Visualization of Version Control Information

by

Xiaomin Wu
B. Eng., Wuhan University of Hydraulic and Electric Engineering, 1995

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming to the required standard

[redacted]

Dr. Margaret-Anne Storey, Supervisor (Department of Computer Science)

[redacted]

Dr. Hausi A. Müller, Supervisor (Department of Computer Science)

[redacted]

Dr. Daniel M. Germán, Department Member (Department of Computer Science)

[redacted]

Dr. Issa Traoré, External Examiner (Department of Electrical and Computer Engineering)

Supervisor: Dr. Margaret-Anne Storey and Dr. Hausi A. Müller

Abstract

Version control is an important activity related to many phases of the software development lifecycle. Many version control systems have been developed to manage both software version history and associated human activities with the intent of producing higher quality software and supporting collaborative development. However, the vast information these version control systems portray is not well presented, and hence poses a barrier for people to understand and explore the version control information space. This barrier degrades the capability of people to understand the software history and collaboration of team members. In this thesis, we approach this problem by applying visualization techniques to the version control domain. We developed a visualization tool called Xia for the navigation and exploration of software version history and associated human activities. We analyzed information from a widely used version control system, CVS, and interpreted it using visual metaphors. We also propose exploration mechanisms to query the information space. This tool was integrated with a modern integrated development environment Eclipse, and hence increases the possibility of Xia being adopted in the real world and providing us with more feedback. In addition, a preliminary user study was conducted to evaluate both the usability and functionality of this tool. The results of the user study have lead to many ideas about new aspects of the research questions, directions for future exploration, and improvements of the tool.

Examiners:

[redacted]

Dr. Margaret-Anne Storey, Supervisor (Department of Computer Science)

[redacted]

Dr. Hausi A. Müller, Supervisor (Department of Computer Science)

[redacted]

Dr. Daniel M. Germán, Department Member (Department of Computer Science)

[redacted]

Dr. Issa Traoré, External Examiner (Department of Electrical and Computer Engineering)

Table of Contents

List of Tables

List of Figures

Acknowledgments

I am very grateful to both of my supervisors, Dr. Margaret-Anne Storey and Dr. Hausi Müller, for their immense help in supporting this research. Furthermore, I wish to express my special thanks to Dr. Storey, who has always warm-heartedly encouraged me to face the difficulties of life.

I would like to thank all my co-workers in the CHISEL group, for bringing ideas and inspiration to this work. In particular Rob Lintern, Jeff Michaud, David Perrin, Yiling Lu, Polly Allan, Tricia d'Entremont, Nasir Rather, Neil Ernst, Victor Chong, and Mechthild Maczewski. I also want to thank Peter Darling, Elizabeth Hargreaves, for providing edits and feedback, and Dr. Daniel Germán, and Dr. Issa Traoré for their generous help.

Lastly, I am grateful to my parents for their endless love and care, and I would like to share this moment of happiness with them.

Dedication

To my parents

Chapter 1 Introduction

1.1 Problem Statement

Version control is an important activity related to many phases of the software development lifecycle. Many version control systems have been developed to manage both software version history and associated human activities with the intent of producing higher quality software and supporting team work. However, the vast information these version control systems portray is not well presented, and hence poses a barrier for people to understand and explore the version control information. This barrier degrades the collaboration of team members and the ability of people to understand the software history.

One of the important aspects of version control systems is to support team work, which has been a predominant style of software development. Usually, a software project is broken down into small components and each of the team members takes care of different components. However, these components are touched or modified by different people in the development and maintenance process. Team workers may have concerns about what happened with a particular file when they

start to work on it. According to the results of a survey we conducted (described in detail in Chapter 3), these concerns can be summarized as follows:

- What happened since I last worked on this project?
- Who made this happen?
- When did it take place?
- Where did it happen?
- Why were these changes made?
- How has a file changed?

Another important aspect of version control systems is to support the management of software versions. People are interested in not only the current version of the project but also old versions of it. For example, through comparison of the current version and an older version, people can understand how the project or a file changes. Also, having a view of project version history is valuable for project management. Project managers can then make further decisions about the direction of structure redesign and software release. Therefore, we consider the "history" problem also a question we need to address.

For brevity, we refer to these questions as "5W+2H", where the 5W's stand for what, who, where, when, why, and the 2H, for how and history. These questions could be related to both the entire project and every single file, presenting different perspectives of team work. Programmers working in a team encounter these questions in their everyday work. We believe the answers to these questions are essential for team work to collaborate efficiently.

By investigating several popular version control tools, we found these questions could mostly be answered by analyzing the repository data of version control systems. The records of software development activities are stored in

repositories, and could be accessed through various interfaces provided by version control systems. However, we discovered that users still find it difficult to understand and explore this information space. For example, a traditional means of browsing or querying data from version control systems is through the command line interface. Command line interfaces incur overhead for humans because users must remember a variety of commands and plain text query results can be difficult to digest. In addition to command line interfaces, modern version control tools also have interfaces that allow people to explore information through a Windows-like GUI [16, 42, 63]. Such interfaces enable most of the core functionality of version control by simple mouse-click and menu use; however, the resulting output is still in plain text and finer querying is not supported (e.g., who is working on which files).

1.2 Objective and Approaches

Regarding the questions posed above, people may ask: is there a better way to help users understand and explore the version control information, find answers to the 5W+2H questions easily and display the answers intuitively? Many studies have shown that visualization techniques could help people understand and explore various information spaces [8]. We hypothesize that visualization techniques could be also applied in the version control domain and help with the comprehension and exploration tasks. The objective of this thesis is to experiment with a new approach to integrate visualization techniques with a version control system to enhance team collaboration and project management by understanding and exploring version control information. A prototype, Xia, was designed and implemented for this purpose.

We chose one of the most popular version control systems, CVS, as the version engine in our experiments, and the Eclipse CVS plug-in as the data retrieving interface. Furthermore, we integrated a visualization tool, called SHriMP, with the Eclipse CVS plug-in to browse version control information in the CVS repositories, and customized SHriMP for the version control domain by implementing additional visual metaphors and querying mechanisms.

Finally, we conducted a user study to test the functionality and usability of Xia. Tasks related to the 5W+2H questions were presented in the study. Positive and negative feedback was collected through observations and questionnaires, leading to ideas for future improvements of the tool and more research questions to be explored.

1.3 Outline of the Thesis

This thesis is organized as follows. Chapter 2 provides background on software configuration management and information visualization, and related works on visualization of version control systems. Chapter 3 describes how we gathered and analyzed requirements for the design of the tool prototype. Chapter 4 describes the details of the implementation of Xia. Chapter 5 introduces an integration of Xia with another tool–Creole, to resolve more problems and summarizes the benefits of the integration. Chapter 6 illustrates our user study of this tool, and describes the interesting results. Finally, Chapter 7 summarizes the contribution of this thesis and proposes future work.

Chapter 2 Background and Related Research

This chapter provides some background that motivated our research and selected related works that gave us inspiration during our research. The first section looks at software configuration management. Then we review some principles of information visualization, and finally review related work.

2.1 Software Configuration Management (SCM)

In the past few decades, there has been considerable research in the domain of Software Configuration Management (SCM). As a result, various SCM environments and tools have been developed. The term Software Configuration Management (SCM) is derived from the term Configuration Management (CM). In this thesis, we use the definition from the IEEE Standard Glossary of Software Engineering Terminology [28] as a standard definition of CM. Configuration management is "a discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements."

Configuration management has been applied to several areas such as hardware, software, and firmware; however, as the concept of CM is mostly used in the software engineering discipline, the term SCM and CM are used synonymously.

2.1.1 Concepts

The concepts of SCM are usually tightly associated with the SCM functionalities and activities. Different SCM systems implement variations of the concepts described in [26]. SCM activities are traditionally grouped into four categories: configuration identification; configuration control; status accounting; and configuration audits and reviews. These four functions are described in IEEE Standard 828-1990 as follows:

- Identification: identify, name, and describe the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled for the project.
- Control: request, evaluate, approve or disapprove, and implement changes.
- Status accounting: record and report the status of project configuration items (i.e., initial approved version, status of requested changes, and implementation status of approved changes).
- Audits and reviews: determine to what extent the actual configuration item reflects the required physical and functional characteristics.

