: STG showing the contributions and relationships between programmers based on the software items they have modified in common.

The thickness of edges represents the number of software items that the associated programmers have changed in common. Thus, the strength of the collaboration relationship between two programmers can be deduced from the thickness of edges. Therefore, this visualization is useful in scenarios where programmer's tasks need to be redistributed due to an unexpected situation or organizational change, where hypothetically, programming tasks performed by *programmer A* may be eventually taken over by *Programmer B*.

The design described in the previous paragraphs is demonstrated by Figure 9.16 (a screenshot of STG) which depicts the contributions of programmers and their collaboration relationships between the years 2001 to 2014 for the project *jEdit*. This figure allows to note that for the given time period the programmer with more contributions is *spestov* (turquoise), followed by *ezust* (orange), *k_satoda* (light yellow), and *daleanson* (light purple). Moreover, it also permits to deduce that *spestov* has intervene in the modification of most software items in *jEdit* as it is the programmer that has more connections to other programmers (highlighted by the red colored edges). Furthermore, the precise identification of the software items that *spestov* modified in common with other programmers could be achieved through the inspection of the correlations between programmers and software items in *Gridmaster*. Although the participation of *spestov* was very intensive during the first years of the evolution of *jEdit* (see sections 9.3.1 and 9.3.2) his contribution to this project stopped in 2005.

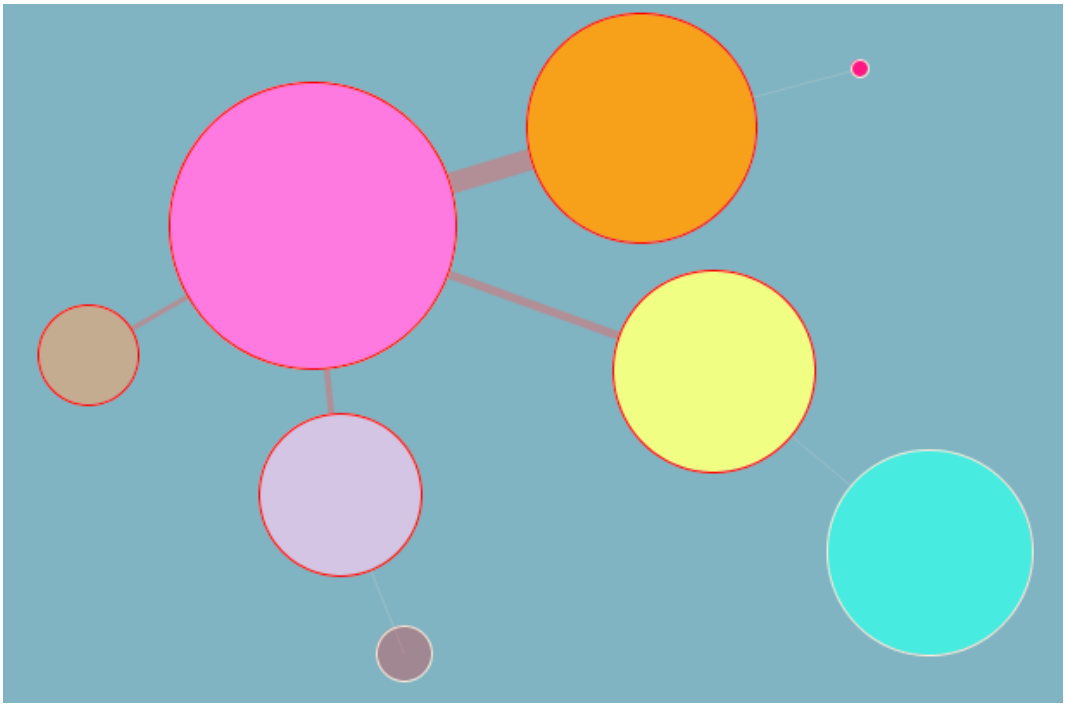


Figure 9.17: Screenshot of STG for *jEdit* and the year 2013.

A relevant aspect to be noted is that the relationships between programmers were minimal during 2013 (see Figure 9.17), where *shlomy* (purple) concentrated most of the relationships to *ezust* (orange), *kpouer* (yellow), *daleanson* (light purple) and *Vampire0* (brown), some of which has been contributing to *jEdit* for several years. Furthermore, *thomasmey* (light green), who made a relative large number of contributions during 2013, established a weak relationship with *kpouer* according to the software items that have changed in common (this can be further investigated in *Gridmaster* after expanding the packages *org–>gjt–>sp–>jedit* and *org–>gjt–>sp–>util*).

The situation described in the previous paragraph is accentuated during the first 4 months of 2014 (January through April), as it is depicted by Figure 9.18. 3 of the 4 programmers that have contributed to *jEdit* during 2014 do not have connections with other programmers and the only relationship that has been established (between *ezust* and *daleanson*) is too weak to be considered relevant in terms of collaboration.

The use of STG is further demonstrated by Figure 9.19 in relation to *JabRef*, where the larger node represents *mortenalver* (green) the programmer who has made more contributions to the project, whereas the other programmers with also important contributions are *coezbek* (dark purple), *kojiyokota* (dark orange), *mspiegel* (yellow), *jzieren* (brown) and *olly98* (light purple) and *nbatada* (pink).

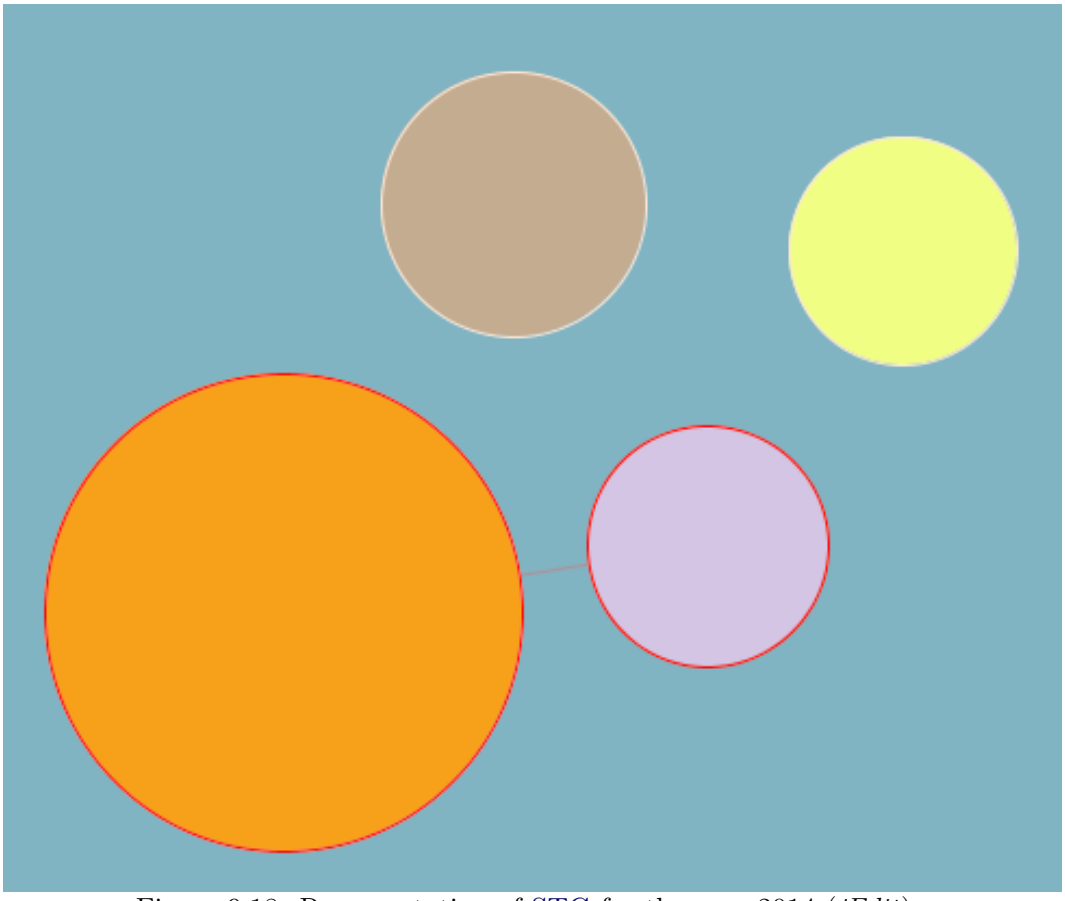


Figure 9.18: Representation of STG for the year 2014 (*jEdit*).

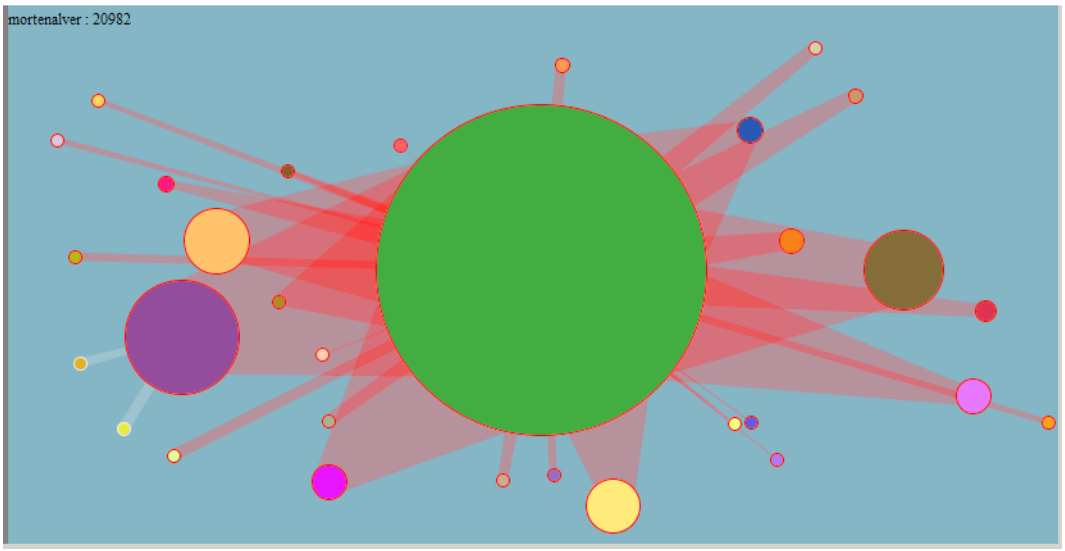


Figure 9.19: STG for the overall evolution of *JabRef*.

The advantage of STG is its capability to represent in a small screen area

several years of data, as shown in Figure 9.19 where 9 years of information has been summarized. While this visualization represents the complete development and maintenance period, it is valuable for understanding the contributions made to the project and the interactions between programmers. Hence, the software project manager requires a representation of a recent and shorter time period for deciding on programmers' substitutions and the collaboration relationships between programmers. Demonstrating this, Figure 9.20 is based on the year 2011 (the last year that the *JabRef* team actively used *Subversion* as their SCM tool) and depicts that *mortenalver* (green) was the programmer with most contributions to the project during that year. So, according to the relationships between the programmers, *kojiyokota* (dark orange) was the most suitable programmer for substituting *olly98* (light purple) in most tasks (the relationships between the other programmers is not even represented because it is extremely weak).

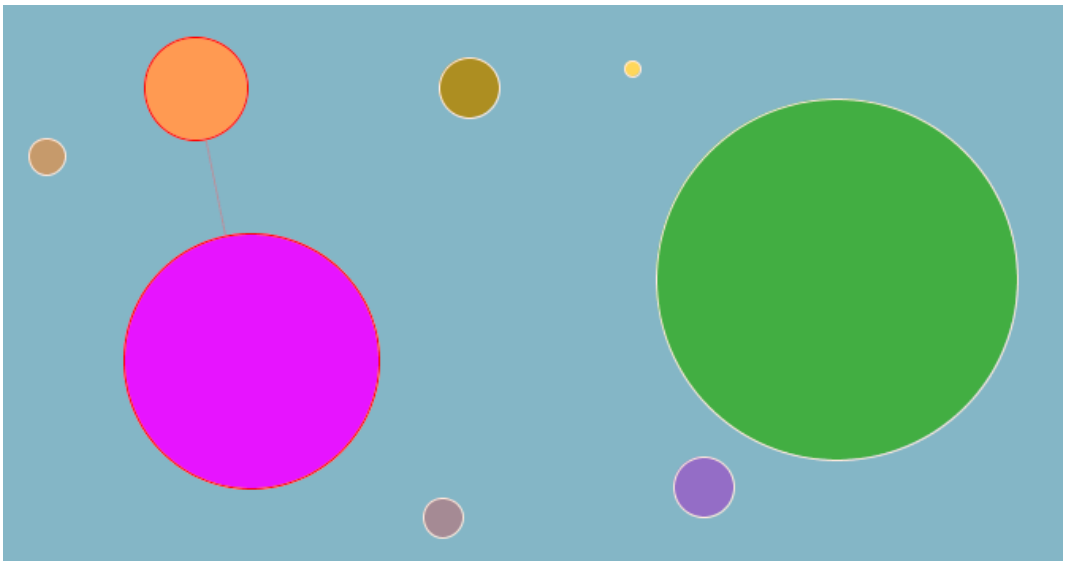


Figure 9.20: Socio-techcnical relationships between programmers for the year 2011 (*JabRef*).

9.4 Discussion and Conclusions

This section focuses on knowledge discovery from socio-technical relationships and looks to answer questions regarding the contributions made by programmers (who has led the development of the software project or has contributed the most?), the contribution patterns of programmers (why has the programmer made so many contributions in such a short time?), the relationship among software items and programmers (who has modified a

given software item?), and the software items programmers have changed in common (which software items have programmers changed in common?).

According to the previous sections in this chapter, the visualizations that conform the VKESE module of *Maleku* convey important details on the socio-technical relationships that are derived from the interactions between programmers and software items as well as architectural patterns. Therefore, the representation of the socio-technical relationships included views at the system and software item levels, and took into account the representation of the following:

1. Statistics on the number of revisions for different granularity levels of time (*e.g.*, years, months, days and hours) and programmers.
2. Socio-technical relationships between programmers and software items.
3. Relationships between programmers according to the software items that they have changed in common.

The representation of the statistics that is carried out by the GT, *Gridmaster* and STG offers a different view of the data. Although some of the functionality of GT and *Gridmaster* is similar as both depict details on the number of revisions for different granularity levels of time and the contributions made by programmers, *Gridmaster* correlates the number of revisions and programmers with software items at the package and file levels. This correlation permits to determine which software items have been changed by each programmer. Hence, it could assists software project managers in the assignment of programming tasks and staff substitutions (due to unexpected situations or staff turnover), according to the previous experience of developers changing specific software items.

Accordingly, STG is a complementary view that allows to get insight on the collaboration between programmers during the development process for a time period, and not on the basis of the software items that programmers have changed in common, which also contributes to the assignment of tasks.

The analysis of *jEdit*, *JabRef* and *JFreeChart* with GT and *Gridmaster* permits to observe that a leading programmer exists during a time interval or the complete development of the project, but also that such leading function is alternated between programmers. Therefore, an hypothesis that arose on this regard is that open source projects are led at the beginning of the development, and sometimes during a large period of time, by the programmer that took the initiative to start the development of a project, but then the leading function is assumed or alternated with other programmers. However, the scope of this research does not take into account a further exploration on this hypothesis.

Section 9.3.2 discussed the features of *Gridmaster* and showed that the use of this visualization allows to uncover the following structural details:

1. The active elements of the project structure based on the contribution patterns and the structure shown by *Gridmaster*.
2. Key packages and software items according to contribution patterns.
3. Classes contained by files.
4. The establishment of inheritance and interface implementation relationships in time.

The depiction of the lifelines of software items permits to convey details of the elements that are under active development. This feature portrayed the current structure of the system and offered clues regarding which software items could require attention to understand the latest changes made to the system. Moreover, it also allowed to identify the key packages and software items of the systems, which could be of great value to draw the attention of programmers and project managers into the more active structural spots of the project. Therefore, this characteristic can help to project managers and programmers to concentrate in the examination of the elements that could constitute the core of the system.

The use of *Gridmaster* also allowed to corroborate its capabilities for depicting details on the software items that are contained by files, as well as information concerned with the evolution of their inheritance and interface implementation relationships. This information is useful for programmers in the understanding of systems and particularly of their structure when they are new to a project.

Chapter 10

Revision Tree: A Case Study on PlasticSCM

Como era costumbre en los momentos de incertidumbre y dificultad, apareció Cucho moviendo la cola y lo arrastró por el pantalón a una gran sala llena de diseños de relojes y piezas para fabricarlos. Güindy se quedó perplejo por un rato, hizo un giro de bailarina para ver en derredor, se sentó en el piso, se volvió a poner de pie, brincó, lanzó un grito ahogado, hasta que finalmente comenzó a revisar documentos, dibujar y juntar piezas de relojes, de las tantas que habían en el lugar. — El viaje de Güindy, A.González

Contents

10.1 Introduction

The origin of RT is rooted in a partnership between the *VisUsal* and *GRIAL* [García-Peñalvo 2012b] research groups of the University of Salamanca (Spain) and *Codice Software*[1]. These research groups have conducted several research projects concerned with IV, VA and HCI in the past few years.

[1] *Codice Software* is a software company that was founded in year 2005 in *Valladolid* (Spain) to design and develop SCM tools.

Therefore, *VisUsal* and *GRIAL* were engaged by *Codice Software* to design a visualization tool based for the representation of the evolution of source code files. Accordingly, these research groups determined that a visualization tool for this purpose should offer focus + context views and provide the following information details regarding the evolution of source code files:

1. Duration of the evolution of a source code file.
2. Number and name of programmers that are participated in coding the software items within the file.
3. Name of the programmer with more contributions to the evolution of the source code file.
4. Number and details (*e.g.*, id, date and time of creation) of the baselines and revisions that constitute the evolution process of the file.
5. Association of the baselines and revisions to branches.
6. Details on the merging of revisions.
7. Patterns of activity during a particular time period.
8. Comparison of activities associated to multiple time periods.

Thereafter, an analysis of Version Tree 3D (VT3D) (a *3D* visualization that was included at that time by *PlasticSCM*), VRCS [Koike 1993, Koike 1997] and *Revision Graph* (*Perforce*) [PerforceSoftware 2014] was carried out to identify additional requirements and visual features for RT (section 10.2).

One the design of RT was performed, its development took two different paths. On the one hand, *VisUsal* and *GRIAL* improved the design (section C.3.4), developed the tool and tested it using source code files from open source projects to evaluate its usefulness in the comprehension of their evolution (section 10.4). In the other hand, *Codice Software* developed its own version and incorporated it into *PlasticSCM*. Thus, the original design that is presented in this chapter was partially adopted by *Codice Software* and named it *Version Tree*.

Consequently, the final design and the tests that are presented in this chapter were carried out with independence of the initial relationship between the *VisUsal* and *GRIAL* research groups and *Codice Software*.

Finally, it is worth to mention that section 10.5 presents the discussion and conclusions of this chapter.

10.2 Analysis of Existing Visualization Tools

The visualization tool included in the first version of *PlasticSCM* was named as VT3D. According to *Codice Software* its development was performed in *3D* because they considered that the use of three dimensions could make it look

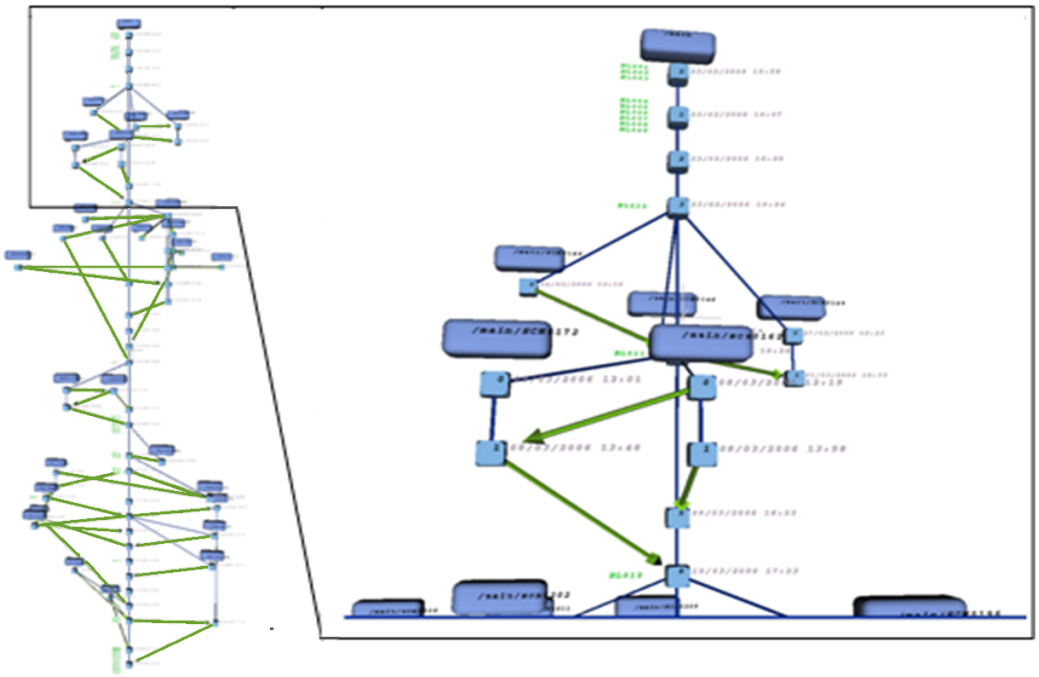


Figure 10.1: PlasticSCM: Version Tree 3D [Therón 2007, Therón 2007].

more striking and interesting, and moreover because its design could allow to understand, in an easy and intuitive manner, the way how a given file has evolved.

VT3D used a line to represent the main line of the development and green arrows to show the merge of revisions associated to a software item. Along with the nodes, it uses labels to indicate information about the baselines and revisions. To illustrate the application of VT3D to the evolution of a file, Figure 10.1 shows two views: the view on the left hand side allows to observe the complete depiction of the evolution of a file, while the view on the right side of the figure shows a partial view of the evolution of the same file using a close up.

However, after a careful analysis it was determined that VT3D has a number of drawbacks:

1. It is a static representation: the user can only change the point of view or choose how far is looking at it (i.e, it was only possible to turn around the tree and zoom into a region to get closer to a node or area).
2. The use of the zoom in functionality increases the size of nodes and it is harder to manipulate the tree, and the context is lost because the visualization lacked a focus + context view.
3. If the user zooms out the visualization, the tree becomes a *3D* shape with no special meaning in the context of SCM.

4. After zooming in, it is difficult to see all the information represented due to occlusion; the front nodes hide the other nodes representing revisions.

Based on the above and the tests performed with this visualization, it was possible to corroborate that users could get disoriented when using the zoom functionalities, and that the interaction possibilities offered by the tool do not allow to obtain information by means of an intuitive approach.

According to the information provided by *Codice Software*, the opinions of customers were divided after some time of having released the first version of VT3D: some of them considered it an attractive tool, whereas it was not attractive to the others. The main objections of customers were the complexity to use and understand evolution details with the use of VT3D (this criteria was different depending on the industry sector of the user).

Similarly, VRCS (illustrated in Figure 10.2), a *3D* visualization comparable to VT3D, try to represent the evolution of source code files. However, whereas VT3D is oriented towards the visualization of the evolution of a single file using a tree structure, the objective of VRCS is to depict the evolution of all the software items within a system by means of a graph structure.

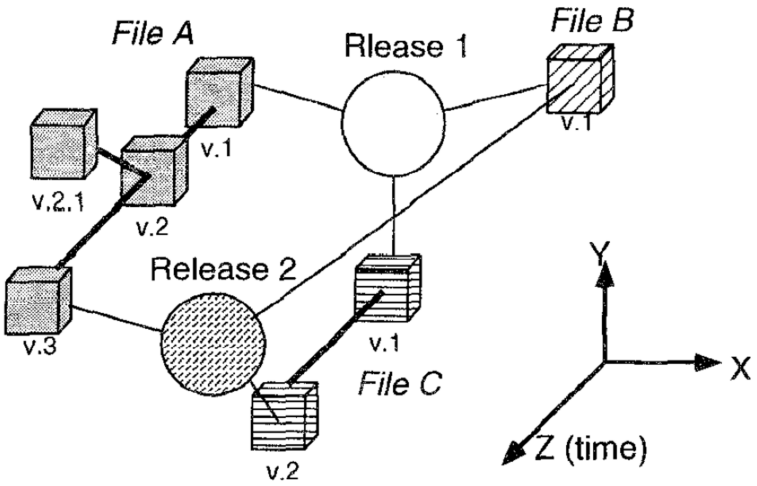


Figure 10.2: Visualization of the evolution of software items with VRCS [Koike 1997].

The main concerns with VRCS [Koike 1993, Koike 1997] are the lack of a focus + context view, the navigation possibilities through the structure, how it can behave with the presentation of complex systems due to the high processor and memory demands of *3D* visualizations and the common occlusion problems in this type of representations. The visualization of large revision histories for one software item using *3D* version trees has some

limitations, being scalability one of the most important. Therefore, the visualization of large repositories with many software items containing lots of baselines and revisions would result in a very large and hard to navigate visualization, which probably would not provide, within a short time, the information required by the user.

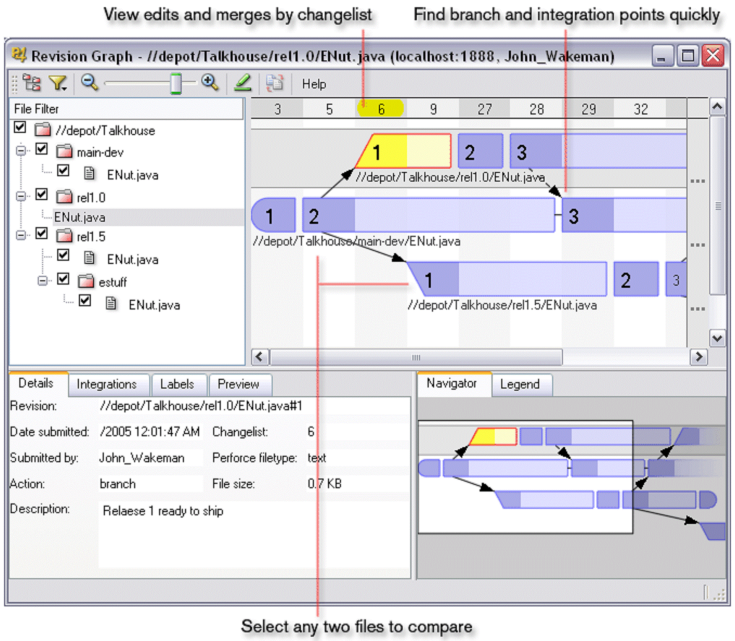


Figure 10.3: Perforce: Visualization of the evolution of a software item with Revision Graph [PerforceSoftware 2014].

After this comparison, it is important to highlight that a large number of SCM such as *Tortoise* and *Subversive* include *2D* visualizations to represent the evolution of software items. However, the design of *Revision Graph* (see Figure 10.3), the visualization offered by *Perforce* [PerforceSoftware 2014], offers an attractive overview + detail approach that shows information about branching and merging as well as details on the date and time of revisions when clicking over the nodes and reviewing the information on the Details tab on the left panel.

However, *Revision Graph* does not provide information about the programmers contributing to the development of the software item, how long the developers have been working on the item, nor in regards to periods without activity. Furthermore, it is not possible to compare two baselines or see the timeline at a first sight. In conclusion, this visualization is static and does not offer interaction possibilities.

Consequently, table 10.1 compares the results of the analysis carried out to the features of VT3D, VRCS and the *Revision Graph* of *Perforce* using as

base the requirements that were portrayed in section 10.1.

Table 10.1: Comparison of visualization tools for the evolution of software items.

Questions	Version Tree 3D	VRCS	Revision Graph
Does the visualization provide a overview + detail view?			x
How many developers are participating in the development of the software item?			
Who are the developers contributing to the evolution?			
Who is the programmer with more contributions to the evolution of the item?			
How many baselines constitute the whole evolution process?	x	x	
Does the tool offer information about dates and times of the creation of baselines and revisions?			x
Is there a revision without been merged after a long time?			x
How long has been the development of the item?	x	x	x
Which baseline has more branches and revisions?			
Which branch has more modification activity?			x
Which is the period of time that does not show any activity?			
Is there a period when the item was stable and then suddenly started having a lot of activity?			
Is it possible to compare baseline activity?			

10.3 Design of Revision Tree

The design of RT came forth as a result of the requirements presented in section 10.1, the analysis of VT3D, VRCS [Koike 1993, Koike 1997] and *Revision Graph* [PerforceSoftware 2014] and following the study of those characteristics desirable in this type of visualization tools [Therón 2007].

RT combines a grid, a timeline and a tree structure to convey the evolution details of a software item, as it can be observed in Figure 10.4 and table 10.2.

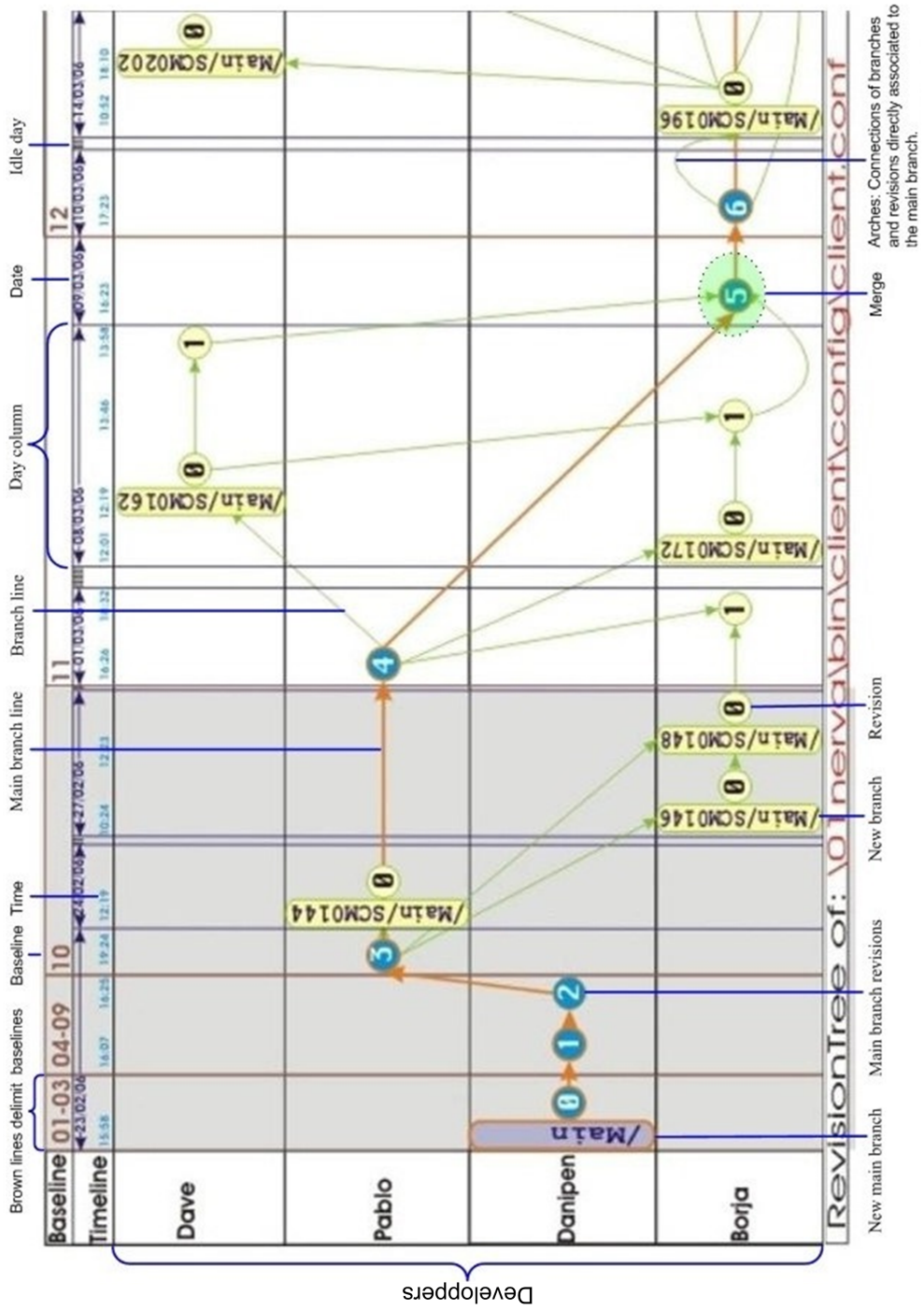


Figure 10.4: Design sketch of the Revision Tree.

Table 10.2: Visual elements and variables represented by Revision Tree.

Visual element	Description	Representation
Authors	Names of the developers.	Label with the name of the developer.
Baseline	Number of the baselines.	Is displayed in the timeline.
Date	Indicates the creation date of branches, baselines and revisions.	Label with the date.
Day column	This is the graphical space for the representation of a day having activity in the creation of branches, baselines and revisions.	A dark blue line with arrows on both ends.
Time	Shows the time when a new branch or revision has been created in the main branch or any other branch.	Label with the time.
New main branch	Indicates the creation of the main branch.	Purple large oval.
New branch	Shows the creation of a new branch.	Yellow large oval.
Main branch line	Highlights the main branch.	Orange arrows.
Arches	Connects the branches and revisions in the main branch.	Green arches.
Main branch revisions	Revisions created in the main branch.	Blue nodes.
Branch line	The branch line connects the main branch with other branches and the revisions within that branch or between two branches.	Green line.
Revision	This symbol represents the creation of a new revision of the software item.	Yellow nodes.
Merge	A merge occurs when one or more branches are combined with the main branch.	Lines coming from other branches into the main branch.

RT shows a large number of details that include the name of programmers, their participation in performing changes, the ownership of a source code file (based on the changes made) at particular periods of time, the id and date of baselines and revisions, and details about branches such as their creation and when these were merged onto the main branch, and the collaboration between programmers in time. This can be observed in Figure 10.5, which shows a side by side comparison of this visualization and VT3D using as reference the complete evolution of a source code file.

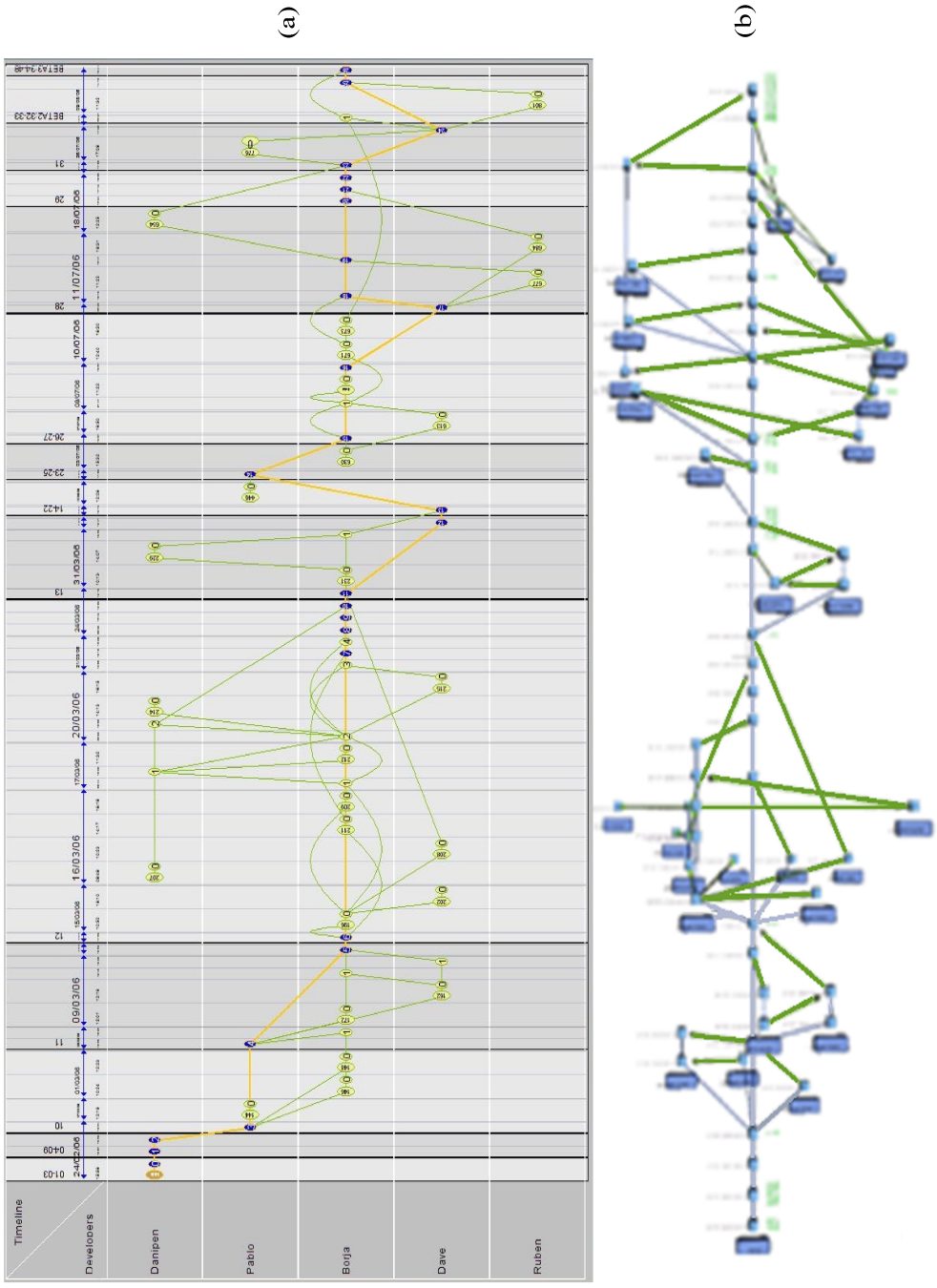


Figure 10.5: Side to side comparison of (a) Version Tree 3D and (b) Revision Tree [Therón 2007, Therón 2008].

This figure allows to see that RT makes details evident at first glance without the need of performing intricate interaction operations, whereas the same details are not possible to be appreciated with VT3D due to the lack of identifiable elements and the occlusion caused by the hiding of nodes in the back layers of the visual plane.

10.3.1 Features of Revision Tree

The evolution of each source code file implicitly holds a temporal attribute, which is the most important element in understanding the development process of any system. Therefore, the visualization of the evolution of files presented several challenges that were addressed by RT. Some of these challenges were associated to the correlation of baselines and revisions and the use of interaction techniques to uncover relevant facts. So, the challenges that were addressed with this design are the following:

* The representation of large revision trees, where the baselines have several branches and each branch many revisions.
* The navigation through the version tree offering a focus + context view.
* Support of interactivity to enable the inspection of more than one baseline at a time and exhibit the collaboration of developers to every baseline.
* The relationships between baselines.
* The hierarchical association between baselines and revisions and correlate all the information with the timeline.

Accordingly, the following sections explains the design details of this visualization and the interaction possibilities it offers.

Grid layout: RT uses a grid-based structure to provide an intuitive mechanism to visualize the working relationship between programmers and baselines by using the rows to represent the programmers and the columns for the baselines (when changes expand during a number of baselines, the column is named after this interval of baselines). Moreover, grid and matrix structures are familiar to developers and the cells can be used as containers for the drawing of nodes of the directed graph representing the flow of revisions for the file.

Timeline: The timeline of RT depicts variable width columns to accommodate the revisions in each baseline (see Figure 10.6). The distribution of the rows is uniform in the timeline and it is made up of two rows, with the first row being used for baseline numbering and the second row representing the temporal attributes about the creation of the revisions (hour and date).

Moreover, the second row includes additional visual elements such as the horizontal blue lines with arrows on both ends to emphasize a particular day and the vertical black lines to indicate the end of a day; the rounded rectangular nodes are used to emphasize the creation of branches and the orange line connecting the blue ovals to outline the main code version.

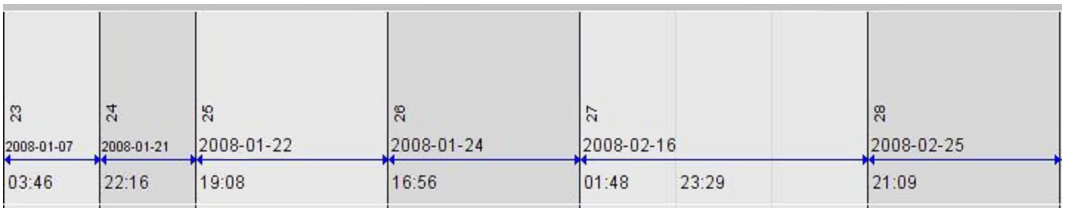


Figure 10.6: Timeline details [Therón 2007, Therón 2008].

RT lays outs revisions, baselines and branches accordingly, based on the timeline information and the developers which are working on revisions and branches. Revisions are located in the intersection between rows (depicting programmers) and columns (representing specific points in the timeline). The revisions are represented by ovals and the revision number is aligned horizontally if it has one digit and vertically if it has more than one. The blue ovals are revisions within the main branch and the others are revisions within branches. The orange line connecting the blue ovals delineates the main code versioning and the green line the branches. However, when there are duplicated branches and where the revisions belongs to more than one branch, the line connecting revisions is composed of more than one color; where each color represents a specific branch.

This representation allows to see all the baselines and revisions at a glance as well as the relationships among baselines and the hierarchical association between branches and revisions.

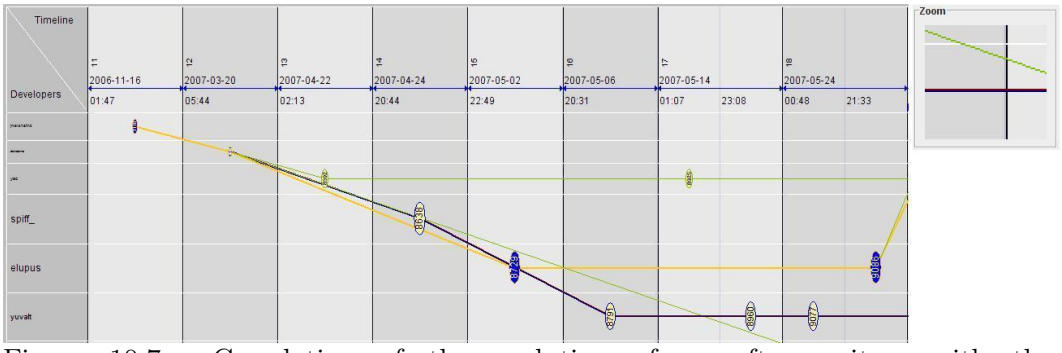


Figure 10.7: Correlation of the evolution of a software item with the timeline [Therón 2007, Therón 2008].

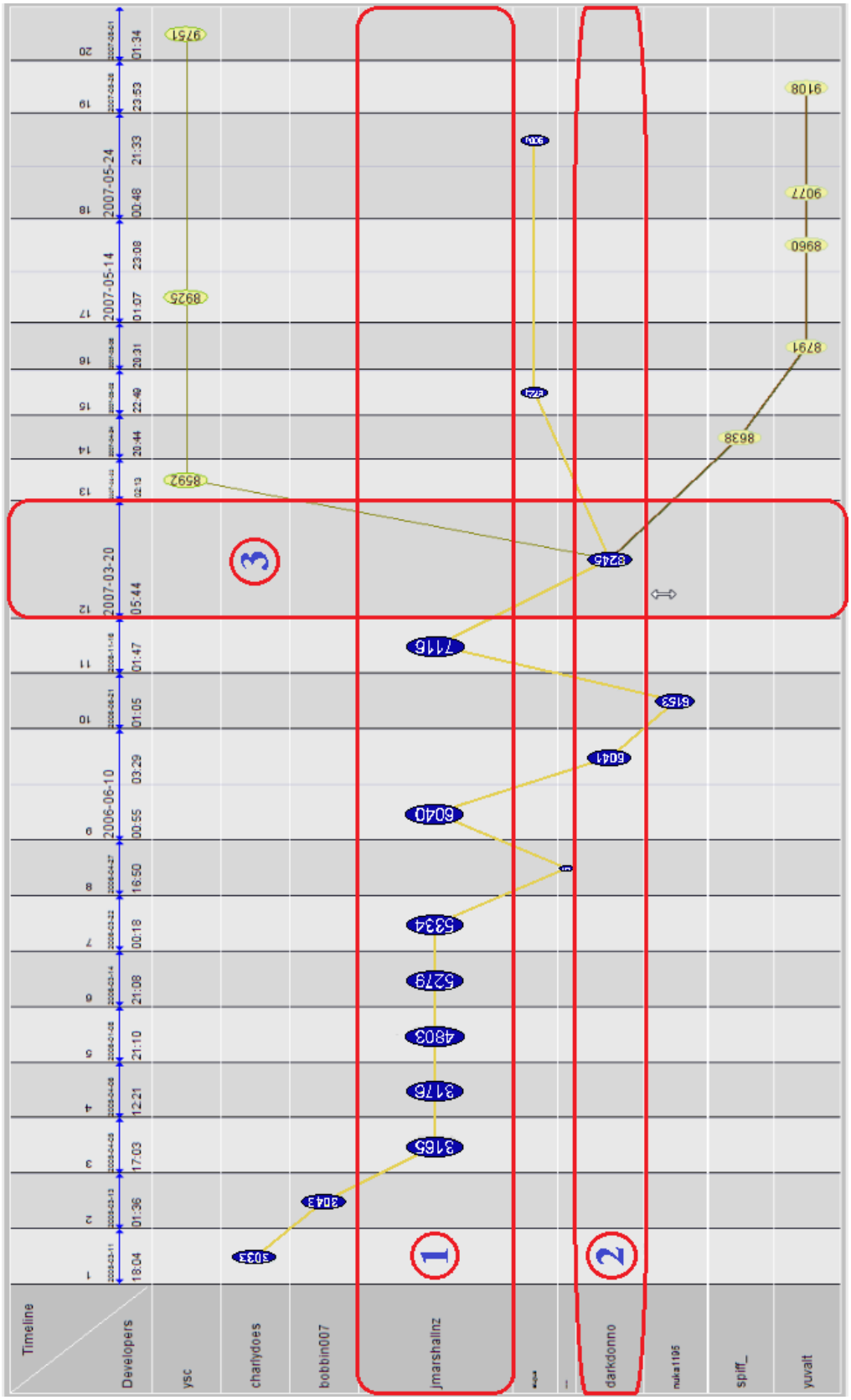


Figure 10.8: Revision Tree: polyphocal display.

Figure 10.7 shows a partial view of RT where the color coding of two branches sharing some of their revisions can be seen; the small square at the top right of the figure shows a zoom of a segment of this branch and allows two different colors to be observed, one for each branch.

Bifocal and polyphocal display: RT uses a focus + context technique that relies on bifocal and polyfocal displays, with rows of the same height and columns of variable width, depending upon the number of revisions in the baselines.

The bifocal display is the capacity of the visualization to expand a row or column of interest, whereas the polyfocal display has the same distortion behavior but allows to focus on more than one area (several rows, columns or a combination of both). Figure 10.8 allows the polyphocal display to be appreciated: two rows and one column are expanded (highlighted by colored red rectangles and numbers *1*, *2* and *3*) while the other rows and columns shrink. When this interaction is carried out, the visualization keeps on the screen all the versioning information of the software item and allows the user to concentrate on the area in which the software item have registered more activity.

Interaction: RT allows users to select the main branch or regular branches in the visualization as a means of uncluttering complex history trees. So, when the main branch of the evolution is selected at some point of the representation the remaining path of such branch is highlighted, as shown in Figure 10.9, and in the case of regular branches the remaining path is highlighted until it is merged with the main branch. This feature is even more valuable when there is a branch parallel to the main branch and it is necessary to highlight the connection between revisions or the merge point of one of these branches, as shown in Figure 10.10.

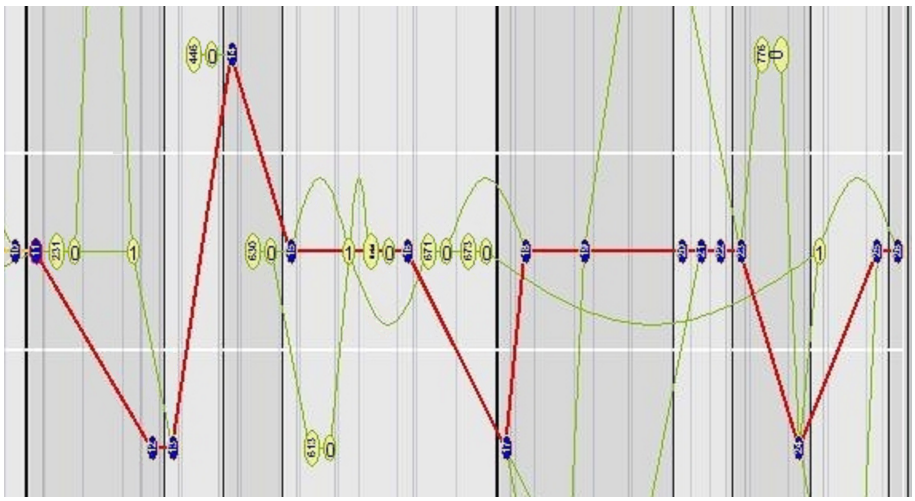


Figure 10.9: Highlighting of main development line.

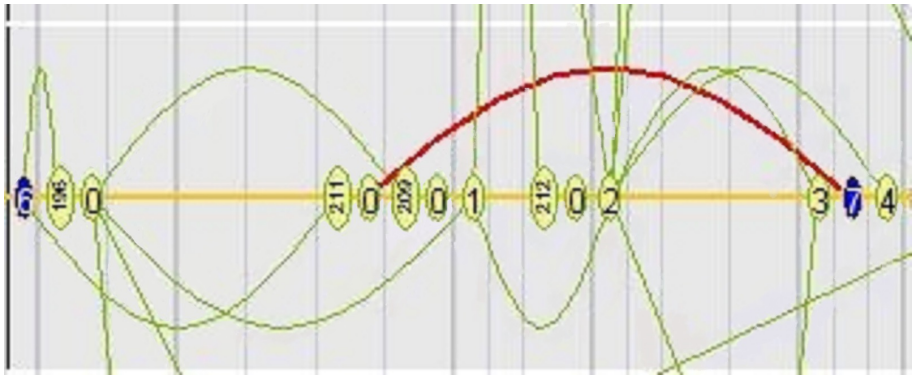


Figure 10.10: Highlighting of a curve shape in the main development line.

Moreover, this visualization supports many other interaction techniques to generate new visualization perspectives and allow the discovery of information that is not visible at first glance. Figure 10.11a shows a normal view, while Figure 10.11b shows the hiding of one developer row and Figure 10.11c shows the hiding of one developer row and one baseline column. This may be useful to have the same representation for a restricted period of time, or to include only the information concerning selected developers. The user can thus select from the entire period of the evolution of the software item or a more restricted period of time to be the object of the study and hide columns of baselines that do not wish to appear in the representation.

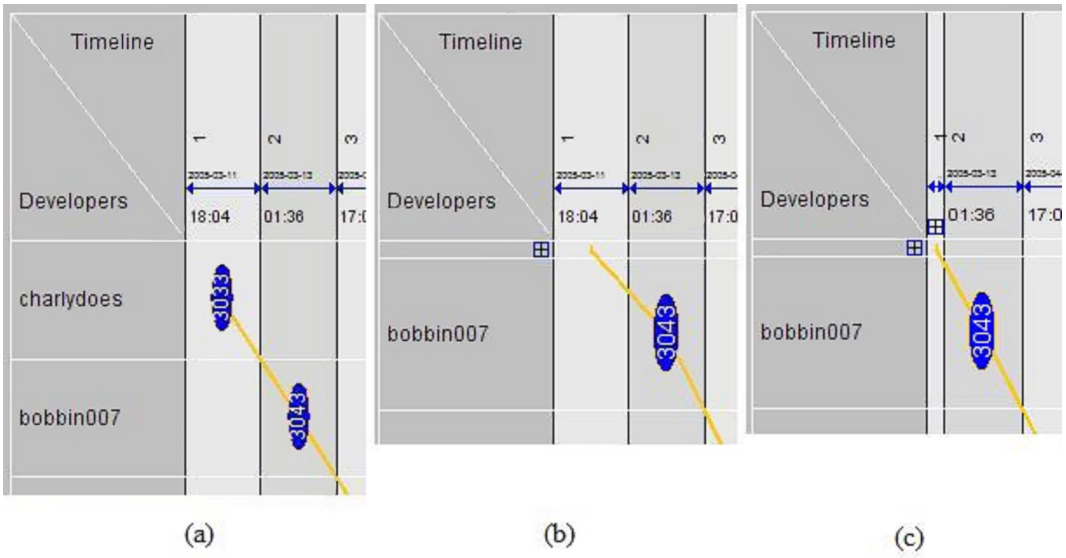


Figure 10.11: Hiding of rows and columns.