With the growth and development of various SCM tools, the Software Engineering Institute (SEI) at Carnegie Mellon University reviewed the existing SCM tools and evolved the definition of SCM by adding three managerial functions [13]. They are:

- Manufacturing: managing the construction and building of the product in an optimal manner.
- Process management: ensuring the carrying out of the organization's procedures, policies and lifecycle model.
- Team work: controlling the work and interactions between multiple users on a product.

These three functions supplement the IEEE standard definition to make the definition of SCM more comprehensive and complete. They also conform to the acknowledged understanding of SCM, which says SCM is not only a process but also a management discipline. However, there are no fixed rules for implementing SCM functionalities. Most of the existing SCM systems implement only part of the full functionality based on the definition given above, and some of them implement a new combination of functionality which is unique to their specific tasks. Since the purpose of SCM is to make good software and there is not a universal standard of what constitutes an SCM system, all systems that provide partial functionality but intend to provide a full functioned SCM to achieve high quality software products are considered SCM systems by the software engineering community [13].

Despite the variation of the SCM implementations, several basic concepts have been adopted into the design and implementation of all SCM systems. These basic concepts include configuration item, version, release, and baseline.

- A software configuration item (SCI) is an aggregation of software that is designated for configuration management and is treated as a single entity in the SCM process [24]. Typical SCIs include specifications, source code segments (e.g., source files), code libraries, documentations, and user or developer manuals.

- Version: A generic term used to describe an initial release or re-release of a software configuration item [27]. Different version control systems have different interpretations of this term. For instance, in CVS, *version* corresponds to the project level while *revision* corresponds to the file level; project versions are produced by *tagging* a group of file revisions in particular moments.
- Release: The formal notification and distribution of an approved version [25].
- Baseline: A software baseline is a set of software items formally designated and fixed at a specific time during the software life cycle [48]. We can also say that a baseline is a specific version used as a reference point for further development [11]. However, a baseline is different from a version. Typically, a baseline represents a conceptual milestone in the software development process.

2.1.2 Version Control System (VCS)

Version control is an important function of SCM. As described above, a system which partially implements the standard SCM model is considered an SCM system; therefore a version control system is considered an SCM system. Version control involves the storage of configuration items, the storage of changes, the management of changes, and a more important function, the facilitation and management of collaborative software development, which enables multiple team workers to work on the same set of source code, commit their changes, and communicate with other team members.

2.1.2.1 Terms

There are some common terms that are shared by various version control systems independent of the basic concepts of SCM described in Section 2.1.1.

- Revision: a committed change in the history of a configuration item. There is a fine line between the definition of *revision* and *version*, and people tend to confuse them. In fact, different version control tools use the terminology in slightly different ways, as we mentioned in Section 2.1.1. In CVS, a revision of a file could belong to more than one project versions, since a revision of a file could be "tagged" multiple times.
- Repository: a centralized library of files that are under version control. The notion of a repository was initially proposed with the Revision Control System (RCS) which was developed by Walter F. Tichy [56]. All the CM information about files and the contents of the files are kept in the repository.
- Workspace: the space within which programmers perform the development tasks.
- Working copy: the copy of the repository files in the user's workspace that can be changed by the user.

2.1.2.2 Common Activities

The purpose of a version control tool is to support the SCM functionality. Although different version control systems implement different version control models, there are some common features among them. The following is a summary of activities associated with common Version Control Systems.

- Data Connection. As mentioned above, the notion of a repository is widely used in version control systems. A repository could be a local file system or a

remote server, and provides a version tree to manage different versions of files. For instance, lots of version control tools use client/server architecture, and the clients/users can get any information about any of the files by accessing the repository.

- Check-in/Check-out. Users commit their changes by checking in software artifacts from their personal workspace to the shared repository, and retrieve software artifacts from the repository by checking them out.
- Versioning. After every commitment, the version control tool assigns a new revision number to the changed files, and records the related information of the commitment such as time, user, etc.
- Diff operation. Almost all version control tools provide the mechanism to enable users comparing the differences between two revisions of selected files. An ideal diff operation would enable the comparison between a users' working copy and any revision in the repository, as well as a comparison between any two revisions in the repository. Some version control tools provide only part of this functionality, while some of them provide a more sophisticated mechanism.
- Get operation. This mechanism is provided for users to retrieve any old revision of files or versions of a project. It is used when users find the old version is better than the current one and want to revert to a previous version.
- Report operation. This operation is used to generate various useful reports about software artifacts, such as history of an artifact, annotations, etc.

2.1.2.3 Overview of Existing Version Control Tools

To select a version control system as a supporting tool for our prototype, we investigated several popular version control systems. We found that some of them are well designed and implemented, and have a considerable number of users.

Rational ClearCase

Rational ClearCase [43] uses a unique Unified Change Management (UCM) model for the purpose of configuration and change management. The basic concepts in UCM are *activity* and *artifact*. An *activity* is a piece of work to be accomplished in the project. An *artifact* is an item, usually a file that should be under version control. The version control model of Rational ClearCase versions every artifact in the software development lifecycle. The artifact could be a single file, a directory, a subdirectory, and all kinds of file system objects. This is a unique feature of Rational ClearCase. Most other VCS only support version control on files.

Perforce

Perforce [42] provides a friendly user interface besides the version control mechanisms. In the Perforce Windows client called P4Win, a native Microsoft Windows user interface is provided to display the information related to all SCM tasks. The works in progress are shown at a glance, and a very easy-to-use drag and drop mechanism is provided for users performing the SCM tasks. The Perforce window provides not only a view of the working directory which is like other VCS, but also views of the repository which includes the entire view of the repository and the client view of the repository (See Fig. 2.1). Another advantage of the Perforce window is that it provides a view of task lists by users, which is beneficial to people

who work in a team to acquire peers' working status and communicate with each other.

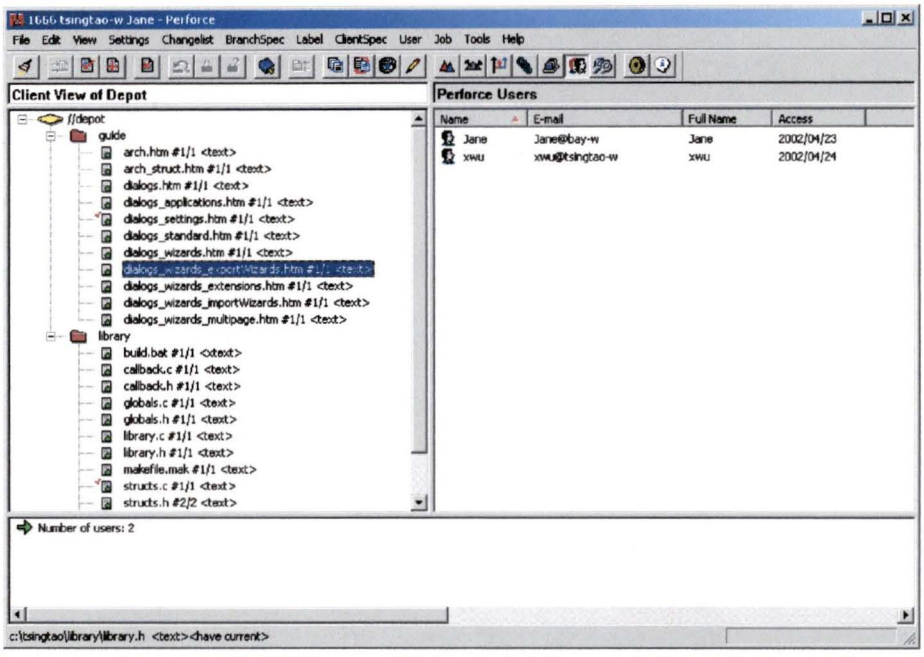


Figure 2.1: A screenshot of Perforce, showing the Client view of the repository and a list of users.

Revision Control System (RCS)

RCS [56] provides simple and basic version control on source files that reside in the repository. The files are stored in the repository in the form of a tree structure. Each file has all of its revisions in an ancestral tree. The root of the tree is the very first revision of this file, and the successive revision is linked to its ancestor which is the previous revision. RCS has its own numbering scheme to assign revision numbers to files. In RCS, only the file deltas are stored in the repository, which means only the differences between revisions are stored. This saves a lot of storage space but increases the access time.

Concurrent Versions System (CVS)

CVS [12] is an open source version control system layered on the top of RCS. That is to say, CVS uses the RCS format for file storage; however, CVS is more sophisticated and supports many more functions. The most outstanding one is that CVS supports a concurrent development environment, while RCS only supports a single user. This function is important nowadays since team programming is a prevalent working style in the software engineering community. However, the advantages of CVS also introduce disadvantages. Because of the powerful functions, more administration is needed for CVS, and the existence of a large number of commands leads to a steep learning curve for beginners. There are also some terms defined specially in CVS. A *tag* is a label assigned to a collection of file revisions in a particular moment of the version history. A project version in CVS is usually determined by a *tag*. Another concept is a *branch.* A CVS branch splits a project's development into separate, parallel histories. Changes made on one branch do not affect the other. In this thesis, we focus our research on software projects that do not use the branching technique in CVS.

By investigating these version control tools, we observed that almost all version control tools have been integrated with other software systems, especially software Integrated Development Environments (IDE). This once again stresses that the role and purpose of SCM is to serve and support the software development process to make good software.

2.2 Information Visualization

The discipline of information visualization has made a great contribution to society by helping with understanding and exploring all kinds of information. Sophisticated techniques have been developed for visualizing massive amounts of data resources.

2.2.1 Basic Concepts and Techniques

What is information visualization? It is "the use of computer-supported, interactive, visual representation of abstract data to amplify cognition" [8]. The definition indicates that the two basic elements in research of information visualization are data models and their visual representations.

2.2.1.1 Data Retrieval

To visualize information, the first stage is to retrieve the data and organize them in a certain form. Mostly, when we are studying a specific information domain, we can get some kind of raw data from the domain. However, in order to abstract more useful information from the data, the raw data needs to be categorized and structured.

Data Types

According to Ware's classification of data [61], real world data can be classified into three categories. They are entities, relationships, and attributes of entities or relationships. Entities are generally the objects of interest [61]. For example, a person, a piece of work, a file, etc. These entities are generally stored in some kind of database. Relationships are the conceptual associations between entities. Attributes

are properties of entities or relationships and derive from entities or relationships. Generally, attributes could be nominal, ordinal, or real-valued data [61]. A nominal attribute usually has a nominal value, e.g. a label, while an ordinal attribute has a numerical value. Real-valued data could be any measurement not falling into the previous two categories. The classification of data types is very valuable when people want to choose a visual metaphor for a specific type of data. Some visual metaphors are not appropriate for some types of data, which means it is hard to use that visual metaphor to interpret the data or the interpretation doesn't make sense. For instance, using size to display nominal attributes hinders users' understanding of the information.

2.2.1.2 Visual Mappings

The next stage is mapping data to visual metaphors. A visual metaphor encodes information with marks and graphical properties. Whether it is a good visual mapping or not depends on how expressive it is and how well it is perceived by people. That is to say, a good visual mapping represents all and only the data that are in the data structure, without any unwanted data appearing; and the mapping should convey more distinctions, or lead to fewer errors [61].

2.2.2 Design of Visual Metaphors

As we know, there can be an unlimited number of ways to characterize an object visually, from a straightforward graph which simply portrays the physical characteristics of the object, to a visual form that also unveils the underlying features of the object. There is no rule or step-by-step guide for designing a visual representation for an object; however, people who are working in this field have

reached an agreement on the criteria that are used to evaluate whether a visual metaphor is good or not [53].

2.2.2.1 Criteria

The criteria for designing a good visual metaphor are tightly tied to the purpose of information visualization. Since the use of information visualization techniques is to help people understand and explore the information space, the criteria of designing a good visual metaphor could be: (1) the visual metaphor should convey as much information as the object has; (2) the visual representation should be expressive enough for human perception; and (3) the visual representation should express the information in a correct way, without misleading the user.

2.2.2.2 Overview of Visual Metaphors in Software Engineering

As we mentioned in Section 2.2.1.1, different types of data may have different appropriate visual metaphors. A visual metaphor may be good for one data type, but not for another one. Since information visualization has evolved over a long time, a rich library of visual metaphors has been accumulated. However, the design space for a visual metaphor is huge since the imagination of the human brain is unlimited. In this section, we review some of the visual metaphors that are related to our work.

Nodes and Arcs

It is generally well accepted that nodes are used to represent entities and arcs are used to represent relationships among entities. In our research, the entities are software artifacts under version control, and the relationships are dependencies among software artifacts.

Color Schemes

Color schemes can be applied to both nominal data and ordinal data. The way of using color schemes could be to apply different colors to different attribute values, or to assign different intensities of a color to different attribute values. An appropriate color scheme is especially useful in highlighting attributes of data.

Size

Size, including length, height, width, and depth, is only suitable to represent numerical data. The use of size to encode numerical data is often appropriate when the designer wants to gain a qualitative impression of the effect of different components and make comparisons [53].

2.3 Visualization of Software under Version Control

Much research has been conducted to study how visualization techniques can help people understand a software system. However, these studies were more focused on a single version of the software while ignoring the software development history and human activities during the development. Recently, the study of visualization for software under version control has attracted people's attention. In this section, several approaches of visualization for software under version control are introduced.

SeeSoft

The SeeSoft software visualization system [3, 18] was developed at AT&T Bell Laboratories to visualize large complex software systems. The tool has been applied to visualize a variety of sources. One of the sources is the version control system which tracks the age of code, the changes of code, programmers, etc. SeeSoft provides a line-level view of the subject software and uses color coding to highlight

specific lines according to the information associated with these lines. For example, lines can be color-coded according to programmers who last modified the code and who might have knowledge about the changes made. Also, SeeSoft provides a color scheme to highlight the code segments to exhibit code decay. Files that have been changed by many programmers and frequently contain bugs are highlighted by multi-colors like a rainbow (See Fig. 2.2).

Figure 2.2: A screenshot of SeeSoft, showing the age of code. The newest lines are in red, and the oldest are in blue, with a rainbow color scale in between.

CVS Activity Viewer

Dourish, at the University of California at Irvine, proposed the CVS Activity Viewer [15]. The graphical design of the CVS Activity Viewer was inspired by SeeSoft. The viewer creates line-by-line visual representations of the code base. Properties such as the programmer working on each line and the rates of change of different parts of the code base are highlighted by different colors. Please see below a screenshot of the tool.

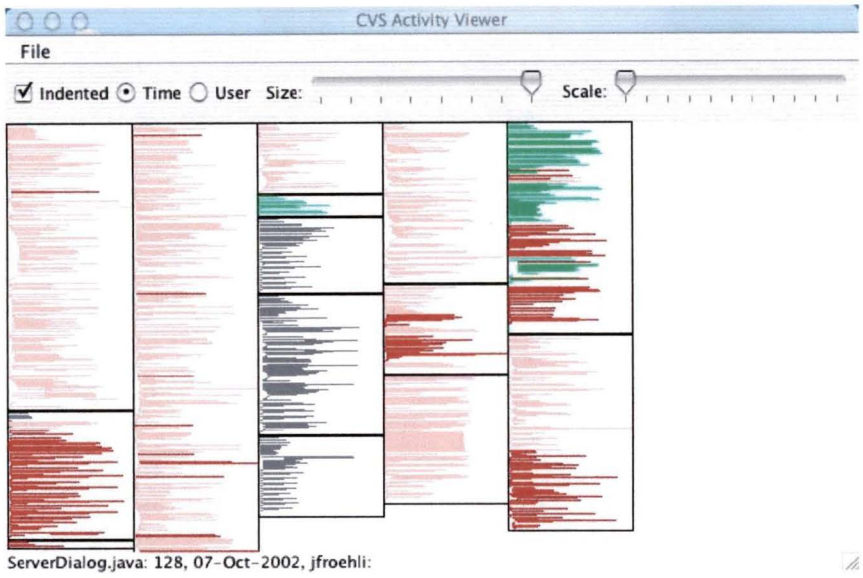


Figure 2.3: A screenshot of CVS Activity Viewer

Bonsai

Bonsai [6] is a web-based query interface to the CVS repository. Through Bonsai, users can perform queries on the contents of a CVS archive; get a list of check-ins, see what check-ins have been made by a given person, or on a given CVS branch, or in a particular time period. Bonsai also includes tools for looking at check-in logs and comments, doing "diff" comparisons between various revisions of a file, and finding out which person is responsible for changing a particular line of code. Figure 2.4 is a screenshot of the Bonsai query interface.