An additional feature of RT is the exchange of rows. Figure 10.12 shows the evolution of a software item in which some elements are cluttered and not clearly visible, whereas Figure 10.13 displays the same representation after

the order of rows was changed. Thus, the perspective of the representation in the latter figure provides a more appealing and clearer depiction that allows to convey information effectively.

Furthermore, when analysis tasks are carried out, this visualization allows to focus on the areas of interest by filtering dates and uses a control panel to display details of the software items, the programmers that intervene in its evolution, as well as details of the baselines and revisions (revision id, date, time, developer, log and relative path of the item), as it is illustrated in Figure 10.14.

10.4 Analysis of the Evolution of Source Code Files

This section is aimed at demonstrating the use of RT in the analysis of the evolution of source code files from open source projects. Accordingly, the evolution of *VFSBrowser.java*, *VFSFileChooserDialog.java* and *JFreechart.java* is studied, where the first two files are part of *jEdit* and the last file is a component of the project*JFreechart* [2015].

HelpViewer.java is a file that contains 6 classes (see section 9.3.2 and Figure 9.10 for further details) whose evolution has been carried by 7 different programmers for nearly 12 years, 40 revisions and 34 baselines distributed between September 2002 and July 2012. Moreover, the contributions to this file (see Figure 10.15)have been made under the main branch and the frequency of revisions lies, mostly, into separate days with only a few ones been carried out together during the same day (baselines 1, 14, 23, 26, 30 and 34).

It is important to highlight that the tests conducted with RT showed that it has some drawbacks to represent more than 80 revisions, so to deal with a larger number of revisions date filtering and, column (baselines and revisions) and row hiding (programmers) are required to avoid occlusion and cluttering. Therefore, the evolution corresponding to *HelpViewer.java* can be easily shown in a standard laptop screen of 15 inches, although the partial view of the evolution of this file in Figure 10.15 only display details of revisions carried out after June 2004 to have a readable screenshot in a small printed page (which does not provide interaction support as the computer visualization does). Furthermore, the decision of filtering out the revisions created previously to 2004 was also supported by the fact that *spestov* was the owner of the file up to that year, and hence the visualization can reveal more interesting patterns of collaboration from that point onwards as the interaction between programmers with the file is more intensive.

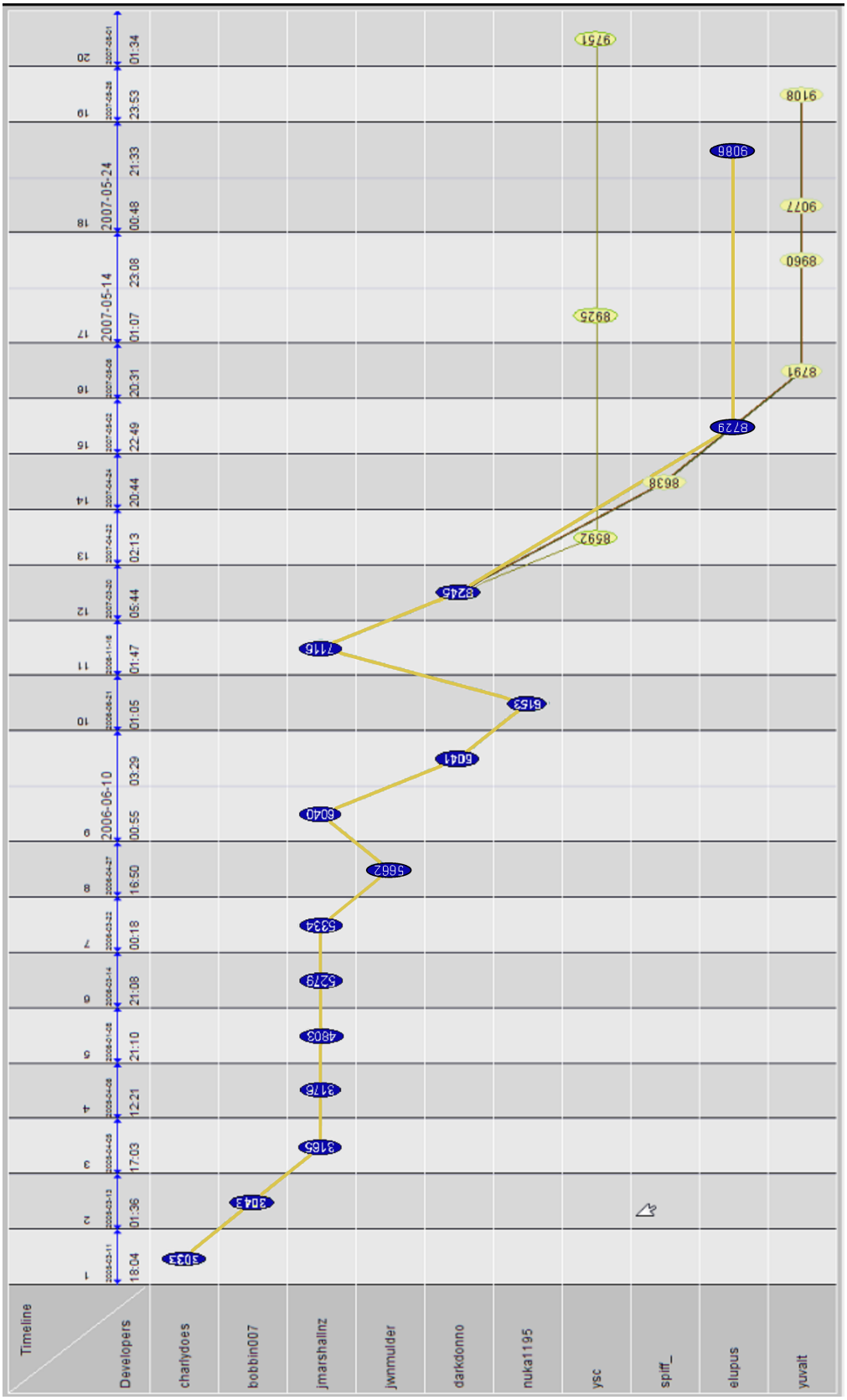


Figure 10.12: Visual representation of a software item with cluttered elements and unordered rows.

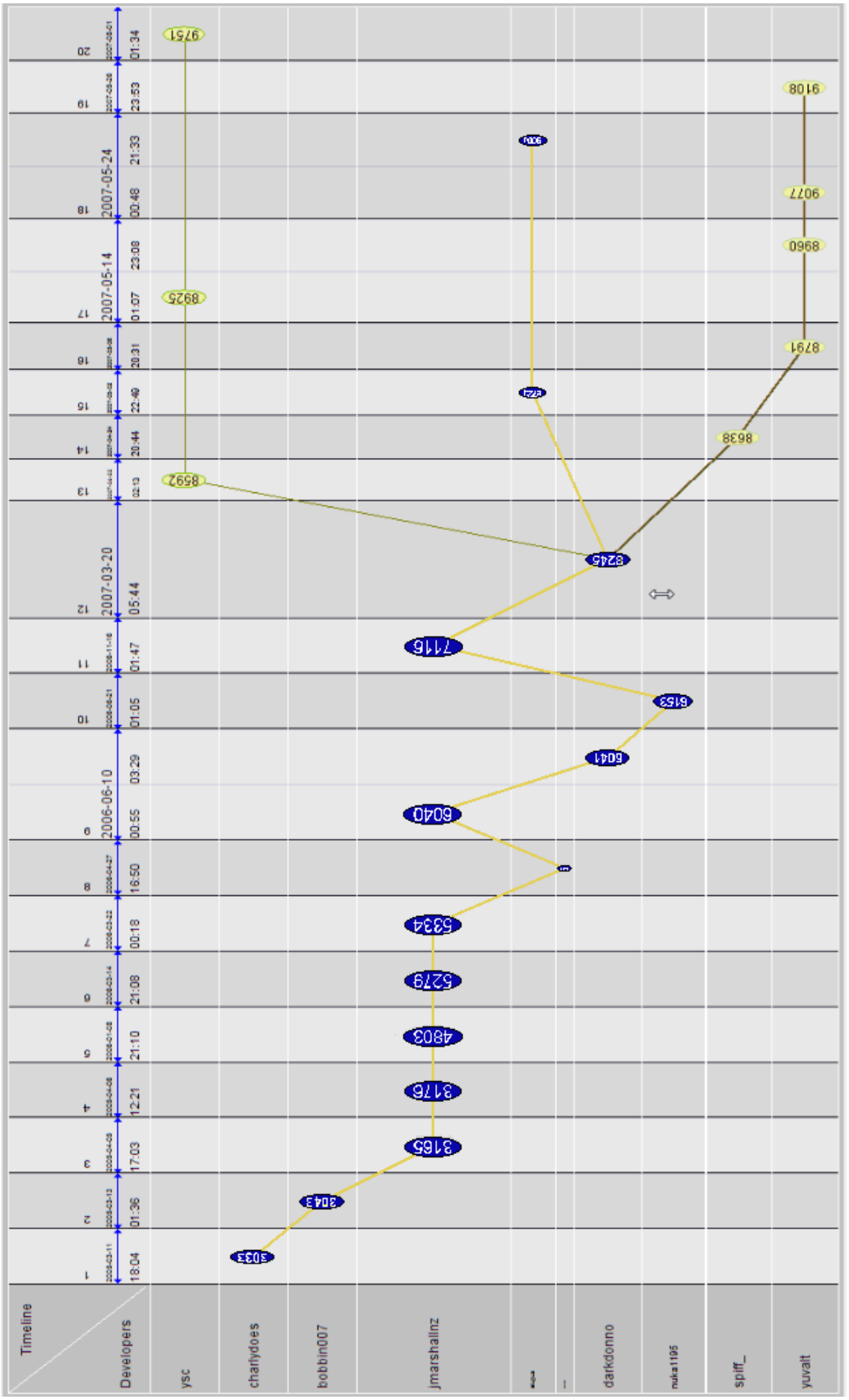


Figure 10.13: Visual representation of a software item with ordered rows.

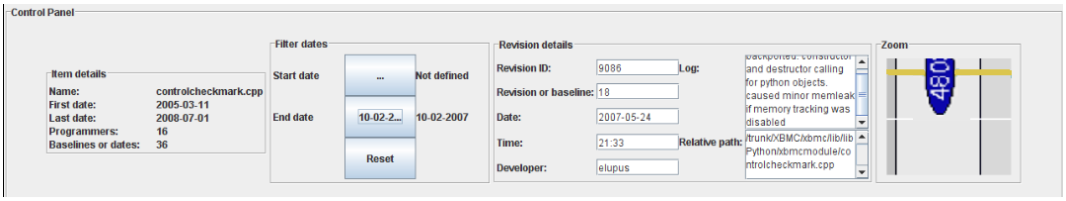


Figure 10.14: Control panel and additional details of the software item and revisions.

The programmers which have contributed to the evolution of *HelpViewer.java* (see Figure 10.15) are *Vampire0*, *spestov*, *ezust*, *karik-af*, *kpouer*, *olearyni* and *shlomy* (*Vampire0* and *kpouer* have duplicated users as it can be noted in the rows of this figure). However, according to what was already stated, before carrying out the filtering of revisions it was possible to observe that *spestov* was the developer with most of the contributions during the first years of the development, followed by *ezust*; who has taken the ownership of the file during some periods of time of the evolution. These results are coincidental with regard to what was made evident by GT and *Gridmaster* based on the information derived from the number of committed revisions: *spestov* was the developer that contributed the most to *jEdit* during the years 2001, 2002, 2003, 2004 and 2005, and later *ezust* was the developer who led the programming of the system.

The activity carried out on the file is continuous during the time period shown in Figure 10.15 as there exist revisions in all years present, although it can be noted that during the year 2010 a higher number of revisions were committed. Therefore, *HelpViewer.java* is continuously evolving and does not appear to be a file which have reached a stable development point.

Figure 10.15 highlights the baselines and revisions from the evolution of *HelpViewer.java* that were hidden, as well as the interaction between programmers, which hence depicts how the ownership of the file changes during the highlighted period of time. Furthermore, it can be noted that the rows corresponding to *spestov*, *ezust*, *kpouer* and *kerik-sf* have been expanded to give more visual space to the programmers that have committed more revisions during the evolution of the source code file. Therefore, the ability to hide, expand and reduce the size of columns and rows as well as the possibility of highlighting branches aid the focus + context capabilities of RT as it can maintain the context information while it allows to focus the attention in particular details. Furthermore, the evolution of other files of *jEdit* such as *OperatingSystem.java*, *VFSFileChooserDialog.java* and *VFSBrowser.java* is similar to that of *HelpViewer.java* in terms of collaboration and ownership patterns, according to the details revealed from the analysis carried out with RT.

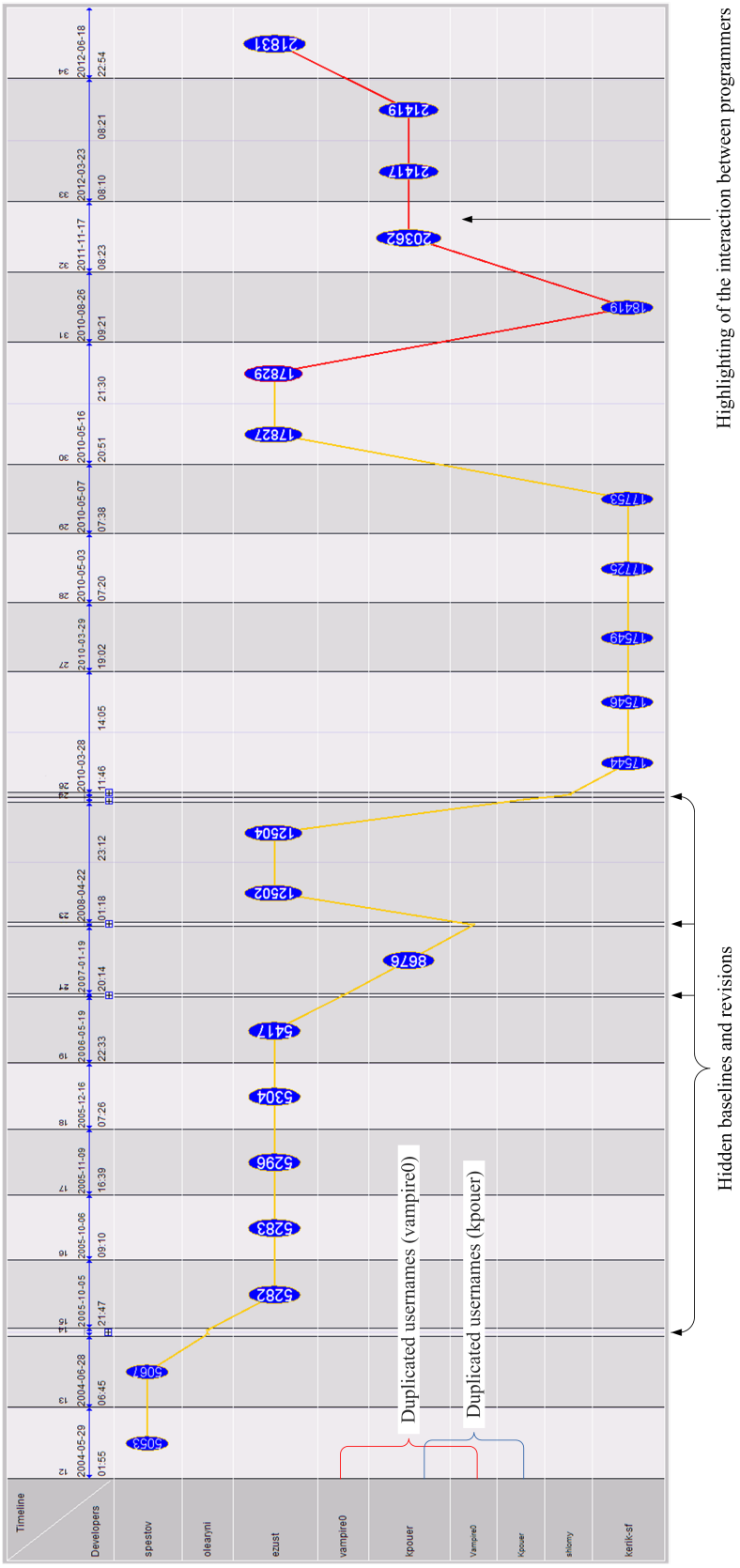


Figure 10.15: Collaboration between programmers during the evolution of *HelpViewer.java*.

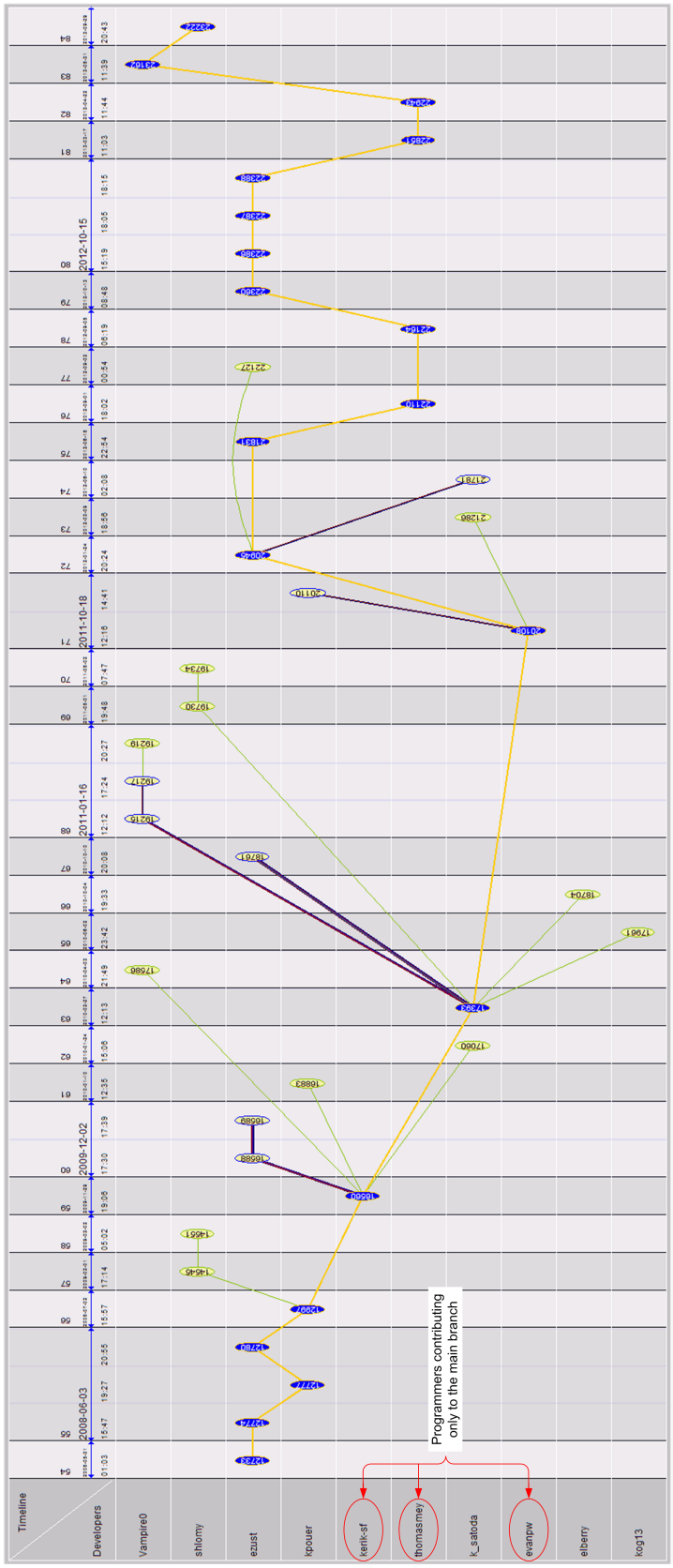


Figure 10.16: Partial view of the evolution and collaboration between programmers for *VFSFileChooserDialog.java*.

However, programmers made contributions to *OperatingSystem.java*, *VFSBrowser.java* and *HelpViewer.java* using a main development branch (orange colored) whereas in the case of *VFSFileChooserDialog.java* they created multiple branches (see Figure 10.16). Therefore, one can think that according to the type of source code file, development branches could be used in different manners.

The development of *VFSFileChooserDialog.java* between the years 2008 to 2012 is portrayed in Figure 10.16, and excludes the first 7 years of evolution of the file (from which *spestov* was the predominant contributor during the initial 5 years). During the time period under consideration the programmers that contributed to the file are *Vampire0, shlomy, ezust, kpouer, kerik-sf, thomasmey, k_satoda, evanpw, elberry and kog13*.

Figure 10.16 exhibit a large number of branches that were created by almost all programmers, with the exception of *kerik-sf*, *thomasmey* and *evanpw* (highlighted with red circles) who only contributed to the source code file under the main development branch. Therefore, some interaction operations where applied to the visualization to show a clearer representation (see Figure 10.17). So, baselines 57, 58, 61, 62, 65, 66, 69 and 70 and the rows corresponding to *elberry* and *kog13* were hidden; the position of some rows were exchanged (*e.g.*, the rows corresponding to *ezust*, *kpouer* and *Vampire0*), and the row corresponding to *ezust* was expanded whereas the height of the row associated to *shlomy*was reduced; and in addition the main branch was highlighted in the section that depicts the evolution period when no secondary branches were created.

The collaboration and evolution patterns that have been discussed so far portrayed the interactions of several programmers with the same file. Thus, one could think that this type of interactions should be simpler in *JFreeChart*, but Figure 10.18 shows that the interactions of *mungady* with *JFreeChart.java* are complex: *mungady* created three branches since the first revision of the file and he has been contributing in parallel to these; however he contributed to the branch that is highlighted in red up to the revision 679, and never merged such branch onto the main branch.

10.5 Discussion and Conclusions

It is important to recall that the design of the VKESE module is based on a focus + context approach. Therefore, GT is a context view that is linked to *Gridmaster*, which in turn is complemented by STG and RT to provide specific details of the evolution of software systems. Thus, RT is concerned with the evolution of individual source code files and meets

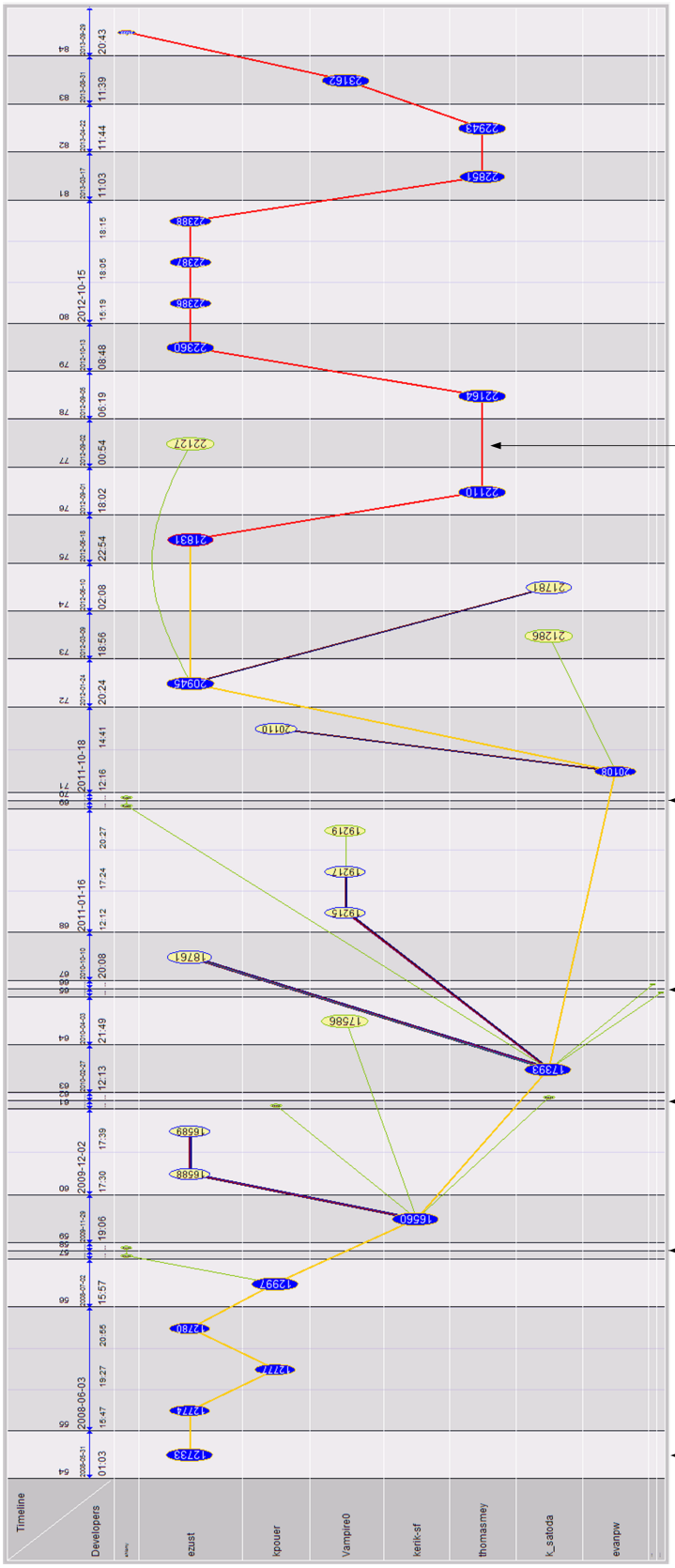


Figure 10.17: View of the evolution of *VFSFileChooserDialog.java* after applying some interaction techniques.

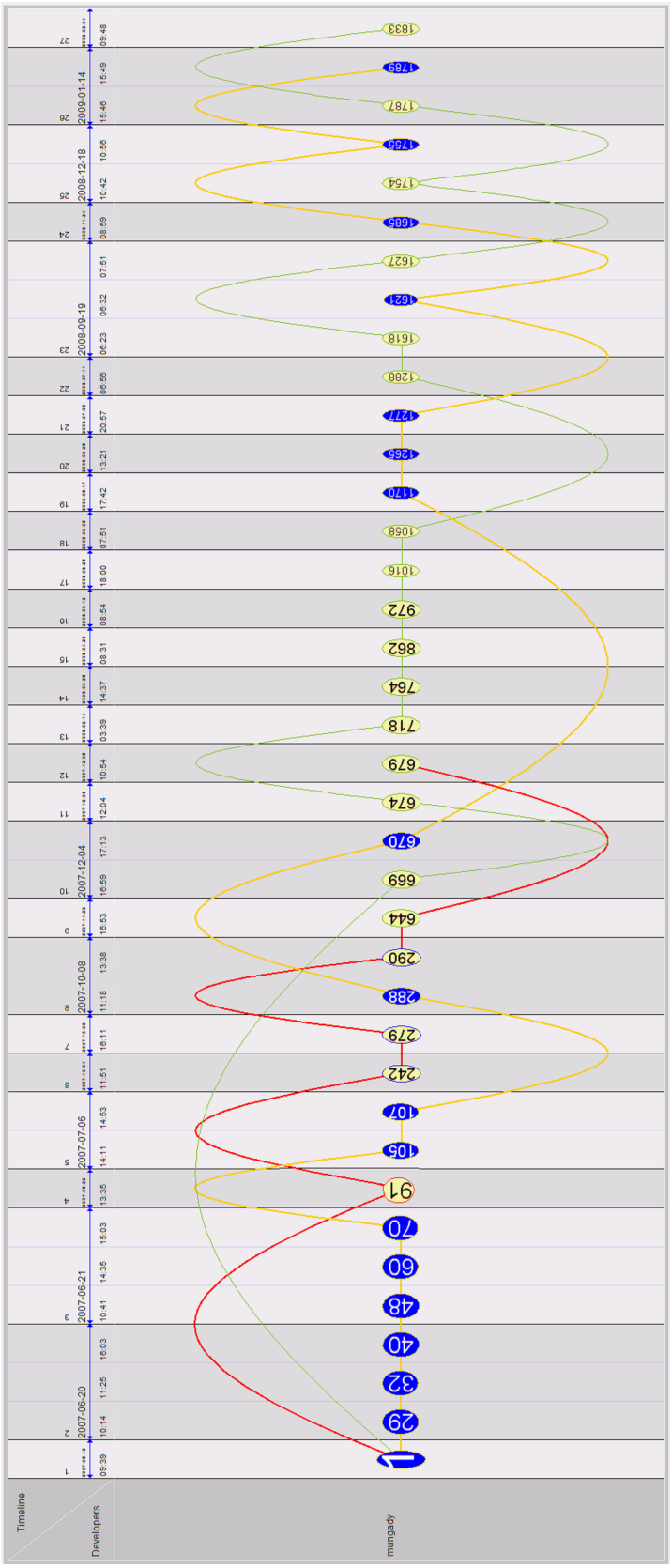


Figure 10.18: Representation of several branches that have been created by the same programmer in *JFreechart.java*.

the role of a detail view (focus), although it also uses a focus + context design to offer different levels of details of the information it represents.

Therefore, the results of the analysis of the evolution of source code files carried out with the use of RT and presented in this chapter were congruent with the outcomes of GT and *Gridmaster* discussed in chapter 8, but at a different granularity level in terms of the collaboration pattern between programmers. Thus, RT showed the interaction of programmers with a particular file, and how these take ownership of it during a time period and create branches for taking responsibility of the development or modification of a specific feature that is implemented by the software items within such source code file.

The analysis carried out in this chapter allowed to corroborate that RT is capable of displaying a large number of details to follow up the contributions of programmers to the development of source code files in a small screen, and satisfy information needs of users upon patterns of interest and time periods with the support of interaction techniques. Furthermore, this visualization assists users to carry out the analysis tasks through an integrated view that allow them to focus on particular details while the representation continues to display information that maintain the context. Moreover, it is worth to mention that the careful checking of each visualization detail in the control panel offers a great deal of additional information. So, the use of interaction and visualization has permitted to confirm the usefulness of these techniques to aid users in knowledge formation and then in decision making.

Accordingly, it was possible to verify that RT provides useful information for project managers with regard to the evolution of source code files because they can become aware of:

* Evolution details of source code files such as the number of baselines, revisions, dates and times.
* The programmers that have been working the most on the development of a source code file.
* If someone has quit or been fired from the company based on the pattern of contributions.
* Discover if the last revisions made by a programmer that is leaving the company were merged.
* The associations between programmers, branches and revisions.
* Branch merges that have never been done.
* The periods of time with higher levels of activity in the file.
* If a file is stable, due to the fact that frequent changes have not been made.

Concerning the relationship of RT with *PlasticSCM*, the release of

PlasticSCM v4 included a *2D* visualization tool that was based on some of the design elements of RT. According to *Codice Software* the feedback provided by most users in relation to the incorporation of the *2D* visualization was positive because they considered it a simple and easy to understand tool (that provides good interaction capabilities), and therefore it was considered more useful than VT3D that was harder to manipulate and produced occlusion between elements.

Consequently, *PlasticSCM* currently includes the *2D* visualization tool and a new version of VT3D will be incorporated in future releases of the system. This addition is targeted to some groups of users such as video game developers who recently have increased their interest on *PlasticSCM*. So, this decision is not based on functionality aspects with regard to the *2D* visualization, but as an alternative to show the evolution of source code files and add an attractive feature to the targeted users.

It is worth to mention that although with the use of RT is possible to obtain a great deal of information, it is recommended to use interaction techniques (*e.g.*, hiding rows and columns and filter by time periods) to make the information more readable. According to the tests that were conducted, when more than 100 revisions are represented possible occlusion issues could occur, which eventually affects the clear depiction of details.

Future improvements to RT include the explicit indication of the software items within a file that are affected by particular revisions and the magnitude of changes carried out by each revision to software items.

Chapter 11

User Assessment Test

Después de algún tiempo, talvez meses o años, Güindy logró fabricar varios modelos de reloj. El día que los presentaron a los clientes, Cucho brincaba, caminaba de lado, de frente, hacia atrás, se echaba y se veía fijamente en el espejo, al tiempo que exhibía uno de esos modelos en su cuello. Al parecer todos parecían felices con los modelos de reloj y su precisión, pero ni Güindy ni Azul lucían felices. Los dos pensaban que las cosas siempre pueden ser mejores. — El viaje de Güindy, A.González

Contents

11.1 Introduction

The assessment of tools is a process that collects data systematically concerning the fulfilling of objectives [Sharp 2011] for which a tool was designed. This type of assessment usually measures the usefulness, efficiency, effectiveness, user satisfaction, learnability and accessibility features of tools [Rubin 2008] with the goal of finding design problems to improve their functionality and usability.

The design of *Maleku* takes into account the three components of the Human Performance Model [Bailey 1989] in order to achieve the functionality and usability required to perform the tasks it aims to support. *Maleku* is targeted to programmers and project managers (the human) who are engaged

in tasks related to software development and maintenance (the activity) within a software development department or a software company (the context).

The purpose of this chapter is to assess VKESE using an assessment test [Rubin 2008] to verify its design, functionality and usability. The selection to this test is based on its usefulness in a intermediate development stage of a tool, as in the current state of *Maleku*. It should be noted that the evidence presented in this chapter are part of an iterative approach to design, implement and test tools, so that the results presented here are part of the first iteration of this iterative process.

The objective of this evaluation is to verify the degree of satisfaction and fulfillment of user expectations [Nielsen 14, Sharp 2011], based on the goals and tasks that the tool is intended to support. Accordingly, this test seeks to confirm that VKESE:

1. Offers statistical information about the revisions performed during the development of a software system.
2. Makes it possible to determine the contributions made by programmers using as a basis their commits (revisions).
3. Provides details about the evolution of the project structure.
4. Permits to identify the lifelines of software items (including packages, files, classes and interfaces).
5. Offers details about the evolution of inheritance and interface implementations.
6. Supports the monitoring of the quality of software systems during their development and maintenance through the use of evolution metrics.
7. Provides details on the collaboration between developers in source code programming.

11.2 Methodology

The research question that the user assessment test is sought to answer, and the steps required to carry out such assessment are outlined in this section. Accordingly, the following is the research question of the user assessment test:

Does the design and the integrated use of the visualizations in VKESE and their results, satisfy users expectations regarding usability and the support offered to SDME processes?

Table 11.1: Background details of the participants in the usability study.

Position	Industry	Age	Degree	Experience		
				Professional		VA
				Current position	Total	
Programmer	Financing	37	Master	15	15	0
		32	Master	1	7	5
	Software company	38	Ph.D.	1	15	0.5
Research and development	University	35	Ph.D.	2	8	6
	Software company	34	Ph.D.	1	12	5
Architect	Mining	41	Master	9	19	0.5
Team leader	Financing	39	Master	7	16	0
Average years of experience		**37**		**5.14**	**13.14**	**2.29**

The steps to carry out the assessment test were the identification of users and tasks, the preparation of the questionnaire and the setup of test environment.

1. **Identification of users and tasks:** The first step in designing the assessment was the identification of experienced users in programming tasks, and preferably with some knowledge in VA. Therefore, the assessment test was carried out with 7 potential users with an average of 13 years of professional experience in programming tasks and 2 years of experience working on the research and development of VA tools. Whereas all the users hold at least a master degree and were distributed as following: 3 hold a master degree, 1 is a VA Ph.D. student and the other 3 hold a Ph.D. (the dissertation of two of them was on VA). All the participants are currently working part-time or full time carrying out programming tasks within software development departments, software companies or research and development departments of universities or multinational companies. The background details of the participants are presented in table 11.1 .

2. **Tasks carried out by users:** The second step consisted in the definition of representative tasks to be carried out by the users and the definition of questions. The tasks that users were asked to undertake are the following:

   (a) Explore the tool for 15 minutes to familiarize themselves with its general operation and the interaction between views.

   (b) Search for two software items in the project under analysis and identify the inheritance and implementation relationships using the

visualizations.

(c) Review the activity of the software project for a given period of time to comprehend its general evolution behavior in terms of which are stable and active software items.

(d) Explore the activities carried out by programmers to understand their contributions and the socio-technical relationships that they have established with the software items. Participants were also asked to obtain details about who has led the project under study and which programmers are actively participating or have ceased its participation in the development of the system.

(e) Carry out an assessment of the visualizations, based on the type of representation used and their ease of understanding and learning.

3. **Preparation of the questionnaire:** The third step involved the preparation of a questionnaire that included 50 questions, from which 21 were closed-ended questions, using a Likert [Likert 1932] scale, and 29 were open-ended questions. The questions were split into tool functionality questions (13) and visualization questions (37). On the one hand, the tool functionality questions were split into questions concerned with the understanding of the project evolution (6 questions) and questions related to management tasks and the comprehension of the collaboration among programmers (7). On the other hand, the visualization design questions were further split into specific questions for each individual visualization (28 questions in total and 7 per visualization) and for the tool in general (9).

4. **Setup of test environment:** The fourth step consisted on carrying out the assessment test on a individual basis, because the users were located in different countries and cities. To this end, users needed to use a software for video conferencing [Skype Communications SARL 2013] as well as a remote access software [TeamViewer GmbH 2013] for accessing a small server, where the tool was running. The usability test required between 90 and 120 minutes to be completed, and it was started with a 15 minutes presentation to explain the tool features and functionality. Then the users were given the names of two software items, a class and an interface, which they needed for using the tool and answer the online questionnaires [Qualtrics, Inc. 2013].

Table 11.2: Question group: Understanding software project evolution.

	Question	Correct answers
$Q1$.	Which interfaces have been implemented by the given class during the project evolution?	7
$Q2$.	Which classes have inherited from the given class during the project evolution?	6
$Q3$.	How the inheritance relationship of the given class with its superclasses have changed during the project evolution?	7
$Q4$.	Which classes have implemented the given interface during the project evolution?	7
$Q5$.	What is the package that is currently under intensive development and which packages did not changed the last year?	6
$Q6$.	How have metrics changed during the project evolution for the given class?	7

11.3 Assessment results

11.3.1 Tool functionality

The target of the questions in this group was to assess the functionality of *Maleku* in tasks related to the understanding of the project evolution and the comprehension of the collaboration among programmers, where the former looks to assist programmers and the latter to aid project managers. Accordingly, this group of questions was further subdivided into two subgroups: Understanding software project evolution and Comprehension of the collaboration among programmers for aiding management tasks.

11.3.1.1 Understand software project evolution

The goal of the questions in this group was to assess the capabilities of the tool for providing information about changes in inheritance and implementation relationships, as well as in software metrics. To this end, the participants were given the name of two software items, an interface and a class, to locate them in the project structure and answer the questions in Table 11.2, which were open-ended questions. The number of correct answers to these questions is shown in the aforementioned table. 4 questions out of 6 were answered correctly by all participants, whereas 2 were answered incorrectly. It is relevant to highlight that the incorrect answers were provided by the same participant. Overall, the results on these questions were positive, as the average of correct answers was 6.66 out of 7 (92.23%). On average the time required to answer these questions was 15 minutes, for an average of 2.5

minutes per question.

Table 11.3: Question group: Collaboration among programmers.

	Question	Results
$Q7$.	Who is the programmer that has created more revisions during the project evolution?	4
$Q8$.	Who is the programmer that led the software project in its early stages?	5
$Q9$.	Select a time period in Gridmaster, see the results in the Socio-Technical view and correlate them with the ones in Gridmaster. Then, based on the software items that programmers have changed in common, describe the relationship among them.	7
$Q10$.	Select the year 2012 in the Gridmaster and review the results in the Socio-Technical view. Based on those results, who is the programmer with more contributions in 2012?	7
$Q11$.	Based on the volume of revisions, who are the programmers that have taken most of the project responsibility during the year 2012?	7
$Q12$.	Based on the number of revisions in 2012, who are the programmers that could substitute the leading programmer in any eventuality?	7
$Q13$.	Who is one of the programmers that could have left the project in the past year?	4

11.3.1.2 Comprehension of the collaboration among programmers

The questions in this group were focused on the comprehension of the relationship among programmers for supporting managers and team leaders in decision making.

The answers to this set of questions were not as accurate as those for questions in the previous group. Questions $Q9$, $Q10$, $Q11$ and $Q12$ were answered correctly by all participants, whereas questions $Q7$, $Q8$ and $Q13$ were answered incorrectly by all participants. It is noteworthy that to answer questions $Q7$, $Q8$ and $Q13$ only one visualization is required, and not several, which contrasts with the other questions in the group that require the use of more than one visualization. In addition, it is also interesting that these questions are related to the contributions of programmers and their continuity in the project.

The participants who answered incorrectly questions $Q7$, $Q8$ and $Q13$ showed confusion because they did not understand which visualization should be used to answer the questions. Their feedback points out that there were

no explicit indications in the screen about which visualization to use for accomplishing each particular task, neither the steps that could be followed. This could indicate that the design of VKESE is not intuitive enough and that its learning requires training to understand better the visualizations that compose it (the training that was offered to participants was of only 15 minutes).

The average of correct answers for this group of questions was 5.86 out of 7 (83.67%), which compared to the previous group, are 11.56% inferior. The questions and the results for this group are listed in table 11.3. The average time required by each user to answer these questions was 25 minutes on average, 3.57 minutes per question.

11.3.2 Visualization design

This group of questions was asked after the users had used the tool, looked for details and answered the group of questions of the previous section. It is composed by closed-ended and open-ended questions that are aimed to evaluate the visualization design of the tool in general and each visualization in particular. Accordingly, the questions in this group are split into 5 smaller groups and the corresponding answers are discussed in the following sections, starting with the individual visualizations and continuing with the results of the overall tool evaluation. Closed-ended questions are graded between 1 and 5, and the results are presented in the corresponding tables as an average value of the answers provided, whereas the entry values provided to open-ended questions are shown in each section as edited comments.

The closed-ended questions assess the visual design, ease to learn and user satisfaction with regard to the visualization. The options available to evaluate the design and ease to learn are shown in table 11.4, while options to evaluate user satisfaction are presented in table 11.5. The average time used to answer questions from these groups was approximately 48 minutes.

Table 11.4: Answers for closed-ended questions that assess the visual design and easy to learn of the visualization.

Weights	Answers
1	Strongly disagree
2	Disagree
3	Neither agree or disagree
4	Agree
5	Strongly agree

Table 11.5: Answers for closed-ended questions that assess the user satisfaction with the visualization.

Weights	Answers
1	Not at all satisfied
2	Not satisfied
3	Partially satisfied
4	Satisfied
5	Highly satisfied

Table 11.6: Question group: Granular Timeline visualization assessment.

	Question	Question type	Results
$Q14$.	Are the time units of this visualization clearly enough represented?	Design and learning	4.28
$Q15$.	Is the use of concentric rings easy to understand for getting insight into the hierarchical relationship between time units	Design and learning	4.14
$Q16$.	Is the visualization easy to learn, use and understand?	Design and learning	4.14
$Q17$.	Which is your satisfaction degree with this visualization?	Satisfaction	3.85
$Q18$.	Which are positive aspects of this visualization	Open-ended	
$Q19$.	Which are negative aspects of this visualization	Open-ended	
$Q20$.	Please add any extra comments you have.	Open-ended	

11.3.2.1 Granular Timeline

The questions in this group are listed in table 11.6 and cover the evaluation of the Granular Timeline visualization. Questions $Q14$, $Q15$, $Q16$ and $Q17$ are closed-ended questions, whose results are displayed in table 11.6. The average assessment value received for those questions is 4.1 out of 5 (82%). The answers to question $Q14$ show that participants assessed positively the representation used, while the answers provided to questions $Q15$ and $Q16$ make evident that they considered that the visualization is easy to understand and learn. Whereas the answers to question $Q17$ point out that the satisfaction degree with this visualization could be improved, taking into account the comments offered in questions $Q19$ and $Q20$. With regard to questions $Q18$, $Q19$ and $Q20$, which are open-ended questions, their answers are summarized as follows:

Question 18: The information is compacted and the radial layout improves the distribution of elements. It allows to represent a large number of elements. The visualization is innovative and allows to explore the information

at different granularities.

Question 19**:** The visualization could get cluttered when representing a large number of years. The user needs some time for understanding the representation and it lacks of customizable features such as color coding.

Question 20**:** The menu options are only visible when right-clicking and the area for each time unit is not re-sizeable for allowing users to focus in a area of his/her interests.

Table 11.7: Question group: Gridmaster visualization assessment.

	Question	**Question type**	**Results**
$Q21$.	Is the relationship between the project structure and the time line units clearly enough represented in this visualization?	Design and learning	4.14
$Q22$.	Does the matrix layout facilitate getting insight of the relationship between the software items and the time line units?	Design and learning	4.00
$Q23$.	Is the visualization easy to learn, use and understand?	Design and learning	3.57
$Q24$.	What is your satisfaction degree with this visualization?	Design and learning	3.57
$Q25$.	Which are positive aspects of this visualization?	Open-ended	
$Q26$.	Which are negative aspects of this visualization?	Open-ended	
$Q27$.	Please add any extra comments you have.	Open-ended	

11.3.2.2 Gridmaster

This group of questions is focused on evaluating the Gridmaster visualization and is listed in table 11.9. Similar to the set of questions in the previous section, the first four questions of this group ($Q21$, $Q22$, $Q23$ and $Q24$) are closed-ended questions. The answers to these questions, although they are good, are not entirely positive.

Answers to questions $Q21$ y $Q22$ reflect that participants consider that the visualization represents, acceptably, the relationships between software items, but the answers to question $Q23$ indicate that the visualization can be improved to facilitate its learning and ease of use. Concerning question $Q24$, the level of satisfaction of participants shows that it is required to take actions to improve the visualization, which could be achieved by taking into account the answers to $Q23$ and the comments provided in questions $Q26$ y $Q27$. The average assessment qualification for questions $Q21$, $Q22$, $Q23$ and $Q24$ is 3.82 out of 5 (76.4%). The answers provided to questions $Q25$, $Q26$ and $Q27$ are the following:

Question 25**:** The use of a tree to represent software projects gives the user a clear overview of the hierarchy. The colors allow the user to clearly distinguish programmers and associated changes. Moreover, the design is user friendly and allows to easily correlate project structure changes with time. The response time of this visualization is good and allows to navigate easily through the project structure.

Question 26**:** The visualization does not have an option for searching and it requires to navigate the tree structure. Moreover, the visualization does not provide details of context concerning to the point where the user is located in the visualization and thus, it complicates the navigation. Additionally, the visualization of metrics is difficult to interpret and it does not allows to navigate into the source code. The layout of the visualization could support more features that the ones that it currently supports.

Question 27**:** The navigation of this visualization needs to be improved to find classes and also to display the associated source code.

Table 11.8: Question group: Socio-Technical Graph assessment.

	Question	Question type	Results
$Q28.$	Is the relationship between programmers and contributions represented in this visualization with enough clarity?	Design and learning	4.14
$Q29.$	Are the relationships among programmers represented in this visualization with enough clarity?	Design and learning	4.42
$Q30.$	Is the visualization easy to learn, use and understand?	Design and learning	4.42
$Q31.$	Which is your satisfaction degree with this visualization?	Design and learning	3.57
$Q32.$	Which are positive aspects of this visualization?	Open-ended	
$Q33.$	Which are negative aspects of this visualization?	Open-ended	
$Q34.$	Please add any extra comments you have.	Open-ended	

11.3.2.3 Socio-Technical Graph

The set of questions in this group was aimed at assessing STG and it is listed in table 11.8. The average assessment value to questions $Q28$, $Q29$, $Q30$ and $Q31$, which are closed-ended questions, is 4.14 out of 5 (82.75%).

The pattern of answers from participants is very similar to that of the above groups of questions. The results for the first three questions ($Q28$, $Q29$ and $Q30$), show that participants believe that the visualization displays information properly and that its learning is simple, although the answers

to question $Q31$ warn that user satisfaction with the visualization must be improved. The commented answers for questions $Q32$, $Q33$ and $Q34$ are the following:

Question 32: This view is easy to understand and accomplish the goal of representing the relationships among programmers and their contributions.

Question 33: The label for programmer's name is not easy to find and the visualization lacks from a rich user interaction and filtering support.

Question 34: This is a nice visualization complement, although it is not an original visualization.

Table 11.9: Question group: Revision Tree assessment.

	Question	Question type	Results
$Q35$.	Does the visualization shows clearly the branches and collaborations of programmers in the item evolution?	Design and learning	5.00
$Q36$.	Are the revisions correlated adequately with their temporal occurrence?	Design and learning	4.42
$Q37$.	Is the visualization easy to learn, use and understand?	Design and learning	3.57
$Q38$.	Which is your satisfaction degree with this visualization?	Design and learning	3.57
$Q39$.	Which are positive aspects of this visualization?	Open-ended	
$Q40$.	Which are negative aspects of this visualization?	Open-ended	
$Q41$.	Please add any extra comments you have.	Open-ended	

11.3.2.4 Revision Tree

The questions in this group were focused on evaluating RT. Similar to the questions in the other groups, the first 4 questions ($Q35$, $Q36$, $Q37$ and $Q38$) are closed-ended questions. The average value of the rate received for these 4 questions is 4.14 out of 5 (82.8%). The pattern of answers is similar to that of the three groups discussed above: participants considered that the design of the visualization is acceptable and give it a 4 or higher rate. However, the answers of participants exhibit that the visualization is not as easy to learn and use as they rate this factor with (3.57 out 5), which is the same rate for their satisfaction with the visualization design (3.57 out of 5).

User comments suggest that RT has many interaction options, but these options are not evident and therefore users have no way of knowing the available options. These comments also make observations about the occlusion that occurs when a large tree is shown, and about the difficulties to use the

interaction option that permits to expand rows and columns. Thus, the difficulty to learn and use RT, indicated by the rates obtained by question $Q37$, may be due to aspects related to the interaction options, and the low satisfaction with the visualization design (question $Q38$).

The summary of answers to questions $Q39$, $Q40$ and $Q41$ is the following:

Question 39**:** The use of a grid structure and a tree representation makes this visualization easy to understand. It allows to easily review which programmers have participated in the development of a software item. Moreover, the possibility to exchange the rows of the visualization is a nice feature.

Question 40**:** The interactions that can be carried out in the visualization are not clear. When the tree is loading, before performing a filtering, it is not possible to see, clearly, the complete evolution of the software item. The possibility to expand rows and columns is complicated and impractical.

Question 41**:** The visualization is nice and has lots of interaction options, but the user has to guess which are the available options, because these are not evident at first glance and no clues about them are offered. The representation of large evolution trees is nor clear neither useful.

11.3.2.5 Evaluation of the tool

The goal of the questions in this section is to assess VKESE, in general. This group includes 9 questions (see table 11.10), where questions $Q42$, $Q43$, $Q44$, $Q45$ and $Q46$ are closed-ended questions and questions $Q47$, $Q48$, $Q49$ and $Q50$ are open-ended questions. Questions $Q42$ and $Q43$ assess the visual design, question $Q44$ how easy to learn, use and understand is the tool and $Q45$ the satisfaction degree. By contrasting the rating given to question $Q42$ with the answers to the questions listed in table 11.3, it can be noted that there is a significant discrepancy between the answers. The reason for this discrepancy could be a misinterpretation of this question, due to conceptual issues, because multiple users did not understand the notion of the term socio-technical clearly enough.

Concerning the answers to question $Q43$, these are satisfactory because this question was aimed to rate some of the key elements that VKESE sought to support. Meanwhile, the rating obtained by question $Q44$ is not positive when a correlation with the comments offered to question $Q49$ is made: participants suggested to improve several aspects regarding the ease to learn of VKESE.

One aspect that stands out is that the rating that participants gave to question $Q45$ is higher than the score given to similar questions for the 4 individual visualizations. The average rate for questions $Q42$, $Q43$, $Q44$ and $Q45$ was 4 out of 5 (80%), as listed in table 11.10.

Table 11.10: Global assessment of VSEKE.

	Question	Question type	Results
$Q42$.	Is the tool clear enough to provide insight into the socio-technical relationships among programmers and software items?	Design and learning	3.85
$Q43$.	Are the tool and its visualization clear enough to provide insight into the project structure, software item relationships (inheritance and interface implementations) and metrics made through the time?	Design and learning	4.28
$Q44$.	Is the visualization easy to learn, use and understand?	Design and learning	3.71
$Q45$.	What is your satisfaction degree with this visualization?	Design and learning	4.14
$Q46$.	Which of the following were disadvantages of the tool evaluated?	see Figure 11.1	
$Q47$.	Which of the following were disadvantages of the tool see Fig. 11.1 evaluated?	Open-ended	
$Q48$.	Which are positive aspects of this visualization?	Open-ended	
$Q49$.	Which are negative aspects of this visualization?	Open-ended	
$Q50$.	Please add any extra comments you have.	Open-ended	

The answers provided for question Q39 are shown in Figure 11.1 and revealead that the main concerns of participants for adopting these kind of tools are User overload, Ambiguity, Over-complexity, Difficulty to understand, training and resources demands and the requirement of previous knowledge. With regard to questions $Q47$, $Q48$, $Q49$ and $Q50$, the text entry comments provided are the following:

Question 47**:** The integration of the tool as an IDE plugin is one of the strongest points of this toolset, because it allows to have everything in the same programming environment. However, the performance of the tool needs to be improved.