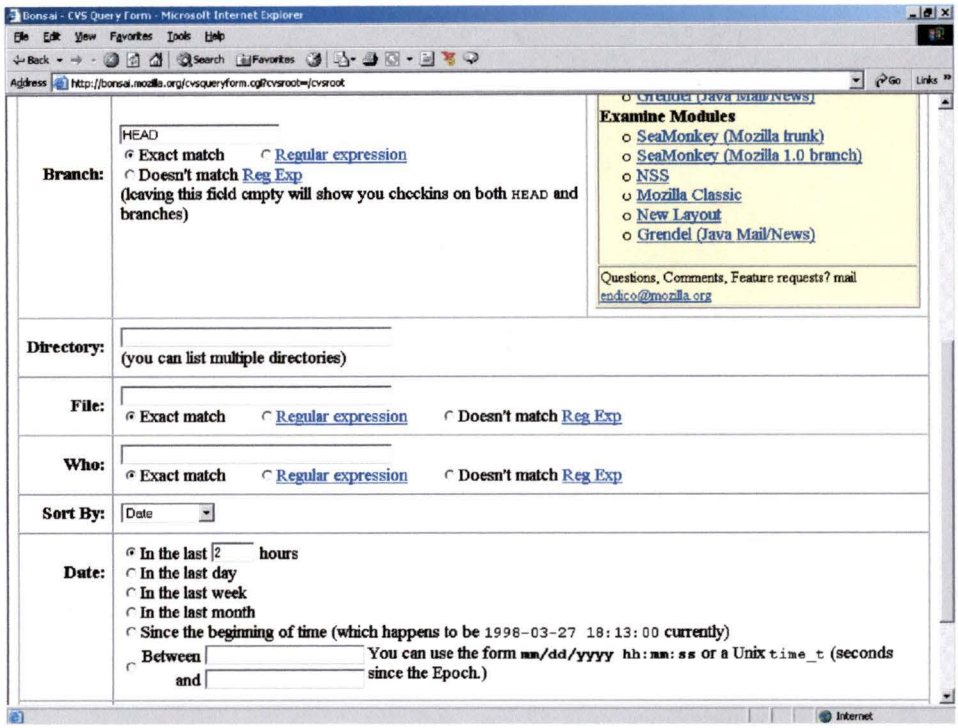


Figure 2.4: Bonsai query interface

Palantir

Palantir [46, 47] is a configuration management workspace awareness tool that provides programmers with insight into other people's workspace, developed in the Department of Information and Computer Science at the University of California, Irvine. Palantir provides a workspace wrapper for each of the programmers workspaces to collect and emit relevant events. All these workspace wrappers are connected with a central event service which provides central management of all the events. Also, each of the programmer's workspaces maintains an internal state to receive and store the events. The events are then extracted and organized by an extractor and shown to a programmer by a visualization component. The graphical visualization provides an overall view of workspace activities. Each software artifact is browsed in a set of cascaded windows that displays the author and event, and can

be sorted by author, event, or severity (amount of changes). Figure 2.4 is a screenshot of Palantir visualization.

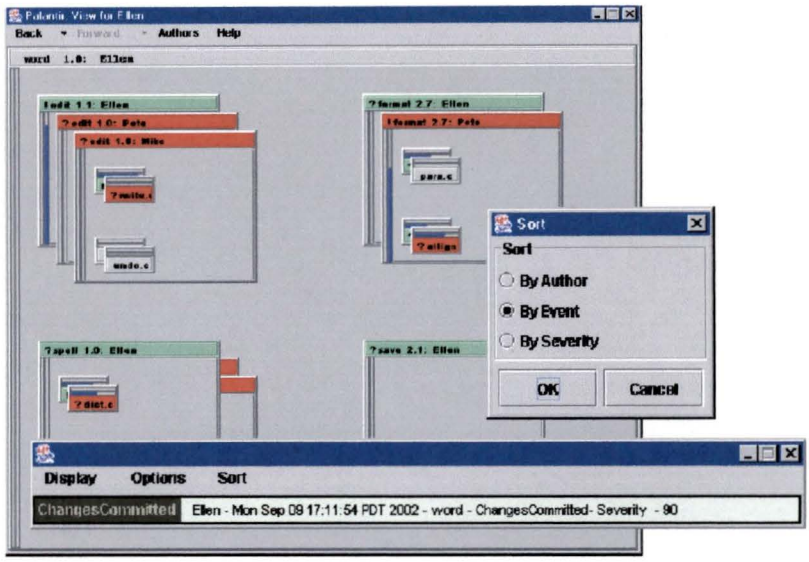


Figure 2.5: Palantir provides an overall view of workspace activities. The artifacts are sorted by event. Green represents the user of current workspace and red for others.

Beagle

Beagle [59] is an integrated environment developed by the Software Architecture Group (SWAG) at the University of Waterloo. Beagle is used to analyze and browse the evolution of software architecture and internal structure of software components. It incorporates different research methods such as evolution metrics, software visualization, and structural evolution analysis tools to allow the user to examine the evolution patterns of software systems from various aspects. Figure 2.5 shows a screenshot of Beagle visualizing the architecture differences between two versions of GCC. The tree structure in the left panel shows the structure of the later version of the program. The graph in the right panel shows the differences between the two versions by color-coding the software entities. For example, red nodes represent entities that are "new" to the current version (the later one); blue nodes

represent entities that were in the old version but deleted from the new version; green nodes are parent-level entities that contain either "new" entities or "deleted" entities.

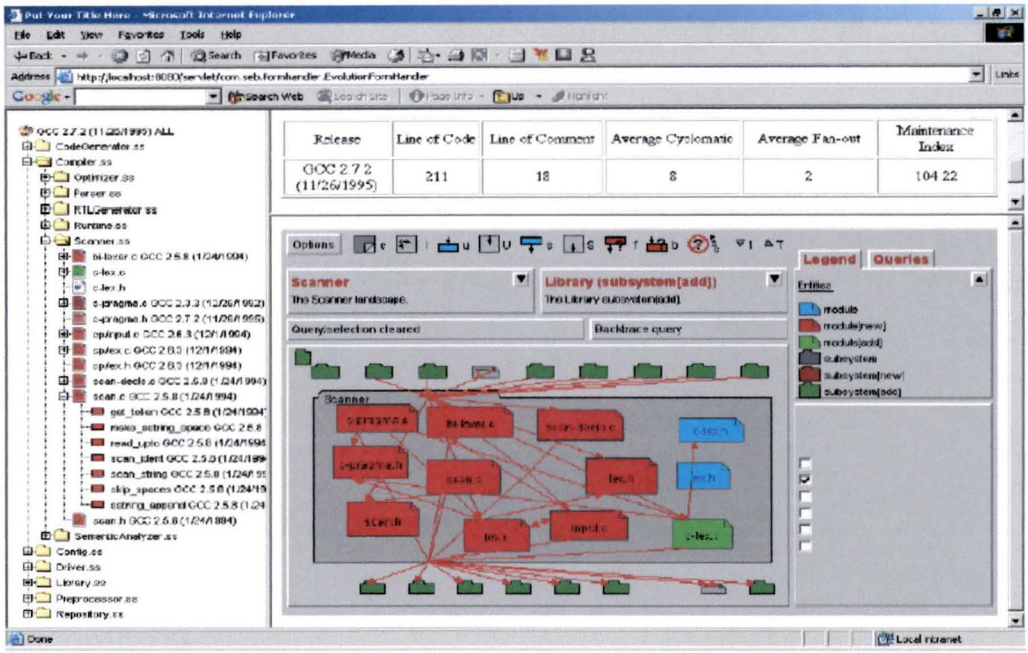


Figure 2.6: A screenshot of Beagle.

Eick's tool set for visualizing software changes

Eick [19] presents a tool set to visualize software changes. In Eick's approach, five visual metaphors were used to browse change-related measurements both independently and in combination. The five metaphors are matrix views, cityscape views, bar and pie charts, data sheets, and network views. The first four metaphors are used to display change measures; changes of the software system are abstracted and analyzed, and visualized through *dimensions* and *measures* associated with the software development process, which include time, change type, software space, developer identification, and other variables. For example, the matrix view could be a 2D grid with rows corresponding to developers and columns corresponding to

software space (e.g. subdirectories in the source code tree containing files with related changes). The network metaphor shows the software structure; nodes represent software units and visual attributes encode measures of association between nodes. For example, nodes are connected by the relationship of relative frequency of files being changed as part of the same modification request. Figure 2.6 shows a screenshot of this approach. All views on the screen are colored by developer. The bar charts index changes by module (top) and developer (bottom) separately. This allows users to identify the programmers who worked on particular modules or to determine where particular programmers worked. The matrix view, in which rows represent developers and columns represent modules, encodes number of changes as the width of the bars in each cell, while the cityscape view encodes the number of changes as the height of the tube.

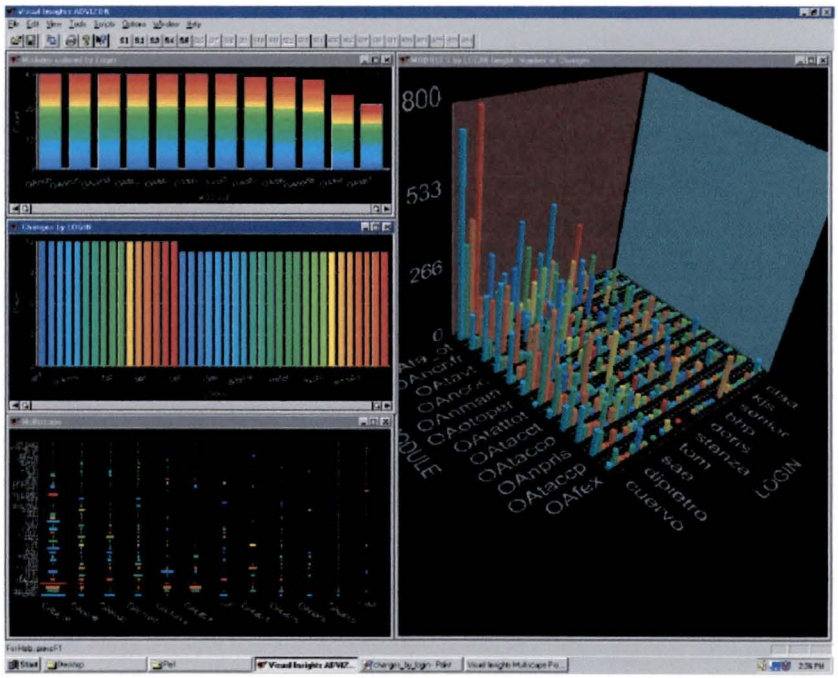


Figure 2.7: Screenshots of Eick's visualization of software changes

GEVOL

GEVOL [10] is a system that visualizes the evolution of software using inheritance, control-flow, and program call-graphs. These graphs are used to display the structure of software programs. GEVOL extracts information about a Java program stored in the CVS version control system, and two attributes of each file, author of each change of each file and the time of each change to each file, are color-coded in these graphs.

Summary of tools

Each of these tools used some kind of visualization to represent data from version control systems and have query mechanisms to explore the data. SeeSoft and the CVS activity viewer used a line-based visualization to display the source code, and color coding to highlight version attributes. However, they don't provide a display of software history, and the line-based view was not evaluated; hence we are unable to tell how successful these visualizations can be. Palantir focuses on bringing awareness of human activities in software development, and provides display for real time data. The display shows details about software artifacts but lacks an overall view of the entire program structure. Beagle focuses more on understanding evolution at the architectural level rather than at the software artifact level, hence it can't provide detailed information of each software artifact. Eick's approach is able to examine extremely large programs, and support statistics on change attributes in the project level. It focuses more on management of software development, but not support for a programmer's daily work. GEVOL also provides relatively high-level views of project changes, and we believe there could be more layouts to display the software structure; also, the attributes of each file taken into account were not adequate, and the details of

changes were not displayed. One of Bonsai's advantages is providing access to CVS repositories through the web; however, the visualization of Bonsai is limited as all data is displayed in a plain text format. Bonsai lacks advanced visualization techniques when compared to other tools. In addition, all these tools don't provide any information about software versions or releases, which is an important issue related to project management; they are not tightly integrated with a full-featured software development environment and hence pose a barrier in getting feedback on the benefits they may provide.

2.4 Summary

This chapter reviews the concepts and principles of software configuration management, information visualization, and research in adopting visualization techniques for software configuration management. Software configuration management has played an important role in the software lifecycle, by generating plenty of information that helps with the understanding of how software is developed. By using visualization techniques, people can gain a better understanding of the software development process, and work more effectively in a collaborative environment.

Chapter 3 Requirements and Design

This chapter discusses the user requirements that guided the design of our prototype and describes our phased design process.

3.1 Requirements from Different Users' Perspectives

We started our work by collecting and analyzing the user requirements. As our purpose of this research is applying visualization techniques to help people understand software version history and improve the efficiency of team work, our work is not only about the software itself but also about the human activities occurring during the software development process. In order to solve this task, we first need to identify the user requirements.

The potential users of this prototype are people working in a team environment. These people play different roles in a team project. Basically they can be divided into two types: project managers and programmers. A project manager and a programmer may have different viewpoints to their project, and their interests to the project are of different levels as well. So it is necessary to collect requirements from both project managers and programmers.

3.1.1 Survey

An informal survey (Appendix A and B) was conducted to collect feedback from a software development group. Six participants answered our survey. Each participant had experience with at least one version control tool, and had experience working on team software projects. Questions in this survey were about what information is of interest to people working on a team project and what levels of views they would like to see. Before setting out the survey, we investigated several version control tools and abstracted their common features. The results of the investigation showed that although these version control tools provide enough features for people to exchange data with each other and work collaboratively, the data generated during the development process are hard to detect and understand because of inefficient visual representations. These data are not well organized and spread out throughout the development environment. To solve this problem, abstraction of the data is needed to generate useful information, and then appropriate visual forms are needed for the information to be displayed and perceived by people. Eight questions were posed in Part A of this survey (see Appendix A), and six questions were posed in Part B (see Appendix B). They are intended to find out the best way to visualize the version history and human activities of software systems.

3.1.2 Results of the Survey

The answers to the survey were very instructive. The participants' points of view can be summarized as follows:

1. The user interfaces of most existing VCS are not satisfactory. There isn't an easy and instant way to let users get their desired information, such as who is working on which part of the project.
2. Project managers are more interested in high-level abstractions and overall views of the software system. Programmers showed their interests at both high-level abstractions of the entire system and their own pieces of work.
3. Different levels of visualization were considered very important to understand the entire system. Programmers want to see different levels of views of the system. For example, a high-level view that contains all the files in the system, names of all the programmers who are working on this system and differences between versions of the entire project are very valuable. Also, they want to go to the details through the high-level view. Continuous navigation, in which the different aspects of the visualization could be switched smoothly, was suggested.
4. People would like to have an overall picture of all the changes of a project.
5. They also want to know the low level details of changes. Typically, a VCS will tell people which file has been changed, and then people can use the "diff" mechanism to compare the file with its latest revision to find out what the changes are. However, people also want to be told which method, not only in the file level, was changed and would like the corresponding method to be highlighted in some way. This requires fine granularity of the configuration management system.
6. Impact of changes is of great concern. People would like to know what parts of the system are affected by a certain change. Visualization can help to bring the awareness of changes to people.

7. They want to know how recently a file has been changed (it could be a way to find dead code) and how frequently a file was changed. The frequency of change and the age of files should be highlighted using some kind of visual metaphors.