Question 48**:** The tool allows to analyze intuitively the life cycle of a project and it could be useful for software maintenance. The tool is simple, easy to learn and allows to represent large datasets cleanly.

Question 49**:** The interaction between the visualizations needs to be improved, as well as the performance of the tool. Learning the features of the tool requires some time, as it is a specialized tool. Furthermore, the volume of details displayed to the users could easily overload them. The visualizations also lack from customizable features.

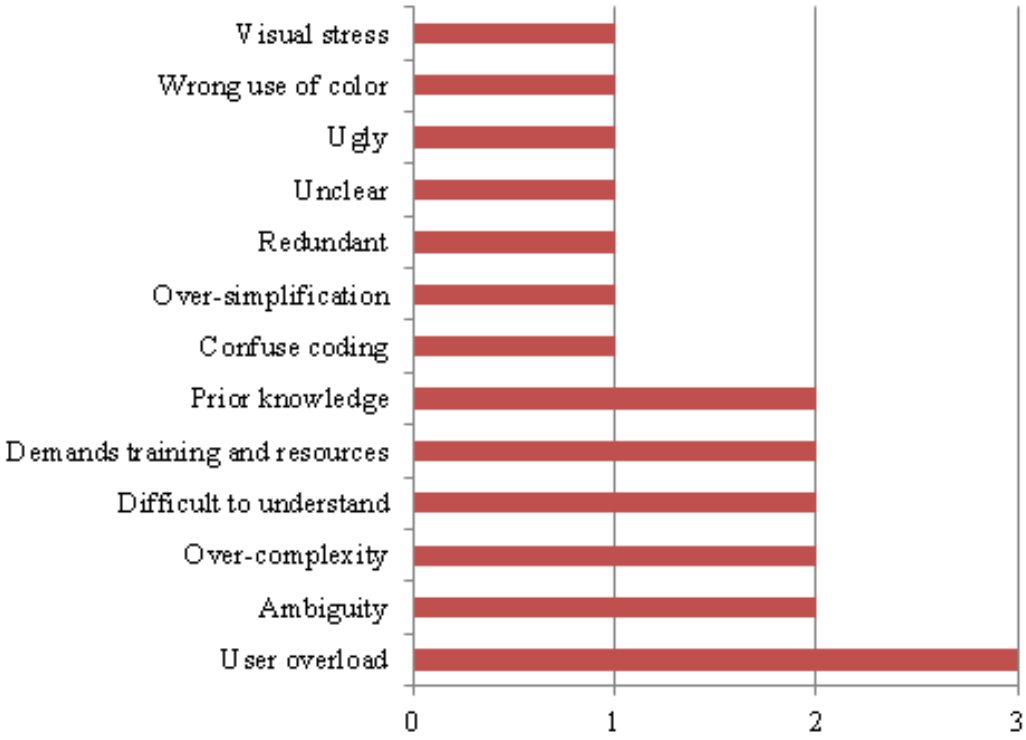


Figure 11.1: Blockers for the adoption of the tool.

Question 50**:** The tool requires to improve the performance of data loading.

11.4 Discussion

The usability testing was divided into two sections: tool functionality and visualization design.

Tool functionality: The aim of the questions that evaluated the tool functionality was to measure if the tool accomplishes the intended goals. These questions were open-ended questions and were prepared taking into account the main tasks for which the tool was designed. Therefore, the first group of questions was aimed at evaluating the functionality of the tool for providing details for understanding the evolution of the project structure and metrics during a period of time or the complete evolution. Whereas, the objective of the second group of questions was to measure the support offered by the tool to project managers and team leaders. The results for this evaluation were positive: the average grading for the tasks in the first group of questions was 95.23% and 83.67% for the second group of questions.

However, it is worth to highlight that questions $Q7$, $Q8$ and $Q13$, which

are related to the contributions of programmers, obtained poor ratings. Thus, the aspects of VKESE to be improved should be reviewed carefully, so users can obtain information more efficiently. However, overall, the answers of participants show that the tool meets the functionality goals that are pursued with its design.

Visualization design: The goal of the assessment of the visualization design was to measure the satisfaction of users when using the tool for finding relevant details to accomplish a given tasks. In this context, several closed-ended questions were used to gather information on the overall satisfaction of users and open-ended questions were used to get further information to improve the functionality and usability of the tool. The results obtained from closed-ended questions in the usability group, in general, were positive, as the average answer was 4 out 5 (80%).

The questions that received the lowest rating in the evaluation, taking into account all the visualizations and the tool in general, are those related to the degree of satisfaction of participants with visualization design ($Q17$, $Q24$, $Q31$, $Q38$ and $Q45$). This rating appears to be related to the ease of use and learn of visualizations and the tool in general, the answers to the relevant questions in that section ($Q16$, $Q23$, $Q30$, $Q37$ and $Q44$) reflect the second lowest rating in the assessment. Therefore, although participants considered that information is adequately represented by each visualization, according to the answers to the relevant questions, this does not imply that it was easy for them to use the visualizations and extract useful knowledge.

Finally, most of the comments from users pointed out on the improvement of the tool performance, the addition of searching and navigation features, the need for additional functionality to get more project details and the inclusion of custom features such as the ability to customize color coding. Another detail to be considered is that the interaction options are not clear to users, so it is advisable to develop strategies to make more intuitive the interaction possibilities of the visualizations. Moreover, the evaluation also allowed to detect some minor bugs that were not previously detected.

11.5 Conclusions

This chapter has presented and discussed the results of the evaluation of VKESE using an assessment test to verify its design, functionality and usability. According to these results, VKESE meets the goals and objectives that were raised during its design to support SDME processes. Thus, the visualizations that constitute this tool represent and provide relevant information in a appropriate form, although the degree of satisfaction and

fulfillment of expectations of users when using them is positive it is not optimal and requires attention in the next iteration of the design and implementation process.

Consequently, the implementation of *Maleku* and VKESE, using an architecture based on the EVSA process has demonstrated the usefulness of the application of VA to SE and therefore, the applicability of such process to the design of tools to support SDME processes.

Part V

Conclusions

Chapter 12

Conclusions

A los días Güindy y Cucho decidieron seguir con su aventura, abandonaron la fábrica y la ciudad. En su nueva travesía tomaron un camino pedregoso, cruzaron varios ríos, subieron y bajaron montañas, hasta que se encontraron frente a un precipicio que parecía ser el inmenso cráter de un volcán. Buscaron la forma de llegar al otro lado para continuar su camino, una opción era bordear, pero la espesura de la selva no les permitía ver qué había más allá, en el horizonte. — El viaje de Güindy, A.González

Contents

12.1 Introduction

This chapter is aimed to present and discuss the concluding remarks of this thesis (section 12.2), the research papers that have been published and are related to this research (section 12.3) and the future works that will be carried out (section 12.4), following the results presented in the previous chapters, in the next iteration of this research work and according to methodology described in Chapter 1.

12.2 Concluding Remarks

Developers and project managers need to understand the software systems they are developing and maintaining, in particular when they have no prior knowledge or documentation of those systems. This situation acquires greater importance with the fact that SDME is a process which usually extends

throughout several years and produces data that shares many of the typical characteristics of *Big Data*. This data are in the form of lines of code, software components, variables, methods, programming structures and details about the relationships between software elements. Thus, the capacities of programmers and project managers are particularly limited when they need to analyze large projects, for several revisions over an extended period of time, and are not able to extract useful information. Accordingly, this research has taken into account that:

1. Software developers and project managers require to understand software systems and the changes that are produced during SDME processes.
2. There is an evident need to use methods of analysis to reduce the volume of data that needs to be examined and studied by programmers and managers.
3. The use of software analysis or SEA, depending on the number of revisions or the time period under study, could lead to the generation of useful knowledge.
4. It is often the case that the results of software analysis and SEA are too voluminous and complex to produce knowledge that could lead to effective problem solving.

Therefore, the previous observations and the positive results of the application of VA to different areas of knowledge (as discussed in chapter 3) motivated to study and analyze their use in the understanding of software systems and their evolution in research (see chapters 4, 5 and 6) and software industry settings (see chapter 7). Consequently, the conclusions from this analysis were the following:

1. There is a large number of research papers that show evidence of the application of IV to software systems and their evolution, but the number of research works that utilized VA for the analysis of software systems is reduced.
2. Most research that is concerned with the application of VA to software systems are focused on theoretical and methodological issues, although do not outline processes, models or architectures that could facilitate the design of tools to carry out the analysis of one or multiple system revisions.
3. A large number of research implement their proposals as plugins of some of the most popular IDEs, such as *Eclipse*.
4. The number of tool proposals that made use of web technologies is reduced (*e.g.*, only one of the research papers that were reviewed in the course of this investigation use those type of technologies).

5. The use of single views is the preferred option of most research, whether classified under the *Sys* or the *Evol* rubric.
6. Proportional similarities exist between the research classified as either *Sys* or *Evol* on the use of *Multiple* and *Multiple linked views*, although research works classified in the category *Evol* represent a greater number of data elements and more complex relationships when compared to research classified under the rubric *Sys*.
7. Companies use SCM and bug tracking tools to record and manage data related to SDME processes, and some of them use these tools in an integrated manner in order to have correlated information from bugs and revisions that could lead to better tracking of changes and project evolution.
8. The curriculum of several university courses does not include contents about the tools that are used empirically in software engineering processes.
9. There is no substantial evidence about the diffusion and transference of the results of research to industry, concerning the application of IV to software systems and their evolution.
10. Most of the simple visualizations used by the software industry or businesses are, in their majority, integrated into SCM and IDE tools, but programmers are not aware of the options that these tools have available.
11. Tasks such as debugging, the navigation of dependencies, the detection of source code clones, refactoring, tracking changes and contributions and SQA metrics monitoring are carried out in the industry without the support of visualization tools.
12. There exist many general descriptions of the process followed by SEV, but a process for the application of VA to SE has not been described.

Following these conclusions, the EVSA process was defined to fulfill the absence of an adequate description of the process involved in the application of VA to SE [González-Torres 2013b, González-Torres 2013a]. This process was subsequently validated by means of implementing an architecture that followed its description. Accordingly, the description of the EVSA process and its validation was divided into three stages:

1. The definition of basic pieces that made possible to construct the characterization of the EVSA process. These building pieces included the explanation of SE terms and concepts, the advanced data analysis process and the definition of the VA process.
2. A detailed analysis and discussion of the use of visualization in software

systems which allowed to determine the tasks supported, data elements and visualization types used in industrial and academic settings.
3. The definition, description and validation of the EVSA process.

The definition of the EVSA process provided details about the principal components (and how they are interrelated and interact together), methods and techniques involved in the transformation of data derived from the analysis of the evolution of software systems into useful knowledge in order to facilitate a deeper understanding of the dynamic that occurs during SDME processes.

Based on the description of the EVSA process, the architecture of *Maleku* was designed and implemented to test whether its implementation was feasible and whether it could be used as a basis for the definition of tool architectures to support SDME processes (see chapter 9). The design of *Maleku* identified and explained the roles, boundaries and interactions between modules, components, and some of the methods and techniques that could be utilized by such process, and allowed to answer the subsidiary research questions.

Thereafter, the validation of *Maleku* was conducted in three steps and had as its main goal the verification of compliance with objectives and functionality, congruent with the design of its architecture. This verification allowed to demonstrate the usefulness of the tool. The three steps of the validation process were carried out as follows:

1. Test the tool in use case scenarios to evaluate its functionality (chapter 9).
2. Case Study on RT and its relation with an commercial SCM tool (chapter 10).
3. User assessment test with expert users (chapter 11).

This provided proof that the description of the process can be followed effectively and thus permit the design and implementation of an architecture to promote the understanding of SE by programmers and project managers, and to support SDME processes according to the goals, tasks and objectives specified.

Furthermore, this ended up confirming the validity of the architecture implemented by means of use case scenarios, a case study and a user assessment test. Stated briefly, the aim of this validation (which was to validate the complete cycle of applying VA to SE in the design and implementation of a tool to support developers and project managers in the development and maintenance of software) was fulfilled and the main research question was answered.

Overall, the usefulness of *Maleku* was demonstrated in providing statistical information in an appropriate manner about the revisions as well as the contributions made by programmers, the evolution of the project structure, the lifelines of software items (including packages, files, classes and interfaces), the evolution of inheritance and interface implementations, evolution metrics, and programmer collaborations. The results of the validation of *Maleku* showed its usefulness to:

1. Aid the comprehension of changes in software quality metrics as well as socio-technical and collaboration relationships during the project evolution or a particular period of time.
2. Assist in the process of understanding changes in software project structures, inheritance and interface implementation for the complete project or a given time period.
3. Support the understanding of changes during software project evolution by comparing time periods.

Consequently, it can be concluded that the definition and description of the EVSA process and its validation were both satisfactory, but it is also important to take into account the following conclusions for the implementation of tools based on this process:

1. The design of tools based on the EVSA could experience several challenges to adequately represent the evolution of software systems. The correlation of information in time adds many additional dimensions to the visual representation that are in correspondence with the number of time units that needs to be depicted. The representation of inheritance for a single system revision requires to model the inheritance tree once, whereas the representation of multiple system revisions could require to compare the inheritance tree to show the differences between each particular revision. Thus, this represent a challenge which complexity is proportional to the size of the inheritance tree and the number of revisions that are involved in the analysis. Although the representation of the temporal dimension in *Maleku* was addressed successfully, the design of the visualizations required to carry out several tests to verify its scalability and usefulness in the analysis of large to medium size open source systems.
2. The design and implementation of tools similar to *Maleku* needs to be prepared for the analysis of large datasets associated to the evolution of systems. The understanding of legacy systems whose evolution has been carried for several years implies that historical data needs to be analyzed and compared. Thus, this type of analysis could take several

hours or even days. In addition, an architecture design should also be prepared to incorporate new data and visual elements when revisions are created, and to address the aggregation of data in a proper manner.
3. Some possible reasons that may have an adverse effect on the adoption of these type of tools are visual stress caused by the visualizations, inadequate design, the complexity of the representations, the time needed to learn how to use the tools; the requirement of prior knowledge and experience of visual tools, as well as aspects related to the lack of clarity and ambiguity of the designs. Therefore, it is recommended that users become active participants throughout the design process of visualization tools on this regard: beginning with requirement elicitation, then design, and later brainstorming reviews, evaluations and usability studies.

Finally, the use of visual tools for assisting programming and management tasks needs to be sponsored by key players in the software industry (e.g., Microsoft, IBM and Borland), incorporating complete toolsets into their IDEs,SCM and bug tracking tools and creating training courses and technical documentation that takes them into account as central elements.

12.3 Publications Related to the Thesis

The research carried out has produced several publications in international conferences and journals that support the work presented in this dissertation. The list of these publications is the following:

Journals

1. González-Torres, A., García-Peñalvo, F. J., Therón, R. Human Computer Interaction in Evolutionary Visual Software Analytics. Computers in Human Behavior, vol. 29, no. 2, pages 486-495 (March 2013) ISSN: 0747-5632 (Impact Factor: 2.273).

2. González-Torres, A., García-Peñalvo, F. J., Therón, R. How Evolutionary Visual Software Analytics Supports Knowledge Discovery. Journal of Information Science and Engineering, vol. 29, no. 1, pages 17-34 (January, 2013) ISSN: 1016-2364 (Impact Factor: 0.333).

3. García, J., Gómez-Aguilar, D. A., González-Torres, A., García-Peñalvo, F. J., Therón, R. A Middleware Framework to Create Data Structures for a Visual Analytics Object Oriented Approach. International Journal

of Knowledge and Learning, Vol. 6, no. 2/3 pages 256-267 (2010) ISSN: 1741-1009.

Master Thesis

1. González-Torres, A. Representación Visual de Sistemas de Software: Evolución y Colaboración. Master's thesis, Universidad de Salamanca, June 2014.

Conferences, Symposiums and Workshops

1. González-Torres, A., García-Peñalvo, F. J., Therón, R. A Framework for the Evolutionary Visual Software Analytics Process. In Information Systems, E-learning, and Knowledge Management Research. Series: Communications in Computer and Information Science. Berlin, Heidelberg: Springer Verlag. VOL. CCIS 278, pages 439-447 (2013). ISBN 978-3-642-35878-4.

2. González-Torres, A., Therón, R., García-Peñalvo, F. J., Wermelinger, M., Yijun, Y. Maleku: An Evolutionary Visual Software Analytics Tool for Providing Insights into Software Evolution. In Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM), USA, pages 594-597 (September 2011) ISBN: 978-1-4577-0663-9.

3. García, J., Gómez-Aguilar, D. A., González-Torres, A., Therón, R., García-Peñalvo, F. J. A Visual Analytics Tool for Software Project Structure and Relationships among Classes. In Proceedings of the 10th International Symposium on Smart Graphics, SG'09, Berlin, Heidelberg: Springer-Verlag. LNCS 5531, pages 203-212 (2009). ISBN 978-3-642-02114-5.

4. González-Torres, A., Therón, R., Telea, A., García-Peñalvo, F. J. Combined Visualization of Structural and Metric Information for Software Evolution Analysis. In Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, IWPSE-Evol'09, New York, NY, USA, ACM. pages 25-30, (2009). ISBN: 978-1-60558-678-6.

5. González-Torres, A., Gómez-Aguilar, D. A., Segrera-Francia, S., Therón, R., Moreno-García, M. N., García-Peñalvo, F. J. Uncovering the Relationships among Clases and Packages in Software Evolution. In Proceedings of the Web Mining and Semantic Web Workshop, MiWebSe '08, Departamento de Informática y Automática, Universidad de Salamanca. pages. 11-20, (September 2008). ISBN: 978-84-691-5945-3

6. Therón, R., González-Torres, A., García-Peñalvo, F. J. Supporting the Understanding of the Evolution of Software Items. SoftVis '08: Proceedings of the 4th ACM symposium on Software visualization, Ammersee, Germany, ACM. pages 189-192, (October 2008). ISBN:978-1-60558-112-5.

7. Therón, R., González-Torres, A., García-Peñalvo, F. J. The Use of Information Visualization to Support Software Configuration Management. Berlin, Heidelberg: Springer-Verlag. LNCS 4663, pages 317-331, (2007) ISBN13: 9783540747994 ISBN10: 3540747990.

8. Therón, R., González-Torres, A., García-Peñalvo, F. J. Visualización de la Colaboración en la Evolución de un Ítem de Software y la Estructura de las Baselines. VIII Congreso Internacional de Interacción Persona-Ordenador, Interacción 2007 (AIPO), II Congreso Español de Informática CEDI'2007, Zaragoza, Spain. pages 69-78, (September 2007). ISBN 978-84-9732-596-7.

9. González-Torres, A., Therón, R., García-Peñalvo, F. J. Visualización de la Colaboración en la Evolución de Items de Software con Base en los Repositorios de las Herramientas SCM. Informe técnico DPTOIA-IT-2007-002, Departamento de Informática y Automática, Universidad de Salamanca. Septiembre (July 2007)

12.4 Future Research

Cloud computing has become popular in the past few years and in consequence, many services based on the cloud are currently being offered. So, a great number of companies are using these services to perform different business tasks. The software industry is not an exception and recently several cloud based IDEs have been made available (*e.g.*, *Codenvy*, *Cloud 9* and *Code Anywhere*). This kind of tools could also be considered in a near future as an alternative to support GSD processes. However, there is no reliable information available about the use of these IDEs at a current time neither about projections of their future use.

An additional point to be taken into account is that web based versions of SCM and bug tracking tools have been in place since many years, and thus, these could be migrated to the cloud or communicated with cloud based IDEs using software interfaces.

The aforementioned trend opens an opportunity to contribute with programmers and project managers that use software development tools

based in the cloud. Accordingly, future work will be carried out to design and implement an architecture based in the EVSA process to be integrated as a plugin into a selected cloud IDE (this architecture could include some components from *Maleku*). Two fundamental requirements of such architecture is that its design will be responsive and use restful web technologies. Furthermore, it will take into account the use of tools of the Hadoop ecosystem to process and store data in a timely and secure fashion.

In addition, the architecture will support the dynamic analysis of system at run-time, when it is applicable because of the programming language of the system under analysis. Moreover, data analysis modules will support at least three of the most common programming languages in use nowadays. The comments made by participants in the usability study in this research will be used as an input for the design of the visualizations (or re-implementation of visualizations used in *Maleku*), interaction support and the tool in general.

Concerning the systematic mapping study presented in chapter 4, it will be expanded to cover research works that have been published since the inception of SEV. Similarly, a detailed study of research projects related to methods and techniques for data extraction and analysis, taking as a reference point the work of Kagdi [Kagdi 2007a], will be carried out.

Finally, the use of visualization tools (during the development and maintenance process) in both the software industry and software development departments of enterprises, will be monitored. To this end, an improved version of the survey introduced in this research will be conducted every two years. The improved survey will include a greater number of topics and participants.

Appendix A

Papers Published per Venue

Table A.1: Papers published per venue.

Venue	Publication venue	Acronym	Year							Total
			2007	2008	2009	2010	2011	2012	2013	
Journal	Computer Graphics Forum	CGF						1	2	3
	Computational Statistics	COMPUSTAT					1			1
	Electronic Notes in Theoretical Computer Science	ENTCS			2					2
	Empirical Software Engineering	ESE						1	1	2
	International Journal of Computers Communications & Control	IJCCC					1			1
	International Journal of Software Engineering and Its Applications	IJSEIA					1		1	2
	International Journal on Software Tools for Technology Transfer	IJSTTT					1			1
	Information and Software Technology	IST						1	1	2
	Information Technology Journal	ITJ							1	1
	Information Visualization	IV			2	1	1	6		10
	Journal of Software Maintenance and Evolution: Research and Practice	JSME			1	1				2

Continued on next page.

Table A.1 Papers published per venue – continued from previous page

Venue	Publication venue	Acronym	Year							Total
			2007	2008	2009	2010	2011	2012	2013	
	Journal of Visual Languages & Computing	JVLC			2	1	1			4
	Science of Computer Programming	SCP				1		1		2
	Software & Systems Modeling	SoSyM					1			1
	ACM Transactions on Computing Education	TOCE			2	1				3
	IEEE Transactions on Software Engineering	TSE		1	1					2
	IEEE Transactions on Visualization and Computer Graphics	TVCG			2		2			4
									Total journal papers	43

Continued on next page.

Table A.1 Papers published per venue – continued from previous page

Venue	Publication venue	Acronym	Year							Total
			2007	2008	2009	2010	2011	2012	2013	
Conference	Australasian Computer Science Conference	ACSC			2		2			4
	Asia - Pacific Software Engineering Conference	APSEC					1			1
	International Working Conference on Advanced Visual Interfaces	AVI			1					1
	Canadian Conference on Electrical and Computer Engineering	CCECE			1					1
	Conference on Human Factors in Computing Systems	CHI				1				1
	European Conference on Software Maintenance and Reengineering	CSMR							1	1
	Annual Computer Software and Applications Conference	COMPSAC					1			1
	International Conference on Supporting Group Work	GROUP	1							1
	International Conference on Information Visualisation	ICIV						1		1
	International Conference on Program Comprehension	ICPC			2	1		1		4

Continued on next page.

Table A.1 Papers published per venue – continued from previous page

Venue	Publication venue	Acronym	Year							Total
			2007	2008	2009	2010	2011	2012	2013	
	International Conference on Software Engineering	ICSE					1	2	1	4
	Interacção - Conferência Interacção Pessoa-Máquina	INTERACCAC		1						1
	Working Conference on Software Visualization VISSOFT	VISSOFT/ SOFTVIS							9	9
	Working Conference on Reverse Engineering	WCRE			1	1				2
	International Conference on 3D Web Technology	WEB3D							1	1
									Total conference papers	30

Continued on next page.

Table A.1 Papers published per venue – continued from previous page

Venue	Publication venue	Acronym	Year							Total
			2007	2008	2009	2010	2011	2012	2013	
Symposium	Symposium on Software Visualization	SOFTVIS	15		19					34
	International Symposium on the Foundations of Software Engineering	FSE			1			1		1
	Symposium on Applied Computing	SAC			1					1
	International Symposium Visual Information Communication	VINCI					1			1
	International Symposium on Visualization for Cyber Security	VIZSEC			1	1				2
									Total symposium papers	40
Workshop	International Workshop on Visualizing Software for Understanding and Analysis	VISSOFT	16		7		9			32
	International Workshop on Cooperative and Human Aspects of Software Engineering	CHASE				1	1			2
	Workshop on Managing Technical Debt	MTD					1			1
	Workshop on Program Analysis for Software Tools and Engineering	PASTE				1				1

Continued on next page.

Table A.1 Papers published per venue – continued from previous page

Venue	Publication venue	Acronym	Year							Total
			2007	2008	2009	2010	2011	2012	2013	
Total workshop papers										36
Total number of papers per year			17	17	26	30	24	17	18	149

Appendix B

Correlation of Research Approaches and Papers

Table B.1: Philosophical Research: Correlation of research approaches.

Research approach	Analysis type	References
Case study	Sys	[Sundararaman 2008]
	Evol	
Classification schema or taxonomy	Sys	
	Evol	[Myller 2009, Xu 2009]
Evaluation	Sys	[Sensalire 2008, Xie 2008, Xie 2009, Ruan 2010]
	Evol	
Framework	Sys	[Gallagher 2008]
	Evol	[von Pilgrim 2008, Pilgrim 2009, Telea 2011] [Madhavi 2011]
Lessons learned	Sys	[D'Ambros 2011]
	Evol	[Sensalire 2009]
Novel technique	Sys	[Holten 2007, Parnin 2008, Caserta 2011b]
	Evol	[D'Ambros 2009b, Parnin 2010, Ogawa 2010] [Beck 2011, Burch 2011, Erra 2012a] [Erra 2012b, Minelli 2013]
Reflections or discussion	Sys	[Parnin 2007, Béron 2008, Moons 2009]
	Evol	[Rilling 2007, D'Ambros 2009a, Ogawa 2009] [Park 2009, Petre 2010, Walny 2011]
Study	Sys	[Sharif 2009a, Sharif 2013]
	Evol	[Boccuzzo 2007, Teyseyre 2009] [Feigenspan 2013]
Survey	Sys	
	Evol	[Kienle 2007, Caserta 2011a, Ben-Ari 2011]
Systematic mapping study	Sys	
	Evol	[Novais 2013]

Continued on next page.

Table B.1 Correlation... – continued from previous page.

Research approach	Analysis type	References
Technique improvement	Sys	[Eichelberger 2008, Ploeger 2008] [Rufiange 2012]
	Evol	[Laval 2009]

Table B.2: Solution Proposal: Correlation of research approaches.

Research approach	Analysis type	References
Detect design flaws	Sys	[Wettel 2008b, Murphy-Hill 2010, Murphy-Hill 2013]
	Evol	[Garousi 2010]
Distributed systems comprehension	Sys	[Cosma 2007, Zeckzer 2008, Pauw 2009, Femmer 2011, Pauw 2013]
	Evol	
Improve software quality	Sys	[Erdemir 2011, Risi 2012, Gouveia 2013]
	Evol	[Bohnet 2011]
Improve source code security	Sys	[Liebrock 2009, Goodall 2010]
	Evol	
Memory allocation analysis	Sys	[Moreta 2007, Reiss 2009, Aftandilian 2010, Myers 2010, Robertson 2010, Choudhury 2011, Kelley 2013, Rosen 2013]
	Evol	
Multithreading execution analysis	Sys	[Trümper 2010, Reiss 2010, Karran 2013, Sigovan 2013, Reiss 2013]
	Evol	
Parallel execution analysis	Sys	[Sigovan 2013]
	Evol	[Bernardin 2008]
Performance analysis	Sys	[Schaeckeler 2009, Pauw 2010, Zimmer 2010]
	Evol	[Alcocer 2013]
Program execution analysis	Sys	[Deelen 2007, Bohnet 2009a, Telea 2009b, Cornelissen 2009, Noda 2009, Helminen 2010, Lin 2010, Wu 2010, Bennedsen 2010, Maoz 2011, Cisar 2011, Ishio 2012]
	Evol	[Adamoli 2010]
Software design and modeling	Sys	[Kagdi 2007b, Harel 2008, Frisch 2010, Frisch 2013]
	Evol	
Software ecosystem comprehension	Sys	
	Evol	[Lungu 2010, Neu 2011]
Software testing	Sys	[Cottam 2008, Shi 2011]
	Evol	[Chan 2010]

Continued on next page.

Table B.2 Correlation... – continued from previous page.

Research approach	Analysis type	References
Support reverse engineering	Sys	[Telea 2008, Quist 2011]
	Evol	[Telea 2009a]
System analysis and understanding	Sys	[Wettel 2007, Bohnet 2007, Shah 2008, Zhang 2009, Islam 2010, Deng 2011]
	Evol	[Telea 2009c, Novais 2011, Novais 2012, Servant 2012]
System refactoring	Sys	[Hermans 2013]
	Evol	[Servant 2010, Vanya 2012]
Team awareness and collaboration	Sys	[Assogba 2010]
	Evol	[de Souza 2007, Ripley 2007, Nestor 2008, Kuhn 2010b, Gómez 2010, Jermakovics 2011, Hattori 2012, Kuhn 2012, Anslow 2013, Benomar 2013]
Understand dependencies	Sys	[Holmes 2007, Cassell 2011]
	Evol	
Understand software changes	Sys	[Broeksema 2011]
	Evol	[D'Ambros 2007b, Bohnet 2009b]
Understand system architectures	Sys	[Ali 2009, Anslow 2010, Bentrad 2013, Luo 2013]
	Evol	[Sawant 2007, Hindle 2007, Beck 2010, Kuhn 2010a, Steinbrückner 2010, Wettel 2011, Reniers 2012, Limberger 2013, Beck 2013, Abuthawabeh 2013, Steinbrückner 2013]

Appendix C

Resumen de la Tesis

Estando Güindy y Cucho al borde del precipicio, decidieron subir a una pequeña montaña que habían dejado atrás hacía unos minutos. Al llegar a la cima se estiraron, gritaron y aúllaron. Volvieron a ver el camino por el que habían venido, luce como una serpentina entre árboles. Vuelven a gritar y aullar, Cucho no sabe por qué, pero lo cierto es que Güindy se había dado cuenta que podrían haber llegado al mismo sitio tomando un camino tres veces más corto. — A.González

Contents

C.1 Introducción

Los sistemas de software son elementos que están omnipresentes en la vida cotidiana y se utilizan en casi todos los dispositivos que son usados por las personas y las empresas. Las personas usan estos dispositivos y el software asociado para hacer algunas de las siguientes cosas [Charette 2005]:

1. Trabajar.
2. Aprender (asistir a cursos en línea, leer e investigar temas de interés).
3. Entretenerse (jugar, ver televisión o videos, escuchar música).
4. Comunicarse (amigos, familia, compañeros de trabajo, participar en foros, colaborar y reuniones de trabajo).
5. Hacer compras.
6. Hacer trámites (bancarios, impuestos y pagos).
7. Teletrabajo.

Esta ha sido la realidad social de los últimos años, es la realidad social actual y seguirá siendo la realidad social cotidiana en el futuro.

La omnipresencia de la tecnología ha llevado a algunos individuos y organizaciones a ser social y económicamente dependientes de los sistemas de software [Boehm 1999b]. Como resultado, el porcentaje de presupuesto que las empresas invierten cada año en esta área, en promedio, es mayor de 4% [Charette 2005, Hall 2013], y el gasto en productos de software crece cada año [Gartner 2013, Gartner 2014].

Dado este entorno, el mercado de software es muy atractivo para los inversores, lo cual tiene como consecuencia que la competencia entre los productores sea intensa. Por tanto, es necesario tener en cuenta que un producto es valioso si permite o ayuda a las personas y las organizaciones a lograr sus metas y objetivos particulares por medio de su uso [Boehm 1999b].

La industria del software en general, y de forma más específica, los departamentos de desarrollo de software internos (que son conscientes de esta situación) buscan desarrollar productos que cumplan con los requisitos y funcionalidad demandadas por los usuarios, con altos estándares de calidad, en el menor tiempo y al menor costo posible. Esta afirmación también es válida en relación con el software de código abierto [Lee 2009].

En este punto, es conveniente tener en cuenta que el desarrollo y mantenimiento de software son procesos complejos en los cuales la posibilidad de fallo está presente en todas las etapas y niveles del proceso de [Kraut 1995, Procaccino 2002, Morisio 2002, Chow 2008]. Al respecto, existen informes anuales que se publican con respecto a este problema [Group 2013] con el fin de evaluar el desempeño general de la industria del software y las tasa de éxito de los proyectos en particular. Se debe mencionar que se han publicado

estudios sobre casos famosos de proyectos cuyo fracaso ha costado la pérdida de millones de dólares [Charette 2005] y que además han provocado la pérdida de prestigio de algunas empresas que han participado en esos proyectos.

Con estos ejemplos en mente, se han hecho grandes esfuerzos para mejorar los elementos del proceso de desarrollo, tales como el cálculo de costos y riesgos, la planificación, la reutilización de los componentes de software [García-Peñalvo 2000, Garcıa-Peñalvo 2002, Laguna 2003] y el diseño y mantenimiento de sistemas [Boehm 1999b, Sullivan 2001, Royce 2009].

Muchos de estos esfuerzos tienen como objetivo mejorar la capacidad técnica de las personas y los procesos con el fin de obtener mejores resultados económicos [Boehm 2000, Colomo-Palacios 2013, Buxmann 2013, Colomo-Palacios 2014]. Cabe indicar que en términos económicos, las técnicas de ingeniería de software tienen valor si facilitan el desarrollo de software más valioso [Boehm 1999b].

Teniendo en cuenta este contexto más general, esta tesis pretende contribuir a la economía del software mediante el apoyo al proceso Desarrollo y Mantenimiento de Software (DMS) a través de la descripción del proceso Análitica Visual (AV), con el fin de facilitar el análisis de los sistemas de software y su evolución.

Algunos factores que influyen en el éxito o fracaso de los proyectos de software que se tienen en cuenta en esta investigación son los siguientes:

1. Control y seguimiento de la calidad del software por medio del uso de métricas [Niazi 2006, Lee 2009, Nasir 2011].
2. La importancia del control y la configuración de cambio [Nasir 2011].
3. La ubicación distribuida (a veces en diferentes países y continentes) de los colaboradores en los proyectos y su nivel de conocimiento de las actividades y los cambios llevados a cabo por el personal del proyecto [Niazi 2006, Fabriek 2008].
4. La necesidad de contar con herramientas automáticas confiables para obtener información sobre el proceso DMS y los cambios que se han realizado en el código fuente del proyecto [Niazi 2006].

C.1.1 Problema de investigación

El objetivo de Análisis de la Evolución de Software (AES) es apoyar a los administradores de proyectos y programadores durante el proceso de cambio y evolución de los sistemas de software en diferentes localidades geográficas [Estublier 1999, Ogawa 2009]. Los administradores de proyectos deben tener una visión general del proceso que les permita controlar la calidad del software; evaluar la productividad; reducir los riesgos de implementación

y mantenimiento, así como tener la capacidad informar sobre todas estas actividades a los niveles superiores de dirección. En tanto los programadores necesitan aprender las nuevas bases de código para entender los cambios estructurales, así como los cambios en la herencia y la implementación de interfaces. Por otra parte, los programadores necesitan entender las dependencias de los elementos de software y comprender las diferencias de las revisiones de código fuente y tener acceso al historial del desarrollo.

AES tiene como fin contribuir de forma activa con los procesos de software a través del análisis y apoyo a la realización de cambios, comprensión de la complejidad, el crecimiento y control de la calidad [Lehman 1997]. Sin embargo, AES produce conjuntos de datos grandes y complejos, debido a que el número de variables que intervienen en el proceso de evolución y las complejidades de sus relaciones son difíciles de entender por parte de los seres humanos. Debido a los motivos expuestos, conviene considerar que aunque AES proporciona elementos valiosos de información, no provee de conocimiento suficiente para llevar a cabo de forma satisfactoria las tareas de comprensión de los cambios y la evolución del software.

C.1.2 Análitica Visual y Mantenimiento de Software

Aunque el resultado del análisis de la evolución de los sistemas de software proporciona información útil, no ofrece información suficiente para llevar a cabo las tareas de comprensión de los cambios al sistema de una manera satisfactoria para apoyar de forma adecuada a los desarrolladores y administradores de proyectos. Por lo tanto, esta investigación ha tenido en cuenta el importante papel que ha tenido Visualización de la Información (VI) en los últimos años al proporcionar conocimiento a partir de grandes y complejos conjuntos de datos mediante el uso de representaciones visuales combinadas con técnicas de interacción.

Debido a lo anterior, los esfuerzos de un sector de la comunidad científica que investigación sobre métodos de ingeniería de software se han centrado en el uso de representaciones visuales con técnicas de interacción para facilitar la comprensión de los sistemas de software y su evolución. Estos esfuerzos de investigación se han centrado en el uso de Visualización de Software (VS) [Diehl 2007] y Visualización de la Evolución de Software (VES) [Voinea 2007]; aunque de forma más reciente algunos esfuerzos de investigación se han centrado en la aplicación de AV a los sistemas de software[1].

[1]La aplicación deAV a los sistemas de software se conoce comoAnalítica Visual de Software (AVS) [Telea 2011], y de forma más reciente también se ha aplicado a la evolución de los sistemas de software [González-Torres 2013a, González-Torres 2013b], con el objetivo

En este punto, merece la pena mencionar que AV combina las fortalezas de las máquinas con las humanas, como la capacidad de análisis, la intuición, la resolución de problemas y la percepción visual. Por lo tanto, las personas están en el corazón de AV [Dix 2010] y Interacción Persona-Ordenador (IPO) es un componente clave para apoyar el descubrimiento de conocimientos. Este es un proceso cuyo objetivo es proporcionar conocimiento a partir de la enorme cantidad de datos científicos, forenses, académicos o de la empresas que son almacenados por formatos heterogéneas como bases de datos, archivos HTML, XML, metadatos y código fuente.

Este proceso recopila información de forma iterativa, realiza el preprocesamiento de datos, efectúa análisis estadístico [Peck 2011], lleva a cabo minería de datos, usa aprendizaje automático [Witten 2005], representación del conocimiento [van Harmelen 2007], interacción del usuario [Sharp 2011], representaciones visuales [Leung 1994a, Johnson 1991, Robertson 1991], la cognición y percepción humana, la exploración y las capacidades humanas para la toma de decisiones [Keim 2006, Llorá 2006].

AV se ha aplicado de forma amplia a problemas tan diversos como la gripe aviar [Proulx 2006], las condiciones paleoceanográficas [Therón 2006c], el análisis organizacional [Card 2006], eLearning [Gómez-Aguilar 2009, Gómez-Aguilar 2015b], la toma de decisiones [Migut 2011, Savikhin 2008], ingeniería de ontologías [García 2012, García-Peñalvo 2012c, García-Peñalvo 2014], patrones temporales [Weaver 2006, Ziegler 2010], las redes sociales [Perer 2011], el análisis de la seguridad [Harrison 2011] y los sistemas de software [Reniers 2012, González-Torres 2013b]. Por lo tanto, se puede decir que el descubrimiento de conocimiento es una propiedad intrínseca de AV, debido a que está destinada a apoyar a los analistas y tomadores de decisiones en la obtención de conocimiento a partir de grandes conjuntos de datos multivariantes [Thomas 2005].

En consecuencia, AV puede ofrecer soluciones al problema que conlleva brindar apoyo efectivo a programadores y administradores de proyectos durante los procesos de implementación de sistemas, teniendo en cuenta que es un proceso que ofrece un enfoque integral que incluye desde la recuperación de información y análisis hasta la representación visual de los resultados del análisis. Además, ofrece la posibilidad de explorar diferentes niveles de detalle utilizando múltiples representaciones visuales, enlazadas y coordinadas entre sí mediante el uso de técnicas de interacción [North 2000]. Esto permite facilitar el descubrimiento de relaciones y conocimiento por medio del razonamiento

de proporcionar mejores resultados. El objetivo de la investigación en esta área es apoyar el proceso de comprensión de los sistemas de software y mejorar el diseño y estrategias de implementación de herramientas dirigidas a satisfacer las necesidades de análisis de los programadores y administradores de proyectos.

analítico del analista.

Teniendo en cuenta estos factores positivos, se puede decir que una de las propiedades de AV es la capacidad de proporcionar apoyo a la toma de decisiones [Savikhin 2008, Mane 2012] utilizando las capacidades cognitivas de los usuarios, por lo cual su aplicación a la evolución de software ofrece grandes oportunidades para apoyar tanto a los programadores como a los administrador de proyectos. Sin embargo, la aplicación de AV a la evolución del software es nueva y las tareas y necesidades de información de los programadores y administradores de proyectos son complejas [Forsberg 2005, de Oliveira Barros 2004, Munch 2004, Paul 1999].

Esto implica que todavía existe un gran número de desafíos que deben ser superados para apoyar con éxito a los administradores de proyectos en la toma de decisiones. Entre esos retos se encuentran los siguientes:

* Facilitar el análisis visual y la evaluación del proceso de desarrollo.
* Proporcionar métodos para controlar visualmente la evolución de la calidad de los elementos de software (clases, paquetes y módulos), teniendo en cuenta el uso de métricas de calidad del software. Lo anterior con el objetivo de mantener la complejidad y la evolución del sistema bajo control, así como asegurar el control de calidad.
* Proporcionar mecanismos visuales para revisar las medidas de ejecución de tareas, y permitir el análisis de rendimiento y la predicción de los avances.
* Apoyar mediante el uso de métodos visuales la gestión de riesgos, el crecimiento y complejidad del ăă producto de software.
* Mantener informados a los administradores de proyectos sobre los patrones de colaboración entre desarrolladores y sobre los elementos que se han modificado de forma sincróna o asincróna, así como sobre las consecuencias (en términos de calidad y funcionalidad) de los cambios que se han llevado a cabo.

Mientras que los desafíos que enfrenta AV para apoyar a los programadores en la comprensión de Evolución de Software (ES), de acuerdo con sus necesidades de información [Sillito 2006b] son las siguientes:

* Ofrecer detalles de los elementos de software a los que están efectuando cambios.
* Proporcionar información sobre los componentes de software que son modificados de forma simultánea por otros programadores.
* Permitir que los programadores comprendan las implicaciones de los cambios realizados con base en las relaciones (herencia y implementación de la interfaz) y asociaciones entre los elementos de software

(composición, de referencia, y de acoplamiento), así como el efecto de la colaboración entre los objetos.
* Proporcionar detalles sobre la creación de variables así como el acceso y modificación de datos por medio de argumentos a los métodos y variables globales.
* Facilitar que los programadores analicen y comparen para dos o más revisiones el control de flujo, la ejecución y la gestión de excepciones entre revisiones.
* Facilitar la identificación de diferencias entre archivos, elementos de software y tipos en varias revisiones.

Merece la pena recordar que DMS [Colomo-Palacios 2012] cubre un alto porcentaje del costo de los sistemas de software modernos [Koschke 2003].

Los conjuntos de datos que se derivan del mantenimiento de software comparten muchos puntos en común con los conjuntos de datos de *Big Data*:

1. Grandes volúmenes de datos que pueden contener campos sin valores (*e.g.*, millones de líneas de código fuente [Baker 1995, Kagdi 2007a] y miles de componentes de software [D'Ambros 2008]).
2. Conjuntos de datos con tipos híbridos y complejos (*e.g.*, grandes bases de datos de métricas de programas, documentos de diseños, resultados de pruebas, logs con registros sobre la ejecución de los sistemas, reportes de fallos [Hassan 2005, Lanza 2005b], atributos numéricos, de categorías o texto, interconectados por numerosos tipos de relaciones como herencia, jerarquía de la estructura, y flujos de llamadas, control y dependencias).
3. Conjuntos de datos que evolucionan con el tiempo (*e.g.*, las revisions de los sistemas de software almacenados en los repositorio de las herramientas Administración de la Configuración de Software (ACS) [Mens 2008].

Las tareas de comprensión de los programas [Koschke 2003] siguen un patrón preciso para la construcción de sentido mediante la creación, refinamiento y validación de hipótesis, algo común en común enAV [Sun 2004, Thomas 2005, Thomas 2006]. Finalmente, las herramientas para la comprensión de programas se basan en la misma combinación de elementos que se basa el análisis de software [Koschke 2003] y los components de VS [Diehl 2007].

La aplicación de AV a ES es un desarrollo reciente, y se requería describir con claridad dicho proceso e identificar los factores, métodos y técnicas que contribuyen con él.

C.1.3 Objetivos y Preguntas de Investigación

La intención de la investigación es definir un proceso para describir y explicar la aplicación de AV a ES. El objetivo es ofrecer orientación en el diseño e implementación de herramientas de software para ayudar a los programadores y administradores de proyectos en el desarrollo y mantenimiento de software. Además, esta investigación también busca contribuir con la comunicación y la comprensión de la investigación llevada a cabo por otros investigadores mediante la descripción de dicho proceso. En consecuencia, la pregunta de investigación principal que plantea esta tesis es la siguiente:

¿Cómo se puede definir de forma adecuada un proceso para describir y explicar la aplicación de la Analítica Visual a la Evolución del Software?

La definición de un proceso, como se indica en la pregunta anterior, requiere, por un lado la explicación de como se aplica AV a ES, y por otro lado, la identificación de los roles, fronteras, interacciones y relaciones de los componentes, métodos y técnicas que intervienen en dicho proceso. Siguiendo este enfoque, las siguientes son las preguntas de investigación subsidiarias para ayudar a explicar y describir de manera adecuada dicho proceso:

1. ¿Cómo están interrelacionados e interactúan los componentes que conforman el proceso de aplicación de la Analítica Visual a la Evolución de Software?
2. ¿Cuál es la composición de los componentes (en términos de métodos y técnicas, funciones e interacciones) en el proceso de aplicación de la Analítica Visual a la Evolución del Software?

Las preguntas de investigación anteriores son necesarias para identificar los componentes, métodos y técnicas que intervienen en el proceso de aplicación de AV a ES, y para caracterizar los roles, relaciones e interacciones entre estos elementos. Además, la utilidad de este proceso en el diseño e implementación de herramientas debe ser probada, como un elemento necesario para responder a la pregunta de investigación planteada. Sobre la base de lo anterior, la siguiente pregunta de investigación subsidiaria ha sido formulada:

3. ¿Cómo se puede probar que la descripción del proceso se puede seguir de manera efectiva al diseñar e implementar una arquitectura para apoyar la comprensión de ES por parte de los programadores y administradores de proyectos?

Para responder a la pregunta 3, es necesario implementar una arquitectura, con base en el proceso de aplicación de AV a ES. Dicha arquitectura debe tener en cuenta los problemas descritos en las secciones anteriores en cuanto a las necesidades de los administradores de proyectos y programadores. Por lo tanto, la implementación de esta arquitectura debe abordar las siguientes preguntas de investigación:

3.1 ¿Cómo pueden los administradores de proyectos de software ser apoyados en la toma de decisiones mediante la comprensión de los cambios en los indicadores de calidad del software, y las relaciones socio-técnicas y de colaboración durante la evolución del proyecto o un período de tiempo determinado?

3.2 ¿Cómo pueden los programadores ser apoyados en la comprensión de los cambios en los indicadores de calidad de software, las estructuras del sistema de software, la herencia y la implementación de interfaces para un periodo de tiempo determinado?

3.3 ¿Cómo pueden los programadores y administradores de proyectos recibir apoyo en la comprensión de los cambios durante la evolución de los proyectos de software mediante la comparación de períodos de tiempo?

Además, la arquitectura debe ser validada a través de una prueba de evaluación de usuario y escenarios de casos de uso. El objetivo de esta validación es probar el ciclo completo de la aplicación AV a ES en el diseño e implementación de una herramienta para apoyar a los programadores y administradores de proyectos en el desarrollo y mantenimiento de software.

C.1.4 Methodología y Organización de la Tesis and Outline

La metodogía utilizada en el desarrollo de la presente tesis es una adaptación del modelo de *Investigación Acción* [Kemmis 2005]. Las fases de esta metodología que fueron seguidas en esta investigación son 5 y se relacionan con los capítulos de la tesis de la siguiente forma:

Planificación y revisión de la planificación: Esta fase corresponde a la introducción de la tesis, así como a los capítulos 2 y3. En esta fase, las metas, los objetivos, preguntas de investigación y problema de investigación se definen y redefinen de forma constante. Del mismo

modo, los conceptos, términos y el proceso de análisis de ES, así como la definición del proceso de AV se revisan en cada iteración de la metodología. Es importante destacar que el objetivo de los capítulos 2 y 3 es definir algunos elementos básicos que tienen por objeto contribuir a responder la pregunta principal de la presente investigación.

Diagnóstico: En esta fase se lleva a cabo un análisis de los trabajos de investigación que han sido publicados y están relacionados con la aplicación de visualizaciones y AV a los sistemas de software (y su evolución). Durante esta etapa, se realizan encuestas a profesionales que trabajan en la industria del software. El objetivo de esta encuesta es obtener información con relación a la situación actual sobre el uso de herramientas de visualización. Posteriormente, utilizando los resultados obtenidos en dicha encuesta, y con el apoyo de las referencias bibliográficas pertinentes, se llevó a cabo una discusión detallada sobre el estado de la investigación en este campo y su impacto en la industria (tomando como punto de partida el uso actual de la visualización y AV para apoyar el proceso de desarrollo y mantenimiento).

En esta fase (chapters 4,5, 6 and 7) se identifican las tareas que son apoyadas por los trabajos de investigación en esta área, los elementos de datos y visualizaciones que son utilizadas en la academia y la industria, y por lo tanto, tiene como objetivo efectuar el diagnóstico de las necesidades que deben ser abordadas por la caracterización del proceso de aplicación de AV a ES.

Tomar acción: En esta fase, se define o redefine (de acuerdo con la iteración de la metodología) el proceso de aplicación de AV a ES. La especificación de una arquitectura, así como el diseño e implementación de una herramienta (que utiliza como base la arquitectura especificada) también se definen o redefinen en cada ciclo.

Evaluación: Tiene por objetivo llevar a cabo la evaluación indirecta de la definición del proceso de aplicación de AV a ES. La meta de la especificación e implementación de la arquitectura es probar la aplicabilidad de este proceso. Como consecuencia, la evaluación y validación de la herramienta que hace uso de esta arquitectura también evalúa y valida el proceso de la aplicación de AV a ES. Los capítulos 9, 10 and 11 presentan los resultados de la evaluación de la herramienta y están asociados a esta fase.

Análisis de los hallazgos: Esta es la última fase de un ciclo de investigación, y tiene por fin analizar los resultados de todo el ciclo

para presentar las principales conclusiones. Con base en los resultados de esta fase, se redefine el plan para el siguiente ciclo del proceso de investigación. Esta fase del modelo se asocia con el capítulo 12 de la presente tesis.

La investigación que se presenta y discute en esta tesis corresponde al primer ciclo de un proceso para proponer modelos y herramientas que contribuyan de manera eficaz con los procesos de desarrollo y mantenimiento de software mediante el uso de AV. De acuerdo con la metodología descrita, cuando se termina un ciclo del proceso de investigación los resultados conducen a un nuevo ciclo del proceso que inicia con una revisión del plan de investigación, las metas, objetivos, preguntas y el problema de investigación.

C.2 Un Proceso de Analítica Visual para la Evolución de Software

En esta sección se resume la definición del proceso de aplicación AV a ES, la cual tiene por objetivo responder a la siguiente pregunta de investigación:

¿Cómo definir el proceso de aplicación de la Analítica Visual a la Evolución de Software?

C.2.1 Analítica Visual y Sistemas de Software

La aplicación de los principios de AV a los sistemas de software [Telea 2010, Reniers 2012] se conoce como AVS [Anslow 2009, Telea 2011]. El uso de AV en este contexto es una mejora con respecto a VS, sobretodo si se tiene en consideración a AV como un proceso integral que incluye el análisis avanzado de datos y el uso de múltiples vistas vinculadas.

Por otra parte, la aplicación de los principios de AV a ES comparte elementos comunes con AVS, pero la principal diferencia entre ambos radica en que el primero tiene en cuenta dos o más revisiones, mientras que el segundo sólo tiene en cuenta el análisis de una revisión del proyecto de software.

La aplicación de AV a ES implica efectuar el análisis individual de cada revisión y luego requiere realizar un análisis adicional para comparar y correlacionar los resultados con el fin de descubrir relaciones, similitudes y diferencias entre estas relaciones, pero además tiene en cuenta que es necesario considerar los siguientes factores adicionales:

* Visualizar los diferentes tipos de datos en diferentes escalas de tiempo (años, meses, días y horas) y correlacionar los datos en estas escalas.