3.1.3 Analysis of Requirements

By analyzing the users' feedback, we found that user requirements could be classified into two categories based on the goals of our research. One is for ease of program understanding, and the other is for improving the efficiency of team work. The ease of program understanding is to be achieved by analyzing the historical information of the software system. The improvement of team work efficiency is to be achieved by analyzing information generated by human activities during the software development process. On the basis of these two categories, we summarized the requirements in the following two sections.

3.1.3.1 Requirements for Ease of Program Understanding

The task of understanding a software program involves a reverse engineering process during which the software artifacts are abstracted and visualized. In our approach, the history of the software program is also considered an important aspect of understanding the software program. Therefore, all the historical information needs to be taken into consideration when abstracting the software artifacts.

The requirements for ease of program understanding could be summarized as follows:

1. All the data produced during the software development process should be taken into consideration—not only the latest version of the software.

2. Multiple levels of abstraction are needed to provide different levels of views.
3. Multiple views are needed to provide views from different angles for different users.
4. There should be an easy way for users to retrieve historical data and to make comparisons on them.
5. There should be an easy way for users to view the associated human activities of each software artifact, so that they know whom to consult if there is any understanding problem.
6. The software artifacts in the system should be organized in a structured view for easy navigation.

3.1.3.2 Requirements for Work Efficiency Improvement

A collaborative work environment brings a lot of advantages. However, it also brings disadvantages. One of the disadvantages is the communication barrier among the team workers. Because of the existence of working distance and asynchronization operation on the same set of files, it's hard for people to know each other's work. Even though the current VCSs do provide features to record people's work, these records are not well organized and have to be retrieved by an inconvenient way, e.g. commands with options that are hard to remember. To improve the efficiency of cooperative working, the records of human activities need to be processed and delivered to every person in an easy way. As such, requirements for work efficiency improvement can be summarized as follows:

1. All the information about human activities should be recorded.
2. The information of human activities should be organized in multiple ways to provide multiple views.

3. Human activities should be associated with software artifacts.
4. There should be a way for people to know each other's work (e.g., who is working on which part, and who made what changes).

3.2 Design

We started our design of a version control visualization tool after we gathered and analyzed the user requirements. While designing the prototype of this tool, we underwent two phases: the conceptual phase and the physical phase. The task of the conceptual phase was to define the challenge and the solution concepts. The task of the physical phase was to apply the conceptual design to physical requirements and constraints.

3.2.1 Conceptual Design Phase

During the conceptual design phase, we identified and documented the project vision based on the user requirements. To specify the functionalities of the tool, several usage scenarios were identified as follows:

Usage scenario 1: A user wants to review the whole version history of a project

The version history of a project contains all versions of the project, and each version of the project is considered an artifact. The version history is a high-level view, through which people could have an understanding of how the software evolves. The project manager is the person who has more interest on this level of abstraction since the managerial decision for the direction of the project is always made at this level.

Usage scenario 2: In the version history view, the user found that one of the versions is especially interesting

In this case, navigating from the whole version history view to the particular version is desired. The user doesn't want to lose the context so that she can go back to the overall view of the version history and pick another version to view.

Usage scenario 3: Then, in that particular project version, the user found that she wanted to know more about a file revision in this version

The user may continue her work in scenario 2, navigating from the selected version to a particular file revision in that version, and performing a task on this file revision. For example, she can browse the history or the contents of the file revision. Also, the user doesn't want to lose the context so that she can go back to the project version and pick another file to view.

Usage scenario 4: A project manager wants to know who is working on which part currently so to assign related tasks in the near future. A programmer also wants to know this so she knows who to talk to if there are some related questions.

When the user wants to know who is working on which part, firstly she needs to know who the programmers are. A project manager or a programmer may have this information in her mind when there aren't too many people. However, if there are lots of people involved in this project or the user is new to the group, a list of all the programmers who have been working on the project would be very helpful. With this information, the user can then find out who is working on which part. Two visual metaphors could be used here to answer the question. The first one is color coding, by which the files could be colored according to the programmer who is working on it;

the second one is filter, by which only the files that the selected programmer is working on are kept on the screen while others are filtered out.

Usage Scenario 5: A programmer left for one week's vacation and came back to work to find lots of files have changed

This circumstance is very common in a project team. Things that changed within the past week made it difficult to resume her work. She has to understand what happened while she was absent. In order to find this out, she needs to focus on all the commitments other programmers made during the past week. The solution is filtering the files according to the commitment time. All file revisions that were committed in the time range, a week ago to today, would be highlighted with others being filtered from the view.

Usage Scenario 6: A user wants to know which file was changed the most times and which file was changed the least

This is to answer the question of "which is the most active file in the project and which is the most stable one". Of course the answer could be found out by lots of calculations and comparisons. However, the searching work would be much easier and the result would be more visible by applying visualization techniques. The solution is performing an interactive query and highlighting the results.

Usage Scenario 7: A user wants to know who can help with his/her current work

A programmer is working on a particular file and encounters some problems of understanding a piece of code. After a frustrating investigation of the source code by herself, she still can't resolve the problem. Under this circumstance, communicating with the previous authors directly is always a good solution. In doing this, the programmer may check who has been working on this file, especially who was the last person working on this file. Then she may also read the annotations of

each commitment which may give her some hints on what was changed in that commitment and why they made the changes. As a solution to this problem, a window that contains all this historical information of the file could be associated with the file and brought up easily.

By summarizing the above scenarios, we concluded that the functionality of accessing and retrieving information from the repository would be a fundamental aspect of our project. We also found that a continuous navigation mechanism would be very useful for easy navigation while still keeping the context of the information. In addition, information should be organized at different levels of detail to answer questions at different levels.

3.2.2 Physical Design Phase

To accomplish our goal for this research, we need to build a prototype to test our assumptions. Some research has been done to determine the physical infrastructure of the prototype, during which the programming model and the development language were selected. The detail functionalities that should be supported and implemented to meet user requirements are discussed. Visual metaphors were also selected for different data types in the version control domain.

3.2.2.1 Physical Infrastructure

The basic component we need in our prototype is a VCS tool. Through our investigation on the existing VCS tools, we chose CVS for our prototype. There are three reasons behind this selection: first, CVS is free software; second, CVS provides

robust functionality for version history management; third, CVS is widely used, hence providing us more opportunities to test our prototype.

Another important part of our project is to retrieve the historical information of subject software and records of associated human activities. This could be achieved by accessing and extracting information from the CVS repository. The access to the CVS repository data could be attained through various CVS user interfaces. Since CVS is open source software, lots of contributions from third parties have been made to it, in improving the compatibility of CVS with different operating systems and applications. As a result, various CVS user interfaces are proposed for users to take more advantage of CVS. Examples of the user interfaces are the CVS command line, WinCvs, MacCvs, gCvs, the Eclipse CVS plug-in, and so on. We chose the Eclipse CVS plug-in [36] as the CVS user interface in our prototype.

As to the visualization part, we could do it from scratch, or make use of currently existing tools. There are some research tools that could be used to visualize various information spaces. Rigi [45] is an interactive visualization tool used to help people understand and re-document software. Rigi's extensible structure allows customization through end-user programming. Also, our other research project, SHriMP [52], provides a visualization environment for large complex information spaces. SHriMP has very elegant visualization features and has been integrated with several other tools. The successful integration between SHriMP and other tools, especially IBM's Flow Composition Builder [44] and Creole [34], encouraged us to make use of SHriMP as a visual front-end in our prototype. According to our previous experiences, it is possible to successfully integrate SHriMP with the Eclipse CVS plug-in through the Eclipse platform [17].

In summary, we chose the Eclipse CVS plug-in as the SCM tool, SHriMP as a visual front-end to display and explore the information from CVS, and the Eclipse platform as the underlying framework of the prototype.

It is also advantageous that both SHriMP and Eclipse was implemented in the same programming language, Java. This eases the integration process and allows the programmer to put more focus on improving the program's functionality.

3.2.2.2 Design of Functionality

By analyzing the user requirements and the usage scenarios, we concluded that understanding software structure is important for team workers' development tasks. We feel that the multiple views and layouts supported by SHriMP are very valuable in displaying program structures from different perspectives. Many other approaches, as discussed in Section 2.3, only provide limited graph layouts of software structure, or do not support software structure at all. For example, Eick's approach contains a network view to show the structure of software space: nodes that represent software units are connected by arcs that represent the correlativity of software units being changed together. Eick addressed their unresolved problem of attempting to locate strongly associated nodes near each other, which could be easily solved by using the Spring Layout (to be described in Section 4.3.2) in SHriMP.

We attempted to use the nested graph (to be described in Section 4.3.2) supported by SHriMP to resolve the scalability issue, which is a big concern of many graph layout problems. We feel that some of the approaches discussed in Section 2.3 give a relatively busy screen when trying to present more information to users (SeeSoft, and the CVS activity viewer). However, the nested graph resolves this problem by dividing information into many levels and holds the screen only for one

level a time; also, the screen space becomes unlimited since the lower levels could always be hidden inside higher levels and brought out when needed.

A project version view is considered important for the project management purpose. In this view, software artifacts would be organized by the project versions, or tags in CVS (see Section 4.2.3 for definition). However, none of the approaches described in Section 2.3 support this.

We also summarized from other approaches the two different ways to deal with version information. One is *structure + visual variable* view, the other is stand alone *statistics* view. The *structure + visual variable* view presents the software structure as the main graph; other information related to changes and versions are treated as visual variables of software artifact nodes in the structure view. The statistics view excludes software structures but only shows the statistics on version attributes. We deemed the statistics view should be tightly bound with the software structure view, otherwise users will find it hard to relate the statistics data to their work.

Version or change attributes attached to software artifacts is another important issue to consider. According to our requirements analysis and scenario discussion, we feel that these attributes, along with the software artifact they associated, should be able to answer the 5W+2H questions we posed in Chapter 1. Therefore, we defined a set of CVS attributes associated with each of the file revisions in CVS repositories (see Section 4.2.2.2). Compared to GEVOL, which only has the author and time attributes, we have more attributes considered in our approach and believe future exploration on this could bring up more sophisticated results.

3.2.2.3 Design of Visual Metaphors

As discussed in Chapter 2, appropriate visual metaphors can be helpful in comprehension tasks, while inappropriate visual metaphors might bring overhead to comprehension tasks or even be misleading. In Chapter 2, we also discussed how different visual metaphors match different data types. In our prototype, the data resource is software artifacts and associated human activities in CVS repositories. We started the design of visual metaphors by analyzing the data types we needed to deal with.

In the CVS domain, software artifacts are entities; the version information and associated human activities are treated as attributes of software artifacts. These attributes can be divided into two major types: nominal data type and ordinal data type. Table 3.1 shows the visual variables we planned for different types of data based on the suitability of it for that kind of data.

In Chapter 4, we describe in detail what these attributes are and how we map the visualization techniques to them.

Visualization \ Attribute Type		Nominal	Ordinal
Tooltip		yes	yes
Color Scheme	Distinct Colors	yes	
	Intensity		yes
Filter	Checkbox	yes	
	Double Slider		yes

Table 3.1: The mapping table of attribute types and visualizations

3.3 Summary

We set out a survey in the early stage of our research and gathered some feedback for our preliminary design. The design process consisted of two phases: the conceptual design phase and the physical design phase. During the conceptual design phase, several usage scenarios were created based on user requirements. Expected results from the scenarios made up the design of our prototype. During the physical design phase, technical considerations were taken into account. Research has been done on existing technology candidates. As a result, SHriMP, the Eclipse Platform, and the Eclipse CVS plug-in were chosen as three major components in the prototype. The basic features our prototype should contain were discussed based on the results of the analysis of user requirements and scenario, and the analysis of related approaches. Also, data in the CVS domain were classified into two categories: nominal data and ordinal data, and mapped to appropriate visual metaphors and mechanisms. The next chapter describes details of the implementation of our prototype.

Chapter 4 The Prototype

This chapter illustrates the implementation of our design, a prototype called Xia. Firstly we introduce some supporting tools for the prototype and then we describe its implementation.

4.1 Architecture

In implementing the design, we built a prototype, named Xia, on the basis of three supporting tools: the Eclipse platform, the Eclipse CVS plug-in, and the SHriMP (Simple Hierarchical Multi-Perspective) visualization environment. These three tools were chosen carefully given our technical and practical considerations. They were integrated using APIs which were already provided by these tools. Figure 4.1 shows the high-level architecture of Xia. In Xia, the Eclipse CVS plug-in serves as a data back-end, SHriMP serves as a visualization engine for this back-end data, and the Eclipse Platform provides a framework to enable the integration.

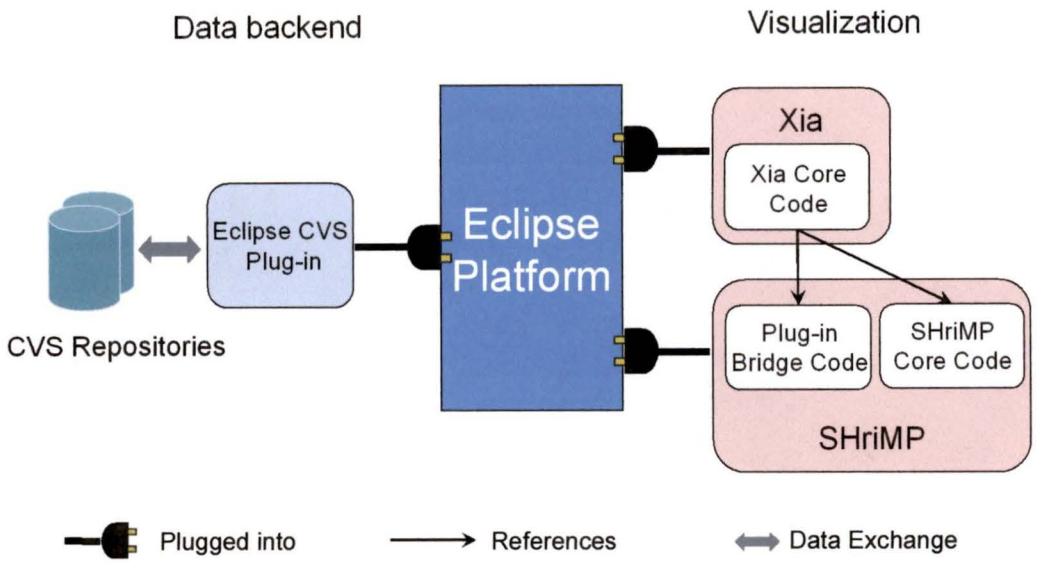


Figure 4.1: The architecture of Xia

The Eclipse Platform

The Eclipse Platform is a subproject of the Eclipse project, which is an open source software development environment. The Eclipse Platform defines core frameworks and services to support software development. It has a unique plug-in based architecture which allows other tool providers to integrate their tools into the platform as a plug-in by following the procedures specified by the platform.

The Eclipse CVS plug-in

CVS has been integrated into the Eclipse Platform by means of a plug-in. This integration is referred to as the Eclipse CVS plug-in in this thesis. The Eclipse CVS plug-in is a reimplementation of the CVS functionalities written in Java by the Eclipse developers. It provides mechanisms to support source file versioning and multiple user development. As a complex tool, the functionality of the CVS plug-in was split across three separate plug-ins: Core, SSH, and UI. The CVS Core plug-in is in charge of the data structure definition and repository management; the CVS SSH plug-in is responsible for the client-server communication; the CVS UI plug-in defines user

interfaces. As the Eclipse CVS plug-in is still in under development, the functionality of CVS is only partially implemented. For example, the "watch" functionality of CVS, a way for programmers to notify each other about who is working on what files at a given time, is not supported so far.

The SHriMP Visualization Tool

SHriMP is a domain-independent information visualization tool developed at the University of Victoria under the direction of Dr. Storey. Advanced visualization techniques are employed in SHriMP to help users browse and explore large and complex information spaces. In SHriMP, artifacts in the domain are represented as nodes on the screen, and relationships or dependencies among artifacts are represented as arcs. An artifact is an entity that carries common characteristics in an information domain. For example, an artifact is a software artifact in the domain of software programs, such as a class. SHriMP provides multiple views to display information at different levels of abstraction. Various layouts are also supported in SHriMP to draw attention to different aspects of the information.