* La representación visual de los cambios estructurales de los sistemas de software es una tarea muy compleja.
* Los desarrolladores y los administradores de proyectos deben ser hábiles y prudentes para poder notar las relaciones y diferencias cuando se requiere analizar varias revisiones del proyecto para llegar a la solución de un determinado problema.

De acuerdo con lo anterior, es posible resaltar que la aplicación de AV a ES es una especialización de AVS. Una analogía práctica es que AVS es como un fotograma de una película, mientras que la aplicación de AV a ES es una película que se compone de un gran número de fotogramas ordenados e interrelacionados de forma temporal. En consecuencia, esta investigación define el proceso de aplicación de AV a ES como AVAES, cuya definición conceptual es la siguiente:

Evolutionary Visual Software Analytics es el proceso de aplicación de la Análitica Visual a la Evolución de Software con el fin de mejorar la comprensión de los cambios del sistema de software con la participación activa de los usuarios por medio de la Interacción Persona-Ordenador.

C.2.2 Evolutionary Visual Software Analytics

El proceso AVAES se describe en la figura C.1 y, en términos generales, esta descripción es compartida por los procesos AVS y AV. Por lo tanto, el proceso utiliza un enfoque modular, donde cada módulo es una colección de componentes que se encuentran conformados por métodos y técnicas. En consecuencia, los principales módulos de este proceso son: ETL, Motor de Análisis Avanzado de la Evolución de Software (MAAES) y Explorador Visual de Conocimiento para la Evolución de Software (EVCES) (ver tabla C.1).

Es importante mencionar que los componentes de visualización del submódulo VES en la figura C.1 fueron identificados en en esta investigación.

Los pasos que sigue el proceso AVAES se organizaron en fases y se enumeran a continuación:

Fase I: Recuperación y Carga de Datos Recupera y lleva a cabo el procesamiento inicial de los datos, para luego almacenarlos en un almacén de datos.

Recuperación de datos: De acuerdo con el tipo de tarea que el investigador o diseñador buscan apoyar el proceso de recuperación se puede llevar a cabo usando los repositorios de software, registros de seguimiento de defectos del sistema, correos electrónicos, código fuente y registros del sistema de prueba. Las técnicas utilizadas en

la recuperación de datos pueden incluir la recuperación de código fuente, consultas sobre la estructura del sistema, búsqueda de patrones y recuperación de texto (Extraer, flecha 1).

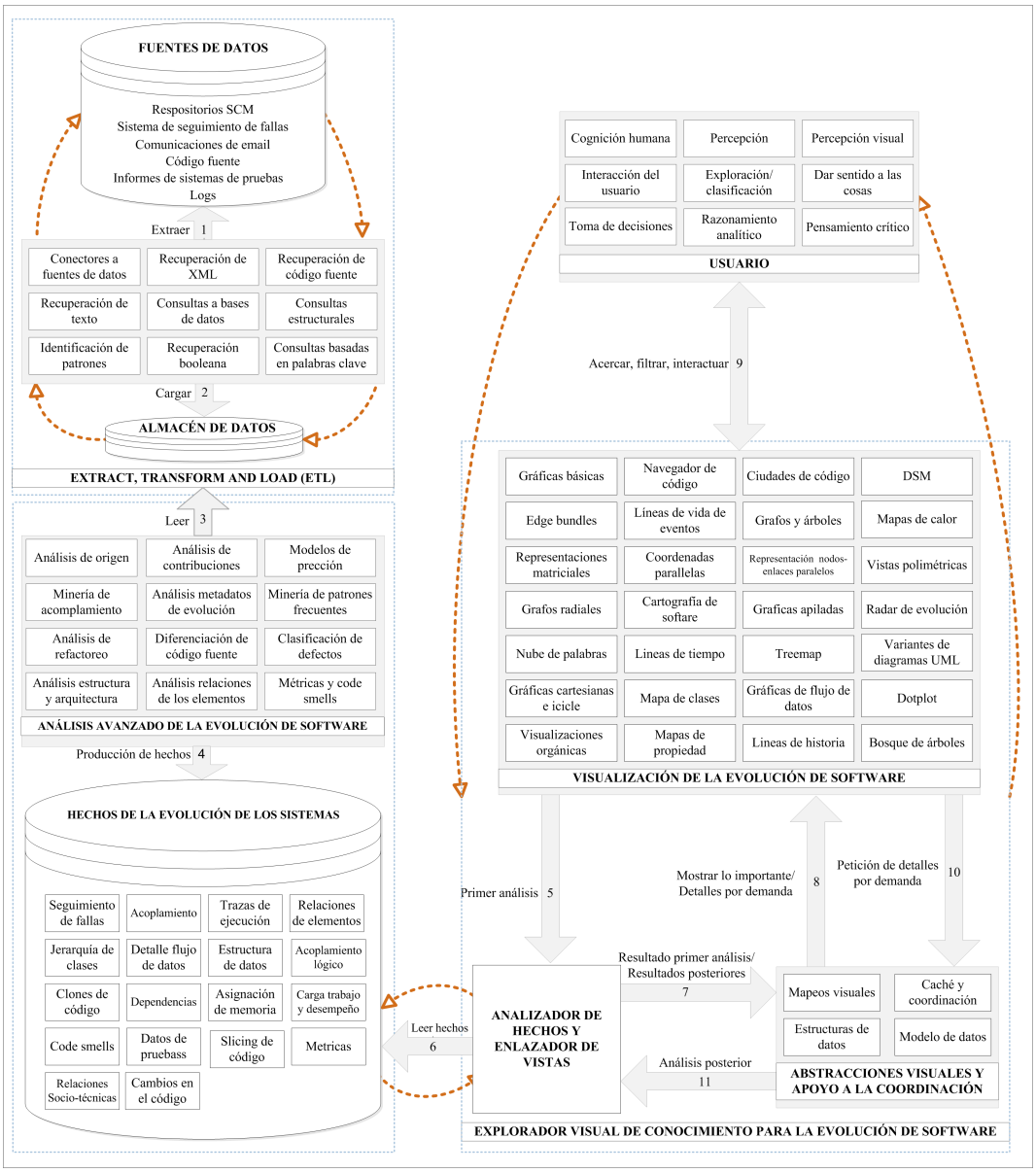


Figure C.1: Descripción general del proceso Evolutionary Visual Software Analytics.

Almacén de datos: Una vez que los datos han sido recuperados, se limpian, integran y correlacionan y guardan en un almacén de datos (Cargar, flecha 2).

Fase II: Análisis de Datos En esta fase se lleva a cabo el análisis y

Table C.1: Responsabilidades y funciones de los módulos que componen el proceso AVAES.

Módulo	Descripción
Extraction, Transformation and Load (ETL)	Este módulo tiene la función de realizar la conexión y recuperación de datos de los repositorios de software, sistemas de seguimiento de defectos (*bug tracking*), correos electrónicos, revisiones de código fuente, sistemas de pruebas, registros y cualquier otra fuente de datos disponible. Cuando se recuperan los datos, se limpian, fusionan y cargan en un almacén de datos.
Motor de Análisis Avanzado de la Evolución de Software (MAAES)	Este módulo se compone de técnicas de análisis [Hassan 2005, Hassan 2006, Kagdi 2007a] que podrían ser utilizadas de forma individual o combinadas con el fin de extraer hechos de conocimiento.
Explorador Visual de Conocimiento para la Evolución de Software (EVCES)	Este módulo está formado por tres componentes: VES, Analizador de Hechos y Enlazador de Vistas (AHEV) y Abstracciones Visuales y Apoyo a la Coordinación (AVAC).

extracción de hechos de ES y luego se procede a almacenar los resultados en una base de datos.

Análisis y extracción de hechos: Cuando se han agregado nuevos datos en el almacén de datos, ETL lee los datos (Leer, flecha 3) y luego MAAES procede con el análisis utilizando una o más técnicas, dependiendo del objetivo con el cual este está siendo efectuado.

Almacenamiento de hechos de evolución: Una vez que el análisis se ha llevado a cabo, los hechos de la evolución se almacenan en la base de datos *Software Evolution Facts* (Producción de hechos, flecha 4).

Fase III: Carga de Estructura y Mapeo Visual Las tareas de esta fase incluyen carga de hechos de ES, la creación de las estructuras de datos y los mapeos visuales, así como la carga de las visualizaciones.

Carga de la visualización: El usuario inicia el componente VES que

utiliza visualizaciones enlazadas.

Solicitud de estructuras de datos con hechos: Cuando se ha cargado el componente VES, las estructuras de datos con los hechos que son requeridas por las visualizaciones son solicitados por AHEV (Primer análisis, flecha 5 y Leer hechos, 6).

Carga de hechos: El componente AHEV lee los hechos de *Software evolution Facts Database* y los pasa al componente AVAC (Resultados del primer análisis, flecha 7).

Estructuras y mapeo visual: AVAC crea y pasa el modelo de datos apropiado, así como las estructuras de datos y mapeos visuales a VES (Mostrar lo importante, flecha 8).

Fase IV: Interacción del Usuario y Detalles por Demanda Esta fase es la etapa final del proceso de transformación de datos en conocimiento. Después de la recuperación, el análisis y mapeo visual de la información, esta fase hace posible un bucle de retroalimentación entre el usuario y el sistema: los usuarios realizar solicitudes de datos adicionales al sistema por medio de las posibilidades de interacción disponibles, y el sistema proporciona los datos solicitados. De acuerdo con las interacciones del usuario el proceso de descubrimiento de conocimiento es refinado y avanza hacia el hallazgo de conocimiento y respuestas útiles.

Interacción del usuario: Durante el proceso de descubrimiento de conocimiento, el usuario navega, filtra y explora diferentes perspectivas de los datos, selecciona los elementos de una o más de las visualizaciones (Acercar, filtrar, interactuar, flecha 9).

Solicitar detalles: De acuerdo con las necesidades e interacciones del usuario, la visualización solicita nuevas estructuras de datos con hechos y mapeos visuales para proporcionar información adicional al usuario, conforme con las opciones seleccionadas (Petición de detalles por demanda, flecha 10).

Detalles adicionales: Si los detalles adicionales que han sido solicitados están disponibles en forma de estructuras de datos con hechos y mapeos visuales, se pasan a VES (Detalles por demanda, flecha 8). Sin embargo, si estos detalles no están disponibles, la solicitud se pasa a AHEV (Análisis posterior, flecha 11), para que lea los hechos adicionales, (Leer hechos, flecha 6), transforma los detalles y luego los pasa a AVAC (Resultados posteriores, flecha 7) para que pueda proceder a crear las estructuras de datos con hechos y los mapeos visuales.

Descubrimiento de conocimiento: El usuario continúa interactuando con el sistema hasta que obtiene el conocimiento necesario o considera que no es posible llegar a una conclusión usando los datos y representaciones disponibles.

C.2.3 Architecture Specification

La definición de la arquitectura de herramientas de software es una tarea compleja que requiere un análisis cuidadoso. Es un reto determinar qué técnicas utilizar y cómo van a estar interrelacionadas. En esta sección se pretende contribuir con los objetivos especificados, responder a la pregunta de investigación formulada en la sección C.1.3, así como apoyar el diseño de herramientas en situaciones donde se ha decidido aplicar AV a ES. Por lo que tomando como referencia AVAES, se diseñó una arquitectura para una herramienta denominada como *Maleku* [González-Torres 2011, González-Torres 2013b, González-Torres 2013a]

Maleku busca apoyar a los programadores y administradores de proyectos de software al correlacionar métricas, la estructura del proyecto, las relaciones de herencia e implementación de interfaces y las relaciones socio-técnicas. Dicha arquitectura se implementó en *Java* y probó en proyectos de código abierto.

Los módulos de la arquitectura (ver figura C.2) son similares a los módulos del proceso descrito en la sección anterior y se les ha dado el mismo nombre. El funcionamiento de los módulos ETL y MAAES es síncrono, mientras que la operación de EVCES es asíncrona, en relación con los otros dos módulos. La arquitectura se basa en el modelo cliente / servidor, en el que los módulos ETL y MAAES son ejecutados por el servidor y EVCES es un plugin de *Eclipse* ejecutado en el lado cliente. Los diferentes módulos y componentes de la arquitectura se describen en el siguiente orden: recuperación de datos, análisis de datos y representación visual.

ETL contiene un sub-módulo (SM) y dos componentes (C), como se muestra en la siguiente lista:

Fuente de Datos (C):[2] Las fuentes de datos utilizadas por *Maleku* consisten en los repositorios ACS de los proyectos de software. La información que se extrae de estos repositorios incluyen los metadatos asociados con los cambios en el código fuente, las actividades de los programadores, la estructura del proyecto y el código fuente.

Sensor de Nuevas Revisiones (C): El *Sensor de Nuevas Revisiones* es un proceso que monitorea de forma continua la adición de nuevas revisiones

[2] C hace referencia a componente.

a los proyectos de software y envía notificaciones al *Extractor de Datos*.

Extractor de Datos (SM):[3] La función de este sub-módulo es extraer los datos necesarios para llevar a cabo el análisis, cuyos resultados se utilizan para alimentar a las visualizaciones de la herramienta de AV. Los componentes de este sub-módulo se describen a continuación:

Recuperación de arquitectura y estructura (C): Este componente se encarga de extraer los detalles de la estructura del proyecto para cada revisión, con especial interés en los paquetes del sistema y su organización.

Recuperación de código fuente (C): Es responsable de recuperar el código fuente de cada una de las revisiones del sistema y de almacenar clases con información básica acerca de su ubicación en la arquitectura del sistema.

Recuperación de metadatos (C): Los datos que este componente es responsable de recuperar, incluye los logs de cada revisión y sus detalles asociados: la fecha en cual se hizo la revisión, el programador que la llevó a cabo y los elementos afectados.

Los submódulos que conforman MAAES son el *Analizador de Código Fuente* y el *Motor de Correlación de Metadatos y Análisis de la Evolución de Software*, cuyas descripciones se presentan a continuación.

Analizador de Código Fuente (SM): Este sub-módulo se encarga de llevar a cabo el análisis de las revisiones del código fuente del proyecto usando los siguientes componentes:

Detección de métricas (C): Este componente se encarga de detectar y calcular métricas usando detalles del código fuente analizadO. Algunas de las métricas que pueden ser calculadas por este componente incluyen LOC, NOM y la Complejidad Ciclomática.

Análisis de relaciones de elementos (C): Las funciones de este componente incluyen la detección de herencia (padre-hijo y hijo-padre) y las relaciones de implementación de interfaces.

Parser de código fuente (C): Este módulo lee cada archivo de código fuente, línea por línea, con el fin de identificar las clases, interfaces, métodos y declaraciones, y aplica reglas de análisis. Además calcula métricas para identificar las relaciones entre los elementos de software.

[3]SM se refiere a sub-módulo.

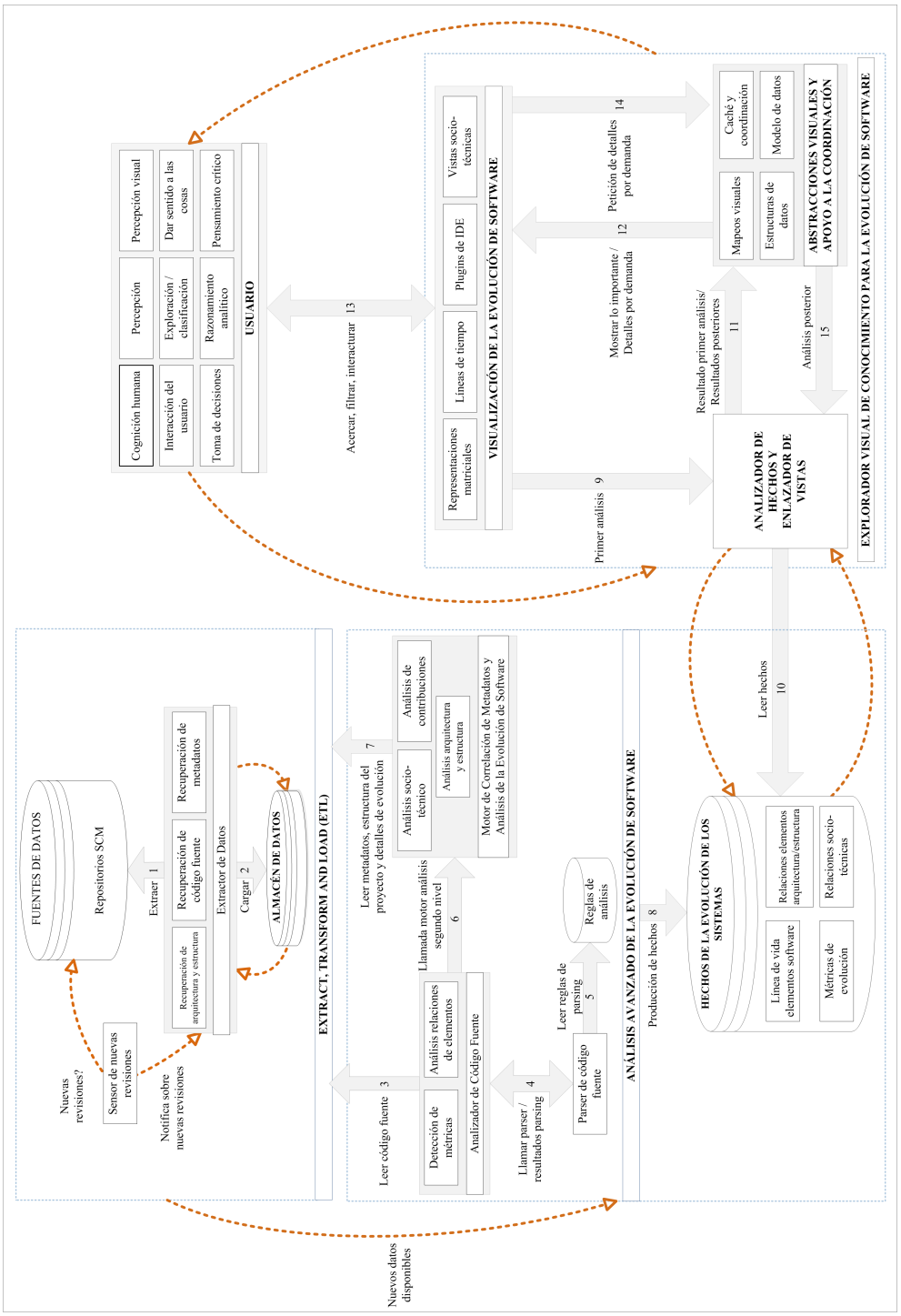


Figure C.2: Vista general de la arquitectura de *Maleku* .

Reglas de análisis (C): El *Parser de código fuente* aplica reglas de análisis, que se almacenan en archivos de texto. Algunas de estas reglas se generan de forma automática, mientras que otras son creadas de forma manual.

Correlación de Metadatos y Análisis de la Evolución (SM): Este sub-módulo es invocado por el sub-módulo de *Analizador de Código Fuente* cuando se termina de ejecutar el análisis. Su función es identificar las relaciones socio-técnicas y determinar las contribuciones hechas por cada programador, así como efectuar el análisis de la arquitectura y la estructura del proyecto para cada revisión que se analiza. Los componentes de este sub-módulo son:

- **Análisis de contribuciones (C):** Con base en los metadatos de los repositorios ACS se lleva a cabo un cálculo acumulativo de los elementos que han sido cambiados por cada revisión y programador.
- **Análisis socio-técnico (C):** Utiliza los metadatos de los repositorios ACS extraer información sobre las relaciones entre los programadores y los elementos de software, así como sobre las relaciones que se crean entre los programadores tomando como base los elementos que han cambiado en común.
- **Análisis arquitectura y estructura (C):** Los resultados producidos por el *Analizador de Código Fuente* y la información obtenida de los metadatos de los repositorios ACS se utilizan para correlacionar la estructura del proyecto de software, las métricas y las relaciones entre los elementos de software. De forma adiciona, recopila información sobre la creación de los elementos de software y su línea de vida durante el proyecto.

Base de datos de hechos de la evolución (C): Esta base de datos almacena los resultados de los análisis que son producidos por otros sub-módulos y componentes de MAAES. Para eso utiliza un diseño que emula la jerarquía de los proyectos de software: proyecto –> revisión –> paquete –> archivo –> elemento de software.

La secuencia de pasos que sigue el proceso de recuperación y análisis de datos (compuesto por ETL y MAAES) son los siguientes:

1. El usuario introduce los parámetros de conexión del repositorio ACS y la base de datos donde se almacenarán los datos del proyecto que será analizado.

2. Cuando el proceso se ha iniciado, los componentes de recuperación que están incluidos en el sub-módulo de *Extractor de Datos* llevan a cabo la recuperación de los datos (Extraer, flecha 1).
3. Una vez que los datos se han recuperado son cargados en el almacén de datos (Cargar, flecha 2). Los datos cargados se utilizan como referencia para los procesos de recuperación posteriores.
4. El *Sensor de Nuevas Revisiones* es el responsable de la supervisión de la disponibilidad de nuevas revisiones en los repositorios ACS, así como de notificar a los módulos de recuperación de manera oportuna.
5. Conforme los datos son recuperados, ETL informa a MAAES que los nuevos datos están disponibles para realizar el análisis de acuerdo con los componentes de análisis disponibles.
6. El sub-módulo de *Analizador de Código Fuente* lee datos del almacen de datos del módulo ETL (Leer código fuente, flecha 3) con el fin de detectar y calcular métricas para las clases y métodos, y analizar la relaciones entre los elementos de software, tales como la jerarquía de clases y la implementación de interfaces.
    6.1. Para llevar a cabo sus tareas el *Analizador de Código Fuente* requiere analizar el código fuente y luego notifica al componente *Parser de código fuente* (Llamar parser, flecha 4).
    6.2. El componente *Parser de código fuente* lee las reglas de análisis de su propia base de datos (Leer reglas de parsing, flecha 5) y realiza el análisis del código fuente.
7. Cuando el sub-módulo de *Analizador de Código Fuente* ha terminado de hacer el análisis, almacena los resultados en la base de datos *Hechos de la Evolución de los Sistemas* y notifica al *Motor de Correlación de Metadatos y Análisis de la Evolución de Software* (Llamada al motor de análisis de segundo nivel, flecha 6).
8. El sub-módulo de *Motor de Correlación de Metadatos y Análisis de la Evolución de Software* lee los hechos de evolución de la base de datos *Hechos de la Evolución de los Sistemas*, así como metadatos, detalles de la estructura del proyecto y la evolución del almacén de datos de ETL (Leer metadatos, estructura del proyecto y detalles de evolución, flecha 7). Con esta información el módulo lleva a cabo un análisis más profundo de las relaciones socio-técnicas, las contribuciones de los programadores y de la arquitectura y la estructura del proyecto de software.
9. El proceso que llevan a cabo ETL and MAAES se ejecuta de manera indefinida para cada uno de los proyectos configurados, hasta que el análisis es detenido por el usuario.

El diseño de la arquitectura de *Maleku* permite la adición de nuevos componentes a los módulos y sub-módulos para permitir nuevas conexiones a otras fuentes de datos, realizar otros tipos de análisis y visualizar los resultados del análisis con nuevas representaciones visuales. Los pasos seguidos por MAAES son los mismos que los que se describen en la sección C.2.2, de modo que se omite la explicación de esos pasos.

C.3 Diseños de las Visualizaciones y Escenarios de Uso

La vista principal de EVCES y las visualizaciones que incluye se muestran en la figura C.3: GT (ver esquina inferior izquierda), *Gridmaster* (situada en el panel superior derecho), STG y RT (estas dos últimas se muestran en el panel inferior derecho).

GT proporciona una vista general de las actividades de un proyecto de software, mientras que *Gridmaster*, STG y RT representan detalles específicos sobre la correlación de la estructura del proyecto, asociaciones, la colaboración entre los miembros del equipo de desarrollo en el tiempo y en diferentes niveles de granularidad de la estructura del proyecto.

Las visualizaciones que incluye *EVCES* son iniciadas al realizar su selección de un menú contextual en el *Explorador de Paquetes* de Eclipse (situado en el panel superior izquierdo, como es indicado por el número 1). Una vez que las visualizaciones han sido iniciadas, el flujo de trabajo (indicado por los números y flechas en la figura C.3) comienza con el análisis de los patrones de los aportes de los programadores al sistema haciendo uso de GT, y la selección en esa visualización de una o más unidades de tiempo para su posterior análisis en *Gridmaster*. Después, el usuario puede seleccionar cualquiera de estas dos opciones desde *Gridmaster*:

1. Un elemento de software en el árbol situado en el lado izquierdo de *Gridmaster* para obtener detalles acerca de la colaboración durante la evolución de ese elemento (utilizando RT).
2. Una unidad de tiempo en *Gridmaster* para la representación de las relaciones socio-técnicas asociadas en STG.

Los hechos sobre ES que han sido tomados en cuenta por las visualizaciones son los siguientes:

1. Líneas de vida de los elementos de software.
2. Métricas de evolución.
3. Correlación de los datos estructurales con métricas.

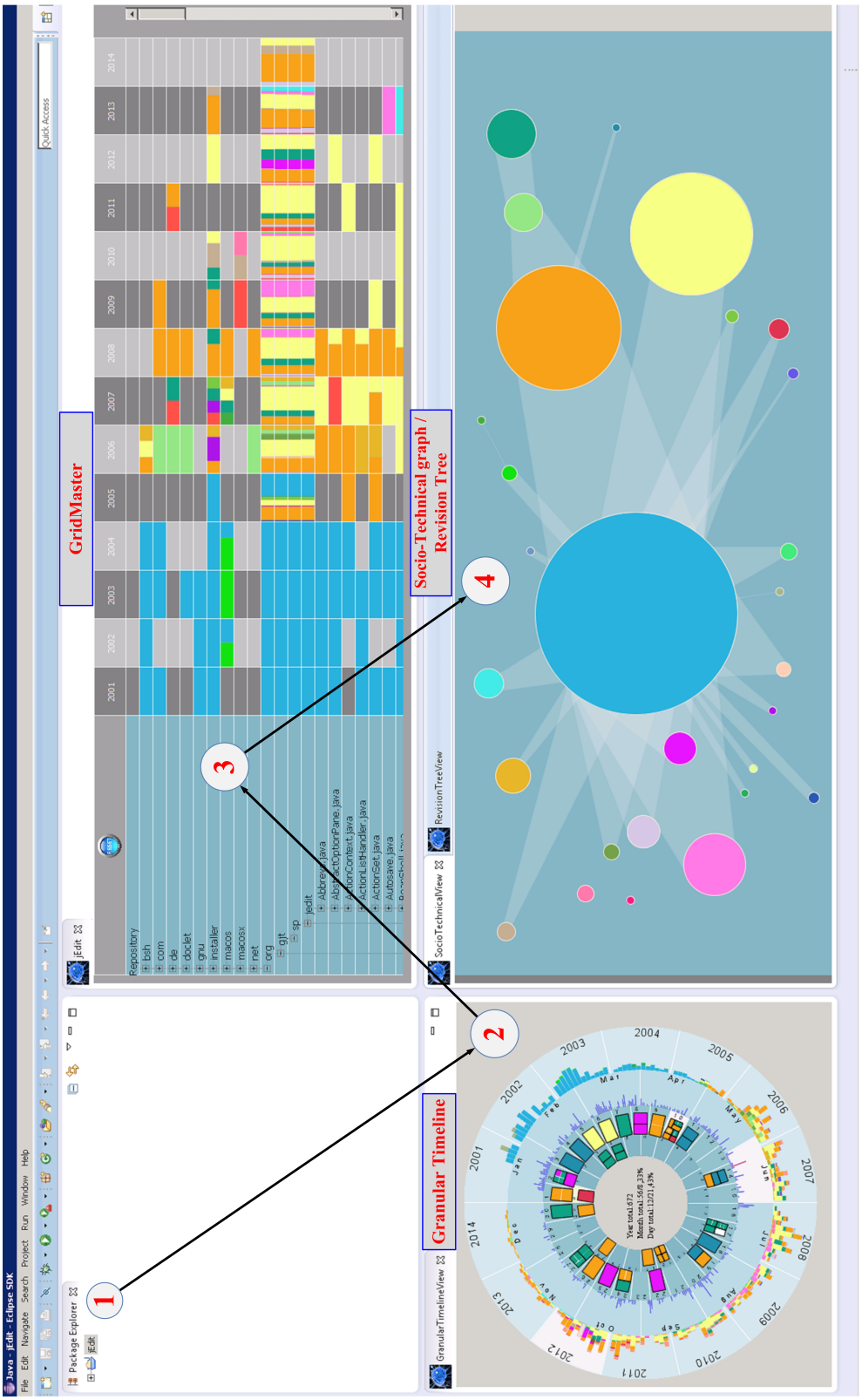


Figure C.3: Flujo de descubrimiento de conocimiento en el módulo *EVCES* de *Malekv*.

4. Relaciones socio-técnicos.
5. Algunas relaciones arquitectónicas / estructural, como, por ejemplo, la herencia, la interfaz la implementación.

En la presente tesis se estudiaron tres proyectos de código libre escritos en Java haciendo uso de 9:

1. *jEdit* es un editor de texto de código abierto para programadores que se encuentra disponible en http://sourceforge.net/projects/jedit y cuyo desarrollo comenzó en diciembre de 1999. En esta investigación se tomaron en cuenta cerca de 1212 clases y 5801 revisiones, las cuales fueron realizadas entre septiembre del 2001 y abril del 2014.
2. *JabRef* es un gestor de referencias bibliográficas que fue liberado en noviembre de 2003 y se encuentra disponible en http://jabref.sourceforge.net. Los datos sobre la evolución de dicho proyecto que son considerados en esta tesis incluye 3719 revisiones y 1236 elementos de software que fueron creados durante 9 años (de octubre de 2003 a noviembre de 2011).
3. *JFreeChart* es una biblioteca para crear gráficos cuyo desarrollo es efectuado por solo un programador (http://www.jfree.org/jfreechart) y comenzó en Febrero de 2000. Esta investigación toma en cuenta 916 revisiones de este proyecto y 1130 elementos de software que han sideo pgoramados entre el 2007 y el 2010.

Sin embargo, en este resumen solo se presentarán ejemplos de *jEdit* para describir las visualizaciones de *EVCES*.

C.3.1 Granular Timeline: Análisis de Estadísticas de las Revisiones y Contribuciones de los Programadores

GT utiliza un diseño gráfico de un anillo circular modificado para mostrar una visión general de la dimensión temporal de un proyecto de software (figura C.4). Los anillos concéntricos representan las diferentes escalas de tiempo en las cuales se registran los eventos de cambio, la granularidad de dicha escala de tiempo va de las unidades grano grueso (años, anillo exterior) a las de grano fino (horas, anillo más interno). Una representación circular similar fue utilizada por Holten *et al.* para visualizar la jerarquía de los proyectos de software [Holten 2007, Cornelissen 2007]. Este tipo de representación compacta permite presentar grandes cantidades de datos y proporciona una estrategia del tipo *vista general + detalle*.

GT es una visualización que representa datos sobre la evolución de los proyectos de software por largos periodos de tiempo y que ofrece mecanismos

de interacción que permiten seguir un camino que se inicia en los niveles de granularidad gruesa y permite pasar a los niveles de detalle de grano fino.

GT puede ser usada para representar cualquier tipo de datos cuantitativos producidos en el tiempo, dada su naturaleza para representar valores numéricos y resultados estadísticos. Sin embargo, en esta investigación se utiliza para representar las estadísticas de los *commits* y las contribuciones de los programadores, tal como se explica a continuación.

Estadísticas sobre revisiones: El espacio dentro de cada celda de la visualización se utiliza para integrar diferentes tipos de visualizaciones que representen el número de revisiones en el nivel de detalle de una celda particular. De forma que las celdas del anillo *years* insertan gráficas de barras que muestran el número de revisiones para cada mes por cada año. Mientras que las celdas del anillo *months* incrusta un gráfico de barras para presentar el número de revisiones realizadas por cada día del mes. El siguiente anillo a continuación, *days*, muestra las revisiones para cada día del mes, por lo que este anillo tiene 28, 29, 30 y 31 celdas (según el número de días del mes que ha sido seleccionado), y las revisiones en este nivel se representan mediante gráficas de barras o *treemaps*, de acuerdo con la selección del usuario (ver figura C.5 para la representación usando treemaps). Por último, el anillo más interno, *hours*, muestra las revisiones conforme con la hora en que ha sido realizadas, por lo que tiene 24 celdas, y representa las revisiones por medio de gráficos de barras o *treemaps*.

La utilización de gráficas de barras proporciona detalles sobre el número de contribuciones en cada nivel de granularidad. Sin embargo, la representación de los datos en el nivel de granularidad *days* y *hours* no saca el máximo provecho de la pequeña área gráfica disponible. Por esa razón se ofrece la posibilidad de representar los datos utilizando *treemaps*. Este tipo de visualización permite aprovechar mejor el espacio, y además, permite resaltar mejor las revisiones individuales.

De forma adicional, algunas estadísticas se muestran en el centro de la visualización conforme se seleccionan las diferentes unidades de tiempo. Observe que la selección de las unidades de tiempo comienza en el anillo exterior que se corresponde con el anillo *years* y sigue la secuencia *months* –> *days* –> *hours* (e.g., 2008 –> junio de 2008 –> 3 de junio de 2008 –> 22 : 00 horas del 3 de junio de 2008), como se encuentra resaltado por las selecciones que se muestran en la figura C.4.

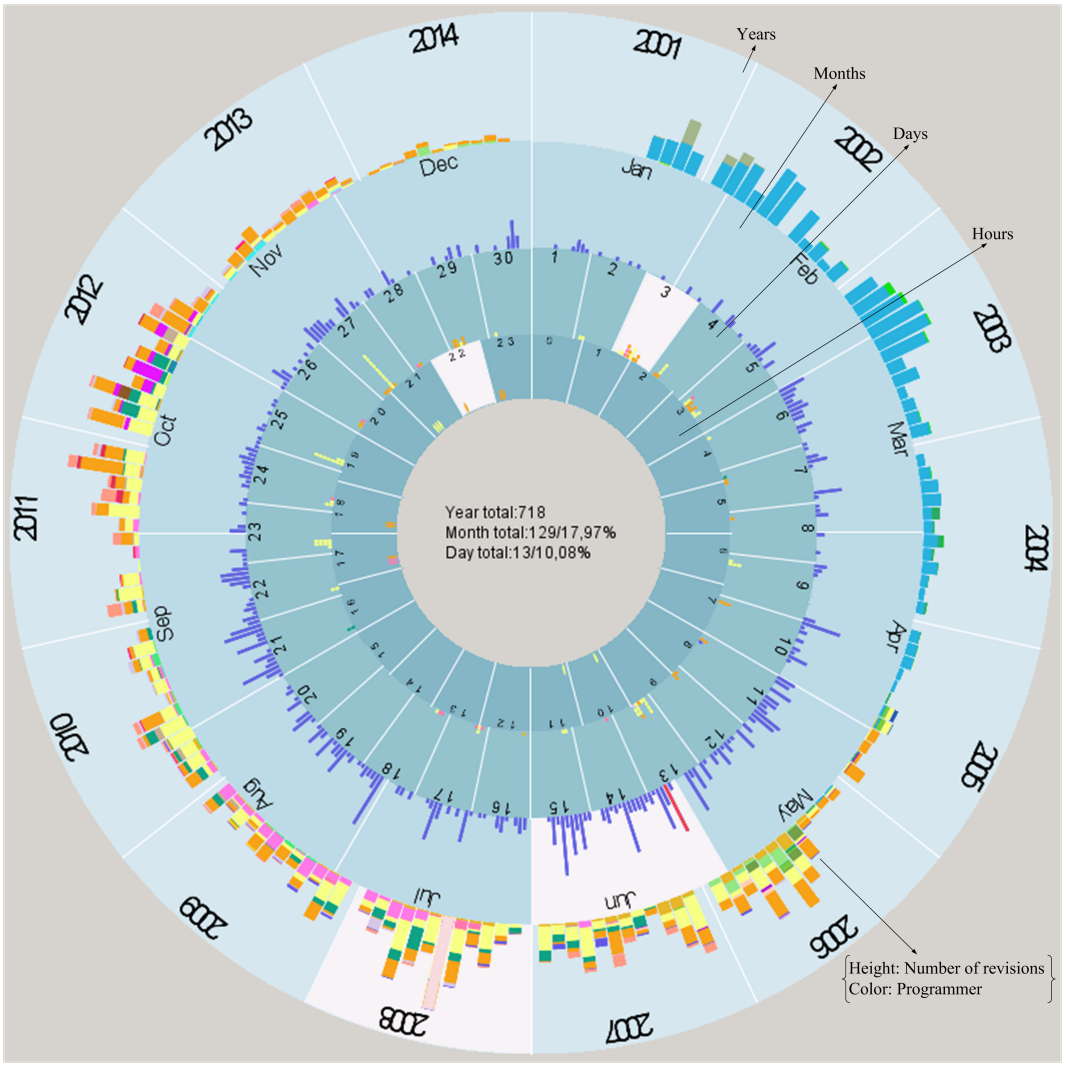


Figure C.4: Visualización de datos estadísticos usando sobre las revisiones de *jEdit* en un lapso de tiempo de 14 años utilizando GT.

Contribuciones de los programadores: Las contribuciones de los programadores están codificados por colores, donde cada color se corresponde con un programador particular. En general, el uso del color permite la adquisición de información de estadísticas sobre las revisiones y quien las ha llevado a cabo a simple de vista.

Por lo que es fácil extraer patrones de los programadores que intervienen en el desarrollo de un proyecto. Esta visualización es simple e intuitiva y se puede utilizar con pocos cambios en la representación de cualquier tipo de estadísticas.

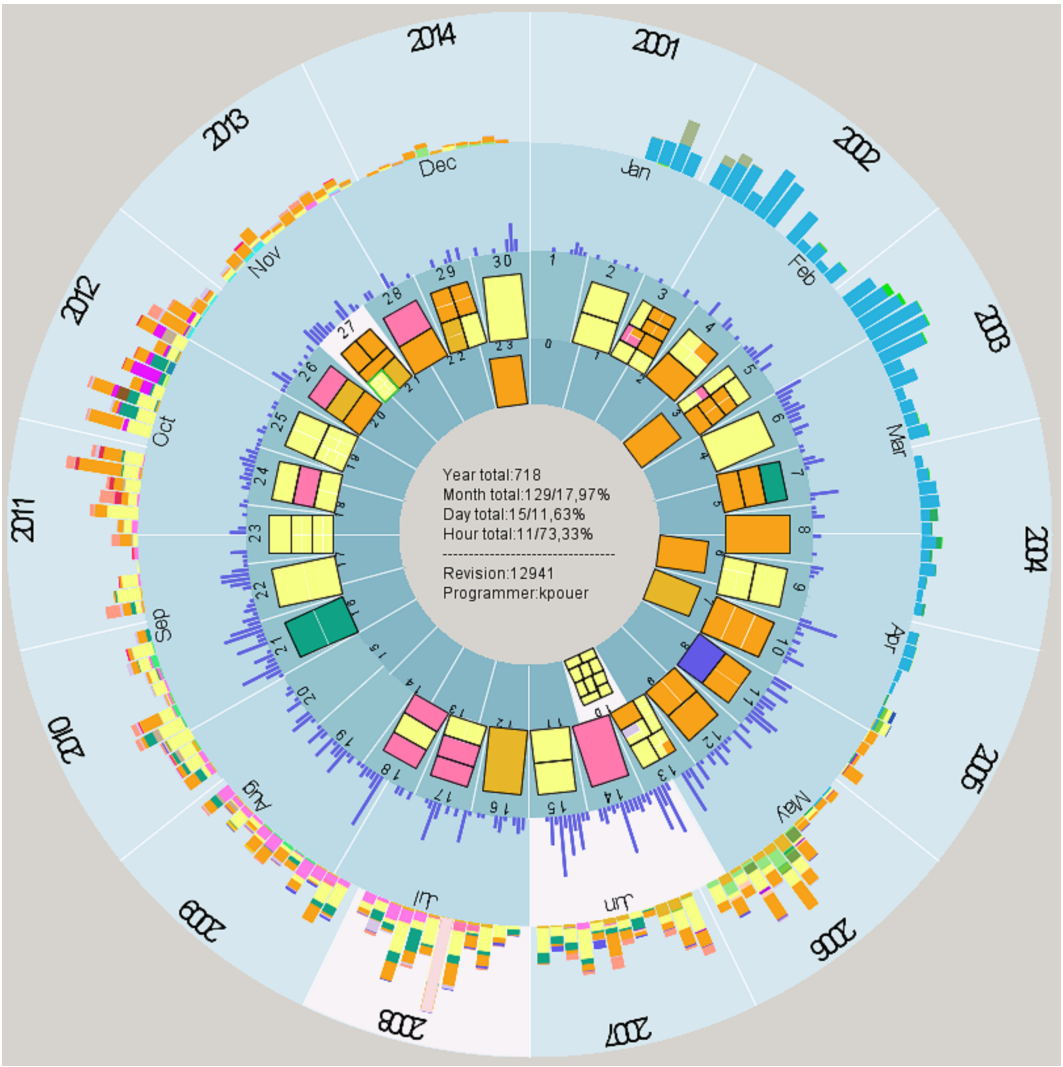


Figure C.5: Representación visual de estadísticas sobre revisiones de *jEdit* usando GT y *treemaps*.

La figura C.4 permite observar a primera vista que los datos de la evolución de *jEdit* representados comprende el periodo que va del año 2001 al año 2014. Además, permite ver detalles relativos a los años con más revisiones (2008 y 2012), así como patrones de colaboración de los programadores. Por lo tanto, *spestov* (turquesa) es el desarrollador que más contribuyó al proyecto durante los primeros 5 años (2001, 2002, 2003, 2004 y 2005), y luego otros programadores como *ezust* (naranja) y *kpouer* (amarillo) lideraron la programación del sistema, de acuerdo con el número de revisiones.

Aunque algunos proyectos de software siguen un patrón similar al de *jEdit* (*e.g.*, un programador principal contribuye con un proyecto al inicio de este), la disposición de las contribuciones de los programadores en el tiempo varía de un proyecto a otro.

C.3.2 Gridmaster: Correlación de Estructura, Relaciones y Métricas

Gridmaster se basa en una representación de un árbol y una matriz, dos estructuras ampliamente conocidas por los programadores (ver figura C.6). La estructura de árbol visualiza todos los paquetes y elementos de software que se han agregado al proyecto durante su evolución, mientras que la matriz es creada por la intersección de filas y columnas: los paquetes y elementos de software se colocan en la estructura de árbol y son asociados a las filas, y las unidades de tiempo están vinculadas a las columnas.

El uso de un árbol y una matriz permite la correlación, de todos los elementos de software, con las contribuciones de los programadores, la creación de los elementos, los cambios en las relaciones arquitectónicas, la adición o eliminación de una relación de herencia y la implementación de interfaces, y las métricas.

En consecuencia, algunas de las características de *Gridmaster* permiten extraer detalles de la colaboración en los diferentes niveles de granularidad, de forma similar a GT. Sin embargo, *Gridmaster* fue diseñada para ser una vista complementaria de GT. En este sentido, GT fue pensada para ofrecer una visión de conjunto, así como estadísticas en diferentes niveles sobre las revisiones de un sistema de software, mientras que *Gridmaster* fue diseñado para correlacionar las contribuciones de los programadores con los elementos de software. De ahí que *Gridmaster* puede ser usada para representar las relaciones socio-técnicas entre los elementos de software y los programadores, así como la línea de vida de los elementos para toda la evolución de un sistema o un determinado período de tiempo a partir de su seleción en GT.

En esta visualización los colores se asignan a los programadores y el área asociada a cada programador depende del número de aportaciones realizadas en términos de revisiones y de forma relativa a la unidad de tiempo representada (ver la Figura C.6).

Este tipo de representación permite delinear la línea de vida de los elementos de software y los paquetes utilizando un enfoque intuitivo. La figura C.6 muestra que las actividades realizadas en el paquete *bsh* de *jEdit* se llevaron a cabo entre el 2001 y el 2006. También muestra que este paquete en la actualidad no forma parte de la última revisión del proyecto, lo cual se puede corroborar al revisar la estructura de *jEdit* en *Eclipse*.

La figura C.6 también permite observar que otros paquetes tales como *com*, *gnu*, *macos* y *macosx* forman parte del primer nivel de la estructura del proyecto en la visualización aunque sus líneas de vida no registran actividad reciente para *macos* y *macosx*, por lo cual se puede intuir que fueron movidos a un nivel de jerarquía inferior, o fueron eliminados del proyecto.

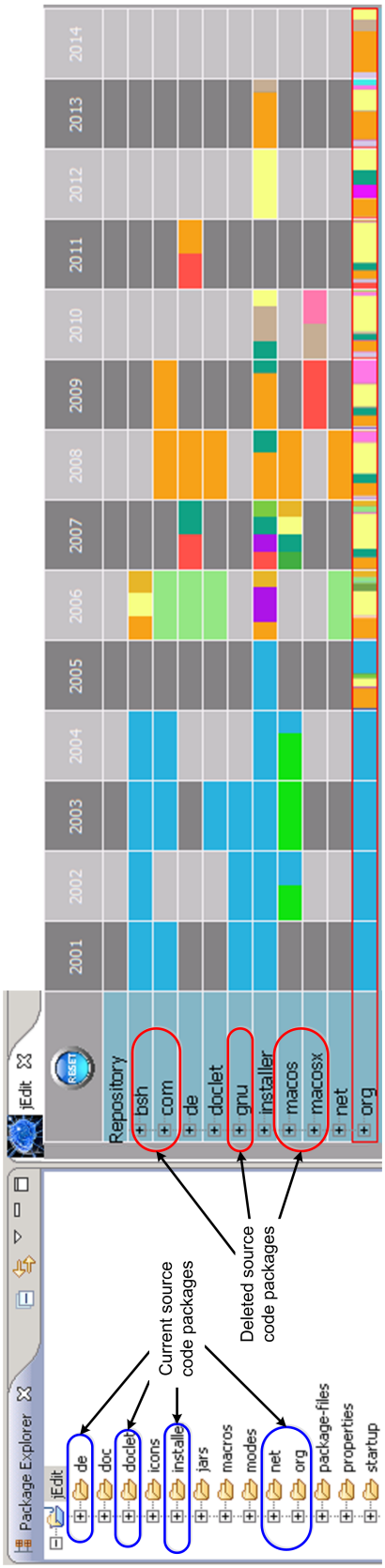


Figure C.6: *Representación absoluta* de las contribuciones de los programadores, la estructura del proyecto y las líneas de vida para *jEdit*.

La figura C.7 muestra los años 2006 y 2008 en forma expandida, de modo que los cambios en las relaciones de herencia y modificaciones de las métricas se pueden observar, y sus distribuciones mensuales pueden ser examinadas.

La estructura de árbol utilizada por *Gridmaster* es una representación escalable que está basada en un árbol plegable para representar paquetes, dentro de los cuales se encuentran los archivos, los cuales a su vez contienen las clases, interfaces, enumeraciones y anotaciones (los cuatro elementos básicos de *Java*). En este contexto las clases son denotadas por un pequeño símbolo cuadrado, mientras que las interfaces son representadas por un pequeño círculo, las enumeraciones por un triángulo y las anotaciones por estrella. Cuando la estructura plegable se expande, es posible ver el contenido de los archivos y los detalles sobre el establecimiento y la terminación de las relaciones de herencia e implementación de interfaces, así como las métricas asociadas a los elementos.

La figura C.7 señala los elementos de software que intervienen en las relaciones de herencia e implementación de interfaces por medio de leyendas de color rojo que indican su ubicación (la API de *Java*, el proyecto que está siendo analizado o una API externa). Además, representa el establecimiento de relaciones por medio de un óvalo verde, mientras que la terminación de relaciones es representada por un óvalo rojo. El procedimiento utilizado para determinar si un elemento de software está incluido en el proyecto bajo análisis, *Java* o una API externa es mostrado en el algoritmo C.3.1.

En cuanto a la métricas, estas son representadas en las filas que corresponden a los elementos de software (las filas que están asociadas a los paquetes, relaciones de herencia e implementación de interfaces no se utilizan para este propósito) utilizando gráficas de barras con la finalidad de destacar los cambios (ver la figura C.7).

Los valores de las métricas se muestran cuando se lleva a cabo el *commit* de un elemento de software, con independencia de que los valores hayan cambiado o no. Es importante mencionar que el diseño de la visualización sólo permite la representación de una métrica a la vez, por lo que el usuario debe seleccionar cual métrica quiere visualizar.

Gridmaster representa las relaciones de herencia tanto de los padres hacia los hijos como de los hijos hacia los padres. Así, cuando un elemento es seleccionado, se pueden observar las relaciones de herencia que se han establecido con el tiempo.

En consecuencia, la herencia se representa *Gridmaster* de la siguiente forma: padres <– elemento de software seleccionado –> hijos. La figura C.8 representa las relaciones que *AbstractOptionPane* ha establecido con otros elementos de software: esta clase hereda de *JPanel*, implementa la interfaz *OptionsPane* y tiene 8 subclases.

Algorithm C.3.1: EXTRACCIÓN DEPENDENCIAS DE ELEMENTOS(*repos*)

```
repository ← repos
while n < repository.lastRevision()
do { revisionClasses ← DataExtractor.getRevisionItems(n)
     for each item ∈ revisionItem
     do { if item.hasParent()
          then { parent ← item.getParent()
                 self ← Analyzer.isParentOnProj(parent)
                 if project
                   then location ← Project
                   else java ← Analyzer.isParentOnJava(parent)
                 if java
                   then location ← Java
                   else location ← Library
                 package ← Analyzer.getClassPackage(parent)
                 extends(x) = ( parent  location  package )
                 vector.add(extends)
          if item.hasImplements()
          then { interfaces ← item.getInterfaces()
                 for each interface ∈ interfaces
                 do { self ← Analyzer.isIntOnProj(interface)
                      if project
                        then location ← Project
                        else java ← Analyzer.isIntOnJava(interface)
                      if java
                        then location ← Java
                        else location ← Library
                      package ← Analyzer.getIntPackage(interface)
                      implements(x) = ( parent  location  package )
                      vector.add(implements)
return (vector)
```

La representación de la implementación de interfaces se lleva a cabo de una manera similar a la representación de las relaciones de herencia y representa las interfaces que han sido implementadas por una clase particular (desde la perspectiva de una clase) y que clases implementan una interfaz específica (desde la perspectiva de una interfaz). Esta característica se muestra en la figura C.9, la cual destaca con un óvalo rojo la interfaz *Compare* y representa que esta interfaz hereda de la interfaz *Comparator* y es implementado por las clases *MenuItemCompare* y *StringCompare* (resaltada por los círculos azules).

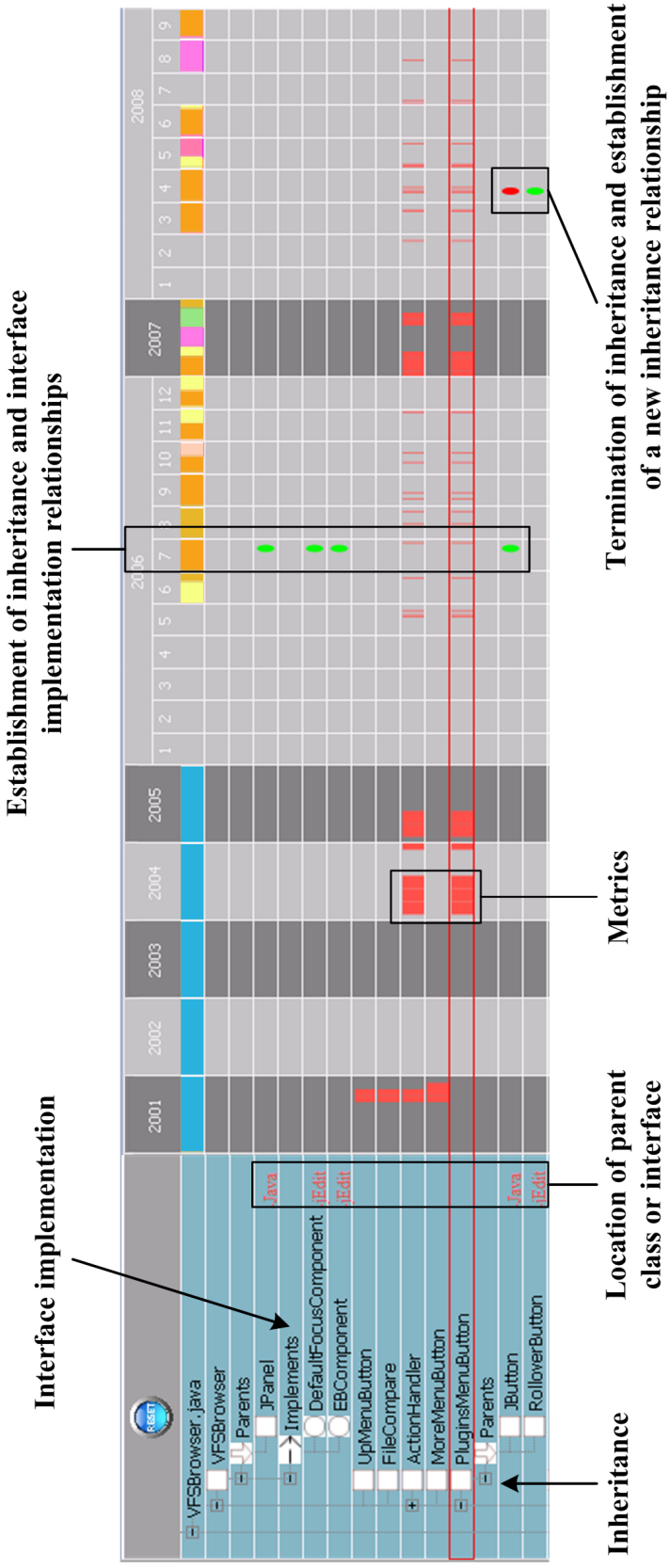


Figure C.7: Relaciones de herencia e implementación de interfaces, incluyendo la expansión de los años y valores de métricas para el archivo *VFSBrowser.java* de *jEdit*.

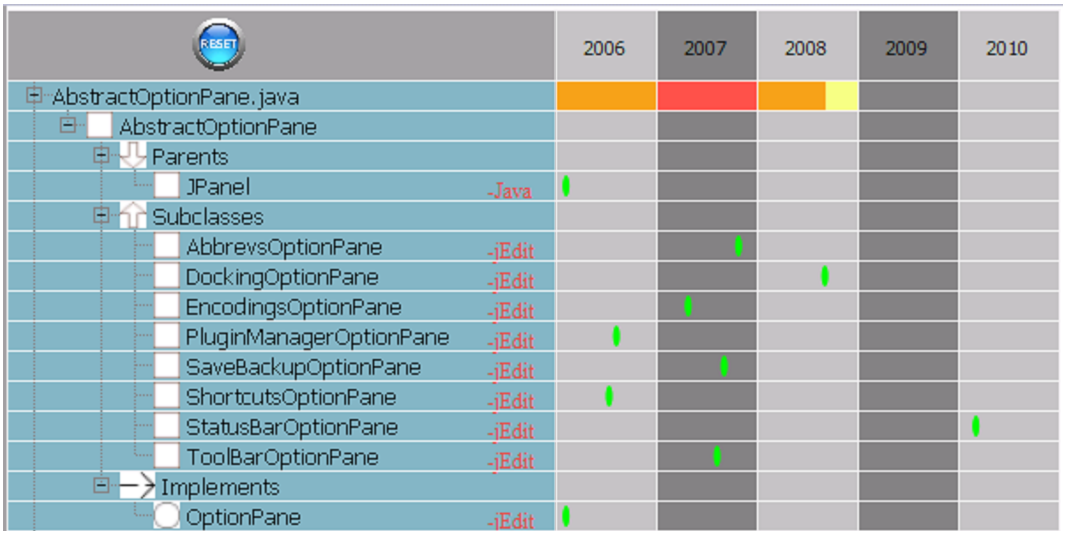


Figure C.8: Herencia de un elemento de software y relaciones de implementación de interfaces para el archivo *AbstractOptionPane.java* en *jEdit*.

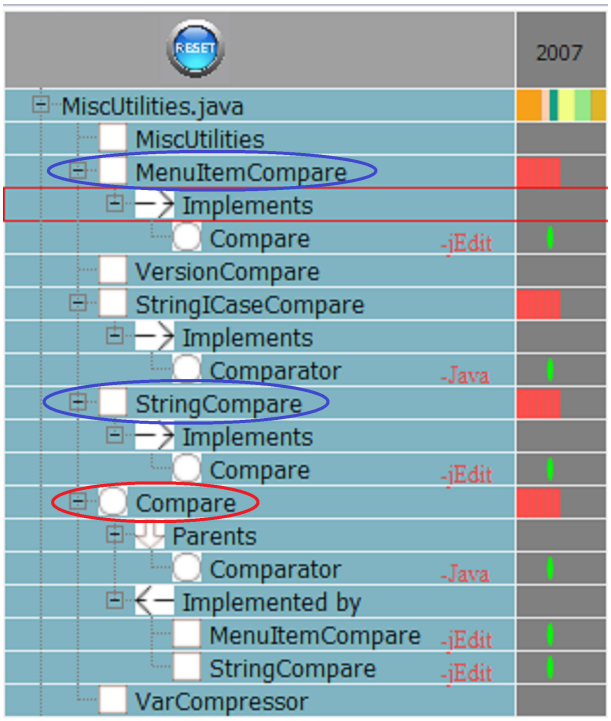


Figure C.9: Relaciones de implementación para la interfaz *Comparator* del proyecto *jEdit*.

La figura C.10 muestra que el archivo *OperatingSystem.java* (ubicado en el paquete *installer* de *jEdit*) contiene 10 clases y varias relaciones de herencia

(ninguna de las clases está implementando una interfaz). Por otra parte dicha figura representa una jerarquía en la cual *OperatingSystem* es una clase que tiene cuatro sub-clases (*HalfAnOs*, *Unix*, *VMS* and *Windows*) de las cuales a su vez heredan otras clases, como es el caso de la clase *MacOS* que hereda de la clase *Unix*. Además, la figura C.10 muestra que la mayoría de las clases del archivo *OperatingSystem.java* establecieron relaciones de herencia durante el año 2003, y también muestra que las métricas para las clases *Unix*, *Windows* y *OSTask* tienen algunos ligeros cambios durante el mismo año.

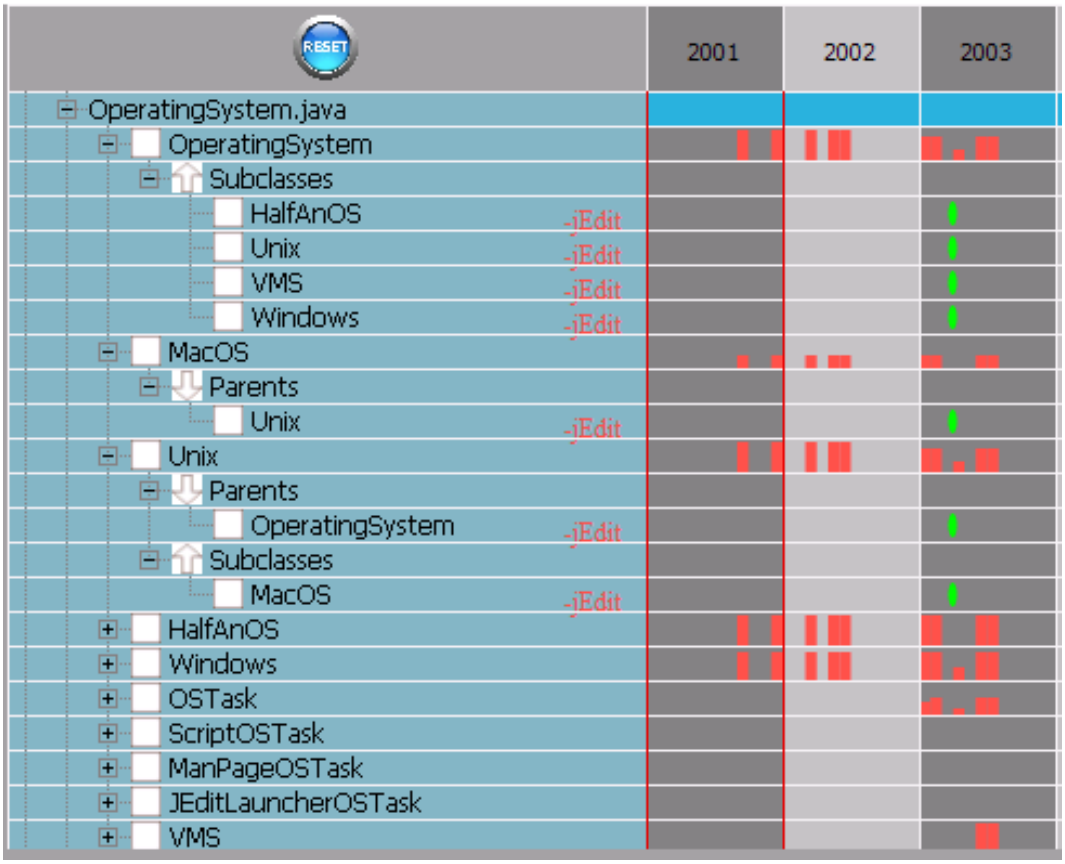


Figure C.10: *OperatingSystem.java*: este archivo contiene 10 clases, las cuales hacen uso intensivo de herencia.

El archivo *HelpViewer.java* ha evolucionado desde el año 2002 hasta el mes de abril de 2014 (una vista parcial de esta línea de vida se muestra en la figura C.11) y contiene 6 clases (*HelpViewer, LinkHandler, KeyHandler, ActionHandler, PropertyChangeHandler* y *AsyncHTMLEditorKit*) que han establecido relaciones de herencia o de implementación de interfaces. En consecuencia, 3 de estas clases han establecido relaciones de herencia con otras clases mientras que 4 de han implementado interfaces.

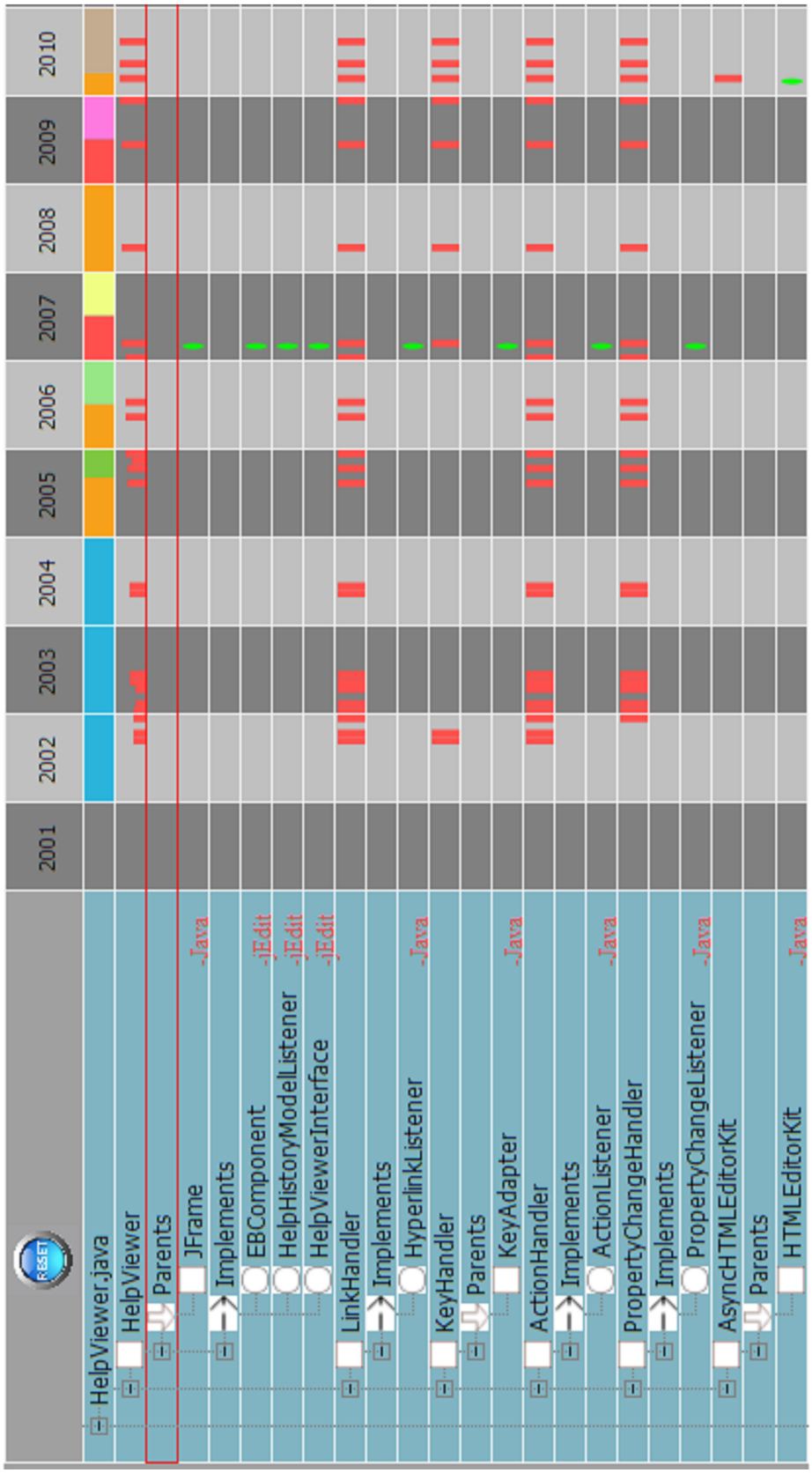


Figure C.11: Relaciones entre los elementos de software de *HelpViewer.java* (*jEdit*).

Además, el valor de la métrica NOM asociada a 5 de las 6 clases no han cambiado con el tiempo, con la excepción de los valores de la clase *HelpViewer* (ver los años 2003, 2005 y 2007 en la figura C.11).

Un detalle importante a destacar es que durante los años 2006, 2007 y 2008 se registró una intensa actividad en el establecimiento o modificación de las relaciones de herencia o de implementación de interfaces en *jEdit*, como se puede observar en las figuras C.7, C.8, C.9, C.10 and C.11. Por lo tanto, esto podría indicar que durante esos años se llevó a cabo una restructuración del sistema, y de forma particular durante el año 2007, que es el año para el cual se muestra una mayor actividad en este sentido.

Por último, merece la pena resaltar que *Gridmaster* utiliza varias técnicas de interacción que incluyen las posibilidades de hacer *zoom-in* y *zoom-out*, la distorsión de ojo de pez, y la capacidad de filtrar los nodos de la estructura. Además, permite selecciomar un año de la línea de tiempo para representar los datos de acuerdo con los meses de ese año.

C.3.3 Socio-Technical Graph: Representación de la Collaboración y Relaciones entre Programadores

STG es una vista complementaria (ver figura C.12) que se basa en una representación gráfica y muestra cuando una unidad de tiempo (*e.g.*, un año o un mes) es seleccionada de *Gridmaster*. Esta visualización está dirigida a la representación de las contribuciones de los programadores (en términos del número de archivos y revisiones) y la relación entre ellos que se deriva de los elementos de software que han cambiado en común. En consecuencia, los nodos representan las contribuciones de los programadores y los colores están asociados a los nombres de usuario de los programadores, mientras que las aristas representan las relaciones de colaboración que se han establecido entre los programadores a partir de los cambios que han hecho a los elementos de software.

El tamaño de los nodos ganglios en STG representa el número de aportaciones realizadas por los programadores, las cuales están determinadas por el número de archivos que han sido modificados en cada *commit* (el cálculo de los pesos de los nodos difiere del cálculo realizado por Jermakowicz *et al.* [Jermakovics 2011] que sólo tiene en cuenta el número de *commits*).

El espesor de las aristas representa el número de elementos de software que los programadores asociados han cambiado en común. Por lo tanto, la fuerza de la relación de colaboración entre dos programadores se puede deducir a partir del espesor de la arista que los une. Por lo tanto, esta visualización es útil en situaciones en las cuales es necesario redistribuir las tareas de un programador debido a una situación inesperada o a

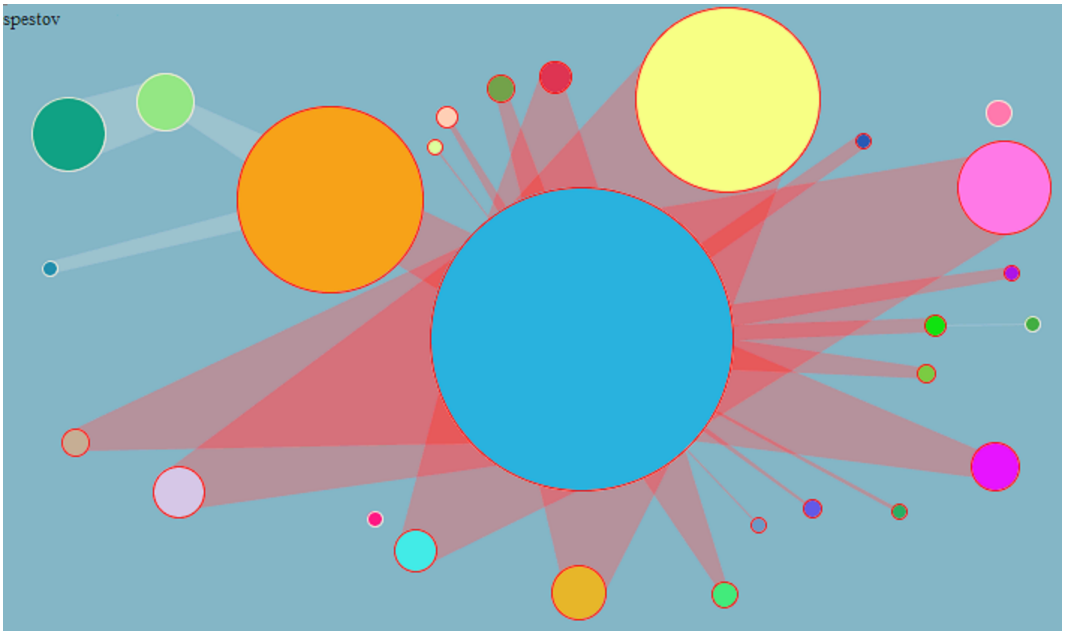


Figure C.12: STG mostrando las contribuciones y relaciones entre los programadores, con base en los elementos de software que han cambiado en común.

cambios organizacionales, donde hipotéticamente, las tareas de programación realizadas por el *programador A* pueden ser llevadas a cabo por el *programador B*.

El diseño descrito se muestra en la figuraC.12 (una captura de pantalla de STG), y representa las contribuciones de los programadores y sus relaciones de colaboración entre los años 2001 y 2014 para *jEdit*. Esta figura permite observar que para el período de tiempo dado los programadores con más contribuciones son *spestov* (turquesa), *ezust* (naranja), *k_satoda* (amarillo claro), y *daleanson* (violeta claro).

Dicha figura también permite deducir que *spestov* ha intervenido en la modificación de la mayoría de los elementos de software en *jEdit* debido a que es el programador que tiene más conexiones con otros programadores (resaltadas por las aristas de color rojo). De forma adicional, la identificación precisa de los elementos de software que *spestov* ha modificado en común con otros programadores podría lograrse mediante la revisión de las correlaciones entre los programadores y los elementos de software en *Gridmaster*. Aunque la participación de *spestov* fue muy intensa durante los primeros años de la evolución de *jEdit* su contribución al proyecto se detuvo en el año 2005.

Un aspecto relevante es que las relaciones entre los programadores fueron mínimas durante el año 2013 (ver figura C.13), donde *shlomy* (violeta) se concentró la mayor parte de las relaciones con *ezust* (naranja), *kpouer* (amarillo), *daleanson* (violeta claro) y *Vampire0* (marrón); algunos de los

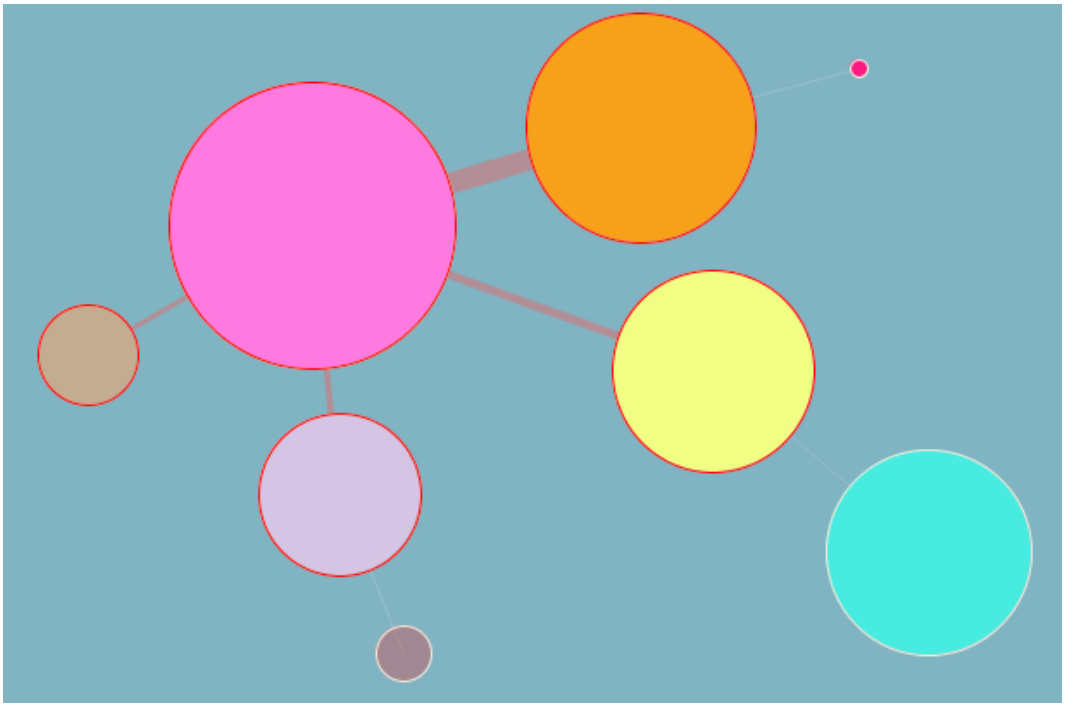


Figure C.13: Captura de pantalla de STG para *jEdit* y el año 2013.

cuales han contribuido a emph jEdit por muchos años. Además, *thomasmey* (verde claro), que hizo un número alto de contribuciones, en términos relativos, durante el año 2013, estableció una relación débil con *kpouer* de acuerdo con los elementos de software que han cambiado en común (esto puede investigarse más a fondo en *Gridmaster* después de expandir los paquetes *org–>gjt–>sp–>jedit* and *org–>gjt–>sp–>util*).

La situación descrita se acentuó durante los primeros 4 meses de 2014 (enero a abril), como es representado por la figura C.13. 3 de los 4 programadores que han contribuido con *jEdit* durante 2014 no tienen conexiones con otros programadores y la única relación que se ha establecido (entre *ezust* y *daleanson*) es demasiado débil para ser considerada relevante en términos de la colaboración.

C.3.4 Diseño de Revision Tree

El diseño de RT se realizó como resultado de un análisis de requerimientos que se llevó a cabo para este tipo de herramientas, así como las características deseables para este tipo de herramientas de visualización que fueron identificadas por Therón *et al.* [Therón 2007].

RT una estructura de matriz, una línea de tiempo y un árbol para representar los detalles de evolución de un elemento de software, como se puede observar en la figura C.14 y la tabla C.2.

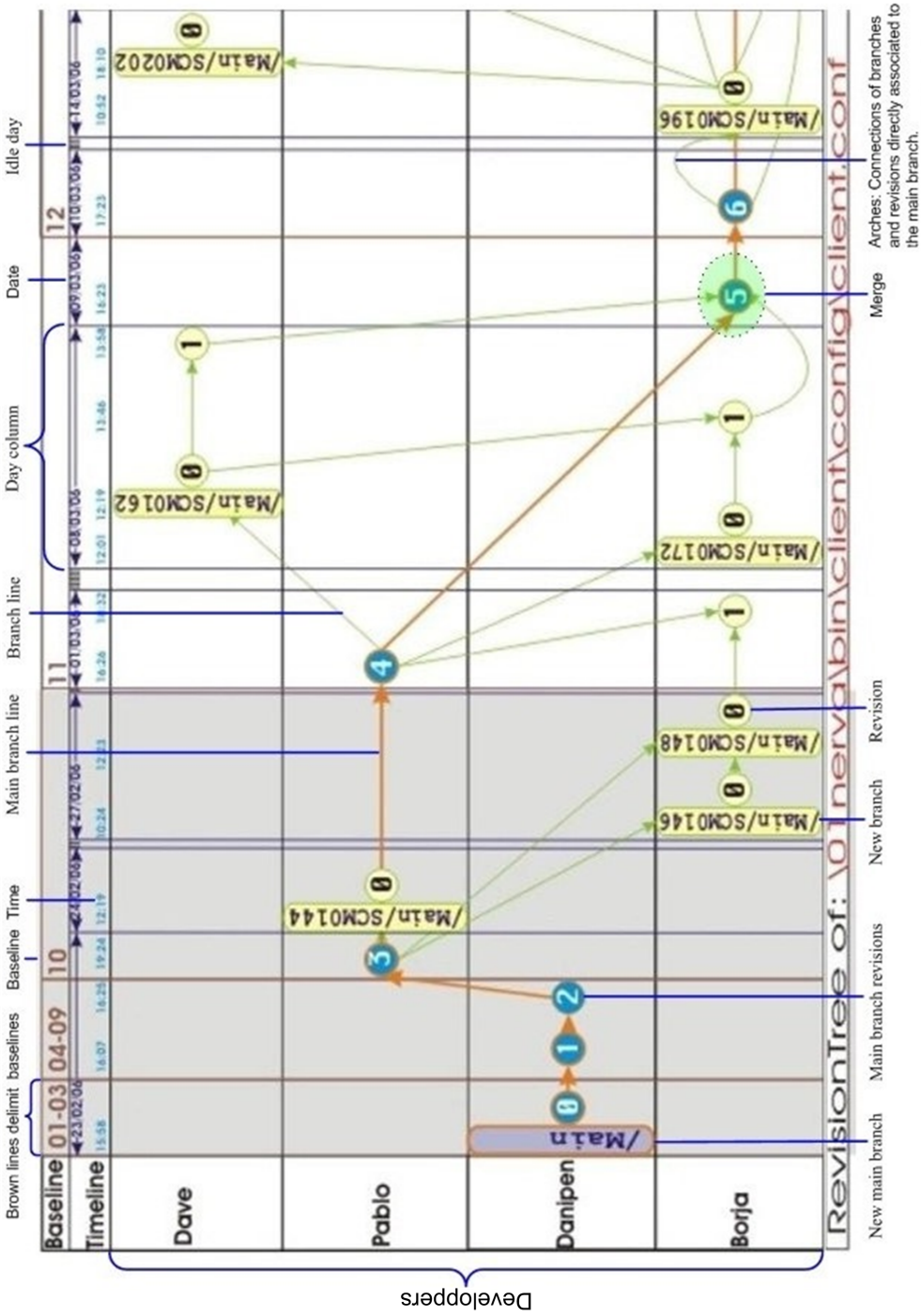


Figure C.14: Bosquejo del diseño de Revision Tree.

Table C.2: Elementos visuales y variables representadas por el Revision Tree.

Elemento visual	**Descripción**	**Representación**
Autores	Nombres de los programadores.	Etiqueta con el nombre del programador.
Baseline	Número de las baselines.	Se despliega en la línea de tiempo.
Fecha	Indica la fecha de creación de las ramas, baselines y revisiones.	Etiqueta con la fecha.
Columna del día	Este es el espacio gráfico para la representación de un día con actividad en la creación de ramas, baselines y revisiones.	Una línea de color azul oscuro con flechas en los extremos.
Hora	Muestra la hora en la cual se ha creado una nueva rama o revisión en la rama principal o cualquier otra rama.	Etiqueta con la hora.
Nueva rama principal	Indica la creación de la rama principal.	Óvalo violeta.
Nueva rama	Muestra la creación de una nueva rama.	Óvalo amarillo.
Línea rama principal	Resalta la rama principal.	Flechas naranja.
Arcos	Conecta las ramas y revisiones en la rama principal.	Archos verdes.
Revisiones rama principal	Revisiones creadas en la rama principal.	Nodos azules.
Línea de rama	La línea de las conectan a la rama principal con otras ramas y las revisiones en esa rama o entre dos ramas.	Línea verde.
Revisión	Representa la creación de una nueva revisión de un elemento de software.	Nodos amarillos.
Combinación	Una combinación o *merge* occure cuano uno o más ramas son combinadas con la rama principal.	Línea entrantes en la rama principal.

RT muestra un gran número de detalles que incluyen el nombre de los programadores, su participación en la realización de cambios, la propiedad de un archivo de código fuente (basado en los cambios realizados) en determinados períodos de tiempo, el *id* y fecha de *baselines* y revisiones, así como detalles sobre la creación y fusión de las ramas, y la colaboración entre programadores en el tiempo.

C.3.4.1 Características de Revision Tree

La evolución de cada archivo de código fuente contiene implícito un atributo temporal, que es el elemento más importante en la comprensión del proceso de desarrollo de cualquier sistema. Por lo tanto, la visualización de la evolución de archivos presenta varios desafíos que fueron abordados por RT. Algunos de estos desafíos se asociaron con la correlación de las *baselines* y las revisiones, así como con el uso de técnicas de interacción para descubrir hechos relevantes. Así, los retos que se abordaron con este diseño son los siguientes:

* La representación de grandes árboles de revisiones, donde las *baselines* tienen varias ramas y cada rama muchas revisiones.
* La navegación del árbol de versiones mediante una vista del tipo foco + contexto.
* Soporte a la interacción para permitir la inspección de más de una *baseline* a la vez y mostrar la colaboración de los desarrolladores en cada *baseline*.
* Las relaciones entra las *baselines*.
* La asociación jerárquica entre las *baseline* y las revisiones, y la correlación de toda la información con la línea de tiempo.

En consecuencia, las siguientes secciones explican los detalles del diseño de esta visualización y las posibilidades de interacción que ofrece.

Diseño de matriz: RT utiliza una estructura basada en una matriz para proporcionar un mecanismo intuitivo para visualizar la relación entre programadores y *baselines* mediante el uso de las filas para representar a los programadores y las columnas para las *baselines*. Además, estructuras de matriz resultan familiares a los desarrolladores y las celdas se pueden usar como contenedores para dibujar los nodos del grafo dirigido que representa el flujo de revisiones para el archivo.

Línea de tiempo: La línea de tiempo de RT utiliza columnas de ancho variable para acomodar las revisiones en cada *baseline* (ver figura C.15). La distribución de las filas es uniforme en la línea de tiempo y se compone de dos filas, la primera fila se utiliza para la numeración de las *baselines* y la segunda fila para representar los atributos temporales sobre la creación de las revisiones

(hora y fecha). Por otra parte, la segunda fila incluye elementos visuales adicionales, tales como las líneas azules horizontales con flechas en ambos extremos para enfatizar un día en particular y las líneas negras verticales para indicar el final de un día; los nodos rectangulares redondeados se utilizan para hacer hincapié a la creación de ramas y la línea naranja que conecta los óvalos azules para delinear la rama principal .

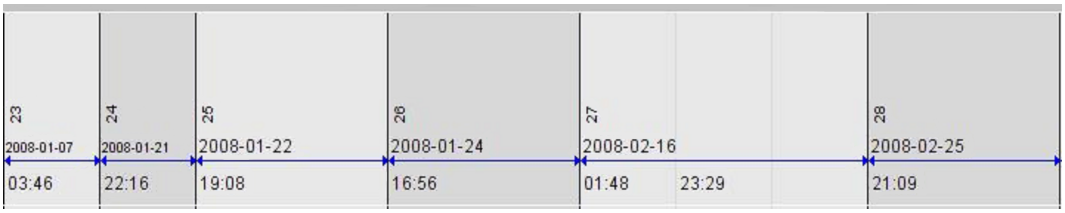


Figure C.15: Detalles de la línea de tiempo [Therón 2007, Therón 2008].

RT representa las *baselines*, revisiones y ramas de acuerdo con la información de la línea de tiempo y los desarrolladores que están trabajando en las revisiones. Las revisiones se encuentran en la intersección entre las filas (que representa a los programadores) y las columnas (que representan puntos específicos en la línea de tiempo). Las revisiones están representadas por óvalos y el número de revisión está alineado de forma horizontal si se compone de un dígito y verticalmente si tiene más de un dígito. Los óvalos azules son las revisiones de la rama principal y los demás óvalos son revisiones dentro de las ramas de desarrollo secundario. La línea naranja que conecta los óvalos azules delinea la rama principal y la línea verde a las ramas secundarias. Sin embargo, cuando el programador trabaja en varias ramas y donde las revisiones pertenecen a más de una rama, la línea que conecta las revisiones se compone de más de un color; donde cada color representa una rama específica.

Esta representación permite ver todas las *baselines* y revisiones de un vistazo, así como las relaciones entre las *baselines* y la asociación jerárquica entre ramas y revisiones.

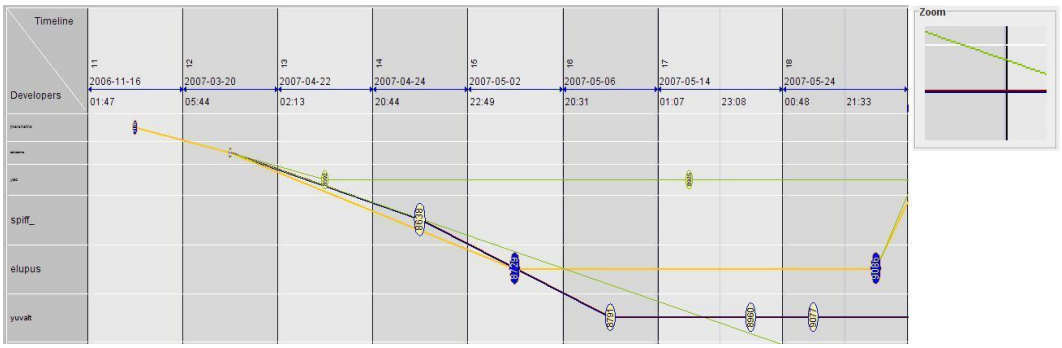


Figure C.16: Correlación de la evolución de un elemento de software con la línea de tiempo [Therón 2007, Therón 2008].

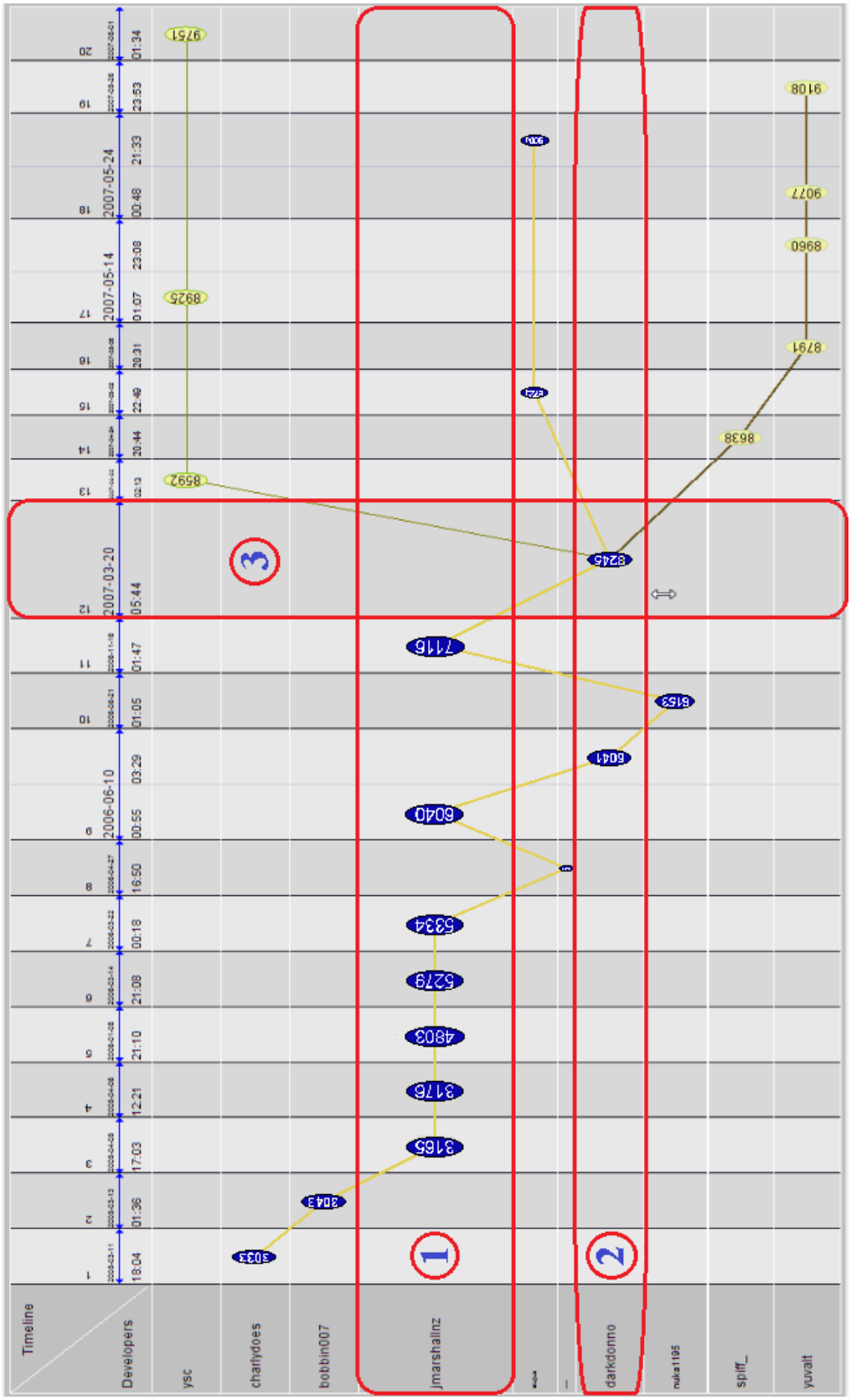


Figure C.17: Revision Tree: vista polifocal.

La figura C.16 muestra una vista parcial de RT. En un pequeño recuadro en la esquina superior derecha de la figura se puede observar el uso de la codificación de colores en la representación de dos ramas que comparten algunas de sus revisiones. El recuadro mencionado muestra el acercamiento a un segmento de las ramas y permite observar dos colores diferentes, uno para cada rama.

Vista bifocal y polifocal: RT utiliza una técnica de foco + contexto que se basa en vistas bifocales y polifocales, con las filas de la misma altura y las columnas de ancho variable, según el número de revisiones en las *baselines*.

La vista bifocal ofrece la posibilidad de expandir una fila o columna, mientras que la vista polifocal ofrece la misma posibilidad pero permite concentrarse en más de una área (varias filas, columnas o una combinación de ambas). La figura C.17 permite apreciar la visualización polifocal: dos filas y una columna están expandidas (resaltadas por rectángulos de color rojo y los números *1*, *2* y *3*) mientras que las otras filas y columnas se encuentran reducidas. Cuando esta interacción se lleva a cabo, la visualización mantiene en la pantalla toda la información de las versiones de un elemento software y permite al usuario concentrarse en la zona en la cual el elemento de software ha registrado mayor actividad.

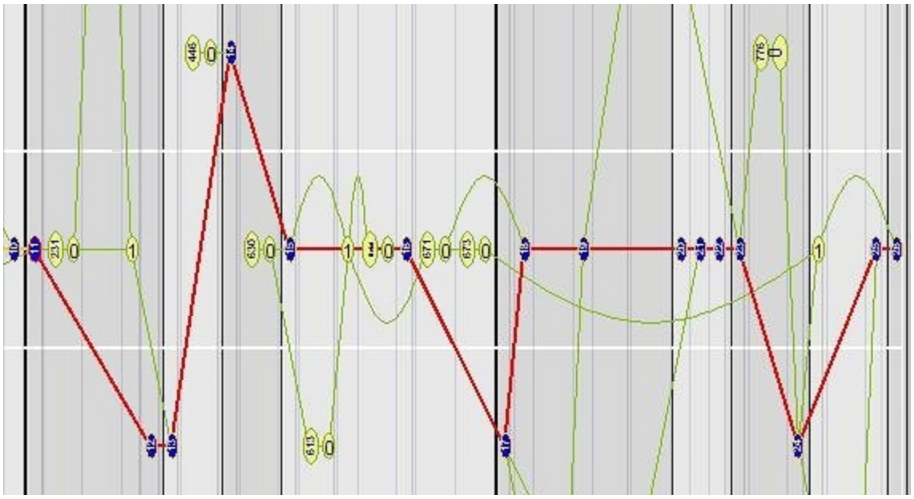


Figure C.18: Línea de desarrollo principal puesta de relieve.

Interacción: RT permite que los usuarios seleccionen la rama principal o las ramas secundarias para resaltarlas y tener una mejor visión de la información cuando la visualización aparece saturada al representar árboles de evolución complejos. Por lo tanto, cuando la rama principal es seleccionada en algún punto de la representación, se pone de relieve el camino restante de dicha rama, como se muestra en la figura C.18, y en el caso de las ramas secundarias el camino restante es resaltado hasta que se fusiona con la rama

principal. Esta característica es aún más valiosa cuando existe una rama paralela la rama principal y es necesario poner de relieve la conexión entre revisiones o el punto de combinación con la rama principal de una de estas ramas, como se muestra en la figura C.19.

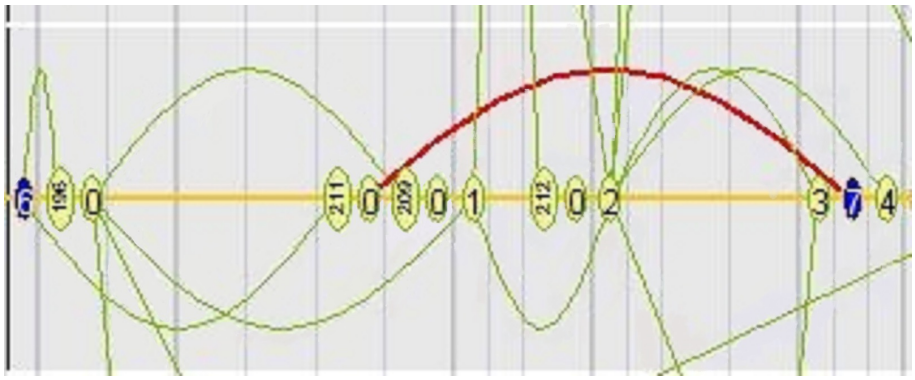


Figure C.19: Resaltado de una rama paralela a la rama principal.

Esta visualización es apoyada por varias técnicas de interacción adicionales para generar nuevas perspectivas de visualización y permitir el descubrimiento de información que no es visible a simple vista. La figura C.20a muestra una vista normal de RT, mientras que la figura C.20b muestra una vista en la cual la fila que corresponde a un desarrollador ha sido ocultada y la figura C.20c presenta una vista en la cual un programador y una *baseline* han sido ocultadas.

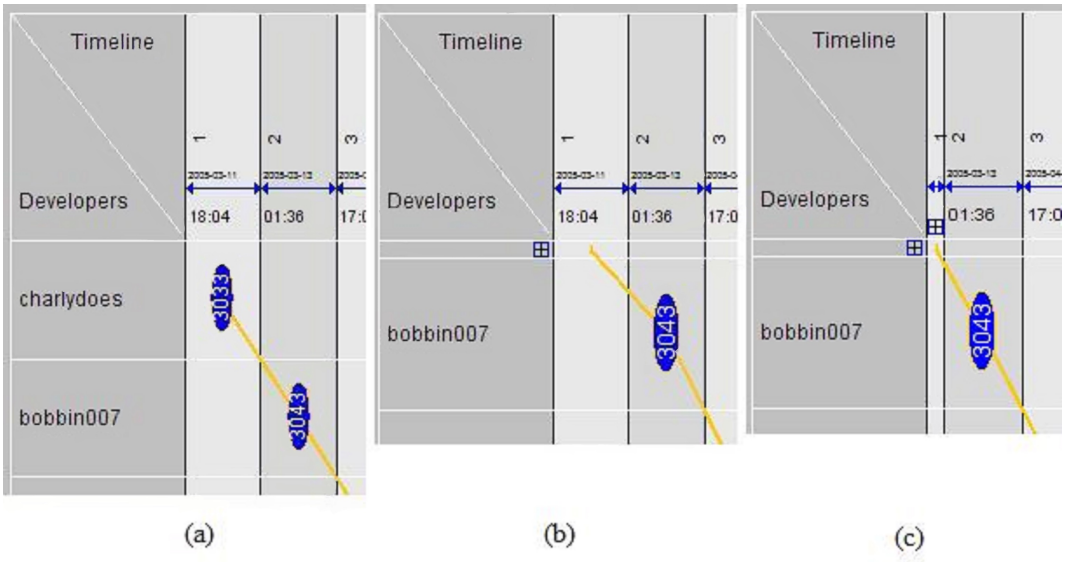


Figure C.20: Ocultado de filas y columnas.

Lo anterior puede ser útil para tener la misma representación para un período limitado de tiempo (sin hacer filtrado por fechas), o para incluir

solamente la información relativa a los desarrolladores seleccionados. El usuario puede seleccionar de este modo todo el período de la evolución del elemento de software o un período más limitado de tiempo y ocultar columnas de *baselines* que no desea que aparezcan en la representación.

C.4 Conclusions

Los desarrolladores y administradores de proyectos necesitan comprender los sistemas de software que están desarrollando y brindando mantenimiento, en particular, cuando no tienen conocimiento previo o documentación de esos sistemas. Esta situación adquiere mayor importancia al considerar que los procesos Desarrollo, Mantenimiento y Evolución de Software (DMES) por lo general se extienden a lo largo de varios años y producen grandes volúmenes de datos. En consecuencia, esta investigación ha tenido en cuenta que:

1. Los desarrolladores de software y administradores de proyectos requieren comprender los sistemas de software y los cambios que se producen durante los procesos DMES.
2. Existe una evidente necesidad de utilizar métodos de análisis para reducir el volumen de datos que necesita ser examinado y estudiado por los programadores y administradores de proyectos.
3. El uso del análisis de software o AES, en función del número de revisiones o el período de tiempo de estudio, puede dar lugar a la generación de conocimiento útil.
4. Es frecuente que los resultados del análisis de software y AES sean voluminosos y complejos para producir conocimiento que permita llevar a cabo la resolución de problemas de forma eficaz.

Por lo tanto, las observaciones anteriores y los resultados positivos de la aplicación de AV a diferentes áreas del conocimiento (como se explica en el capítulo 3) motivó el estudio y análisis sobre su uso en la comprensión de los sistemas de software y su evolución en investigaciones académicas (capítulos 4, 5 and 6) y en la industria del software (véase el capítulo 7). En consecuencia, las conclusiones de este análisis fueron las siguientes:

1. Existe un gran número de trabajos de investigación que muestran evidencia de la aplicación de VI a los sistemas de software y su evolución, pero el número de trabajos de investigación que utilizan AV en el análisis de sistemas de software es reducido.
2. La mayor parte de la investigación que se ocupa de la aplicación de AV a los sistemas de software se centran en cuestiones teóricas y

metodológicas, pero no definen procesos, modelos o arquitecturas que faciliten la elaboración de herramientas para llevar a cabo el análisis de uno o múltiples revisiones de los sistemas.

3. Un gran número de investigaciones implementan sus propuestas como plugin de algunos de los IDE más populares, como por ejemplo *Eclipse*.
4. El número de propuestas de herramientas que hacen uso de las tecnologías web es también reducido (*e.g.*, sólo uno de los trabajos de investigación que fueron revisados en esta investigación utilizan ese tipo de tecnologías).
5. El uso de vistas individuales es la opción preferida de la mayoría de trabajos de investigación clasificados tanto como *Sys* como *Evol*.
6. Existen similitudes proporcionales entre la investigación clasificada como *Sys* y *Evol* en el uso de *Vistas Múltiples* y *Vistas múltiples Vinculadas*, aunque los trabajos de investigación clasificados en la categoría de *Evol* representa un mayor número de elementos de datos y relaciones más complejas en comparación con la investigación clasificada bajo la categoría *Sys*.
7. Las empresas utilizan herramientas ACS y de control de errores (*bug tracking*) para registrar y gestionar los datos relacionados con los procesos DMES, y algunas de estas además utilizan estas herramientas de forma integrada con el fin de contar con información correlacionada sobre los errores o fallas con las revisiones. El principal objetivo de esto último es que podría conducir a un mejor seguimiento de los cambios y la evolución del proyecto.
8. El plan de estudios de muchos cursos universitarios no incluyen contenidos sobre las herramientas que se utilizan de forma empírica en los procesos de ingeniería de software.
9. No existe evidencia sustancial sobre la difusión y transferencia de los resultados de investigación a la industria, con respecto a la aplicación de VI a los sistemas de software y su evolución.
10. La mayoría de las visualizaciones que son utilizadas por la industria de software o empresas son sencillas y están, en su mayoría, integradas en las herramientas ACS y los IDE. De forma adicional, los programadores no son conscientes de las opciones de herramientas de este tipo que tienen disponibles los IDE.
11. Tareas como la depuración, navegación de dependencias, detección de copias de código fuente, refactorización, el seguimiento de los cambios y contribuciones y el seguimiento de los cambios en las métricas de calidad se lleva a cabo en la industria sin el apoyo de herramientas de visualización.
12. Existen muchas descripciones generales del proceso seguido por VES,

pero no se ha descrito el proceso para la aplicación de AV a ES.

A partir de estas conclusiones, se definió el proceso AVAES para satisfacer la necesidad de efectuar una descripción adecuada de los procesos involucrados en la aplicación de AV to ES [González-Torres 2013b, González-Torres 2013a]. Este proceso validado por medio de la implementación de una arquitectura que siguió su descripción. En consecuencia, la descripción de dicho proceso y su validación se dividió en tres etapas:

1. La definición de los básicos que hicieron posible la construcción de la caracterización de AVAES. Estos elementos incluyen la explicación de los términos y conceptos de ES y el proceso de análisis avanzado de datos.
2. El análisis y discusión del uso de la visualización en los sistemas de software permitió determinar las tareas que son apoyadas, así como los elementos de datos y tipos de visualización utilizados en los entornos industriales y académicos.
3. La definición, descripción y validación del proceso AVAES.

La definición de AVAES proceso proporciona detalles acerca de los principales componentes (y la forma en que se interrelacionan e interactúan entre sí), métodos y técnicas que intervienen en la transformación de los datos derivados del análisis de la evolución de los sistemas de software en conocimiento útil para facilitar una comprensión más profunda de los procesos de DMES.

Con base en la descripción de AVAES, la arquitectura de *Maleku* fue diseñada e implementada para probar si la aplicación de este proceso era factible y si podría ser utilizado como base para la definición de arquitecturas de herramientas para apoyar los procesos de DMES. El diseño de *Maleku* identificó y explicó los roles, límites e interacciones entre módulos, componentes, y algunos de los métodos y técnicas que podrían ser utilizados por dicho proceso, y permitió responder a las preguntas de investigación subsidiarias.

A partir de ahí, la validación de *Maleku* fue llevada a cabo en tres pasos y tenía como principal objetivo la verificación del cumplimiento de los objetivos y funciones de acuerdo con el diseño de su arquitectura. Esta verificación permitió demostrar la utilidad de la herramienta. Los tres pasos del proceso de validación que se llevaron a cabo son los siguientes:

1. Probar la herramienta mediante escenarios de uso con el fin de evaluar su funcionalidad.

2. Caso de estudio sobre RT y su relación con una herramienta ACS comercial.
3. Prueba de evaluación con usuarios expertos.

Esto permitió probar que la descripción de AVAES se puede seguir de manera eficaz y por lo tanto permite el diseño e implementación de arquitecturas para ayudar a comprender los datos derivados de ES por parte de los programadores y administradores de proyectos, y por lo tanto permite apoyar los procesos de DMES conforme con las metas, tareas y objetivos especificados.

Además, este terminó de confirmar la validez de la arquitectura implementada mediante el uso de escenarios, un caso de estudio y una prueba de evaluación de usuario. En síntesis, el objetivo de esta validación (que fue validar el ciclo completo de la aplicación AV a ES en el diseño e implementación de una herramienta para apoyar a los desarrolladores y administradores de proyectos en el desarrollo y mantenimiento de software) se cumplió y la pregunta principal de investigación fue respondida.

En general, la utilidad de *Maleku* se demostró en el suministro de información estadística sobre las revisiones, así como las aportaciones que han realizado los programadores, la evolución de la estructura del proyecto, las líneas de vida de los elementos de software (incluyendo los paquetes, archivos, clases e interfaces), la evolución de la herencia y la implementación de interfaces, métricas de evolución, y colaboraciones de programador. Los resultados de la validación de *Maleku* mostraron la utilidad de esta herramienta para:

1. Ayudar en la comprensión de los cambios en las métricas de calidad de software, así como las relaciones socio-técnicos y de colaboración durante la evolución del proyecto o un periodo de tiempo determinado.
2. Ayudar en el proceso de comprensión de los cambios en las estructuras del proyecto de software, la herencia y la implementación de interfaces para el proyecto completo o un período de tiempo determinado.
3. Apoyar la comprensión de los cambios durante la evolución del proyecto de software mediante la comparación de los períodos de tiempo.

Por último, el uso de herramientas visuales para ayudar a las tareas de programación y de gestión debe ser patrocinada por jugadores clave en la industria del software (*e.g.*, Microsoft, IBM y Borland). Podría ser utilidad la incorporación de herramientas visuales completas en sus IDEs, ACS y herramientas de gestión de errores, así como la creación de cursos de formación y documentación técnica que las tome en cuenta como elementos centrales.

C.5 Trabajos Futuros

La computación en la nube se ha popularizado en los últimos años y como consecuencia, actualmente se están ofreciendo muchos servicios basados en la nube. Así, un gran número de empresas están utilizando estos servicios para llevar a cabo diferentes tareas de negocio. La industria del software no es una excepción y, recientemente, un gran número de IDEs basados en la nube se encuentra disponibles (*e.g.*, *Codenvy*, *Cloud 9* y *Code Anywhere*). Este tipo de herramientas también se pueden considerar en un futuro cercano como una alternativa para apoyar los procesos GSD. Sin embargo, no existe información fiable sobre el uso de estos IDEs ni proyecciones sobre su uso futuro.

Un punto adicional a tener en cuenta es que las versiones basadas en web de herramientas ACS y de gestión de errores han estado disponibles desde hace muchos años, y por lo tanto, éstas herramientas podrían ser migradas a la nube o utilizar interfaces de software para conectarlas con los IDE basados en la nube.

La tendencia mencionada abre una oportunidad para contribuir con los programadores y administradores de proyectos que utilizan herramientas de desarrollo basadas en la nube. En consecuencia, como trabajo futuro se llevará a cabo el diseño e implementación de una arquitectura basada en AVAES que será integrada como un plugin en un IDE en la nube seleccionado (esta arquitectura podía incluir algunos componentes de *Maleku*). Dos requisitos fundamentales de esta arquitectura es que su diseño debe ser *responsive* y utilizar tecnologías *restful*. Además, tendrá en cuenta el uso de herramientas del ecosistema de Hadoop para procesar y almacenar datos de manera oportuna y segura.

Además, la arquitectura apoyará el análisis dinámico de los sistemas en tiempo de ejecución, cuando sea aplicable de acuerdo con el lenguaje de programación del sistema bajo análisis. Además, los módulos de análisis de datos apoyarán al menos tres de los lenguajes de programación más comunes hoy en día. Los comentarios realizados por los participantes en el estudio de usabilidad en esta investigación serán utilizados como insumo para el diseño de las visualizaciones (o la re-implementación de las visualizaciones utilizadas en *Maleku*), la interacción y la herramienta en general.

Por último, el uso de herramientas de visualización (durante el proceso de desarrollo y mantenimiento), tanto en la industria del software y como en los departamentos de desarrollo de las empresas, será objeto de seguimiento. Con este fin, una versión mejorada de la encuesta introducida en esta investigación se llevará a cabo cada dos años. La versión mejorada de la encuesta incluirá un mayor número de temas y participantes.

Bibliography

[2015] *JFreeChart*. http://www.jfree.org/index.html/, 2015. [Online; accessed 28-March-2012]. 259

[Abuthawabeh 2013] Ala Abuthawabeh, Fabian Beck, Dirk Zeckzer and Stephan Diehl. *Finding structures in multi-type code couplings with node-link and matrix visualizations*. In First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pages 1–10, 2013. 122, 123, 144, 306

[Academies 2000] The National Academies. How people learn: brain, mind, experience, and school. National Academy Press, 2000. 203

[Adamoli 2010] Andrea Adamoli and Matthias Hauswirth. *Trevis: a context tree visualization analysis framework and its use for classifying performance failure reports*. In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 73–82, New York, NY, USA, 2010. ACM. 305

[Aftandilian 2010] Edward E. Aftandilian, Sean Kelley, Connor Gramazio, Nathan Ricci, Sara L. Su and Samuel Z. Guyer. *Heapviz: interactive heap visualization for program understanding and debugging*. In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 53–62, New York, NY, USA, 2010. ACM. 305

[Agerfalk 2006] Pär J. Agerfalk and Brian Fitzgerald. *Flexible and Distributed Software Processes: Old Petunias in New Bowls?* Communications of the ACM, vol. 49, no. 10, pages 26–34, October 2006. 147, 150

[Aggarwal 2005] K.K. Aggarwal, Yogesh Singh, Pravin Chandra and Manimala Puri. *Measurement of Software Maintainability Using a Fuzzy Model*. Journal of Computer Sciences, vol. 1, no. 4, 2005. 41

[Agrafiotis 2010] Dimitris K. Agrafiotis and John J. M. Wiener. *Scaffold Explorer: An Interactive Tool for Organizing and Mining Structure-Activity Data Spanning Multiple Chemotypes*. Journal of Medicinal Chemistry, vol. 53, no. 13, pages 5002–5011, 2010. PMID: 20524668. 48, 49

[Agrawal 1990] Hiralal Agrawal and Joseph R. Horgan. *Dynamic program slicing.* In Proceedings of the ACM SIGPLAN 1990 conference on Programming language design and implementation, PLDI '90, pages 246–256, New York, NY, USA, 1990. ACM. 43

[ah Kang 2011] Youn ah Kang, Carsten Görg and John Stasko. *How Can Visual Analytics Assist Investigative Analysis: Design Implications from an Evaluation.* IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 5, pages 570 –583, may 2011. 49

[ah Kang 2012] Youn ah Kang and John Stasko. *Examining the Use of a Visual Analytics System for Sensemaking Tasks: Case Studies with Domain Experts.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, pages 2869–2878, 2012. 49

[Aigner 2005] Wolfgang Aigner, Silvia Miksch, Bettina Thurnher and Stefan Biffl. *PlanningLines: Novel Glyphs for Representing Temporal Uncertainties and Their Evaluation.* In Proceedings of the Ninth International Conference on Information Visualisation, IV '05, pages 457–463, Washington, DC, USA, 2005. IEEE Computer Society. 52, 54

[Alcocer 2013] Juan Pablo Sandoval Alcocer, Alexandre Bergel Stéephane Ducasse and Marcus Denker. *Performance evolution blueprint: Understanding the impact of software evolution on performance.* In First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pages 1–9, 2013. 305

[Ali 2009] Jauhar Ali. *Cognitive support through visualization and focus specification for understanding large class libraries.* Journal of Visual Languages & Computing, vol. 20, no. 1, pages 50 – 59, 2009. 124, 306

[Alsallakh 2012] Bilal Alsallakh, Wolfgang Aigner, Silvia Miksch and M. Eduard Groller. *Reinventing the Contingency Wheel: Scalable Visual Analytics of Large Categorical Data.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, pages 2849–2858, 2012. 49

[Amicis 2009] Raffaele De Amicis, Giuseppe Conti, Bruno Simões, Raimondo Lattuca, Nicolò Tosi, Stefano Piffer and Giuseppe Pellitteri. *Geo-visual analytics for urban design in the context of future internet.* International Journal on Interactive Design and Manufacturing, vol. 3, pages 55–63, 2009. 10.1007/s12008-009-0060-1. 49

[André 2007] Paul André, Max L. Wilson, Alistair Russell, Daniel A. Smith, Alisdair Owens and m.c. schraefel. *Continuum: designing timelines for hierarchies, relationships and scale.* In UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology, pages 101–110, New York, NY, USA, 2007. ACM. 54

[Andrews 1998] Keith Andrews and Helmut Heidegger. *Information Slices: Visualising and Exploring Large Hierarchies using Cascading, Semi-Circular Discs.* Late Breaking Hot Topic Paper, IEEE Symposium on Information Visualization (InfoVis'98), 1998. 52, 58

[Andrienko 2007] Gennady Andrienko and Natalia Andrienko. *Coordinated Multiple Views: a Critical View.* International Conference on Coordinated and Multiple Views in Exploratory Visualization, vol. 0, pages 72–74, 2007. 48

[Andrienko 2010] Gennady Andrienko, Natalia Andrienko, Sebastian Bremm, Tobias Schreck, Tatiana Von Landesberger, Peter Bak and Daniel Keim. *Space-in-Time and Time-in-Space Self-Organizing Maps for Exploring Spatiotemporal Patterns.* Computer Graphics Forum, vol. 29, no. 3, pages 913–922, 2010. 49

[Andrienko 2012a] Gennady Andrienko, Natalia Andrienko, Michael Burch and M Daniel Weiskopf. *Visual Analytics Methodology for Eye Movement Studies.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, pages 2889–2898, 2012. 49

[Andrienko 2012b] Gennady Andrienko, Natalia Andrienko, Martin Mladenov, Michael Mock and Christian Pölitz. *Identifying Place Histories from Activity Traces with an Eye to Parameter Impact.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 5, pages 675 –688, may 2012. 48

[Andrienko 2013a] Gennady Andrienko, Natalia Andrienko, Christophe Hurter, Salvatore Rinzivillo, and Stefan Wrobel. *Scalable Analysis of Movement Data for Extracting and Exploring Significant Places.* IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 7, pages 1078–1094, 2013. 48

[Andrienko 2013b] Natalia Andrienko and Gennady Andrienko. *A visual analytics framework for spatio-temporal analysis and modelling.* Data Mining and Knowledge Discovery, vol. 27, no. 1, pages 55–83, 2013. 49

[Andrienko 2013c] Natalia Andrienko and Gennady Andrienko. *Visual analytics of movement: An overview of methods, tools and procedures.* Information Visualization, vol. 12, no. 1, pages 3–24, 01 2013. 48

[Anslow 2009] Craig Anslow, James Noble, Stuart Marshall and Ewan Tempero. *Towards Visual Software Analytics.* In Proceedings of the Australasian Computing Doctoral Consortium (ACDC), Wellington, New Zealand, 2009. 76, 203, 317

[Anslow 2010] Craig Anslow, Stuart Marshall, James Noble, Ewan Tempero and Robert Biddle. *User evaluation of polymetric views using a large visualization wall.* In Proceedings of the 5th International Symposium on Software visualization, SOFTVIS '10, pages 25–34, New York, NY, USA, 2010. ACM. XIV, 163, 306

[Anslow 2013] Craig Anslow, Stuart Marshall, James Noble and Robert Biddle. *SourceVis: Collaborative software visualization for co-located environments.* In First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pages 1–10, 2013. XIV, 164, 306

[Arias-Hernandez 2012] Richard Arias-Hernandez, Tera M. Green and Brian Fisher. *From Cognitive Amplifiers to Cognitive Prostheses: Understandings of the Material Basis of Cognition in Visual Analytics.* Interdisciplinary Science Reviews, vol. 37, no. 1, pages 4 – 18, 2012. 49

[Assogba 2010] Yannick Assogba and Judith Donath. *Share: a programming environment for loosely bound cooperation.* In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '10, pages 961–970, New York, NY, USA, 2010. ACM. XV, 170, 171, 172, 173, 306

[Bade 2004] Ragnar Bade, Stefan Schlechtweg and Silvia Miksch. *Connecting time-oriented data and information to a coherent interactive visualization.* In CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 105–112, New York, NY, USA, 2004. ACM. 54, 55

[Bailey 1989] Robert W. Bailey. Human performance engineering: Using human factors/ergonomics to achieve computer system usability (2nd ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. 270

[Baker 1995] Marla J. Baker and Stephen G. Eick. *Space-filling Software Visualization.* Journal of Visual Languages & Computing, vol. 6, no. 2, pages 119 – 133, 1995. 116, 143, 196, 313

[Ball 1996] Thomas Ball and Stephen G. Eick. *Software visualization in the large.* Computer, vol. 29, no. 4, pages 33–43, Apr 1996. 112

[Balzer 2005a] Michael Balzer and Oliver Deussen. *Exploring Relations within Software Systems Using Treemap Enhanced Hierarchical Graphs.* In 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005., pages 1–6, 2005. XIII, 112, 119, 120, 121, 143

[Balzer 2005b] Michael Balzer, Oliver Deussen and Claus Lewerentz. *Voronoi treemaps for the visualization of software metrics.* In SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization, pages 165–172, New York, NY, USA, 2005. ACM Press. XIII, 56, 117, 143

[Barbara 1987] A. Kitchenham Barbara. *Controlling software projects.* Electronics and Power, vol. 33, no. 5, pages 312–315, May 1987. 154

[Barlowe 2011] Scott Barlowe, Yujie Liu, Jing Yang, Dennis R. Livesay, Donald J. Jacobs, James Mottonen and Deeptak Verma. *WaveMap: Interactively Discovering Features From Protein Flexibility Matrices Using Wavelet-based Visual Analytics.* Computer Graphics Forum, vol. 30, no. 3, pages 1001–1010, 2011. 49

[Basole 2012] Rahul C. Basole, Mengdie Hu, Pritesh Patel and John T. Stasko. *Visual Analytics for Converging-Business-Ecosystem Intelligence.* IEEE Computer Graphics and Applications, vol. 32, no. 1, pages 92 –96, jan.-feb. 2012. 49, 50, 62

[Bass 2003] Len Bass, Paul Clements and Rick Kazman. Software architecture in practice, second edition. Addison-Wesley Professional, April 2003. 112

[Battista 1998] Giuseppe Di Battista, Peter Eades, Roberto Tamassia and Ioannis G. Tollis. Graph drawing: Algorithms for the visualization of graphs. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st édition, 1998. 52, 59

[Battke 2010] Florian Battke, Stephan Symons and Kay Nieselt. *Mayday - integrative analytics for expression data.* BMC Bioinformatics, vol. 11, pages 121 – 130, 2010. 48

[Batty 2013] Michael Batty. *Visually-Driven Urban Simulation: exploring fast and slow change in residential location.* Environment and Planning, vol. 45, no. 3, pages 532–552, 2013. 48

[Baxter 1998] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna and Lorraine Bier. *Clone Detection Using Abstract Syntax Trees.* In Proceedings of the International Conference on Software Maintenance, ICSM '98, pages 368–, Washington, DC, USA, 1998. IEEE Computer Society. 44

[Baysal 2007] Olga Baysal and Andrew J. Malton. *Correlating Social Interactions to Release History During Software Evolution.* In Proceedings of the Fourth International Workshop on Mining Software Repositories, MSR '07, pages 7–, Washington, DC, USA, 2007. IEEE Computer Society. 43, 44

[Beck 2010] Fabian Beck and Stephan Diehl. *Visual comparison of software architectures.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 183–192, New York, NY, USA, 2010. ACM. 306

[Beck 2011] Fabian Beck, Radoslav Petkov and Stephan Diehl. *Visually exploring multi-dimensional code couplings.* In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–8, 2011. 303

[Beck 2013] Fabian Beck and Stephan Diehl. *Visual comparison of software architectures.* Information Visualization, vol. 12, no. 2, pages 178–199, 04 2013. XIII, XIV, 123, 124, 134, 135, 306

[Ben-Ari 2011] Mordechai Ben-Ari, Roman Bednarik, Ronit Ben-Bassat Levy, Gil Ebel, Andrés Moreno, Niko Myller and Erkki Sutinen. *A decade of research and development on program animation: The Jeliot experience.* Journal of Visual Languages & Computing, vol. 22, no. 5, pages 375 – 384, 2011. 303

[Benestad 2009] Hans Christian Benestad, Bente Anda and Erik Arisholm. *Understanding software maintenance and evolution by analyzing individual changes: a literature review.* Journal of Software Maintenance and Evolution: Research and Practice, vol. 21, no. 6, pages 349–378, 2009. 43

[Bennedsen 2010] Jens Bennedsen and Carsten Schulte. *BlueJ Visual Debugger for Learning the Execution of Object-Oriented Programs?*

Transactions on Computing Education, vol. 10, no. 2, pages 8:1–8:22, June 2010. 305

[Bennett 2000] Keith H. Bennett and Václav T. Rajlich. *Software Maintenance and Evolution: A Roadmap.* In Proceedings of the Conference on The Future of Software Engineering, ICSE '00, pages 73–87, New York, NY, USA, 2000. ACM. 26, 29

[Benomar 2013] Omar Benomar, Houari Sahraoui and Pierre Poulin. *Visualizing software dynamicities with heat maps.* In First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pages 1–10, 2013. 306

[Bentrad 2013] Sassi Bentrad and Djamel Meslati. *Visualizing and Analyzing the Structure of AspectJ Software under the Eclipse Platform.* International Journal of Software Engineering and Its Applications, vol. 7, no. 3, pages 353–376, May 2013. 116, 143, 306

[Bernardin 2008] Tony Bernardin, Brian C. Budge and Bernd Hamann. *Stacked-widget visualization of scheduling-based algorithms.* In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 165–174, New York, NY, USA, 2008. ACM. 305

[Béron 2008] Mario Béron, Daniela da Cruz, Maria João Varanda Pereira, Pedro Rangel Henriques and Roberto Uzal. *Evaluation Criteria of Software Visualization Systems used for Program Comprehension.* In Universidade de Évora, editeur, Interaccão'08 – 3Âł Conferência Interaccão Pessoa-Máquina, Oct 2008. 303

[Bertini 2011] Enrico Bertini and Giuseppe Santucci. *Improving visual analytics environments through a methodological framework for automatic clutter reduction.* Journal of Visual Languages and Computing, vol. 22, no. 3, pages 194 – 212, 2011. 49

[Beyer 2006] Dirk Beyer and Ahmed E. Hassan. *Evolution Storyboards: Visualization of Software Structure Dynamics.* In 14th IEEE International Conference on Program Comprehension, 2006. ICPC 2006., pages 248–251, 2006. XIV, 135, 137, 144

[Biersack 2012] Ernst Biersack, Quentin Jacquemart, Fabian Fischer, Johannes Fuchs, Olivier Thonnard, Georgios Theodoridis, Dimitrios Tzovaras and Pierre-Antoine Vervier. *Visual analytics for BGP monitoring and prefix hijacking identification.* IEEE Network, vol. 26, no. 6, pages 33–39, 2012. 48

[Boccuzzo 2007] Sandro Boccuzzo and Harald Gall. *CocoViz: Towards Cognitive Software Visualizations.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 72–79, 2007. 303

[Boehm 1988] Barry W. Boehm. *A spiral model of software development and enhancement.* IEEE Computer, vol. 21, no. 5, pages 61–72, May 1988. 21

[Boehm 1999a] Barry Boehm, Alexander Egyed, Dan Port, Archita Shah, Julie Kwan and Ray Madachy. *A Stakeholder Win to Win Approach to Software Engineering Education.* Annals of Software Engineering, vol. 6, no. 1-4, pages 295–321, April 1999. 21

[Boehm 1999b] Barry W. Boehm and Kevin J. Sullivan. *Software economics: status and prospects.* Information & Software Technology, vol. 41, no. 14, pages 937–946, 1999. 4, 308, 309

[Boehm 2000] Barry W. Boehm and Kevin J. Sullivan. *Software Economics: A Roadmap.* In Proceedings of the Conference on The Future of Software Engineering, ICSE '00, pages 319–343, New York, NY, USA, 2000. ACM. 4, 309

[Bohner 2002] Shawn A. Bohner. *Extending Software Change Impact Analysis into COTS Components.* In Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02), SEW '02, pages 175–, Washington, DC, USA, 2002. IEEE Computer Society. 44

[Bohnet 2007] Johannes Bohnet and Jürgen Döllner. *Facilitating Exploration of Unfamiliar Source Code by Providing 21/2D Visualizations of Dynamic Call Graphs.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 63–66, 2007. 306

[Bohnet 2009a] Johannes Bohnet, Martin Koeleman and Juergen Doellner. *Visualizing massively pruned execution traces to facilitate trace exploration.* In 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009., pages 57–64, 2009. 305

[Bohnet 2009b] Johannes Bohnet, Stefan Voigt and Jürgen Döllner. *Projecting code changes onto execution traces to support localization of recently introduced bugs.* In Proceedings of the 2009 ACM Symposium

on Applied Computing, SAC '09, pages 438–442, New York, NY, USA, 2009. ACM. 306

[Bohnet 2011] Johannes Bohnet and Jürgen Döllner. *Monitoring code quality and development activity by software maps.* In Proceedings of the 2nd Workshop on Managing Technical Debt, MTD '11, pages 9–16, New York, NY, USA, 2011. ACM. 305

[Booch 2005] Grady Booch, James Rumbaugh and Ivar Jacobson. Unified modeling language user guide, the (2nd edition) (addison-wesley object technology series). Addison-Wesley Professional, 2005. 128

[Boukhelifa 2003] Nadia Boukhelifa and Peter J. Rodgers. *A model and software system for coordinated and multiple views in exploratory visualization.* Information Visualization, vol. 2, no. 4, pages 258–269, December 2003. 47

[Bresciani 2009] Sabrina Bresciani and Martin J. Eppler. Identität und vielfalt der kommunikations-wissenschaft, chapitre The Risks of Visualization: a Classification of Disadvantages Associated with Graphic Representations of Information. UVK Verlagsgesellschaft mbH, 2009. 189

[Briand 1999] Lionel C. Briand, Jürgen Wüst and Hakim Lounis. *Using Coupling Measurement for Impact Analysis in Object-Oriented Systems.* Proceedings IEEE International Conference on Software Maintenaince (ICSM '99), 1999. 89

[Broeksema 2011] Bertjan Broeksema and Alexandru Telea. *Visual support for porting large code bases.* In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–8, 2011. 306

[Buja 1996] Andreas Buja, Dianne Cook and Deborah F. Swayne. *Interactive High-Dimensional Data Visualization.* Journal of Computational and Graphical Statistics, vol. 5, no. 1, pages 78–99, 1996. 63

[Burch 2011] Michael Burch, Corinna Vehlow, Fabian Beck, Stephan Diehl and Daniel Weiskopf. *Parallel Edge Splatting for Scalable Dynamic Graph Visualization.* IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 12, pages 2344–2353, 2011. 303

[Buse 2012] Raymond P. L. Buse and Thomas Zimmermann. *Information Needs for Software Development Analytics.* In Proceedings of the 34th

International Conference on Software Engineering, ICSE '12, pages 987–996, Piscataway, NJ, USA, 2012. IEEE Press. 38, 40

[Buss 1994] E. Buss, R. De Mori, W. M. Gentleman, J. Henshaw, H. Johnson, K. Kontogiannis, E. Merlo, H. A.Müller, J. Mylopoulos, S. Paul, A. Prakash, M. Stanley, S. R. Tilley, J. Troster and K. Wong. *Investigating Reverse Engineering Technologies for the CAS Program Understanding Project.* IBM Systems Journal, vol. 33, no. 3, pages 477–500, July 1994. 44

[Buxmann 2013] Peter Buxmann, Heiner Diefenbach and Thomas Hess. The software industry: Economic principles, strategies, perspectives, chapitre Economic Principles in the Software Industry, pages 19–53. Springer Publishing Company, Incorporated, 2013. 4, 309

[Cain 2012] Aurora A. Cain, Robert Kosara and Cynthia J. Gibas. *GenoSets: Visual Analytic Methods for Comparative Genomics.* PLoS ONE, vol. 7, no. 10, pages 1 – 9, 2012. 48

[Card 1999a] Stuart K. Card, Jock Mackinlay and Ben Shneiderman. Readings in information visualization: Using vision to think. 1999. 51

[Card 1999b] Stuart K. Card, Jock D. Mackinlay and Ben Shneiderman, editeurs. Readings in information visualization: using vision to think. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. 47, 160, 198

[Card 2006] Stuart K. Card, Bongwon Suh, Bryan A. Pendleton and Jeffrey Heer. *TimeTree: exploring time changing hierarchies.* In IEEE Symposium on Visual Analytics Science and Technology 2006 (VAST 2006), Baltimore; MD; USA. Piscataway NJ, 2006. IEEE Computer Society. 9, 55, 311

[Carmel 1999] Erran Carmel. Global software teams: Collaborating across borders and time zones. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999. 24, 147

[Carmel 2001] Erran Carmel and Ritu Agarwal. *Tactical Approaches for Alleviating Distance in Global Software Development.* IEEE Software, vol. 18, no. 2, pages 22–29, March 2001. 149

[Caserta 2011a] Pierre Caserta and Olivier Zendra. *Visualization of the Static Aspects of Software: A Survey.* IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 7, pages 913–933, 2011. 303

[Caserta 2011b] Pierre Caserta, Olivier Zendra and Damien Bodénès. *3D Hierarchical Edge bundles to visualize relations in a software city metaphor.* In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–8, 2011. 303

[Cassell 2011] Keith Cassell, Craig Anslow, Lindsay Groves and Peter Andreae. *Visualizing the Refactoring of Classes via Clustering.* In Mark Reynolds, editeur, Australasian Computer Science Conference (ACSC 2011), volume 113 of *CRPIT*, pages 63–72, Perth, Australia, 2011. ACS. 306

[Castellanos-Garzón 2013] José A. Castellanos-Garzón, Carlos Armando García, Paulo Novais and Fernando Díaz. *A visual analytics framework for cluster analysis of DNA microarray data.* Expert Syst. Appl., vol. 40, no. 2, pages 758–774, 2013. 48

[Cataldo 2007] Marcelo Cataldo, Matthew Bass, James D. Herbsleb and Len Bass. *On Coordination Mechanisms in Global Software Development.* In Second IEEE International Conference on Global Software Engineering, 2007. ICGSE 2007., pages 71–80, Aug 2007. 152

[Cegielski 2006] Casey G. Cegielski and Dianne J. Hall. *What Makes a Good Programmer?* Communications of the ACM, vol. 49, no. 10, pages 73–75, October 2006. 24

[Chan 2010] Brian Chan, Ying Zou, Ahmed E. Hassan and Anand Sinha. *Visualizing the Results of Field Testing.* In IEEE 18th International Conference on Program Comprehension (ICPC), 2010, pages 114–123, 2010. 305

[Charette 2005] Robert N. Charette. *Why software fails [software failure].* IEEE Spectrum, vol. 42, no. 9, pages 42–49, Sept 2005. 3, 4, 308, 309

[Chawla 2003] Sanjay Chawla, Bavani Arunasalam and Joseph Davis. *Mining Open Source Software (OSS) Data Using Association Rules Network.* In Kyu-Young Whang, Jongwoo Jeon, Kyuseok Shim and Jaideep Srivastava, editeurs, Advances in Knowledge Discovery and Data Mining, volume 2637 of *Lecture Notes in Computer Science*, pages 461–466. Springer Berlin Heidelberg, 2003. 43

[Chen 2002] Chaomei Chen, Timothy Cribbin, Jasna Kuljis and Robert Macredie. *Footprints of information foragers: behaviour semantics of*

visual exploration. International Journal of Human-Computer Studies, vol. 57, no. 2, pages 139–163, August 2002. 62

[Chen 2006] Chaomei Chen. *CiteSpace II: Detecting and visualizing emerging trends and transient patterns in scientific literature*. Journal of the American Society for Information Science and Technology, vol. 1, no. 57, pages 359–377, 2006. 56

[Chen 2010] Chaomei Chen, Jian Zhang and Michael Vogeley. *Making sense of the evolution of a scientific domain: a visual analytic study of the Sloan Digital Sky Survey research*. Scientometrics, vol. 83, pages 669–688, 2010. 10.1007/s11192-009-0123-x. 48, 49

[Chen 2013] Chaomei Chen. *Hindsight, insight, and foresight: a multi-level structural variation approach to the study of a scientific field*. Technology Analysis and Strategic Management, vol. 25, no. 6, pages 619–640, 2013. 49

[Chi 2000] Ed H. Chi. *A Taxonomy of Visualization Techniques Using the Data State Reference Model*. In Proceedings of the IEEE Symposium on Information Vizualization 2000, INFOVIS '00, pages 69–, Washington, DC, USA, 2000. IEEE Computer Society. 47, 198

[Chiara 2011] Davide De Chiara, Vincenzo Del Fatto, Robert Laurini, Monica Sebillo and Giuliana Vitiello. *A chorem-based approach for visually analyzing spatial data*. Journal of Visual Languages and Computing, vol. 22, no. 3, pages 173 – 193, 2011. 49

[Chidamber 1994] Shyam R. Chidamber and Chris F. Kemerer. *A Metrics Suite for Object Oriented Design*. IEEE Transactions in Software Engineering, vol. 20, no. 6, pages 476–493, June 1994. 41

[Chillarege 1992] Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray and Man-Yuen Wong. *Orthogonal Defect Classification-A Concept for In-Process Measurements*. IEEE Transactions in Software Engineering, vol. 18, no. 11, pages 943–956, November 1992. 44

[Chinchor 2010] Nancy A. Chinchor, James J. Thomas, Pak Chung-Wong, Michael G. Christel and William Ribarsky. *Multimedia Analysis + Visual Analytics = Multimedia Analytics*. Computer Graphics and Applications, IEEE, vol. 30, no. 5, pages 52 –60, sept.-oct. 2010. 48

[Choudhury 2011] A.N.M. Imroz Choudhury and Paul Rosen. *Abstract visualization of runtime memory behavior.* In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–8, 2011. 305

[Chow 2008] Tsun Chow and Dac-Buu Cao. *A Survey Study of Critical Success Factors in Agile Software Projects.* Journal of Systems and Software, vol. 81, no. 6, pages 961–971, June 2008. 4, 308

[Chui 2011] Kenneth K. H. Chui, Julia B. Wenger, Steven A. Cohen and Elena N. Naumova. *Visual Analytics for Epidemiologists: Understanding the Interactions Between Age, Time, and Disease with Multi- Panel Graphs.* PLoS ONE, vol. 6, no. 2, pages 1 – 8, 2011. 48

[Chung-Wong 2009] Pak Chung-Wong, L. Ruby Leung, Michael J. Scott Ning Lu, Patrick Mackey, Harlan Foote, James Correia Jr., Z. Todd Taylor, Stephen D. Unwin Jianhua Xu and Antonio Sanfilippo. *Designing a Collaborative Visual Analytics Tool for Social and Technological Change Prediction.* Computer Graphics and Applications, IEEE, vol. 29, no. 5, pages 58 –68, sept.-oct. 2009. 48

[Chung-Wong 2012a] Pak Chung-Wong, Han-Wei Shen, Christopher R. Johnson, Chaomei Chen and Robert B. Ross. *The Top 10 Challenges in Extreme-Scale Visual Analytics.* IEEE Computer Graphics and Applications, vol. 32, no. 4, pages 63–67, 2012. 50, 62

[Chung-Wong 2012b] Pak Chung-Wong, Han-Wei Shen and Valerio Pascucci. *Exploratory Visualization Involving Incremental, Approximate Database Queries and Uncertainty.* IEEE Computer Graphics and Applications, vol. 32, no. 4, pages 55–62, 2012. 49

[Cisar 2011] Sanja Maravic Cisar, Dragica Radosav, Robert Pinter and Petar Cisar. *Effectiveness of Program Visualization in Learning Java: a Case Study with Jeliot 3.* International Journal of Computers Communications & Control, vol. 6, no. 4, pages 669–682, 2011. 305

[Cockburn 2000] Andy Cockburn and Bruce McKenzie. *An evaluation of cone trees.* In Proceedings of the 2000 British Computer Society Conference on Human-Computer Interaction, 2000. 57

[Cockburn 2009] Andy Cockburn, Amy Karlson and Benjamin B. Bederson. *A review of overview+detail, zooming, and focus+context interfaces.* Journal ACM Computing Surveys, vol. 41, no. 1, pages 2:1–2:31, January 2009. 53, 63

[Collard 2004] Michael L. Collard. *Meta-differencing: An Infrastructure for Source Code Difference Analysis*. PhD thesis, Kent, OH, USA, 2004. AAI3147487. 44

[Collberg 2003] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts and Kevin Wampler. *A system for graph-based visualization of the evolution of software*. In SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization, pages 77–ff, New York, NY, USA, 2003. ACM. XIV, 140, 141, 144

[Collins-Sussman 2004] B. Collins-Sussman, B. Fitzpatrick and M. Pilato. Version control with subversion. Sebastopol, CA USA: O'Reilly Media, Inc., 2004. ISBN: 0-596-00448-6. XII, 34

[Colomo-Palacios 2012] Ricardo Colomo-Palacios, Pedro Soto-Acosta, Francisco J. García-Peñalvo and Ángel García-Crespo. *A Study of the Impact of Global Software Development in Packaged Software Release Planning*. Journal of Universal Computer Science, vol. 18, no. 19, pages 2646–2668, nov 2012. 11, 24, 38, 313

[Colomo-Palacios 2013] Ricardo Colomo-Palacios, Cristina Casado-Lumbreras, Pedro Soto-Acosta, Francisco J. García-Peñalvo and Edmundo Tovar-Caro. *Competence gaps in software personnel: A multi-organizational study*. Computers in Human Behavior, vol. 29, no. 2, pages 456–461, 2013. 4, 150, 309

[Colomo-Palacios 2014] Ricardo Colomo-Palacios, Cristina Casado-Lumbreras, Pedro Soto-Acosta, Francisco José García-Peñalvo and Edmundo Tovar. *Project Managers in Global Software Development Teams: A Study of the Effects on Productivity and Performance*. Software Quality Control, vol. 22, no. 1, pages 3–19, March 2014. 4, 24, 150, 309

[Comfort 2007] Louise K. Comfort. *Crisis Management in Hindsight: Cognition, Communication, Coordination, and Control*. Public Administration Review, vol. 67, pages 189–197, 2007. 149, 151, 153, 154

[Conchúir 2009] Eoin Ó. Conchúir, Par J. Agerfalk, Helena H. Olsson and Brian Fitzgerald. *Global Software Development: Where are the Benefits?* Communications of the ACM, vol. 52, no. 8, pages 127 – 131, 2009. 24, 147

[Cooke 2013] Nancy J. Cooke, Jamie C. Gorman, Christopher W. Myers and Jasmine L. Duran. *Interactive Team Cognition.* Cognitive Science, vol. 37, no. 2, pages 255–285, 2013. 156

[Cornelissen 2007] Bas Cornelissen, Danny Holten, Andy Zaidman, Leon Moonen, Jarke J. van Wijk and Arie van Deursen. *Understanding Execution Traces Using Massive Sequence and Circular Bundle Views.* In Proceedings of the 15th IEEE International Conference on Program Comprehension, ICPC '07, pages 49–58, Washington, DC, USA, 2007. IEEE Computer Society. 219, 329

[Cornelissen 2009] Bas Cornelissen, Andy Zaidman, Arie van Deursen and Bart van Rompaey. *Trace visualization for program comprehension: A controlled experiment.* In IEEE 17th International Conference on Program Comprehension, 2009. ICPC '09., pages 100–109, 2009. 305

[Cosma 2007] Dan C. Cosma and Radu Marinescu. *Distributable Features View: Visualizing the Structural Characteristics of Distributed Software Systems.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 55–62, 2007. 305

[Cottam 2008] Joseph A. Cottam, Joshua Hursey and Andrew Lumsdaine. *Representing unit test data for large scale software development.* In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 57–66, New York, NY, USA, 2008. ACM. 305

[Crouser 2012] R. Jordan Crouser, Daniel E. Kee, Dong Hyun Jeong and Remco Chang. *Two Visualization Tools for Analyzing Agent-Based Simulations in Political Science.* IEEE Computer Graphics and Applications, vol. 32, no. 1, pages 67–77, 2012. 48

[Dainton 2015] Marianne Dainton and Elaine D. Zelley. Applying Communication Theory for Professional Life : A Practical Introduction. Sage Publications, Inc, 2015. 150

[D'Ambros 2006a] Marco D'Ambros and Michele Lanza. *Applying the Evolution Radar to PostgreSQL.* In Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR '06, pages 177–178, New York, NY, USA, 2006. ACM. XIV, 140, 142, 144

[D'Ambros 2006b] Marco D'Ambros and Michele Lanza. *Software Bugs and Evolution: A Visual Approach to Uncover Their Relationship.* In Proceedings of the Conference on Software Maintenance and

Reengineering, CSMR '06, pages 229–238, Washington, DC, USA, 2006. IEEE Computer Society. 181

[D'Ambros 2006c] Marco D'Ambros, Michele Lanza and Mircea Lungu. *The Evolution Radar: Visualizing Integrated Logical Coupling Information*. In Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR '06, pages 26–32, New York, NY, USA, 2006. ACM. 140, 144

[D'Ambros 2007a] Marco D'Ambros and Michele Lanza. *BugCrawler: Visualizing Evolving Software Systems*. In Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on, pages 333–334, March 2007. 181

[D'Ambros 2007b] Marco D'Ambros, Michele Lanza and Martin Pinzger. *"A Bug's Life" Visualizing a Bug Database*. In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 113–120, 2007. 306

[D'Ambros 2008] Marco D'Ambros, Harald C. Gall, Michele Lanza and Martin Pinzger. *Analyzing software repositories to understand software evolution*. In Software Evolution, 2008. 8, 30, 196, 313

[D'Ambros 2009a] Marco D'Ambros and Michele Lanza. *Visual software evolution reconstruction*. Journal of Software Maintenance and Evolution: Research and Practice, vol. 21, no. 3, pages 217–232, 2009. 30, 303

[D'Ambros 2009b] Marco D'Ambros, Michele Lanza and Mircea Lungu. *Visualizing Co-Change Information with the Evolution Radar*. IEEE Transactions on Software Engineering, vol. 35, no. 5, pages 720–735, 2009. 41, 90, 140, 144, 303

[D'Ambros 2011] Marco D'Ambros, Michele Lanza, Mircea Lungu and Romain Robbes. *On porting software visualization tools to the web*. International Journal on Software Tools for Technology Transfer, vol. 13, no. 2, pages 181–200, 2011. 303

[Danese 2010] María Danese, Urska Demsar, Nicola Masini and Martin Charlton. *Investigating material decay of historic building using visual analytics with multi-temporal infrared thermographic data*. Archaeometry, vol. 52, no. 3, pages 482–501, 2010. 48

[Dang 2013] Tuan Nhon Dang, Anushka Anand and Leland Wilkinson. *TimeSeer: Scagnostics for High-Dimensional Time Series.* IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 3, pages 470–483, 2013. 49

[Davenport 2006] Thomas H. Davenport. *Competing on Analytics.* Harvard Business Review, vol. 84, no. 1, pages 98–107, January 2006. 6

[de Bono 2012] Bernard de Bono, Pierre Grenon and Stephen John Sammut. *ApiNATOMY: A novel toolkit for visualizing multiscale anatomy schematics with phenotype-related information.* Human Mutation, vol. 33, no. 5, pages 837–848, 2012. 48

[de Oliveira Barros 2004] Márcio de Oliveira Barros, Cláudia Maria Lima Werner and Guilherme Horta Travassos. *Supporting risks in software project management.* Journal of Systems and Software, vol. 70, no. 1-âĂŞ2, pages 21 – 35, 2004. 10, 312

[de Souza 2007] Cleidson R. B. de Souza, Stephen Quirk, Erik Trainer and David F. Redmiles. *Supporting collaborative software development through the visualization of socio-technical dependencies.* In Proceedings of the 2007 International ACM Conference on Supporting Group Work, GROUP '07, pages 147–156, New York, NY, USA, 2007. ACM. 306

[Deelen 2007] Pieter Deelen, Frank van Ham, Cornelis Huizing and Huub van de Wetering. *Visualization of Dynamic Program Aspects.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 39–46, 2007. 305

[Deng 2011] Fang Deng, Nicholas DiGiuseppe and James A. Jones. *Constellation visualization: Augmenting program dependence with dynamic information.* In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–8, 2011. 306

[Diehl 2007] Stephan Diehl. Software visualization visualizing the structure, behaviour, and evolution of software. Springer Berlin Heidelberg New York, 2007. 11, 196, 310, 313

[Dinkla 2011] Kau Dinkla, Michel A. Westenberg, Hau Timmerman, Sacha A. F. T. van Hijum and Jack van Wijk. *Comparison of Multiple Weighted*

Hierarchies: Visual Analytics for Microbe Community Profiling. Computer Graphics Forum, vol. 30, no. 3, pages 1141–1150, 2011. 49

[Dix 1998] Alan Dix and Geoffrey Ellis. *Starting simple: adding value to static visualisation through simple interaction.* In Proceedings of the working conference on Advanced visual interfaces, AVI '98, pages 124–134, New York, NY, USA, 1998. ACM. 63

[Dix 2010] Alan Dix, Margit Pohl and Geoffrey Ellis. Mastering the information age solving problems with visual analytics, chapitre Perception and Cognitive Aspects, pages 109 – 130. Eurographics Association, 2010. 50, 197, 311

[Dransch 2010] Doris Dransch, Patrick Kothur, Sven Schulte, Volker Klemann and Henryk Dobslaw. *AssessiDang the quality of geoscientific simulation models with visual analytics methods-a design study.* International Journal of Geographical Information Science, vol. 24, no. 10, pages 1459–1479, October 2010. 49

[Draper 2009] Geoffrey M. Draper, Yarden Livnat and Richard F. Riesenfeld. *A Survey of Radial Methods for Information Visualization.* IEEE Transactions on Visualization and Computer Graphics, vol. 15, no. 5, pages 759–776, 2009. 59

[Drigas 2011] Athanasios Drigas, Lefteris Koukianakis and Yannis Papagerasimou. *Towards an ICT-based psychology: E-psychology.* Computers in Human Behavior, vol. 27, no. 4, pages 1416 – 1423, July 2011. 202

[Ducasse 2005] Stéphane Ducasse and Michele Lanza. *The Class Blueprint: Visually Supporting the Understanding of Classes.* IEEE Transactions in Software Engineering, vol. 31, no. 1, pages 75–90, January 2005. 127, 144

[Ducheneaut 2005] Nicolas Ducheneaut. *Socialization in an Open Source Software Community: A Socio-Technical Analysis.* Journal Computer Supported Cooperative Work, vol. 14, no. 4, pages 323–368, August 2005. 43, 44

[Ebert 2001a] Christof Ebert and Philip De Neve. *Surviving Global Software Development.* IEEE Software, vol. 18, no. 2, pages 62–69, March 2001. 24

[Ebert 2001b] Christof Ebert, Casimiro Hernandez Parro, Roland Suttels and Harald Kolarczyk. *Improving Validation Activities in a Global Software Development.* In Proceedings of the 23rd International Conference on Software Engineering, ICSE '01, pages 545–554, Washington, DC, USA, 2001. IEEE Computer Society. 24

[Eichelberger 2008] Holger Eichelberger. *Automatic layout of UML use case diagrams.* In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 105–114, New York, NY, USA, 2008. ACM. 304

[Eick 2002] Stephen G. Eick, Todd L. Graves, Alan F. Karr, Audris Mockus and Paul Schuster. *Visualizing Software Changes.* IEEE Transactions in Software Engineering, vol. 28, no. 4, pages 396–412, 2002. 161

[El-Nasr 2013] Magy Seif El-Nasr, Heather Desurvire, Bardia Aghabeigi and Anders Drachen. *Game Analytics for Game User Research, Part 1: A Workshop Review and Case Study.* IEEE Computer Graphics and Applications, vol. 33, no. 2, pages 6–11, 2013. 49

[El-Sappagh 2011] Shaker H. Ali El-Sappagh, Abdeltawab M. Ahmed Hendawi and Ali Hamed El Bastawissy. *A proposed model for data warehouse {ETL} processes.* Journal of King Saud University - Computer and Information Sciences, vol. 23, no. 2, pages 91 – 104, 2011. 198

[Elmqvist 2012] Niklas Elmqvist and David S. Ebert. *Leveraging Multidisciplinarity in a Visual Analytics Graduate Course.* IEEE Computer Graphics and Applications, vol. 32, no. 3, pages 84–87, 2012. 49

[Endsley 1995] Mica R. Endsley. *Toward a Theory of Situation Awareness in Dynamic Systems.* Human Factors: The Journal of the Human Factors and Ergonomics Society, vol. 37, no. 1, pages 32–64, 1995. 157, 160

[Erdemir 2011] Ural Erdemir, Umut Tekin and Feza Buzluca. *E-Quality: A graph based object oriented software quality visualization tool.* In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–8, 2011. 305

[Erra 2012a] Ugo Erra and Giuseppe Scanniello. *Towards the visualization of software systems as 3D forests: the CodeTrees environment.* In Proceedings of the 27th Annual ACM Symposium on Applied

Computing, SAC '12, pages 981–988, New York, NY, USA, 2012. ACM. 303

[Erra 2012b] Ugo Erra, Giuseppe Scanniello and Nicola Capece. *Visualizing the Evolution of Software Systems Using the Forest Metaphor*. In 16th International Conference on Information Visualisation (IV), 2012, pages 87–92, 2012. 303

[Estublier 1999] Jacky Estublier. *Distributed Objects for Concurrent Engineering*. In Proceedings of the 9th International Symposium on System Configuration Management, SCM-9, pages 172–185, London, UK, UK, 1999. Springer-Verlag. 5, 23, 309

[Estublier 2000] Jacky Estublier. *Software configuration management: a roadmap*. In Proceedings of the Conference on The Future of Software Engineering, ICSE '00, pages 279–289, New York, NY, USA, 2000. ACM. 8, 30, 33, 34

[Fabriek 2008] Matthias Fabriek, Mischa van den Brand, Sjaak Brinkkemper, Frank Harmsen and Remko Helms. *Reasons for Success and Failure in Offshore Software Development Projects*. In Proceedings of the 16th European Conference on Information Systems, ECIS 2008, Galway, Ireland, 2008, pages 446–457, 2008. 5, 309

[Fallick 2006] Bruce Fallick, Charles A. Fleischman and James B. Rebitzer. *Job-Hopping in Silicon Valley: Some Evidence Concerning the Microfoundations of a High-Technology Cluster*. Review of Economics and Statistics, vol. 88, no. 3, pages 472 – 481, October 2006. 7, 27

[Feigenspan 2013] Janet Feigenspan, Christian Kästner, Sven Apel, Jörg Liebig, Michael Schulze, Raimund Dachselt, Maria Papendieck, Thomas Leich and Gunter Saake. *Do background colors improve program comprehension in the ifdef hell?* Empirical Software Engineering, vol. 18, no. 4, pages 699–745, 2013. 303

[Femmer 2011] Henning Femmer, Nora Broy, Marin Zec, Asa MacWilliams and Roland Eckl. *Dynamic Software Visualization with BusyBorg - A Proof of Concept*. In IEEE 35th Annual Computer Software and Applications Conference (COMPSAC), 2011, pages 492–497, 2011. 305

[Fenton 2000] Norman E. Fenton and Martin Neil. *Software Metrics: Roadmap*. In Proceedings of the Conference on The Future of Software Engineering, ICSE '00, pages 357–370, New York, NY, USA, 2000. ACM. 44

[Fernandez-Ramil 2008] Juan Fernandez-Ramil, Angela Lozano, Michel Wermelinger and Andrea Capiluppi. Software evolution, chapitre Empirical Studies of Open Source Evolution, pages 263 – 288. Sp, 2008. 8

[Fiore 2004a] Stephen M. Fiore and Eduardo Salas. Team cognition: Understanding the factors that drive process and performance, chapitre Advances in measuring team cognition., pages 83–106. American Psychological Association, Washington, DC, US, 2004. 148, 155, 156, 159

[Fiore 2004b] Stephen M. Fiore and Eduardo Salas. Team cognition: Understanding the factors that drive process and performance, chapitre Why we need team cognition., pages 235–248. American Psychological Association, Washington, DC, US, 2004. 156, 157

[Fischer 1978] Kurt F. Fischer. *Software Quality Assurance Tools: Recent Experience and Future Requirements*. SIGSOFT Software Engineering Notes, vol. 3, no. 5, pages 116–121, January 1978. 154

[Fischer 2005] Michael Fischer, Johann Oberleitner, Harald Gall and Thomas Gschwind. *System evolution tracking through execution trace analysis*. In Proceedings. 13th International Workshop on Program Comprehension, 2005. IWPC 2005., pages 237–246, May 2005. 43

[Fluri 2007] Beat Fluri, Michael Wuersch, Martin Pinzger and Harald Gall. *Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction*. IEEE Transactions in Software Engineering, vol. 33, no. 11, pages 725–743, November 2007. 44

[Forsberg 2005] Kevin Forsberg, Hal Mooz and Howard Cotterman. Visualizing project management: Models and frameworks for mastering complex systems. John Wiley & Sons, 3rd édition, September 2005. 10, 312

[Frisch 2010] Mathias Frisch and Raimund Dachselt. *Off-screen visualization techniques for class diagrams*. In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 163–172, New York, NY, USA, 2010. ACM. 305

[Frisch 2013] Mathias Frisch and Raimund Dachselt. *Visualizing offscreen elements of node-link diagrams*. Information Visualization, vol. 12, no. 2, pages 133–162, 04 2013. 305

[Fry 2008] Ben Fry. Visualizing data - exploring and explaining data with the processing environment. O'Reilly, 2008. 47, 198

[Fua 1999] Ying-Huey Fua, Matthew O. Ward and Elke A. Rundensteiner. *Hierarchical parallel coordinates for exploration of large datasets.* In Proceedings of the conference on Visualization '99: celebrating ten years, VIS '99, pages 43–50, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press. 60

[Furnas 1986] George W. Furnas. *Generalized Fisheye Views.* In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '86, pages 16–23, New York, NY, USA, 1986. ACM. 52, 53

[Gaither 2012] Kelly P. Gaither, Hank Childs, Karl W. Schulz, Cyrus Harrison, William Barth, Diego Donzis and Pui-Kuen Yeung. *Visual Analytics for Finding Critical Structures in Massive Time-Varying Turbulent-Flow Simulations.* IEEE Computer Graphics and Applications, vol. 32, no. 4, pages 34–45, 2012. 49

[Gall 1998] Harald Gall, Karin Hajek and Mehdi Jazayeri. *Detection of logical coupling based on product release history.* In Proceedings., International Conference on Software Maintenance, 1998., pages 190–198, Nov 1998. 44, 90

[Gall 2003] Harald Gall, Mehdi Jazayeri and Jacek Krajewski. *CVS Release History Data for Detecting Logical Couplings.* In IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution, page 13, Washington, DC, USA, 2003. IEEE Computer Society. 41, 44, 141

[Gallagher 2008] Keith Gallagher, Andrew Hatch and Malcolm Munro. *Software Architecture Visualization: An Evaluation Framework and Its Application.* IEEE Transactions on Software Engineering, vol. 34, no. 2, pages 260–270, 2008. 303

[Gansner 2000] Emden R. Gansner and Stephen C. North. *An Open Graph Visualization System and Its Applications to Software Engineering.* Software Practice and Experience, vol. 30, no. 11, pages 1203–1233, September 2000. 140

[García-Peñalvo 2000] Francisco J. García-Peñalvo. Modelo de reutilización soportado por estructuras complejas de reutilización denominadas mecanos, volume 53. Ediciones Universidad de Salamanca, Colección Vitor, 2000. 4, 309

[Garcıa-Peñalvo 2002] Francisco J. Garcıa-Peñalvo, Juan-Antonio Barras, Miguel Ángel Laguna and José Manuel Marqués. *Product Line Variability Support by FORM and Mecano Model Integration.* SIGSOFT Software Engineering Notes, vol. 27, no. 1, pages 35–38, January 2002. 4, 309

[Garcıa-Peñalvo 2011] Francisco J. Garcıa-Peñalvo, Ricardo Colomo Palacios, Pedro Soto-Acosta, Isabel Martínez-Conesa and Enric Serradell-López. *SemSEDoc: Utilización de tecnologías semánticas en el aprovechamiento de los repositorios documentales de los proyectos de desarrollo de software.* Information Research, vol. 16, no. 4, 2011. 8

[García-Peñalvo 2012a] Francisco J. García-Peñalvo, Ricardo Colomo-Palacios, Juan García and Roberto Therón. *Towards an ontology modeling tool. A validation in software engineering scenarios.* Expert Systems with Applications, vol. 39, no. 13, pages 11468 – 11478, 2012. 48

[García-Peñalvo 2012b] Francisco J. García-Peñalvo, María J. Rodríguez Conde, Antonio Miguel Seoane Pardo, Miguel Angel Conde GonzÁlez, Valentina Zangrando and Alicia García Holgado. *GRIAL (GRupo de investigación en InterAcción y eLearning), USAL.* IE Comunicaciones: Revista Iberoamericana de Informática Educativa, no. 15, pages 85–94, 2012. 14, 245

[García-Peñalvo 2012c] Francisco J. García-Peñalvo, Ricardo Colomo Palacios, Juan García and Roberto Therón. *Towards an ontology modeling tool. A validation in software engineering scenarios.* Expert Systems Application, vol. 39, no. 13, pages 11468–11478, 2012. 9, 14, 311

[García-Peñalvo 2014] Francisco J. García-Peñalvo, Patricia Ordóñez de Pablos, Juan García and Roberto Therón. *Using OWL-VisMod through a decision-making process for reusing OWL ontologies.* Behaviour & IT, vol. 33, no. 5, pages 426–442, 2014. 9, 14, 48, 311

[García 2009a] Carlos Armando García, Roberto Therón, Rafael Peláez, José Luis López-Pérez and Gustavo Santos-García. *Visual Evaluation of Clustered Molecules in the Process of New Drugs Design.* In Smart Graphics, 9th International Symposium, SG 2009, Salamanca, Spain, May 28-30, 2009. Proceedings, pages 3–14, 2009. 14, 49

[García 2009b] Juan García, Antonio González-Torres, Diego A. Gómez-Aguilar, Roberto Therón and Francisco J. García-Peñalvo. *A Visual Analytics Tool for Software Project Structure and Relationships among Classes.* In Proceedings of the 10th International Symposium on Smart Graphics, SG '09, pages 203–212, Berlin, Heidelberg, 2009. Springer-Verlag. XIII, 117, 118, 143

[García 2012] Juan García. *Analítica Visual Aplicada a la Ingeniería de Ontologías.* PhD thesis, University of Salamanca, 2012. 9, 14, 48, 311

[Garousi 2010] Vahid Garousi and James Leitch. *IssuePlayer: An extensible framework for visual assessment of issue management in software development projects.* Journal of Visual Languages & Computing, vol. 21, no. 3, pages 121 – 135, 2010. 305

[Gartner 2013] Inc. Gartner. *Gartner Says Worldwide IT Spending on Pace to Reach 3.7 Trillion in 2013.* Website, July 2013. 4, 308

[Gartner 2014] Inc. Gartner. *Gartner Says Worldwide IT Spending on Pace to Reach 3.8 Trillion in 2014.* Website, January 2014. 4, 308

[German 2006] Daniel M. German and Abram Hindle. *Visualizing the Evolution of Software Using Softchange.* International Journal of Software Engineering and Knowledge Engineering, vol. 16, no. 1, pages 5–22, 2006. 41

[Gethers 2012] Malcom Gethers, Bogdan Dit, Huzefa Kagdi and Denys Poshyvanyk. *Integrated Impact Analysis for Managing Software Changes.* In Proceedings of the 34th International Conference on Software Engineering, ICSE '12, pages 430–440, Piscataway, NJ, USA, 2012. IEEE Press. 43, 44

[Gibson 2013] Helen Gibson, Joe Faith and Paul Vickers. *A survey of two-dimensional graph layout techniques for information visualisation.* Information Visualization, vol. 12, no. 3-4, pages 324–357, 07 2013. 52, 60

[Gîrba 2004] Tudor Gîrba, Stéphane Ducasse and Michele Lanza. *Yesterday's Weather: guiding early reverse engineering efforts by summarizing the evolution of changes.* In Proceedings. 20th IEEE International Conference on Software Maintenance, 2004., pages 40–49, Sept 2004. 44

[Gîrba 2005] Tudor Gîrba, Adrian Kuhn, Mauricio Seeberger and Stéphane Ducasse. *How Developers Drive Software Evolution.* In Proceedings of the Eighth International Workshop on Principles of Software Evolution, IWPSE '05, pages 113–122, Washington, DC, USA, 2005. IEEE Computer Society. XIV, 164, 165, 167

[Godart 2001] Claude Godart, Gilles Halin, Jean-Claude Bignon, Christophe Bouthier, O Malcurat and Pascal Molli. *Implicit or explicit coordination of virtual teams in building design.* In Proceedings of the Sixth Conference on Computer Aided Architectural Design Research in Asia, pages 429–434, 2001. 151

[Godfrey 2005] Michael W. Godfrey and Lijie Zou. *Using Origin Analysis to Detect Merging and Splitting of Source Code Entities.* IEEE Transactions in Software Engineering, vol. 31, pages 166–181, February 2005. 43, 44

[Gómez-Aguilar 2009] Diego A. Gómez-Aguilar, Roberto Therón and Francisco J. García-Peñalvo. *Semantic Spiral Timelines Used as Support for e-Learning.* Journal of Universal Computer Science, vol. 15, no. 7, pages 1526–1545, April 2009. 9, 14, 48, 55, 311

[Gómez-Aguilar 2010] Diego A. Gómez-Aguilar, Cristóbal Súarez Guerrero, Roberto Therón and Francisco J. García-Peñalvo. Advances in learning processes, chapitre Visual Analytics to Support E-learning, pages 207–228. InTech, January 2010. 55

[Gómez-Aguilar 2014] Diego A. Gómez-Aguilar, Francisco J. García-Peñalvo and Roberto Therón. *Analítica visual en e-learning.* El Profesional de la Información, vol. 23, no. 3, pages 236–245, 2014. 14, 48

[Gómez-Aguilar 2015a] Diego A. Gómez-Aguilar. *Analítica Visual en eLearning.* PhD thesis, Universidad de Salamanca, April 2015. 14, 48, 60

[Gómez-Aguilar 2015b] Diego A. Gómez-Aguilar, Ángel Hernández-García, Francisco J. García-Peñalvo and Roberto Therón. *Tap into visual analysis of customization of grouping of activities in eLearning.* Computers in Human Behavior, vol. 47, no. 0, pages 60 – 67, 2015. Learning Analytics, Educational Data Mining and data-driven Educational Decision Making. 9, 14, 48, 60, 311

[Gómez 2010] Verónica Uquillas Gómez, Stéphane Ducasse and Theo D'Hondt. *Visually Supporting Source Code Changes Integration: The*

Torch Dashboard. In 17th Working Conference on Reverse Engineering (WCRE), 2010, pages 55–64, 2010. 306

[González-Torres 2009] Antonio González-Torres, Roberto Therón, Alexandru Telea and Francisco J. García-Peñalvo. *Combined visualization of structural and metric information for software evolution analysis*. In Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, IWPSE-Evol '09, pages 25–30, New York, NY, USA, 2009. ACM. XIV, 132, 133, 196

[González-Torres 2011] Antonio González-Torres, Roberto Ther ón, Francisco J. García-Peñalvo, Michel Wermelinger and Yijun Yu. *Maleku: an evolutionary visual software analytics tool for providing insights into software evolution*. In IEEE Computer Society, editeur, IEEE International Conference on Software Maintenance (ICSM), 2011. 207, 322

[González-Torres 2013a] Antonio González-Torres, Francisco J. García-Peñalvo and Roberto Therón. *How Evolutionary Visual Software Analytics Supports Knowledge Discovery*. Journal of Information Science and Engineering, vol. 29, no. 1, pages 17–34, 1 2013. 49, 207, 289, 310, 322, 353

[González-Torres 2013b] Antonio González-Torres, Francisco J. García-Peñalvo and Roberto Therón. *Human-Computer interaction in evolutionary visual software analytics*. Computers in Human Behavior, vol. 29, no. 2, pages 486–495, March 2013. 9, 49, 207, 289, 310, 311, 322, 353

[González-Torres 2014] Antonio González-Torres. Representación visual de sistemas de software: Evolución y colaboración. Master's thesis, Universidad de Salamanca, June 2014. XVI, 215

[Goodall 2010] John R. Goodall, Hassan Radwan and Lenny Halseth. *Visual analysis of code security*. In Proceedings of the 7 International Symposium on Visualization for Cyber Security, VizSec '10, pages 46–51, New York, NY, USA, 2010. ACM. 305

[Gotz 2008] David Gotz and Michelle X. Zhou. *Characterizing users' visual analytic activity for insight provenance*. In IEEE Symposium on Visual Analytics Science and Technology, 2008. VAST '08., pages 123 –130, oct. 2008. 49

[Goulão 2012] Miguel Goulão, Nelson Fonte, Michel Wermelinger and Fernando Brito e Abreu. *Software Evolution Prediction Using Seasonal Time Analysis: A Comparative Study*. In Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering, CSMR '12, pages 213–222, Washington, DC, USA, 2012. IEEE Computer Society. 44

[Gouveia 2013] Carlos Gouveia, José Campos and Rui Abreu. *Using HTML5 visualizations in software fault localization*. In Software Visualization (VISSOFT), 2013 First IEEE Working Conference on, pages 1–10, 2013. 305

[Gra 2002] Graph Drawing 2002. Rings: A technique for visualization of large hierarchies. Springer-Verlag, April 2002. 52, 57, 58

[Green 2011] Pam Green, Peter C.R. Lane, Austen Rainer and Sven-Bodo Scholz. Research and development in intelligent systems xxvii, chapitre Selecting Features in Origin Analysis, pages 379–392. Springer London, January 2011. 43

[Green 2012] Tera M. Green and Brian Fisher. *Impact of personality factors on interface interaction and the development of user profiles: Next steps in the personal equation of interaction*. Information Visualization, vol. 11, no. 3, pages 205–221, 07 2012. 49

[Gribov 2010] Alexander Gribov, Martin Sill, Sonja Luck, Frank Rucker, Konstanze Dohner, Lars Bullinger, Axel Benner and Antony Unwin. *SEURAT: Visual analytics for the integrated analysis of microarray data*. BMC Medical Genomics, vol. 3, no. 1, page 21, 2010. 48

[Group 2013] The Standish Group. *The Chaos Manifesto*, 2013. 4, 308

[Grubb 2003] Penny Grubb and Armstrong A. Takang. Software maintenance: Concepts and practice. World Scientific, 2nd édition, 2003. 21, 27

[Guo 2011] Diansheng Guo and Hai Jin. *iRedistrict: Geovisual analytics for redistricting optimization*. Journal of Visual Languages and Computing, vol. 22, no. 4, pages 279 – 289, 2011. 49

[Hall 2013] Jamie Guevara Eric Stegman Linda Hall. *Gartner IT Key Metrics Data: 2013 IT Enterprise Summary Report*. Website, March 2013. 4, 308

[Hao 2010] Ming C. Hao, Ratnesh K. Sharma, Daniel A. Keim, Umeshwar Dayal, Chandrakant D. Patel and Ravigopal Vennelakanti. *Application of Visual Analytics for Thermal State Management in Large Data Centres*. Computer Graphics Forum, vol. 29, no. 6, pages 1895–1904, 2010. 48

[Hardisty 2010] Frank Hardisty and Alexander Klippel. *Analysing spatio-temporal autocorrelation with LISTA-Viz*. International Journal of Geographical Information Science, vol. 24, no. 10, pages 1515 – 1526, 2010. 49

[Harel 2008] David Harel and Itai Segall. *Visualizing inter-dependencies between scenarios*. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 145–153, New York, NY, USA, 2008. ACM. 305

[Harrison 2011] L. Harrison, Wenwen Dou, Aidong Lu, W. Ribarsky and Xiaoyu Wang. *Guiding security analysis through visualization*. In 2011 IEEE Conference on Visual Analytics Science and Technology (VAST), pages 317 –318, oct. 2011. 9, 311

[Hasenauer 2012] Jan Hasenauer, Julian Heinrich, Malgorzata Doszczak, Peter Scheurich, Daniel Weiskopf and Frank Allgower. *A visual analytics approach for models of heterogeneous cell populations*. EURASIP Journal on Bioinformatics and Systems Biology, vol. 2012, no. 1, page 4, 2012. 48

[Hassan 2005] Ahmed E. Hassan. *Mining software repositories to assist developers and support managers*. PhD thesis, Waterloo, Ont., Canada, Canada, 2005. 8, 42, 196, 204, 313, 320

[Hassan 2006] Ahmed E. Hassan. *Mining Software Repositories to Assist Developers and Support Managers*. In ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance, pages 339–342, Washington, DC, USA, 2006. IEEE Computer Society. 42, 204, 320

[Hassine 2005] Jameleddine Hassine, Juergen Rilling, Jacqueline Hewitt and Rachida Dssouli. *Change Impact Analysis for Requirement Evolution Using Use Case Maps*. In Proceedings of the Eighth International Workshop on Principles of Software Evolution, IWPSE '05, pages 81–90, Washington, DC, USA, 2005. IEEE Computer Society. 43, 44

[Hattori 2012] LilePalma Hattori, Michele Lanza and Romain Robbes. *Refining code ownership with synchronous changes*. Empirical Software Engineering, vol. 17, no. 4-5, pages 467–499, 2012. 164, 306

[He 2007] Jun He, Brian Butler and William King. *Team Cognition: Development and Evolution in Software Project Teams*. Journal of Management Information Systems, vol. 24, no. 2, pages 261–292, October 2007. 148, 149, 155, 156, 158

[Healey 2012] Christopher G. Healey and James T. Enns. *Attention and Visual Memory in Visualization and Computer Graphics*. IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 7, pages 1170–1188, 2012. 49

[Heimerl 2012] Florian Heimerl, Steffen Koch, Harald Bosch and Thomas Ertl. *Visual Classifier Training for Text Document Retrieval*. IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, pages 2839–2848, 2012. 48

[Heitlager 2007] Ilja Heitlager, Tobias Kuipers and Joost Visser. *A Practical Model for Measuring Maintainability*. In 6th International Conference on the Quality of Information and Communications Technology, 2007. QUATIC 2007., pages 30–39, 2007. 41

[Heller 2011] Brandon Heller, Eli Marschner, Evan Rosenfeld and Jeffrey Heer. *Visualizing collaboration and influence in the open-source software community*. In Proceedings of the 8th Working Conference on Mining Software Repositories, MSR '11, pages 223–226, New York, NY, USA, 2011. ACM. 172

[Helminen 2010] Juha Helminen and Lauri Malmi. *Jype-a program visualization and programming exercise tool for Python*. In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 153–162, New York, NY, USA, 2010. ACM. 305

[Hemerly 2013] Jess Hemerly. *Public Policy Considerations for Data-Driven Innovation*. IEEE Computer, vol. 46, no. 6, pages 25–31, 2013. 6

[Herbsleb 2001a] James D. Herbsleb, Audris Mockus, Thomas A. Finholt and Rebecca E. Grinter. *An Empirical Study of Global Software Development: Distance and Speed*. In Proceedings of the 23rd International Conference on Software Engineering, ICSE '01, pages 81–90, Washington, DC, USA, 2001. IEEE Computer Society. 24, 150

[Herbsleb 2001b] James D. Herbsleb and Deependra Moitra. *Global software development.* IEEE Software, vol. 18, no. 2, pages 16–20, Mar 2001. 23, 147

[Herbsleb 2003] James D. Herbsleb and Audris Mockus. *An Empirical Study of Speed and Communication in Globally Distributed Software Development.* IEEE Transactions in Software Engineering, vol. 29, no. 6, pages 481–494, June 2003. 23, 150, 151

[Herman 2000] Ivan Herman, Guy Melançon and M. Scott Marshall. *Graph Visualization and Navigation in Information Visualization: A Survey.* IEEE Transactions on Visualization and Computer Graphics, vol. 6, no. 1, pages 24–43, January 2000. 59

[Hermans 2013] Felienne Hermans, Ben Sedee, Martin Pinzger and Arie van Deursen. *Data clone detection and visualization in spreadsheets.* In Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pages 292–301, Piscataway, NJ, USA, 2013. IEEE Press. 306

[Heuer 1999] Richards J. Heuer. Psychology of intelligence analysis. United States Government Printing, November 1999. 202

[Hindle 2007] Abram Hindle, Zhen Ming Jiang, Walid Koleilat, Michael W. Godfrey and Richard C. Holt. *YARN: Animating Software Evolution.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 129–136, 2007. XIV, 134, 136, 144, 306

[Hitz 1995] Martin Hitz and Behzad Montazeri. *Measuring Coupling and Cohesion In Object-Oriented Systems.* Symposium on Applied Corporate Computing, 1995. 44

[Hochheiser 2004] Harry Hochheiser and Ben Shneiderman. *Dynamic query tools for time series data sets: timebox widgets for interactive exploration.* Information Visualization, vol. 3, no. 1, pages 1–18, 2004. 55

[Hollender 2010] Nina Hollender, Cristian Hofmann, Michael Deneke and Bernhard Schmitz. *Integrating cognitive load theory and concepts of human computer interaction.* Computers in Human Behavior, vol. 26, no. 6, pages 1278–1288, November 2010. 49

[Holmes 2007] Reid Holmes and Robert J. Walker. *Task-specific source code dependency investigation.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 100–107, 2007. 306

[Holten 2007] Danny Holten, Bas Cornelissen and Jarke J. van Wijk. *Trace Visualization Using Hierarchical Edge Bundles and Massive Sequence Views.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 47–54, 2007. XIII, 129, 144, 219, 303, 329

[Hyun 2009] Jeong Dong Hyun, Caroline Ziemkiewicz, Brian Fisher, William Ribarsky and Remco Chang. *iPCA: An Interactive System for PCA-based Visual Analytics.* Computer Graphics Forum, vol. 28, no. 3, pages 767 – 774, 2009. 48

[Inselberg 1985] Alfred Inselberg. *The plane with parallel coordinates.* The Visual Computer, vol. 1, no. 2, pages 69–91, 1985. 60

[Inselberg 2009] Alfred Inselberg. Parallel coordinates: Visual multidimensional geometry and its applications. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2009. 60

[Institute 2011] McKinsey Global Institute. *Big data: The next frontier for innovation, competition, and productivity.* Rapport technique, McKinsey & Company, 2011. 64

[Isenberg 2009] Petra Isenberg and Danyel Fisher. *Collaborative Brushing and Linking for Co-located Visual Analytics of Document Collections.* Computer Graphics Forum, vol. 28, no. 3, pages 1031–1038, 2009. 49

[Isenberg 2012] Petra Isenberg, Danyel Fisher, Sharoda A. Paul, Meredith Ringel Morris, Kori Inkpen and Mary Czerwinski. *Co-Located Collaborative Visual Analytics around a Tabletop Display.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 5, pages 689 –702, may 2012. 49

[Ishio 2012] Takashi Ishio, Shogo Etsuda and Katsuro Inoue. *A lightweight visualization of interprocedural data-flow paths for source code reading.* In IEEE 20th International Conference on Program Comprehension (ICPC), 2012, pages 37–46, 2012. 305

[Islam 2010] Syed S. Islam, Jens Krinke and David Binkley. *Dependence cluster visualization.* In Proceedings of the 5th international

symposium on Software visualization, SOFTVIS '10, pages 93–102, New York, NY, USA, 2010. ACM. 306

[ISO 2014] *Systems and software engineering-Software life cycle processes*, December 2014. 20

[Jackson 2000] Daniel Jackson and Martin Rinard. *Software Analysis: A Roadmap*. In Proceedings of the Conference on The Future of Software Engineering, ICSE '00, pages 133–145, New York, NY, USA, 2000. ACM. 43

[Javed 2013] Waqas Javed and Niklas Elmqvist. *Stack Zooming for Multifocus Interaction in Skewed-Aspect Visual Spaces*. IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 8, pages 1362–1374, 2013. 49

[Jedlitschka 2009] Andreas Jedlitschka. *An empirical model of software managers' information needs for software engineering technology selection: a framework to support experimentally-based software engineering technology selection*. PhD thesis, 2009. 37

[Jensen 2003] Matt Jensen. *Visualizing Complex Semantic Timelines*. NewsBlip Technical Report NBTR2003-001, 2003. 54

[Jermakovics 2011] Andrejs Jermakovics, Alberto Sillitti and Giancarlo Succi. *Mining and visualizing developer networks from version control systems*. In Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '11, pages 24–31, New York, NY, USA, 2011. ACM. XV, 171, 172, 174, 238, 306, 341

[Jiménez 2009] Miguel Jiménez, Mario Piattini and Aurora Vizcaíno. *Challenges and Improvements in Distributed Software Development: A Systematic Review*. Advances in Software Engineering, vol. 2009, 2009. 23, 38

[Johnson 1991] Brian Johnson and Ben Shneiderman. *Tree-Maps: a space-filling approach to the visualization of hierarchical information structures*. In VIS '91: Proceedings of the 2nd conference on Visualization '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press. 9, 47, 52, 56, 116, 311

[Kagdi 2007a] Huzefa Kagdi, Michael L. Collard and Jonathan I. Maletic. *A survey and taxonomy of approaches for mining software repositories*

in the context of software evolution. Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, no. 2, pages 77–131, 2007. 8, 42, 43, 196, 204, 295, 313, 320

[Kagdi 2007b] Huzefa Kagdi and Jonathan I. Maletic. *Onion Graphs for Focus+Context Views of UML Class Diagrams.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 80–87, 2007. 42, 305

[Kamiya 2002] Toshihiro Kamiya, Shinji Kusumoto and Katsuro Inoue. *CCFinder: a multilinguistic token-based code clone detection system for large scale source code.* IEEE Transactions on Software Engineering, vol. 28, no. 7, pages 654–670, Jul 2002. 44

[Kan 2002] Stephen H. Kan. Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd édition, 2002. 154

[Karolak 1999] Dale W. Karolak. Global software development: Managing virtual teams and environments. IEEE Computer Society Press, Los Alamitos, CA, USA, 1st édition, 1999. 24

[Karran 2013] Benjamin Karran, Jonas Trümper and Jürgen Döllner. *SYNCTRACE: Visual thread-interplay analysis.* In First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pages 1–10, 2013. 305

[Kasprzyka 2013] Joseph R. Kasprzyka, Shanthi Nataraj, Patrick M. Reeda and Robert J. Lempert. *Many objective robust decision making for complex environmental systems undergoing change.* Environmental Modelling and Software, vol. 42, no. 0, pages 55 – 71, 2013. 48

[Kazman 1996] Rick Kazman, Gregory Abowd, Len Bass and Paul Clements. *Scenario-based analysis of software architecture.* IEEE Software, vol. 13, no. 6, pages 47–55, Nov 1996. 112

[Keim 2006] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind and Hartmut Ziegler. *Challenges in Visual Data Analysis.* In IV '06, Proceedings of the conference on Information Visualization, pages 9–16, Washington, DC, USA, 2006. IEEE Computer Society. 9, 47, 198, 311

[Keim 2008a] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer and Guy Melancon. *Visual Analytics:*

Definition, Process, and Challenges. In Andreas Kerren, John T. Stasko, Jean-Daniel Fekete and Chris North, editeurs, Information Visualization, volume 4950 of *Lecture Notes in Computer Science*, pages 154–175. Springer Berlin Heidelberg, 2008. 47

[Keim 2008b] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind, Jim Thomas and Hartmut Ziegler. Visual data mining, chapitre Visual Analytics: Scope and Challenges, pages 76–90. Springer-Verlag, Berlin, Heidelberg, 2008. 198

[Keivanloo 2011] Iman Keivanloo, Christopher Forbes, Juergen Rilling and Philippe Charland. *Towards Sharing Source Code Facts Using Linked Data*. In Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation, SUITE '11, pages 25–28, New York, NY, USA, 2011. ACM. 43

[Kelley 2013] Sean Kelley, Edward Aftandilian, Connor Gramazio, Nathan Ricci, Sara L. Su and Samuel Z. Guyer. *Heapviz: Interactive heap visualization for program understanding and debugging*. Information Visualization, vol. 12, no. 2, pages 163–177, 04 2013. 305

[Kemmis 2005] Stephen Kemmis and Robin McTaggart. The sage handbook of qualitative research (3rd ed.), chapitre Participatory Action Research: Communicative Action and the Public Sphere., pages 559–603. Sage Publications Ltd, 2005. 13, 315

[Kendall 2012] Wesley Kendall, Jian Huang and Tom Peterka. *Geometric Quantification of Features in Large Flow Fields*. IEEE Computer Graphics and Applications, vol. 32, no. 4, pages 46–54, 2012. 48

[Khan 2010] Mumtaz Muhammad Khan, Sulaiman Aziz Lodhi and Muhammad Abdul Majid Makk. *Measuring Team Implicit Coordination*. Australian Journal of Basic and Applied Sciences, vol. 4, no. 6, pages 1211–1136, 2010. 151

[Khan 2012] Taimur Khan, Henning Barthel, Achim Ebert and Peter Liggesmeyer. *Visualization and Evolution of Software Architectures*. In Christoph Garth, Ariane Middel and Hans Hagen, editeurs, Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011, volume 27 of *OpenAccess Series in Informatics (OASIcs)*, pages 25–42, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 113

[Kiczales 1997] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin. *Aspect-oriented programming.* In Mehmet Aksit and Satoshi Matsuoka, editeurs, ECOOP'97 - Object-Oriented Programming, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin Heidelberg, 1997. 91

[Kiekel 2011] Preston A . Kiekel and Nancy J . Cooke. Handbook of human factors in web design, second edition, chapitre Human Factor Aspects of Team Cognition, pages 107 – 123. CRC Press, 2011. 148, 149, 156

[Kienle 2007] Holger M. Kienle and Hausi A. Müller. *Requirements of Software Visualization Tools: A Literature Survey.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 2–9, 2007. 303

[Kilpi 1997] Tapani Kilpi. *Choosing a SCM-tool: a framework and evaluation.* In Eighth Conference on Software Engineering Environments, pages 164–172, 1997. 33

[Kim 2006] Sunghun Kim, Kai Pan and Jr. Emmet James Whitehead. *Micro Pattern Evolution.* In Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR '06, pages 40–46, New York, NY, USA, 2006. ACM. 44

[Kim 2011] Miryung Kim. *An Exploratory Study of Awareness Interests About Software Modifications.* In Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '11, pages 80–83, New York, NY, USA, 2011. ACM. 38, 41

[Kim 2012] Jinah Kim and Jinah Park. *Visualizing Marine Environmental Changes to the Saemangeum Coast.* IEEE Computer Graphics and Applications, vol. 32, no. 6, pages 82–87, 2012. 48

[Kirsch 1996] Laurie J. Kirsch. *The Management of Complex Tasks in Organizations: Controlling the Systems Development Process.* Organization Science, vol. 7, no. 1, pages 1–21, 1996. 154

[Kitchenham 1989] Barbara A. Kitchenham and John G. Walker. *A quantitative approach to monitoring software development.* Software Engineering Journal, vol. 4, no. 1, pages 2–13, Jan 1989. 154

[Klimoski 1994] Richard Klimoski and Susan Mohammed. *Team mental model: construct or metaphor?* Journal of Management, vol. 20, no. 2, pages 403 – 437, 1994. A Special Issue of The Journal of Management. 148, 155, 156

[Ko 2007] Andrew J. Ko, Robert DeLine and Gina Venolia. *Information Needs in Collocated Software Development Teams.* In Proceedings of the 29th International Conference on Software Engineering, ICSE '07, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society. 25, 38, 40

[Ko 2012] Sungahn Ko, Ross Maciejewski, Yun Jang and David S. Ebert. *MarketAnalyzer: An Interactive Visual Analytics System for Analyzing Competitive Advantage Using Point of Sale Data.* Computer Graphics Forum, vol. 31, no. 3pt3, pages 1245–1254, 2012. 48

[Koch 2011] Steffen Koch, Harald Bosch, Mark Giereth and Thomas Ertl. *Iterative Integration of Visual Insights during Scalable Patent Search and Analysis.* IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 5, pages 557 –569, may 2011. 48

[Kohlhammer 2012] Jörn Kohlhammer, Kawa Nazemi, Tobias Ruppert and Dirk Burkhardt. *Toward Visualization in Policy Modeling.* IEEE Computer Graphics and Applications, vol. 32, no. 5, pages 84–89, 2012. 48

[Koike 1993] Hideki Koike. *The Role of Another Spatial Dimension in Software Visualization.* ACM Transactions in Information Systems, vol. 11, no. 3, pages 266–286, July 1993. 246, 248, 250

[Koike 1997] Hideki Koike and Hui-Chu Chu. *VRCS: Integrating Version Control and Module Management using Interactive 3D graphics.* In Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97), page 168, Washington, DC, USA, 1997. IEEE Computer Society. XVII, 246, 248, 250

[Koschke 2003] Rainer Koschke. *Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey.* Journal of Software Maintenance and Evolution: Research and Practice, vol. 15, no. 2, pages 87–109, 2003. 11, 184, 313

[Kotlarsky 2008] Julia Kotlarsky, Paul C. van Fenema and Leslie P. Willcocks. *Developing a knowledge-based perspective on coordination: The case of*

global software projects. Information and Management, vol. 45, no. 2, pages 96 – 108, 2008. 151

[Kraut 1995] Robert E. Kraut and Lynn A. Streeter. *Coordination in Software Development*. Communications of the ACM, vol. 38, no. 3, pages 69–81, March 1995. 4, 151, 152, 153, 308

[Krishnan 2013] Krish Krishnan. Data warehousing in the age of big data. The Morgan Kaufmann Series on Business Intelligence Series. Elsevier Science & Technology Books, 2013. 6

[Kuhn 2010a] Adrian Kuhn, David Erni, Peter Loretan and Oscar Nierstrasz. *Software Cartography: thematic software visualization with consistent layout*. Journal of Software Maintenance and Evolution: Research and Practice, vol. 22, no. 3, pages 191–210, 2010. XIV, 138, 139, 306

[Kuhn 2010b] Adrian Kuhn, David Erni and Oscar Nierstrasz. *Embedding spatial software visualization in the IDE: an exploratory study*. In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 113–122, New York, NY, USA, 2010. ACM. 139, 306

[Kuhn 2012] Adrian Kuhn and Mirko Stocker. *CodeTimeline: Storytelling with versioning data*. In 34th International Conference on Software Engineering (ICSE), 2012, pages 1333–1336, 2012. XV, 166, 168, 306

[Laguna 2003] Miguel A. Laguna, José M. Marqués and Francisco J. García-Peñalvo. *DocFlow: workflow based requirements elicitation*. Information and Software Technology, vol. 45, no. 6, pages 357 – 369, 2003. 4, 309

[Lai 2003] Su-Ying Lai, Richard Heeks and Brian Nicholson. *Uncertainty and Coordination in Global Software Projects: A UK/India-Centred Case Study*. In Development informatics working paper series, numéro 17 de Development informatics working paper series. Manchester : Institute for Development Policy and Management, University of Manchester, 2003. 152, 153

[Laird 2006] Linda M. Laird and M. Carol Brennan. Software measurement and estimation: A practical approach (quantitative software engineering series). Wiley-IEEE Computer Society Pr, 2006. 90

[LaMantia 2008] Matthew J. LaMantia, Yuanfang Cai, Alan D. MacCormack and John Rusnak. *Analyzing the Evolution of Large-Scale Software*

Systems Using Design Structure Matrices and Design Rule Theory: Two Exploratory Cases. In Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), WICSA '08, pages 83–92, Washington, DC, USA, 2008. IEEE Computer Society. 44

[Lamping 1995] John Lamping, Ramana Rao and Peter Pirolli. *A focus+context technique based on hyperbolic geometry for visualizing large hierarchies.* In CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. 57

[Lanza 2001a] Michele Lanza. *The evolution matrix: recovering software evolution using software visualization techniques.* In IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution, pages 37–42, New York, NY, USA, 2001. ACM Press. XIII, 125, 126, 144

[Lanza 2001b] Michele Lanza and Stéphane Ducasse. *A Categorization of Classes Based on the Visualization of Their Internal Structure: The Class Blueprint.* SIGPLAN Notices, vol. 36, no. 11, pages 300–311, October 2001. XIII, 127, 128, 144

[Lanza 2003] Michele Lanza and Stéphane Ducasse. *Polymetric Views-A Lightweight Visual Approach to Reverse Engineering.* IEEE Transactions in Software Engineering, vol. 29, no. 9, pages 782–795, September 2003. XIII, 126, 127, 144

[Lanza 2005a] Michele Lanza, Stéphane Ducasse, Harald Gall and Marting Pinzger. *CodeCrawler - an information visualization tool for program comprehension.* In Proceedings 27th International Conference on Software Engineering, 2005. ICSE 2005., pages 672–673, May 2005. 125

[Lanza 2005b] Michele Lanza, Radu Marinescu and Stéphane Ducasse. Object-oriented metrics in practice - using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. 41, 44, 89, 182, 196, 313

[Lanza 2010] Michele Lanza, Lile Hattori and Anja Guzzi. *Supporting Collaboration Awareness with Real-Time Visualization of Development Activity.* In Proceedings of the 2010 14th European Conference on

Software Maintenance and Reengineering, CSMR '10, pages 202–211, Washington, DC, USA, 2010. IEEE Computer Society. XV, 173, 174, 175

[Laumer 2011] Sven Laumer, Christian Maier, Andreas Eckhardt and Tim Weitzel. *The trend is our friend: german IT personnel's perception of job-related factors before, during and after the economic downturn.* In Proceedings of the 49th SIGMIS annual conference on Computer personnel research, SIGMIS-CPR '11, pages 65–70, New York, NY, USA, 2011. ACM. 7, 27

[Laval 2009] Jannik Laval, Simon Denier, Stéphane Ducasse and Alexandre Bergel. *Identifying Cycle Causes with Enriched Dependency Structural Matrix.* In 16th Working Conference on Reverse Engineering, 2009. WCRE '09., pages 113–122, 2009. 304

[LaValle 2010] Steve LaValle, Michael S. Hopkins, Eric Lesser, Rebecca Shockley and Nina Kruschwitz. *Analytics: The new path to value.* Mit Sloan Management Review, 2010. 6

[Lee 2006] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete and Nathalie Henry. *Task taxonomy for graph visualization.* In Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization, BELIV '06, pages 1–5, New York, NY, USA, 2006. ACM. 60

[Lee 2009] Sang-Yong Tom Lee, Hee-Woong Kim and Sumeet Gupta. *Measuring open source software success.* Omega: The International Journal of Management Science, vol. 37, no. 2, pages 426 – 438, 2009. 4, 5, 308, 309

[Lee 2011] Teng-Yok Lee. *Data Triage and Visual Analytics for Scientific Visualization.* PhD thesis, The Ohio State University, 2011. 62

[Lehman 1997] Meir M. Manny Lehman, Juan F. Ramil, Paul D. Wernick, Dewayne E. Perry and Wladyslaw M Turski. *Metrics and Laws of Software Evolution - The Nineties View.* In Proceedings of the 4th International Symposium on Software Metrics, METRICS '97, pages 20–, Washington, DC, USA, 1997. IEEE Computer Society. 5, 30, 41, 310

[Leinonen 2005] Piritta Leinonen, Sanna Järvelä and Päivi Häkkinen. *Conceptualizing the Awareness of Collaboration: A Qualitative Study*

of a Global Virtual Team. Computer Supported Cooperative Work, vol. 14, no. 4, pages 301–322, August 2005. 158

[Lemieux 2011] Victoria L. Lemieux. *Visual Analytics: A New Way to Manage Data Deluge in E-Discovery.* Information Management Journal, vol. 45, no. 2, pages 38 – 40, 2011. 48

[Leung 1994a] Ying K. Leung and Mark D. Apperley. *A review and taxonomy of distortion-oriented presentation techniques.* ACM Transactions in Computer-Human Interaction, vol. 1, no. 2, pages 126–160, June 1994. 9, 47, 119, 311

[Leung 1994b] Ying K. Leung and Mark D. Apperley. *A review and taxonomy of distortion-oriented presentation techniques.* ACM Transactions in Computer Human Interaction, vol. 1, no. 2, pages 126–160, 1994. 51, 52, 62

[Li 2012] Kaiming Li, Lei Guo, Carlos Faraco, Dajiang Zhu, Hanbo Chen, Yixuan Yuan, Jinglei Lv, Fan Deng, Xi Jiang, Tuo Zhang, Xintao Hu, Degang Zhang, L. Stephen Miller and Tianming Liu. *Visual analytics of brain networks.* NeuroImage, vol. 61, no. 1, pages 82 – 97, 2012. 48

[Liebrock 2009] Daniel A. Quistand Lorie M. Liebrock. *Visualizing compiled executables for malware analysis.* In 6th International Workshop on Visualization for Cyber Security, 2009. VizSec 2009., pages 27–32, 2009. 305

[Likert 1932] Rensis Likert. *A technique for the measurement of attitudes.* Archives of Psychology, vol. 22, no. 140, pages 1–55, 1932. 273

[Limberger 2013] Daniel Limberger, Benjamin Wasty, Jonas Trümper and Jürgen Döllner. *Interactive software maps for web-based source code analysis.* In Proceedings of the 18th International Conference on 3D Web Technology, Web3D '13, pages 91–98, New York, NY, USA, 2013. ACM. 116, 306

[Lin 2010] Shen Lin, François Taïani, Thomas C. Ormerod and Linden J. Ball. *Towards anomaly comprehension: using structural compression to navigate profiling call-trees.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 103–112, New York, NY, USA, 2010. ACM. 305

[Lincke 2008] Rüdiger Lincke, Jonas Lundberg and Welf Löwe. *Comparing Software Metrics Tools.* In Proceedings of the 2008 International

Symposium on Software Testing and Analysis, ISSTA '08, pages 131–142, New York, NY, USA, 2008. ACM. 43

[Lintern 2003] Rob Lintern, Jeff Michaud, Margaret-Anne Storey and Wu Xiaomin. *Plugging-in visualization: experiences integrating a visualization tool with Eclipse.* In Proceedings of the 2003 ACM symposium on Software visualization, SoftVis '03, pages 47–ff, New York, NY, USA, 2003. ACM. 191

[Livnat 2012] Yarden Livnat, Theresa-Marie Rhyne and Matthew H. Samore. *Epinome: A Visual-Analytics Workbench for Epidemiology Data.* Computer Graphics and Applications, IEEE, vol. 32, no. 2, pages 89 –95, march-april 2012. 48

[Llorá 2006] X. Llorá, K. Sastry, F. Alías, D. E. Goldberg and M. l. Welge. *Analyzing Active Interactive Genetic Algorithms Using Visual Analytics.* In GECCO '06, Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 1417–1418, New York, NY, USA, 2006. ACM. 9, 47, 311

[Long 2009] Tran Van Long and Lars Linsen. *MultiClusterTree: Interactive Visual Exploration of Hierarchical Clusters in Multidimensional Multivariate Data.* Computer Graphics Forum, vol. 28, no. 3, pages 823–830, 2009. 60

[Luhmann 1992] Niklas Luhmann. *What is Communication?* Communication Theory, vol. 2, no. 3, pages 251–259, 1992. 150

[Lungu 2010] Mircea Lungu, Michele Lanza, Tudor Gîrba and Romain Robbes. *The Small Project Observatory: Visualizing software ecosystems.* Science of Computer Programming, vol. 75, no. 4, pages 264 – 275, 2010. 305

[Luo 2012] Dongning Luo, Jing Yang, Milos Krstajic, William Ribarsky and Daniel A. Keim. *EventRiver: Visually Exploring Text Collections with Temporal References.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 1, pages 93 –105, jan. 2012. 48

[Luo 2013] Yi Luo and Yanying Han. *Source Code Visualization in Linux Environment Based on Hierarchica Layout Algorithm.* Information Technology Journal, vol. 12, no. 8, pages 1522–1530, August 2013. 306

[Maciejewski 2010] Ross Maciejewski, Travis Drake, Stephen Rudolph, Abish Malik and David S. Ebert. *Data Aggregation and Analysis for Cancer Statistics - A Visual Analytics Approach.* In 43rd Hawaii International Conference on System Sciences (HICSS), pages 1 –5, jan. 2010. 48

[Maciejewski 2011] Ross Maciejewski, Ryan Hafen, Stephen Rudolph, Stephen G. Larew, Michael A. Mitchell, William S. Cleveland and David S. Ebert. *Forecasting Hotspots-A Predictive Analytics Approach.* IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 4, pages 440 –453, april 2011. 49

[Mackinlay 1991] Jock D. Mackinlay, George G. Robertson and Stuart K. Card. *The perspective wall: detail and context smoothly integrated.* In CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 173–176, New York, NY, USA, 1991. ACM Press. 52, 54

[MacMillan 2004] Jean MacMillan, Elliot E. Entin and Daniel Serfaty. Team cognition: Understanding the factors that drive process and performance, chapitre Communication overhead: The hidden cost of team cognition., pages 61–82. American Psychological Association, Washington, DC, US, 2004. 150, 151, 156, 157

[Madhavi 2011] Karanam Madhavi and Akepogu Anand Rao. *A Framework for Visualizing Model-Driven Software Evolution-Its Evaluation.* International Journal of Software Engineering and Its Applications, vol. 5, pages 135–148, 2011. 303

[Mahoney 2009] Mark Mahoney. *Software evolution and the moving picture metaphor.* SIGPLAN Notices, vol. 44, no. 10, pages 525–528, October 2009. 196

[Mahyar 2012] Narges Mahyar, Ali Sarvghad and Melanie Tory. *Note-taking in co-located collaborative visual analytics: Analysis of an observational study.* Information Visualization, vol. 11, no. 3, pages 190–204, 07 2012. 49

[Maletic 2002] Jonathan I. Maletic, Andrian Marcus and Michael L. Collard. *A Task Oriented View of Software Visualization.* IEEE Workshop of Visualizing Software for Understanding and Analysis (VISSOFT), vol. 26, pages 32–40, 2002. 161, 183

[Maletic 2004] Jonathan I. Maletic and Michael L. Collard. *Supporting source code difference analysis.* In Proceedings of the 20th IEEE International

Conference on Software Maintenance, 2004., pages 210–219, Sept 2004. 44

[Mane 2012] Ketan K. Mane, Chris Bizon, Charles Schmitt, Phillips Owen, Bruce Burchett, Ricardo Pietrobon and Kenneth Gersing. *VisualDecisionLinc: A visual analytics approach for comparative effectiveness-based clinical decision support in psychiatry.* Journal of Biomedical Informatics, vol. 45, no. 1, pages 101 – 106, 2012. 10, 48, 312

[Maoz 2011] Shahar Maoz and David Harel. *On tracing reactive systems.* Software & Systems Modeling, vol. 10, no. 4, pages 447 – 468, 2011. 305

[McCabe 1976] Thomas J. McCabe. *A Complexity Measure.* IEEE Transactions on Software Engineering, vol. 2, no. 4, pages 308–320, 1976. 41

[Mens 2001] Tom Mens and Serge Demeyer. *Future trends in software evolution metrics.* In Proceedings of the 4th International Workshop on Principles of Software Evolution, IWPSE '01, pages 83–86, New York, NY, USA, 2001. ACM. 41, 44

[Mens 2008] Tom Mens and Serge Demeyer, editeurs. Software evolution. Springer, 2008. 7, 196, 313

[Merriam-Webster Online 2009] Merriam-Webster Online. *Merriam-Webster Online Dictionary*, 2009. 25

[Meyer 2012] Joerg Meyer, E. Wes Bethel, Jennifer L. Horsman, Susan S. Hubbard, Harinarayan Krishnan, Alexandru Romosan, Elizabeth H. Keating, Laura Monroe, Richard Strelitz, Phil Moore, Glenn Taylor, Ben Torkian, Timothy C. Johnson and Ian Gorton. *Visual Data Analysis as an Integral Part of Environmental Management.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, pages 2088–2094, 2012. 49

[Meyers 2007] Timothy M. Meyers and David Binkley. *An Empirical Study of Slice-based Cohesion and Coupling Metrics.* ACM Transactions in Software Engineering Methodologies, vol. 17, no. 1, pages 2:1–2:27, December 2007. 44

[Migut 2011] Malgorzata Migut, Jan van Gemert and Marcel Worring. *Interactive decision making using dissimilarity to visually represented*

prototypes. In 2011 IEEE Conference on Visual Analytics Science and Technology (VAST), pages 141 –149, oct. 2011. 9, 311

[Migut 2012] Malgorzata Migut and Marcel Worring. *Visual exploration of classification models for various data types in risk assessment*. Information Visualization, vol. 11, no. 3, pages 237–251, 07 2012. 49

[Minelli 2013] Roberto Minelli and Michele Lanza. *Software Analytics for Mobile Applications–Insights amp;amp; Lessons Learned*. In 17th European Conference on Software Maintenance and Reengineering (CSMR), 2013, pages 144–153, 2013. 303

[Mintzberg 1991] Henry Mintzberg. *The Effective Organization: Forces and Forms*. Mit Sloan Management Review, January, 15 1991. 24

[Misra 2013] Sanjay Misra, Ricardo Colomo-Palacios, Tolga Pusatli and Pedro Soto-Acosta. *A discussion on the role of people in global software development*. Tehnicki vjesnik / Technical Gazette, vol. 20, no. 3, pages 525 – 531, 2013. 24, 38

[Mockus 2001] Audris Mockus and David M. Weiss. *Globalization by Chunking: A Quantitative Approach*. IEEE Software, vol. 18, no. 2, pages 30–37, March 2001. 24, 149, 150

[Moons 2009] Jan Moons and Carlos De Backer. *Rationale Behind the Design of the EduVisor Software Visualization Component*. Electronic Notes in Theoretical Computer Science, vol. 224, no. 0, pages 57 – 65, 2009. 303

[Moreta 2007] Sergio Moreta and Alexandru Telea. *Visualizing Dynamic Memory Allocations*. In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 31–38, 2007. 305

[Morisaki 2007] Shuji Morisaki, Akito Monden, Tomoko Matsumura, Haruaki Tamada and Ken ichi Matsumoto. *Defect Data Analysis Based on Extended Association Rule Mining*. In Proceedings of the Fourth International Workshop on Mining Software Repositories, MSR '07, pages 3–, Washington, DC, USA, 2007. IEEE Computer Society. 43

[Morisio 2002] Maurizio Morisio, Michel Ezran and Colin Tully. *Success and failure factors in software reuse*. IEEE Transactions on Software Engineering, vol. 28, no. 4, pages 340–357, Apr 2002. 4, 308

[Morris 2003] Steven A. Morris, G. Yen, Zheng Wu and Benyam Asnake. *Time line visualization of research fronts*. Journal of the American Society for Information Science and Technology, vol. 54, no. 5, pages 413–422, 2003. 55

[Munch 2004] Jurgen Munch and Jens Heidrich. *Software project control centers: concepts and approaches*. Journal of Systems and Software, vol. 70, no. 1,2, pages 3–19, 2004. 10, 312

[Murphy-Hill 2010] Emerson Murphy-Hill and Andrew P. Black. *An interactive ambient visualization for code smells*. In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 5–14, New York, NY, USA, 2010. ACM. 305

[Murphy-Hill 2013] Emerson Murphy-Hill, Titus Barik and Andrew P. Black. *Interactive ambient visualizations for soft advice*. Information Visualization, vol. 12, no. 2, pages 107–132, 04 2013. 305

[Murphy 1997] Gail C. Murphy and David Notkin. *Reengineering with Reflection Models: A Case Study*. IEEE Computer, vol. 30, no. 8, pages 29–36, 1997. 7, 27

[Myers 2010] Colin Myers and David Duke. *A map of the heap: revealing design abstractions in runtime structures*. In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 63–72, New York, NY, USA, 2010. ACM. 305

[Myller 2009] Niko Myller, Roman Bednarik, Erkki Sutinen and Mordechai Ben-Ari. *Extending the Engagement Taxonomy: Software Visualization and Collaborative Learning*. Transactions on Computing Education, vol. 9, no. 1, pages 7:1–7:27, March 2009. 303

[Nam 2013] Julia EunJu Nam and Klaus Mueller. *TripAdvisorN̂-D: A TourismInspired HighDimensional Space Exploration Framework with Overview and Detail*. IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 2, pages 291–305, 2013. 49

[Nasir 2011] Mohd Hairul Nizam Nasir and Shamsul Sahibuddin. *Critical success factors for software projects: A comparative study*. Scientific Research and Essays, vol. 6, no. 10, pages 2174–2186, May 2011. 5, 309

[Nestor 2008] Daren Nestor, Steffen Thiel, Goetz Botterweck, Ciarán Cawley and Patrick Healy. *Applying visualisation techniques in software*

product lines. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 175–184, New York, NY, USA, 2008. ACM. 306

[Neu 2011] Sylvie Neu, Michele Lanza, Lile Hattori and Marco D'Ambros. *Telling stories about GNOME with Complicity*. In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–8, 2011. 305

[Niazi 2006] Mahmood Niazi, David Wilson and Didar Zowghi. *Critical success factors for software process improvement implementation: an empirical study*. Software Process: Improvement and Practice, vol. 11, no. 2, pages 193–211, 2006. 5, 309

[Nielsen 14] Jakob Nielsen and Don Norman. *The Definition of User Experience*. Website, 3 14. 271

[Noda 2009] Kunihiro Noda, Takashi Kobayashi, Kiyoshi Agusa and Shinichiro Yamamoto. *Sequence Diagram Slicing*. In Software Engineering Conference, 2009. APSEC '09. Asia-Pacific, pages 291–298, 2009. 305

[North 2000] Chris North and Ben Shneiderman. *Snap-together visualization: can users construct and operate coordinated visualizations*. International Journal of Human-Computer Studies, vol. 53, no. 5, pages 715 – 739, 2000. 10, 47, 198, 200, 311

[Novais 2011] Renato L. Novais, Caio A. N. Lima, Glauco de F. Carneiro, Paulo R. M. S. Júnior and Manoel Mendonca. *An interactive differential and temporal approach to visually analyze software evolution*. In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–4, 2011. 306

[Novais 2012] Renato Lima Novais, Camila Nunes, Caio Lima, Elder Cirilo, Francisco Dantas, Alessandro Garcia and Manoel Mendonca. *On the proactive and interactive visualization for feature evolution comprehension: An industrial investigation*. In 34th International Conference on Software Engineering (ICSE), 2012, pages 1044–1053, 2012. 306

[Novais 2013] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonca and Nico Zazworka. *Software evolution*

visualization: A systematic mapping study. Information and Software Technology, no. 0, pages –, 2013. 110, 303

[Oeltze 2011] Steffen Oeltze, Wolfgang Freiler, Reyk Hillert, Helmut Doleisch, Bernhard Preim and Walter Schubert. *Interactive, Graph-based Visual Analysis of High-dimensional, Multi-parameter Fluorescence Microscopy Data in Toponomics.* IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 12, pages 1882 –1891, dec. 2011. 48

[Ogawa 2009] Michael Ogawa and Kwan-Liu Ma. *code_swarm: A Design Study in Organic Software Visualization.* IEEE Transactions on Visualization and Computer Graphics, vol. 15, no. 6, pages 1097–1104, nov 2009. 5, 23, 303, 309

[Ogawa 2010] Michael Ogawa and Kwan-Liu Ma. *Software evolution storylines.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 35–42, New York, NY, USA, 2010. ACM. 303

[Olchi 1978] William G. Olchi. *The Transmission of Control Through Organizational Hierarchy.* Academy of Management Journal, vol. 2, no. 2, pages 173–192, June 1978. 154

[Omer 2010] Itzhak Omer, Peter Bak and Tobias Schreck. *Using space-time visual analytic methods for exploring the dynamics of ethnic groups' residential patterns.* International Journal of Geographical Information Science, vol. 24, no. 10, pages 1481–1496, October 2010. 49

[Omoronyia 2010] Inah Omoronyia, John Ferguson, Marc Roper and Murray Wood. *A review of awareness in distributed collaborative software engineering.* Software: Practice and Experience, vol. 40, no. 12, pages 1107–1133, 2010. 23, 38

[Ooms 2012] Kristien Ooms, Gennady Andrienko, Natalia Andrienko, Philippe De Maeyer and Veerle Fack. *Analysing the spatial dimension of eye movement data using a visual analytic approach.* Expert Systems with Applications, vol. 39, no. 1, pages 1324 – 1332, 2012. 48

[Owens 2011] Dawn Owens and Deepak Khazanchi. *Best Practices for Retaining Global IT Talent.* In System Sciences (HICSS), 2011 44th Hawaii International Conference on, pages 1 –12, jan. 2011. 7, 27

[Panas 2003] Thomas Panas, Rebecca Berrigan and John Grundy. *A 3D metaphor for software production visualization.* In Proceedings. Seventh International Conference on Information Visualization, 2003. IV 2003., pages 314–319, July 2003. 114, 143

[Panas 2005] Thomas Panas, Rüdiger Lincke and Welf Löwe. *Online-configuration of Software Visualizations with Vizz3D.* In Proceedings of the 2005 ACM Symposium on Software Visualization, SoftVis '05, pages 173–182, New York, NY, USA, 2005. ACM. 114, 143

[Park 2009] Yunrim Park and Carlos Jensen. *Beyond pretty pictures: Examining the benefits of code visualization for Open Source newcomers.* In 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009., pages 3–10, 2009. 303

[Parnin 2007] Chris Parnin and Carsten Görg. *Design Guidelines for Ambient Software Visualization in the Workplace.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 18–25, 2007. 303

[Parnin 2008] Chris Parnin, Carsten Görg and Ogechi Nnadi. *A catalogue of lightweight visualizations to support code smell inspection.* In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 77–86, New York, NY, USA, 2008. ACM. 303

[Parnin 2010] Chris Parnin, Carsten Görg and Spencer Rugaber. *CodePad: interactive spaces for maintaining concentration in programming environments.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 15–24, New York, NY, USA, 2010. ACM. 303

[Paul 1999] Raymond A. Paul, Tosiyasu L. Kunii, Yoshihisa Shinagawa and Muhammad F. Khan. *Software metrics knowledge and databases for project management.* IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, pages 255 –264, jan/feb 1999. 10, 312

[Pauw 2006] Wim De Pauw, Sophia Krasikov and John F. Morar. *Execution patterns for visualizing web services.* In Proceedings of the 2006 ACM symposium on Software visualization, SoftVis '06, pages 37–45, New York, NY, USA, 2006. ACM. 44

[Pauw 2009] Wim De Pauw and Henrique Andrade. *Visualizing large-scale streaming applications.* Information Visualization, vol. 8, no. 2, pages 87–106, 2009. 305

[Pauw 2010] Wim De Pauw and Steve Heisig. *Zinsight: a visual and analytic environment for exploring large event traces.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 143–152, New York, NY, USA, 2010. ACM. 305

[Pauw 2013] Wim De Pauw, Joel Wolf and Andrey Balmin. *Visualizing jobs with shared resources in distributed environments.* In First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pages 1–10, 2013. 305

[Pauwels 2010] Stefan L. Pauwels, Christian Hübscher, Javier A. Bargas-Avila and Klaus Opwis. *Building an interaction design pattern language: A case study.* Computer in Human Behaviour, vol. 26, no. 3, pages 452–463, May 2010. 63

[Pavlo 2006] Andrew Pavlo, Christopher Homan and Jonathan Schull. *A parent-centered radial layout algorithm for interactive graph visualization and animation.* ArXiv Computer Science e-prints, jun 2006. 52, 58

[Peck 2011] R. Peck, C. Olsen and J.L. Devore. Introduction to statistics and data analysis. Available Titles Aplia Series. Cengage Learning, 2011. 9, 47, 311

[Peláez 2008] Rafael Peláez, Roberto Therón, Carlos Armando García, José Luis López-Pérez and Manuel Medarde. *Design of New Chemoinformatic Tools for the Analysis of Virtual Screening Studies: Application to Tubulin Inhibitors.* In 2nd International Workshop on Practical Applications of Computational Biology and Bioinformatics, IWPACBB 2008, Salamanca, Spain, 22th-24th October 2008, pages 189–196, 2008. 14, 49

[Pelekis 2012] Nikos Pelekis, Gennady Andrienko, Natalia Andrienko, Ioannis Kopanakis, Gerasimos Marketos and Yannis Theodoridis. *Visually exploring movement data via similarity-based analysis.* Journal of Intelligent Information Systems, vol. 38, pages 343–391, 2012. 10.1007/s10844-011-0159-2. 48, 49

[Perer 2011] A. Perer, I. Guy, E. Uziel, I. Ronen and M. Jacovi. *Visual social network analytics for relationship discovery in the enterprise.* In 2011

IEEE Conference on Visual Analytics Science and Technology (VAST), pages 71 –79, oct. 2011. 9, 311

[Perer 2013] Adam Perer, Ido Guy, Erel Uziel, Inbal Ronen and Michal Jacovi. *The Longitudinal Use of SaNDVis: Visual Social Network Analytics in the Enterprise*. IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 7, pages 1095–1108, 2013. 49

[Pérez 2013] Carlos Armando García Pérez. *Analítica visual aplicada al diseño de nuevos fármacos*. PhD thesis, University of Salamanca, 2013. 14, 49

[PerforceSoftware 2014] PerforceSoftware. *Introducing Perforce*, 12 2014. XVII, 246, 249, 250

[Petersen 2008] Kai Petersen, Robert Feldt, Shahid Mujtaba and Michael Mattsson. *Systematic mapping studies in software engineering*. In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08, pages 68–77, Swinton, UK, UK, 2008. British Computer Society. 71

[Peterson 2012] Elena S. Peterson, Lee Ann McCue, Alexandra C. Schrimpe-Rutledge, Jeffrey L Jensen, Hyunjoo Walker, Markus A Kobold, Samantha R. Webb, Samuel H. Payne, Charles Ansong, Joshua N. Adkins, William R. Cannon and Bobbie-Jo M. Webb-Robertson. *VESPA: software to facilitate genomic annotation of prokaryotic organisms through integration of proteomic and transcriptomic data*. BMC Genomics, vol. 13, no. 1, page 131, 2012. 48

[Petre 1998] Marian Petre, Alan F. Blackwell and Thomas R. G. Green. Software visualization: Programming as a multi-media experience, chapitre Cognitive Questions in Software Visualisation, pages 453–480. MIT Press, January 1998. 196

[Petre 2010] Marian Petre. *Mental imagery and software visualization in high-performance software development teams*. Journal of Visual Languages & Computing, vol. 21, no. 3, pages 171 – 183, 2010. 303

[Pileggi 2012] Hannah Pileggi, Charles D. Stolper, J. Michael Boyle, and John T. Stasko. *SnapShot: Visualization to Propel Ice Hockey Analytics*. IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, pages 2819–2828, 2012. 49

[Pilgrim 2009] Jens Von Pilgrim, Kristian Duske and Paul Mcintosh. *Eclipse GEF3D: Bringing 3D to existing 2D editors*. Information Visualization, vol. 8, no. 2, pages 107–119, Summer 2009. 303

[Pinelle 2005] David Pinelle and Carl Gutwin. *A Groupware Design Framework for Loosely Coupled Workgroups*. In Proceedings of the Ninth Conference on European Conference on Computer Supported Cooperative Work, ECSCW'05, pages 65–82, New York, NY, USA, 2005. Springer-Verlag New York, Inc. 24

[Pinzger 2005] Martin Pinzger, Harald Gall, Michael Fischer and Michele Lanza. *Visualizing Multiple Evolution Metrics*. In Proceedings of the 2005 ACM Symposium on Software Visualization, SoftVis '05, pages 67–75, New York, NY, USA, 2005. ACM. 182

[Pirolli 2001] Peter Pirolli, Stuart K. Card and Mija M. Van Der Wege. *Visual information foraging in a focus + context visualization*. In Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '01, pages 506–513, New York, NY, USA, 2001. ACM. 62

[Plaisant 1998] Catherine Plaisant, Daniel Heller, Jia Li, Ben Shneiderman, Rich Mushlin and John Karat. *Visualizing medical records with LifeLines*. In CHI '98: CHI 98 conference summary on Human factors in computing systems, pages 28–29, New York, NY, USA, 1998. ACM. 52, 54

[Ploeger 2008] Bas Ploeger and Carst Tankink. *Improving an interactive visualization of transition systems*. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 115–124, New York, NY, USA, 2008. ACM. 304

[PMI 2002] Project Management Institute PMI. Project manager competency development (pmcd) framework. Project Management Institute, Inc., 2002. 23

[Pohl 2012] Margit Pohl, Michael Smuc, and Eva Mayr. *The User Puzzle-Explaining the Interaction with Visual Analytics Systems*. IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, pages 2908–2916, 2012. 49

[Predonzani 1998] Paolo Predonzani, Giancarlo Succi and Tullio Vernazza. *Skill management in software engineering*. In Proceedings of the Thirteen International Conference and Symposium on Computer and Information Sciences, 1998. 38

[Prikladnicki1 2003] Rafael Prikladnicki1, Jorge Luis Nicolas Audy and Roberto Evaristo. *Global software development in practice lessons learned*. Software Process: Improvement and Practice, vol. 8, no. 4, pages 267–281, 2003. 23, 24

[Procaccino 2002] J. Drew Procaccino, June M. Verner, Scott P. Overmyer and Marvin E. Darter. *Case study: factors for early prediction of software development success*. Information & Software Technology, vol. 44, no. 1, pages 53–62, 2002. 4, 308

[Proulx 2006] Pascale Proulx, Sumeet Tandon, Adam Bodnar, David Schroh, Robert Harper and William Wright. *Avian Flu Case Study with nSpace and GeoTime*. In 2006 IEEE Symposium On Visual Analytics Science And Technology, pages 27 –34, 31 2006-nov. 2 2006. 9, 311

[Qualtrics, Inc. 2013] Qualtrics, Inc. *Qualtrics Web Survey Tool*, 2013. `www.qualtrics.com`. 178, 273

[Quist 2011] Daniel A. Quist and Lorie M. Liebrock. *Reversing Compiled Executables for Malware Analysis via Visualization*. Information Visualization, vol. 10, no. 2, pages 117–126, 04 2011. 306

[Ramesh 2006] Balasubramaniam Ramesh, Lan Cao, Kannan Mohan and Peng Xu. *Can Distributed Software Development Be Agile?* Communications of the ACM, vol. 49, no. 10, pages 41–46, October 2006. 150

[Rao 1994] Ramana Rao and Stuart K. Card. *The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information*. Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence, pages 318–322, 1994. 52

[Reiss 2005] Frederick Reiss and Joseph M. Hellerstein. *Data Triage: an adaptive architecture for load shedding in TelegraphCQ*. In Proceedings of the 21st International Conference on Data Engineering, 2005. ICDE 2005., pages 155–156, 2005. 62

[Reiss 2009] Steven P. Reiss. *Visualizing the Java heap to detect memory problems*. In 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009., pages 73–80, 2009. 305

[Reiss 2010] Steven P. Reiss and Suman Karumuri. *Visualizing threads, transactions and tasks.* In Proceedings of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, PASTE '10, pages 9–16, New York, NY, USA, 2010. ACM. 305

[Reiss 2013] Steven P. Reiss and Alexander Tarvo. *Automatic categorization and visualization of lock behavior.* In First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pages 1–10, 2013. 305

[Reniers 2012] Dennie Reniers, Lucian Voinea, Ozan Ersoy and Alexandru Telea. *The Solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product.* Science of Computer Programming, 2012. 9, 49, 203, 306, 311, 317

[Rilling 2007] Juergen Rilling, Wen Jun Meng, Fuzhi Chen and Philippe Charland. *Software Visualization - A Process Perspective.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 10–17, 2007. 303

[Rios-Berrios 2012] Miguel Rios-Berrios, Puneet Sharma, Tak Yeon Lee, Rachel Schwartz and Ben Shneiderman. *TreeCovery: Coordinated dual treemap visualization for exploring the Recovery Act.* Government Information Quarterly, vol. 29, no. 2, pages 212 – 222, 2012. 48

[Ripley 2007] Roger M. Ripley, Anita Sarma and Andre van der Hoek. *A Visualization for Software Project Awareness and Evolution.* In Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on, pages 137 –144, june 2007. XV, 167, 169, 306

[Risi 2012] Michele Risi and Giuseppe Scanniello. *MetricAttitude: a visualization tool for the reverse engineering of object oriented software.* In Proceedings of the International Working Conference on Advanced Visual Interfaces, AVI '12, pages 449–456, New York, NY, USA, 2012. ACM. 305

[Roberts 2007] Jonathan C. Roberts. *State of the Art: Coordinated Multiple Views in Exploratory Visualization.* In Coordinated and Multiple Views in Exploratory Visualization, 2007. CMV '07. Fifth International Conference on, pages 61 –71, july 2007. 47

[Robertson 1991] George G. Robertson, Jock D. Mackinlay and Stuart K. Card. *Cone Trees: animated 3D visualizations of hierarchical information.* In CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 189–194, New York, NY, USA, 1991. ACM Press. 9, 47, 52, 57, 311

[Robertson 2010] George G. Robertson, Trishul Chilimbi and Bongshin Lee. *AllocRay: memory allocation visualization for unmanaged languages.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 43–52, New York, NY, USA, 2010. ACM. 305

[Robillard 2003] Martin P. Robillard and Gail C. Murphy. *Static Analysis to Support the Evolution of Exception Structure in Object-oriented Systems.* ACM Transactions on Software Engineering and Methodology, vol. 12, no. 2, pages 191–221, April 2003. 43, 44

[Rodríguez 2004] Oscar M. Rodríguez, Ana I. Martínez, Aurora Vizcaíno, Jesús Favela and Mario Piattini. *Identifying Knowledge Management Needs in Software Maintenance Groups: A Qualitative Approach.* In Proceedings of the Fifth Mexican International Conference in Computer Science, ENC '04, pages 72–79, Washington, DC, USA, 2004. IEEE Computer Society. 38

[Rook 1986] Paul Rook. *Controlling software projects.* Software Engineering Journal, vol. 1, no. 1, pages 7–, January 1986. 152, 153, 154

[Rosen 2013] Paul Rosen. *A Visual Approach to Investigating Shared and Global Memory Behavior of CUDA Kernels.* Computer Graphics Forum, vol. 32, no. 3pt2, pages 161–170, 2013. 305

[Roth 2012] Robert E. Roth. *Cartographic Interaction Primitives: Framework and Synthesis.* The Cartographic Journal, vol. 49, no. 4, pages 376–395, 2012. 49

[Roy 2009] Chanchal K. Roy, James R. Cordy and Rainer Koschke. *Comparison and evaluation of code clone detection techniques and tools: A qualitative approach.* Journal Science of Computer Programming, vol. 74, no. 7, pages 470 – 495, 2009. Special Issue on Program Comprehension (ICPC 2008). 44

[Royce 1970] Walker W. Royce. *Managing the development of large software systems: concepts and techniques.* Proceedings of the IEEE WESTCON, Los Angeles, pages 1–9, August 1970. 20, 21

[Royce 2009] Walker Royce. *Improving Software Economics: Top 10 Principles of Achieving Agility at Scale*, May 2009. 4, 309

[Ruan 2010] Haowei Ruan, Craig Anslow, Stuart Marshall and James Noble. *Exploring the inventor's paradox: applying jigsaw to software visualization.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 83–92, New York, NY, USA, 2010. ACM. 303

[Rubin 2008] Jeffrey Rubin and Dana Chisnell. Handbook of usability testing: How to plan, design, and conduct effective tests. Wiley Publishing, 2 édition, 2008. 270, 271

[Rufiange 2012] Sébastien Rufiange, Michael J. McGuffin and Christopher P. Fuhrman. *TreeMatrix: A Hybrid Visualization of Compound Graphs.* Computer Graphics Forum, vol. 31, no. 1, pages 89–101, 2012. 304

[Sack 2006] Warren Sack, Francoise Détienne, Nicolas Ducheneaut, Jean-Marie Burkhardt, Dilan Mahendran and Flore Barcellini. *A Methodological Framework for Socio-Cognitive Analyses of Collaborative Design of Open Source Software.* Journal of Computer Supported Cooperative Work (CSCW), vol. 15, no. 2-3, pages 229–250, 2006. 43

[Salas 1995] Eduardo Salas, Carolyn Prince, David P. Baker and Lisa Shrestha. *Situation Awareness in Team Performance: Implications for Measurement and Training.* Human Factors: The Journal of the Human Factors and Ergonomics Society, vol. 37, no. 1, pages 123–136, 1995. 159, 160

[Saldaña-Ramos 2014] Javier Saldaña-Ramos, Javier García Ana Sanz-Esteban and Antonio Amescua. *Skills and abilities for working in a global software development team: a competence model.* Journal of Software: Evolution and Process, vol. 26, no. 3, pages 329–338, 2014. 25, 147

[Salmon 2013] Paul M. Salmon and Neville A. Stanton. *Situation awareness and safety: Contribution or confusion? Situation awareness and safety editorial.* Safety Science, vol. 56, no. 0, pages 1 – 5, 2013. Situation Awareness and Safety. 148, 157

[Sangal 2005] Neeraj Sangal, Ev Jordan, Vineet Sinha and Daniel Jackson. *Using Dependency Models to Manage Complex Software Architecture.* In Proceedings of the 20th Annual ACM SIGPLAN Conference on

Object-oriented Programming, Systems, Languages, and Applications, OOPSLA '05, pages 167–176, New York, NY, USA, 2005. ACM. XIII, 121, 122, 123, 144

[Santamaría 2009] Rodrigo Santamaría and Roberto Therón. *Treevolution: visual analysis of phylogenetic trees.* Bioinformatics, vol. 25, no. 15, pages 1970–1971, August 2009. 14, 48, 55, 58

[Santamaría 2014] Rodrigo Santamaría, Roberto Therón and Luis Quintales. *BicOverlapper 2.0: visual analysis for gene expression.* Bioinformatics, vol. 30, no. 12, pages 1785–1786, 2014. 14, 48

[Sarewitz 2008] Daniel Sarewitz and Richard R. Nelson. *Progress in Know-How: Its Origins and Limits.* Innovations: Technology, Governance, Globalization, vol. 3, no. 1, pages 101–117, January 2008. 25

[Savikhin 2008] Anya Savikhin, Ross Maciejewski and David S. Ebert. *Applied visual analytics for economic decision-making.* In IEEE Symposium on Visual Analytics Science and Technology, 2008. VAST '08., pages 107 –114, oct. 2008. 9, 10, 311, 312

[Sawant 2007] Amit P. Sawant and Naveen Balit. *DiffArchViz: A Tool to Visualize Correspondence Between Multiple Representations of a Software Architecture.* In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 121–128, 2007. 306

[Scacchi 2004] Walt Scacchi. *Socio-Technical Interaction Networks in Free/Open Source Software Development Processes.* In Software Process Modeling, pages 1–27. Springer Science + Business Media Inc, 2004. 43, 44, 90

[Schaeckeler 2009] Stefan Schaeckeler, Weijia Shang and Ruth Davis. *Compiler Optimization Pass Visualization: The Procedural Abstraction Case.* Transactions in Computing Education, vol. 9, no. 2, pages 14:1–14:13, June 2009. 305

[Schatz 2013] Michael C. Schatz, Adam M. Phillippy, Daniel D. Sommer, Arthur L. Delcher, Daniela Puiu, Giuseppe Narzisi, Steven L. Salzberg and Mihai Pop. *Hawkeye and AMOS: visualizing and assessing the quality of genome assemblies.* Briefings in Bioinformatics, vol. 14, no. 2, pages 213–224, 2013. 48

[Schreck 2013] Tobias Schreck and Daniel Keim. *Visual Analysis of Social Media Data*. IEEE Computer, vol. 46, no. 5, pages 68–75, 2013. 49

[Schumann 2011] Heidrun Schumann and Christian Tominski. *Analytical, visual and interactive concepts for geo-visual analytics*. Journal of Visual Languages and Computing, vol. 22, no. 4, pages 257 – 267, 2011. 49

[Sensalire 2008] Mariam Sensalire, Patrick Ogao and Alexandru Telea. *Classifying desirable features of software visualization tools for corrective maintenance*. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 87–90, New York, NY, USA, 2008. ACM. 181, 183, 303

[Sensalire 2009] Mariam Sensalire, Patrick Ogao and Alexandru Telea. *Evaluation of software visualization tools: Lessons learned*. In 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009., pages 19–26, 2009. 303

[Servant 2010] Francisco Servant, James A. Jones and André van der Hoek. *CASI: preventing indirect conflicts through a live visualization*. In Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '10, pages 39–46, New York, NY, USA, 2010. ACM. 38, 306

[Servant 2012] Francisco Servant and James A. Jones. *History slicing: assisting code-evolution tasks*. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12, pages 43:1–43:11, New York, NY, USA, 2012. ACM. 306

[Shah 2008] Hina Shah, Carsten Görg and Mary Jean Harrold. *Visualization of exception handling constructs to support program understanding*. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 19–28, New York, NY, USA, 2008. ACM. 306

[Sharif 2009a] Bonita Sharif and Jonathan I. Maletic. *The effect of layout on the comprehension of UML class diagrams: A controlled experiment*. In 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009., pages 11–18, 2009. 303

[Sharif 2009b] Khaironi Yatim Sharif and Jim Buckley. *Observation of Open Source programmers' information seeking.* In IEEE 17th International Conference on Program Comprehension, 2009. ICPC '09., pages 307 –308, may 2009. 7

[Sharif 2013] Bonita Sharif, Grace Jetty, Jairo Aponte and Esteban Parra. *An empirical study assessing the effect of seeit 3D on comprehension.* In First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pages 1–10, 2013. 303

[Sharp 2011] Helen Sharp, Yvonne Rogers and Jenny Preece. Interaction design: Beyond human-computer interaction. second edition. John Wiley, UK, 2011. 9, 47, 270, 271, 311

[Shaverdian 2012] Anna A. Shaverdian, Hao Zhou, George Michailidis, and Hosagrahar V. Jagadish. *A Graph Algebra for Scalable Visual Analytics.* IEEE Computer Graphics and Applications, vol. 32, no. 4, pages 26–33, 2012. 48, 49

[Sheldon 2002] Frederick T. Sheldon, Kshamta Jerath and Hong Chung. *Metrics for maintainability of class inheritance hierarchies.* Journal of Software Maintenance: Researh and Practice, vol. 14, no. 3, pages 147–160, May 2002. 41

[Shi 2011] Jian Shi, Ying Qiao and Hongan Wang. *Visualizing inference process of a rule engine.* In Proceedings of the 2011 International Symposium Visual Information Communication, VINCI '11, pages 10:1–10:9, New York, NY, USA, 2011. ACM. 305

[Shneiderman 1996] Ben Shneiderman. *The eyes have it: a task by data type taxonomy for information visualizations.* In Visual Languages, 1996. Proceedings., IEEE Symposium on, pages 336 –343, sep 1996. 62, 63, 198

[Sigovan 2013] Carmen Sigovan, Chris W. Muelder and Kwan-Liu Ma. *Visualizing Large-scale Parallel Communication Traces Using a Particle Animation Technique.* Computer Graphics Forum, vol. 32, no. 3pt2, pages 141–150, 2013. 305

[Sillito 2006a] Jonathan Sillito. *Asking and Answering Questions During a Programming Change Task.* PhD thesis, The Faculty of Graduate Studies, Computer Science. The University Of British Columbia, December 2006. 39

[Sillito 2006b] Jonathan Sillito, Gail C. Murphy and Kris De Volder. *Questions programmers ask during software evolution tasks.* In Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, SIGSOFT '06/FSE-14, pages 23–34, New York, NY, USA, 2006. ACM. 10, 312

[Sillito 2006c] Jonathan Sillito, Gail C. Murphy and Kris De Volder. *Questions programmers ask during software evolution tasks.* In Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, SIGSOFT '06/FSE-14, pages 23–34, New York, NY, USA, 2006. ACM. 39

[Sillito 2008] Jonathan Sillito, Gail C. Murphy and Kris De Volder. *Asking and Answering Questions during a Programming Change Task.* IEEE Transactions in Software Engineering, vol. 34, no. 4, pages 434–451, 2008. 39, 40

[Sips 2012] Mike Sips, Patrick Kothur, Andrea Unger, Hans-Christian Hege and Doris Dransch. *A Visual Analytics Approach to Multiscale Exploration of Environmental Time Series.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, pages 2899–2907, 2012. 49

[Skype Communications SARL 2013] Skype Communications SARL. *Skype*, 2013. `www.skype.com`. 273

[Skytree 2013] Skytree. *Big data analytic 2013: industry report.* Rapport technique, Skytree, 2013. 64

[Sommerville 2011] Ian Sommerville. Software engineering 9. Pearson Education, 2011. 21

[Spence 1982] Robert Spence and Mark Apperley. *Data base navigation: an office environment for the professional.* Behaviour & Information Technology, vol. 1, no. 1, pages 43–54, 1982. 53

[Spence 2000] Robert Spence. *Information Visualization.* ACM Press, 2000. 51

[STA 2005] *IEEE Standard for Software Configuration Management Plans.* IEEE Std 828-2005 (Revision of IEEE Std 828-1998), pages 1–19, 2005. XII, 8, 27, 28, 31, 32

[STA 2006] *International Standard - 14764-2006 - IEEE ISO/IEC 14764:2006, Standard for Software Engineering - Software Life Cycle Processes - Maintenance.* ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998), pages 1 –46, 2006. 7

[STA 2010] *Systems and software engineering – Vocabulary.* ISO/IEC/IEEE 24765:2010(E), pages 1 –418, 15 2010. 7, 27

[Stanton 2001] Neville A. Stanton, P.R.G Chambers and J Piggott. *Situational awareness and safety.* Safety Science, vol. 39, no. 3, pages 189 – 204, 2001. 157

[Stanton 2006] Neville A. Stanton, Rebecca Stewart, Don Harris, Robert J. Houghton, Chris Baber, Richard McMaster, Paul Salmon, Geoff Hoyle, Guy Walker, Mark S. Young, Mark Linsell, Roy Dymott and Damian Green. *Distributed situation awareness in dynamic systems: theoretical development and application of an ergonomics methodology.* Ergonomics, vol. 49, no. 12-13, pages 1288–1311, 2006. 158

[Stasko 2000] John Stasko and Eugene Zhang. *Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations.* In INFOVIS '00: Proceedings of the IEEE Symposium on Information Vizualization 2000, page 57, Washington, DC, USA, 2000. IEEE Computer Society. 52, 58

[Steinbrückner 2010] Frank Steinbrückner and Claus Lewerentz. *Representing development history in software cities.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 193–202, New York, NY, USA, 2010. ACM. 306

[Steinbrückner 2013] Frank Steinbrückner and Claus Lewerentz. *Understanding software evolution with software cities.* Information Visualization, vol. 12, no. 2, pages 200–216, 04 2013. XIV, 131, 132, 133, 143, 306

[Storey 1998] Margaret-Anne Darragh Storey. *A Cognitive Framework for Describing and Evaluating Software Exploration Tools.* PhD thesis, Burnaby, BC, Canada, Canada, 1998. AAINQ37756. 183, 184

[Streit 2012] Marc Streit, Hans-Jörg Schulz, Alexander Lex, Dieter Schmalstieg and Heidrun Schumann. *Model-Driven Design for the Visual Analysis of Heterogeneous Data.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 6, pages 998 –1010, june 2012. 49

[Sukhoo 2005] Aneerav Sukhoo, Andries Barnard, Mariki M. Eloff, John A. Van der Poll and Mahendrenath Motah. *Accommodating Soft Skills in Software Project Management*. In Issues in Informing Science and Information Technology, 2005. 25

[Sullivan 2001] Kevin J. Sullivan, William G. Griswold, Yuanfang Cai and Ben Hallen. *The Structure and Value of Modularity in Software Design*. SIGSOFT Software Engineering Notes, vol. 26, no. 5, pages 99–108, September 2001. 4, 309

[Sun 2004] Dabo Sun and Ken Wong. *On understanding software tool adoption using perceptual theories*. In Proceedings ACSE 2004: 4th IEEE International Workshop on Adoption-Centric Software Engineering, pages 51–55, 2004. 11, 313

[Sun 2013a] Alexander Sun. *Enabling collaborative decision-making in watershed management using cloud-computing services*. Environmental Modelling and Software, vol. 41, no. 0, pages 93 – 97, 2013. 48

[Sun 2013b] GuoDao Sun, RongHua Liang, FuLi Wu and HuaMin Qu. *A Web-based visual analytics system for real estate data*. Science China Information Sciences, vol. 56, no. 5, pages 1–13, 2013. 49

[Sundararaman 2008] Jaishankar Sundararaman and Godmar Back. *HDPV: interactive, faithful, in-vivo runtime state visualization for C/C++ and Java*. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 47–56, New York, NY, USA, 2008. ACM. 303

[Takatalo 2008] Jari Takatalo, Gote Nyman and Leif Laaksonen. *Components of human experience in virtual environments*. Computers, pages 1–15, January 2008. 49

[Talaei-Khoei 2012] Amir Talaei-Khoei, Pradeep Ray, Nandan Parameshwaran and Lundy Lewis. *A framework for awareness maintenance*. Journal of Network and Computer Applications, vol. 35, no. 1, pages 199 – 210, 2012. 23, 38

[Tao 2012] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang and Sunghun Kim. *How do software engineers understand code changes?: an exploratory study in industry*. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12, pages 51:1–51:11, New York, NY, USA, 2012. ACM. 39, 40

[TeamViewer GmbH 2013] TeamViewer GmbH. *TeamViewer*, 2013. www.teamviewer.com. 273

[Telea 2008] Alexandru Telea and Lucian Voinea. *An interactive reverse engineering environment for large-scale C++ code*. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 67–76, New York, NY, USA, 2008. ACM. 306

[Telea 2009a] Alexandru Telea, Heorhiy Byelas and Lucian Voinea. *A Framework for Reverse Engineering Large C++ Code Bases*. Electronic Notes in Theoretical Computer Science, vol. 233, no. 0, pages 143 – 159, 2009. 306

[Telea 2009b] Alexandru Telea, Hessel Hoogendorp, Ozan Ersoy and Dennie Reniers. *Extraction and visualization of call dependencies for large C/C++ code bases: A comparative study*. In 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009., pages 81–88, 2009. 305

[Telea 2009c] Alexandru Telea and Lucian Voinea. *Case study: Visual analytics in software product assessments*. In 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009., pages 65–72, 2009. 182, 306

[Telea 2010] Alexandru Telea, Lucian Voinea and Ozan Ersoy. *Visual Analytics in Software Maintenance: Challenges and Opportunities*. In D. Keim and J. Kohlhammer, editeurs, Proceedings of the 1st European Symposium on Visual Analytics (EuroVAST). Eurographics, 2010. 203, 317

[Telea 2011] Alexandru Telea and Lucian Voinea. *Visual software analytics for the build optimization of large-scale software systems*. Computational Statistics, vol. 26, no. 4, pages 635–654, March 2011. 49, 196, 203, 303, 310, 317

[Teyseyre 2009] Alfredo R. Teyseyre and Marcelo R. Campo. *An Overview of 3D Software Visualization*. IEEE Transactions on Visualization and Computer Graphics, vol. 15, no. 1, pages 87–105, 2009. 303

[Thakur 2011] Sidharth Thakur and Melissa A. Pasquinelli. *Adapting Visual-Analytical Tools for the Exploration of Structural and Dynamical Features of Polymer Conformations*. Macromolecular Theory and Simulations, vol. 20, no. 4, pages 286–298, 2011. 49

[Thayer 1988] Richard H. Thayer. Software engineering project management, chapitre Software engineering project management: A top-down view, pages 15–53. IEEE Computer Science Press, 1988. 25

[Therón 2006a] Roberto Therón. *Hierarchical-temporal Data Visualization using a Ring Tree Metaphor.* Lecture Notes in Computer Science. Smart Graphics, 2006. 55, 58

[Therón 2006b] Roberto Therón. *Hierarchical-Temporal Data Visualization Using a Tree-Ring Metaphor.* In Andreas Butz, Brian Fisher, Antonio Krüger and Patrick Olivier, editeurs, Smart Graphics, volume 4073 of *Lecture Notes in Computer Science*, pages 70–81. Springer, 2006. 9, 52

[Therón 2006c] Roberto Therón. *Visual Analytics of Paleoceanographic Conditions.* In 2006 IEEE Symposium On Visual Analytics Science And Technology, pages 19 –26, 31 2006-nov. 2 2006. 9, 48, 311

[Therón 2007] Roberto Therón, Antonio González-Torres, Francisco J. García-Peñalvo and Pablo Santos. *The Use of Information Visualization to Support Software Configuration Management.* Lecture Notes in Computer Science, vol. Volume 4663/2007, pages 317–331, 2007. XVII, XVIII, 247, 250, 253, 255, 343, 347

[Therón 2008] Roberto Therón, Antonio González-Torres and Francisco J. García-Peñalvo. *Supporting the understanding of the evolution of software items.* In SoftVis '08: Proceedings of the 2008 ACM symposium on Software visualization, New York, NY, USA, 2008. ACM. XVII, XVIII, 253, 255, 347

[Therón 2010] Roberto Therón and Laura Casares. *Visual Analysis of Time-Motion in Basketball Games.* In Smart Graphics, 10th International Symposium on Smart Graphics, Banff, Canada, June 24-26, 2010, Proceedings, pages 196–207, 2010. 49

[Therón 2013] Roberto Therón and Laura Fontanillo. *Diachronic-information visualization in historical dictionaries.* Information Visualization, 2013. 56

[Thomas 2005] James J. Thomas and Kristin A. Cook. Illuminating the path: Research and development agenda for visual analytics. IEEE-Press, 2005. 9, 11, 47, 50, 311, 313

[Thomas 2006] J.J. Thomas and K.A. Cook. *A Visual Analytics Agenda.* IEEE Computer Graphics and Applications, vol. 26, no. 1, pages 10–13, Jan.-Feb. 2006. 11, 313

[Tidwell 2011] Jenifer Tidwell. Designing interfaces. O'Reilly Media, second edition édition, January 2011. 63

[Tockey 1999] Steve Tockey. *Recommended skills and knowledge for software engineers.* In Proceedings. 12th Conference on Software Engineering Education and Training, 1999., pages 168–176, Mar 1999. 26

[Tomaszewski 2011] Brian Tomaszewski, Justine Blanford, Kevin Ross, Scott Pezanowski and Alan M. MacEachren. *Supporting geographically-aware web document foraging and sensemaking.* Computers, Environment and Urban Systems, vol. 35, no. 3, pages 192 – 207, 2011. 48, 49

[Treiber 2009] Martin Treiber, Hong-Linh Truong and Schahram Dustdar. *Service-Oriented Computing — ICSOC 2008 Workshops.* chapitre On Analyzing Evolutionary Changes of Web Services, pages 284–297. Springer-Verlag, Berlin, Heidelberg, 2009. 44

[Trümper 2010] Jonas Trümper, Johannes Bohnet and Jürgen Döllner. *Understanding complex multithreaded software systems by using trace visualization.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 133–142, New York, NY, USA, 2010. ACM. 305

[Tu 1992] Qiang Tu. On navigation and analysis of software architecture evolution. Master's thesis, University of Waterloo, 1992. 44

[Tufte 1990] Edward Tufte. Envisioning information. Graphics Press, Cheshire, CT, USA, 1990. 51

[Tufte 1997] Edward R. Tufte. Visual explanations: images and quantities, evidence and narrative. Graphics Press, Cheshire, CT, USA, 1997. 51

[Turley 1995] Richard T. Turley and James M. Bieman. *Competencies of Exceptional and Nonexceptional Software Engineers.* Journal of Systems and Software, vol. 28, no. 1, pages 19–38, January 1995. 26

[Tyakht 2012] Alexander V. Tyakht, Anna S. Popenko, Maxim S. Belenikin, Ilya A. Altukhov, Alexander V. Pavlenko, Elena S. Kostryukova, Oksana V. Selezneva amd Andrei K. Larin, Irina Y. Karpova and Dmitry G. Alexeev. *MALINA: a web service for visual analytics of human gut microbiota whole-genome metagenomic reads.* Source Code for Biology and Medicine, vol. 7, no. 1, pages 1–5, 2012. 48

[Valetto 2007] Giuseppe Valetto, Mary Helander, Kate Ehrlich, Sunita Chulani, Mark Wegman and Clay Williams. *Using Software Repositories to Investigate Socio-technical Congruence in Development Projects*. In Proceedings of the Fourth International Workshop on Mining Software Repositories, MSR '07, pages 25–, Washington, DC, USA, 2007. IEEE Computer Society. 90

[van Harmelen 2007] Frank van Harmelen, Vladimir Lifschitz and Bruce Porter, editeurs. Handbook of knowledge representation (foundations of artificial intelligence). Elsevier Science, 2007. 9, 47, 311

[van Wijk 2005] Jarke J. van Wijk. *The Value of Visualization*. Visualization Conference, IEEE, vol. 0, page 11, 2005. 198

[Vanya 2012] Adam Vanya, Rahul Premraj and Hans van Vliet. *Resolving unwanted couplings through interactive exploration of co-evolving software entities-An experience report*. Information and Software Technology, vol. 54, no. 4, pages 347 – 359, 2012. 306

[Vasa 2009] Rajesh Vasa, Markus Lumpe, Philip Branch and Oscar Nierstrasz. *Comparative analysis of evolving software systems using the Gini coefficient*. In IEEE International Conference on Software Maintenance, 2009. ICSM 2009., pages 179–188, Sept 2009. 43, 44

[Vassiliadis 2002] Panos Vassiliadis, Alkis Simitsis and Spiros Skiadopoulos. *Conceptual modeling for ETL processes*. In Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, DOLAP '02, pages 14–21, New York, NY, USA, 2002. ACM. 198

[Vassiliadis 2009] Panos Vassiliadis and Alkis Simitsis. *Near Real Time ETL*. In Stanislaw Kozielski and Robert Wrembel, editeurs, New Trends in Data Warehousing and Data Analysis, volume 3 of *Annals of Information Systems*, pages 1–31. Springer US, 2009. 198

[Vicente 2010] Rodrigo Santamaría Vicente. *Visual analysis of gene expression data by means of biclustering*. PhD thesis, University of Salamanca, 2010. 14, 48

[Viégas 2004] Fernanda B. Viégas, Martin Wattenberg and Kushal Dave. *Studying Cooperation and Conflict Between Authors with History Flow Visualizations*. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '04, pages 575–582, New York, NY, USA, 2004. ACM. 56

[Voinea 2007] Stefan-Lucian Voinea. *Software Evolution Visualization*. PhD thesis, Advanced School for Computing and Imaging, Technische Universiteit Eindhoven, 2007. 196, 310

[von Pilgrim 2008] Jens von Pilgrim and Kristian Duske. *Gef3D: a framework for two-, two-and-a-half-, and three-dimensional graphical editors*. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 95–104, New York, NY, USA, 2008. ACM. 303

[Vrhoveca 2013] Simon L. R. Vrhoveca, Marina Trkmana, Ales Kumera, Marjan Krispera and Damjan Vavpotica. *Outsourcing as an Economic Development Tool in Transition Economies: Scattered Global Software Development*. Information Technology for Development, vol. 0, no. 0, pages 1–15, 2013. 24

[Walny 2011] Jagoda Walny, Jonathan Haber, Marian Dörk, Jonathan Sillito and Sheelagh Carpendale. *Follow that sketch: Lifecycles of diagrams and sketches in software development*. In 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pages 1–8, 2011. 303

[Wang 2010] Xiaoyu Wang, Wenwen Dou, Shen-En Chen, William Ribarsky and Remco Chang. *An Interactive Visual Analytics System for Bridge Management*. Computer Graphics Forum, vol. 29, no. 3, pages 1033–1042, 2010. 48

[Wang 2011] Taowei Wang, Krist Wongsuphasawat, Catherine Plaisant and Ben Shneiderman. *Extracting Insights from Electronic Health Records: Case Studies, a Visual Analytics Process Model, and Design Recommendations*. Journal of Medical Systems, vol. 35, pages 1135–1152, 2011. 10.1007/s10916-011-9718-x. 49

[Wang 2012] Xiaoyu Wang, Dong Jeong, Remco Chang and William Ribarsky. *RiskVA: A visual analytics system for consumer credit risk analysis*. Tsinghua Science and Technology, vol. 17, no. 4, pages 440–451, 2012. 49

[Ware 2004] Colin Ware. Information Visualization, Second Edition: Perception for Design (Interactive Technologies). Morgan Kaufmann, 2 édition, April 2004. 160

[Weaver 2006] Chris Weaver, David Fyfe, Anthony Robinson, Deryck Holdsworth and Donna Peuquet. *Visual Analysis of Historic Hotel*

Visitation Patterns. In 2006 IEEE Symposium On Visual Analytics Science And Technology, pages 35 –42, 31 2006-nov. 2 2006. 9, 311

[Weber 2001] Marc Weber, Marc Alexa and Wolfgang Muller. *Visualizing time-series on spirals*. Information Visualization, 2001. INFOVIS 2001. IEEE Symposium on, pages 7–13, 2001. 55

[Wehrend 1990] Stephen Wehrend and Clayton Lewis. *A problem-oriented classification of visualization techniques*. In Visualization '90. Proceedings of the First IEEE Conference on Visualization, pages 139 –143, 469, oct 1990. 61

[Wei 2012] Jishang Wei, Hongfeng Yu, Ray W. Grout, Jacqueline H. Chen and Kwan-Liu Ma. *Visual Analysis of Particle Behaviors to Understand Combustion Simulations*. IEEE Computer Graphics and Applications, vol. 32, no. 1, pages 22 –33, jan.-feb. 2012. 49

[Weiser 1981] Mark Weiser. *Program slicing*. In ICSE '81: Proceedings of the 5th international conference on Software engineering, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press. 43, 90

[Wettel 2007] Richard Wettel and Michele Lanza. *Visualizing Software Systems as Cities*. In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007., pages 92–99, 2007. XIII, 115, 143, 306

[Wettel 2008a] Richard Wettel and Michele Lanza. *Visual Exploration of Large-Scale System Evolution*. In 15th Working Conference on Reverse Engineering, 2008. WCRE '08., pages 219–228, Oct 2008. XIII, 115, 130, 131, 143

[Wettel 2008b] Richard Wettel and Michele Lanza. *Visually localizing design problems with disharmony maps*. In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 155–164, New York, NY, USA, 2008. ACM. 305

[Wettel 2011] Richard Wettel, Michele Lanza and Romain Robbes. *Software systems as cities: a controlled experiment*. In Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, pages 551–560, New York, NY, USA, 2011. ACM. 306

[Wetzel 2004] Kai Wetzel. *Pebbles: Using Circular Treemaps to visualize disk usage*. SourceForge.net, Setiembre 2004. 56

[Wieringa 2006] Roel Wieringa, Neil Maiden, Nancy Mead and Colette Rolland. *Requirements engineering paper classification and evaluation criteria: a proposal and a discussion.* Requirements Engineering, vol. 11, no. 1, pages 102–107, 2006. 71

[Wilkinson 2005] Leland Wilkinson. The grammar of graphics (statistics and computing). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. 63

[Willems 2010] Niels Willems, Willem Robert van Hageb, Gerben de Vriesc, Jeroen H.M. Janssens and Véronique Malaisé. *An integrated approach for visual analysis of a multisource moving objects knowledge base.* International Journal of Geographical Information Science, vol. 24, no. 10, pages 1543–1558, October 2010. 49

[Witten 2005] Ian H. Witten and Eibe Frank. Data mining: Practical machine learning tools and techniques, second edition (morgan kaufmann series in data management systems). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. 9, 47, 311

[Wu 2010] Yongzheng Wu, Roland H.C. Yap and Felix Halim. *Visualizing windows system traces.* In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 123–132, New York, NY, USA, 2010. ACM. 305

[Xie 2008] Shaohua Xie, Eileen Kraemer, R. E. K. Stirewalt, Laura K. Dillon and Scott D. Fleming. *Assessing the benefits of synchronization-adorned sequence diagrams: two controlled experiments.* In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 9–18, New York, NY, USA, 2008. ACM. 303

[Xie 2009] Shaohua Xie, Eileen Kraemer, R. E. K. Stirewalt, Laura K. Dillon and Scott D. Fleming. *Design and evaluation of extensions to UML sequence diagrams for modeling multithreaded interactions.* Information Visualization, vol. 8, no. 2, pages 120–136, Summer 2009. 303

[Xu 2009] Shaochun Xu, Xuhui Chen and Dapeng Liu. *Classifying software visualization tools using the Bloom's taxonomy of cognitive domain.* In Canadian Conference on Electrical and Computer Engineering, 2009. CCECE '09., pages 13–18, 2009. 39, 303

[Yang 2003] Jing Yang, Matthew O. Ward, Elke A. Rundensteiner and Anilkumar Patro. *InterRing: a visual interface for navigating and manipulating hierarchies.* Information Visualization, vol. 2, no. 1, pages 16–30, 2003. 52, 58

[Yang 2005] Hong Yul Yang, Ewan Tempero and Rebecca Berrigan. *Detecting indirect coupling.* In Proceedings of the Australian Software Engineering Conference, ASWEC 2005., pages 212–221, 2005. 44

[Yang 2007] Hong Yul Yang and E. Tempero. *Measuring the Strength of Indirect Coupling.* In Proceedings of the Australian Software Engineering Conference, 2007. ASWEC 2007., pages 319–328, 2007. 44, 89

[Yang 2013] Jing Yang, Yujie Liu, Xin Zhang, Xiaoru Yuan, Ye Zhao, Scott Barlowe and Shixia Liu. *PIWI: Visually Exploring Graphs Based on Their Community Structure.* IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 6, pages 1034–1047, 2013. 48

[Young 1998] Peter Young and Malcolm Munro. *Visualising software in virtual reality.* In Proceedings., 6th International Workshop on Program Comprehension, 1998. IWPC '98., pages 19–26, Jun 1998. 161, 162

[Yu 2004] Liguo Yu, Stephen R. Schach, Kai Chen and Jeff Offutt. *Categorization of Common Coupling and Its Application to the Maintainability of the Linux Kernel.* IEEE Transactions in Software Engineering, vol. 30, no. 10, pages 694–706, October 2004. 44

[Yuan 2010] Xiaoru Yuan, He Xiao, Hanqi Guo, Peihong Guo, W. Kendall, Jian Huang and Yongxian Zhang. *Scalable Multi-variate Analytics of Seismic and Satellite-based Observational Data.* IEEE Transactions on Visualization and Computer Graphics, vol. 16, no. 6, pages 1413 –1420, nov.-dec. 2010. 48

[Zeckzer 2008] Dirk Zeckzer, Robert Kalcklösch, Lutz Schröder, Hans Hagen and T. Klein. *Analyzing the reliability of communication between software entities using a 3D visualization of clustered graphs.* In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, pages 37–46, New York, NY, USA, 2008. ACM. 305

[Zhang 2009] Dehua Zhang, E. Duala-Ekoko and L. Hendren. *Impact analysis and visualization toolkit for static crosscutting in AspectJ.* In IEEE

17th International Conference on Program Comprehension, 2009. ICPC '09., pages 60–69, 2009. 306

[Zhang 2012] Leishi Zhang, Andreas Stoffel, Michael Behrisch, Sebastian Mittelstadt, Tobias Schreck, Rene Pompl, Stefan Weber, Holger Last and Daniel Keim. *Visual analytics for the big data era: A comparative review of state-of-the-art commercial systems.* In IEEE Conference on Visual Analytics Science and Technology (VAST), 2012, pages 173–182, 2012. 64

[Zhao 2002] Jianjun Zhao, Hongji Yang, Liming Xiang and Baowen Xu. *Change Impact Analysis to Support Architectural Evolution.* Journal of Software Maintenance and Evolution: Research and Practice, vol. 14, no. 5, pages 317–333, September 2002. 44

[Zhao 2005] Shengdong Zhao, Michael J. McGuffin and Mark H. Chignell. *Elastic hierarchies: combining treemaps and node-link diagrams.* In IEEE Symposium on Information Visualization, 2005. INFOVIS 2005., pages 57–64, Oct 2005. XIII, 119, 120

[Ziegler 2010] H. Ziegler, M. Jenny, T. Gruse and D.A. Keim. *Visual market sector analysis for financial time series data.* In 2010 IEEE Symposium on Visual Analytics Science and Technology (VAST), pages 83 –90, oct. 2010. 9, 311

[Zimmer 2010] Stephan Zimmer and Stephan Diehl. *Visual Amortization Analysis of Recompilation Strategies.* In 14th International Conference Information Visualisation (IV), 2010, pages 509–514, 2010. 305

[Zou 2003] Lijie Zou and Michael W. Godfrey. *Detecting merging and splitting using origin analysis.* In Proceedings Working Conference Reverse Engineering (WCRE), pages 146–154. IEEE Computer Society Press, 2003. 43, 44

List of Acronyms

ACS	Administración de la Configuración de Software . 314, 323, 326, 327, 353, 355, 356
ADA	Advanced Data Analysis 198–200, 202
AES	Análisis de la Evolución de Software 310, 311, 352
AHEV	Analizador de Hechos y Enlazador de Vistas 321, 322
API	Application Programming Interface 230, 336
ASEA	Advanced Software Evolution Analysis Engine . . . 204, 205, 207, 208, 211, 212
AV	Análitica Visual 310–319, 323, 324, 352, 354, 355
AVAC	Abstracciones Visuales y Apoyo a la Coordinación 321, 322
AVAES	Analítica Visual Aplicada a la Evolución de Software XVI, 319, 321, 323, 354–356
AVS	Analítica Visual de Software 311, 318, 319
BI	Business Intelligence 6, 64
CMV	Coordinated and Multiple Views 47, 198
CVS	Concurrent Versioning System 165
DMES	Desarrollo, Mantenimiento y Evolución de Software 352–355
DMS	Desarrollo y Mantenimiento de Software 310, 314
DSA	Distributed Situation Awareness 147, 158, 159
DSM	Dependency Structure Matrix 114, 121, 123, 125, 134
EDS	EBSCO Discovey Service 67, 70
ERP	Enterprise Resource Planning 64
ES	Evolución de Software 313–319, 321, 323, 328, 354, 355
ETL	Extraction, Transformation and Load . . 198, 199, 202, 204, 205, 207, 208, 211, 212, 319, 321, 323, 326, 327
EVCES	Explorador Visual de Conocimiento para la Evolución de Software 319, 321, 323, 328

EVSA	Evolutionary Visual Software Analytics XVI, 76, 198, 203, 204, 207, 212, 286, 290–292, 296
GPS	Global Positioning System 6
GSD	Global Software Development . . . . XV, 24, 25, 38, 146–151, 159, 162, 173, 174, 176, 214, 295, 356
GT	Granular Timeline . XII, XIV, 216, 217, 219–224, 226, 227, 230, 243, 263, 266, 269, 328, 330–334
HCI	Human-Computer Interaction 47, 49, 61, 197, 199, 245
HEB	Hierarchical Edge Bundles 94, 101, 129
HPC	High Performance Computing 50
HTML	HyperText Markup Language 9, 49, 199, 312
IDE	Integrated Development Environment 29, 42, 45, 76, 79, 85, 139, 184–186, 191, 192, 196, 197, 289, 290, 293, 295, 296, 353, 355, 356
IMMV	Interactive Multi-Matrix Visualization . . . IX, 122, 123, 144
IPO	Interacción Persona-Ordenador 312
IV	Information Visualization 9, 47, 48, 51, 53, 62, 67, 178, 181, 198–200, 202, 245, 289, 290
LOC	Lines of Source Code 8, 210, 324
MAAES	Motor de Análisis Avanzado de la Evolución de Software . . 319, 321, 323, 324, 326–328
MSV	Massive Sequence View 129
NOM	Number of Methods 210, 234, 324, 342
PM	Project Manager 24, 25, 37–39, 43, 45, 160
PNL	Parallel Node-Link . . . . . IX, 123, 124, 144
RFID	Radio Frequency Identification 6
RT	Revision Tree . . 214, 216, 217, 234, 245, 246, 250, 251, 253, 255, 256, 258–260, 263, 266, 269, 270, 281, 282, 291, 328, 344, 347, 348, 350, 351, 355
SA	Situation Awareness 157–160
SAW	Situation Awareness Workspace 159, 160, 215, 216

SCM	Software Configuration Management .. VIII, XI, XV, 7, 8, 20, 30, 32–36, 42, 45, 119, 154, 162, 167, 173, 181–184, 186, 187, 190–192, 196, 208, 210, 211, 242, 245, 247, 249, 290, 291, 293, 296
SDM	Software Development and Maintenance .. 4, 5, 11, 14, 21, 147, 195, 196
SDME	Software Development, Maintenance and Evolution... XII, 113, 147, 150, 157, 159–161, 175, 214–216, 272, 285, 286, 289–291
SE	Software Evolution . 7–14, 19, 20, 23, 29, 74, 191, 196–198, 203, 205, 207, 217, 286, 290, 291
SEA	Software Evolution Analysis 5, 8, 9, 20, 30, 42–45, 196, 289
SEV	Software Evolution Visualization.. 107, 110, 196, 204, 205, 207, 290, 296
SHV	System Hotspots View 163, 164
SOFTVIS	ACM Symposium on Software Visualization..... 70
SQA	Software Quality Assurance .. 154, 155, 178, 179, 182, 186, 190, 290
STG	Socio-Technical Graph . XIII, XIV, 216, 217, 238–241, 243, 266, 280, 328, 342–344
SV	Software Visualization 11, 107, 192, 196, 203
TSA	Team Situation Awareness 147, 157
UML	Unified Modeling Language 91, 95
VA	Visual Analytics 4–6, 9–14, 46–51, 62–64, 66, 67, 69, 71, 76–78, 81, 100, 101, 107–110, 178, 181, 191, 196–200, 202–204, 207, 208, 245, 273, 286, 289–291
VACS	Visualization Abstractions and Coordination Support 198–200, 202, 204, 205, 207
VES	Visualización de la Evolución de Software... 311, 319, 321, 322, 353
VI	Visualización de la Información 311, 352, 353
VISSOFT	IEEE International Workshop on Visualizing Software for Understanding and Analysis..... 69, 70

VKE	Visual Knowledge Explorer 198–200
VKESE	Visual Knowledge Explorer for Software Evolution . . . 204, 207, 208, 213, 214, 216, 243, 266, 272, 277, 282, 285, 286
VLFA	Views Linker and Facts Analyzer . . 198–200, 202, 204, 205, 207
VRCS	Visual Revision Control System . . . . . XIII, 246, 248, 250
VS	Visualización de Software 311, 314, 318
VSA	Visual Software Analytics 203, 204
VT3D	Version Tree 3D 246–248, 250, 253, 255, 270
Wi-Fi	Wireless Fidelity 6
XML	eXtensible Markup Language 9, 49, 199, 312