Integration

The architectures of each of the above three tools (the Eclipse Platform, the Eclipse CVS plug-in, and SHriMP) enabled the tools to be easily integrated with each other. The Eclipse platform provides plug-in mechanisms to achieve tool integration, and the Eclipse CVS plug-in is one of these tools. SHriMP has a component based architecture using the Java Bean design [4], which allows tool interoperability via different levels of integration. The above three tools were integrated at both the data and presentation level. At the data integration level, data in the CVS repository was extracted through the Eclipse CVS plug-in and imported into SHriMP via its data bean. At the presentation level, SHriMP was embedded into a common Eclipse view.

In the following sections, we describe in more detail how we performed this integration.

4.2 Data Processing

The data processing phase is mainly concerned with data integration and information abstraction. In the data integration stage, the backend data in the CVS repository is extracted through the CVS plug-in and imported into SHriMP. Once in SHriMP, the data is analyzed to produce useful information for visualization. In this section, we describe how data is stored in a CVS repository, how it is extracted from the repository, how we abstract information from the data, and how we define the data structure in our prototype.

4.2.1 Data Storage in a CVS Repository

As introduced in Section 2.3.1, the configuration item of CVS is a source file. A software program that uses CVS as its version control management tool stores all the historical information of the source files and related human activities in the CVS repository. Developers in the group share their work with each other by committing changes to the repository and updating their local workspaces with changes from the repository. Data storage in the repository adopts the common RCS format in which only file deltas are stored. Source files are organized in the structure of a directory tree, very similar to that of the MS Windows file system.

4.2.2 Data Extraction and Abstraction

As mentioned above, SHriMP implements a Java Bean design model. The functionality of SHriMP is encapsulated within eight distinct components, or "beans". Each of them accomplishes a specific part of the overall functionality. The common elements throughout the system among each of the beans are the artifacts and relationships. The Data Bean is responsible for wrapping an outside data resource, and provides mechanisms for traversing and manipulating the data hierarchy. The artifacts and relationships of the domain are abstracted in the Data Bean and organized in a hierarchical structure. In our case, the outside data resources are the source file revisions and related log messages stored in the CVS repository. Our data manipulation consists of two phases: data extraction and data abstraction. Both of them are implemented in a new CVS Data Bean which extends the abstract Data Bean class in SHriMP. Methods in the CVS Data Bean are customized to make function calls to access the CVS repository through the Eclipse CVS plug-in. Then the extracted data is classified and manipulated to build a structured information space.

4.2.2.1 Data Extraction

An advanced method of data integration provided in SHriMP involves implementing a new Data Bean that is specific to the subject domain. This technique relies on the external program providing methods to access its data [5]. To find out the methods that access the CVS repository data, we made some educated guesses. By investigating the Eclipse API and its plug-in structure, we conjectured that there should be some existing methods that provide access to the CVS repository, and we also conjectured that these methods reside in the CVS Core plug-in. However, unlike other Eclipse plug-ins, the Eclipse CVS Plug-in does not provide us with an official

API, which unfortunately increased our workload in understanding the program. We therefore proposed some reverse engineering techniques to resolve this problem. In this small reverse engineering process, we firstly produced a Javadoc based on the published Eclipse CVS plug-in source code. The Javadoc provided us with easy navigation of the program hierarchy, and quickly helped us find the classes to target. We then focused on these classes, by reading the source code as well as navigating the hierarchy, to find the exact methods we needed to call.

The investigation on the Eclipse CVS plug-in's source code provided us with a clear picture of its data structure. The Eclipse CVS plug-in maintains a tree structure of a software project and treats its folders and file revisions as software artifacts. Folders that contain files are "parents" of these "children" files; the "root parent" is the project. This hierarchical data structure matches the SHriMP data bean very well, as the SHriMP data bean was also designed to maintain a hierarchical structure.

Our next step was to write the CVS Data Bean in SHriMP. In the SHriMP CVS Data Bean, methods in the Eclipse CVS plug-in are called to access the CVS repository. The data we extracted from the repository consists of the directory structure, the file revisions and related log messages. In the next subsection, we describe how we generated artifacts from these data and fit them into the hierarchy.

4.2.2.2 Data Abstraction

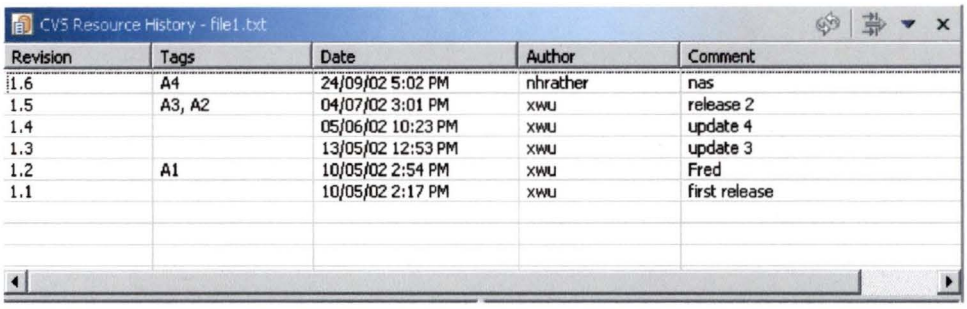
So far we have described how we obtain data from the CVS repository. However, a question remains: Does this data represent all the information we want and is it in the appropriate format for understanding? It is sometimes necessary to analyze and reorganize raw data to elicit useful information. Some calculations may even be required to produce more information. As discussed in Section 2.2.1.1, data

could be classified into three categories: entities, relationships, and attributes of entities or relationships. In our domain, the entities are configuration items (i.e. software artifacts under CVS control). By analyzing the domain data, three types of artifacts were defined: project versions, folders, and file revisions. The relationships among the artifacts were quite simple. Since the data structure that appears in the CVS domain is similar to a directory tree, the only relationship maintained by CVS is a "contains" relationship, which means an element in a higher layer of the tree contains the elements in the lower layer. For instance, a project contains several folders, where folders contain files. The artifacts and relationships in the CVS domain were defined in the CVS Artifact and CVS Relationship classes, which were implementations of the SHriMP Artifact and Relationship interfaces.

Having dealt with the basic data elements: artifacts and relationships, we found that the third type of data, attributes of artifacts, maintains especially useful information for people understanding the project development history and involved human activities. As a result of the data extraction, this attribute information is attached to each software artifact and treated as attributes of the artifacts. As we shall see, proper calculations on this attribute data have the potential to produce meaningful information for visualization, helping to answer the questions posed in this thesis.

The attributes associated with each file revision consist of the log message of each commitment, the size of the file revision (e.g. lines of code), and the history of file revisions which maintains a list of all the log messages. A log message includes the author who made the commitment, the time of the commitment, the revision number, and the comments of the author who made the commitment. This information is retrieved directly from the repository. Figure 4.2 displays a screenshot of the CVS Resource History View in the Eclipse CVS Perspective. The table in the window

shows log messages of a file in the CVS repository. Each row of the table represents a single log message, which contains the revision number, tags, date, author, and comments.


CVS Resource History - file1.txt

Revision	Tags	Date	Author	Comment
1.6	A4	24/09/02 5:02 PM	nhrather	nas
1.5	A3, A2	04/07/02 3:01 PM	xwu	release 2
1.4		05/06/02 10:23 PM	xwu	update 4
1.3		13/05/02 12:53 PM	xwu	update 3
1.2	A1	10/05/02 2:54 PM	xwu	Fred
1.1		10/05/02 2:17 PM	xwu	first release

Figure 4.2: Example: log messages of file1.txt

To improve understanding of the project, some calculations were made to derive meaningful information in addition to the log message. This information constitutes an attribute set associated with each artifact. A list of measurements is summarized as follows, which involves both the raw data and calculated data:

- Revision number of a file revision
- Tags of a file revision
- Time of commitment of a particular file revision
- Author who changed the file most recently. This attribute can be obtained directly from the latest log message.
- Author who changed the file most in a particular time period. This measurement is very important when people want to find out who is a potential source of changes made in that time period.
- Comments associated with each commitment. Comments are annotations made by developers when they commit any changes to the repository. Comments are a good resource of what changes were made and why the author made the changes.

- Number of Changes associated with a particular file revision. This measurement answers the question of *how many times a file has been changed in the development history*. In our approach, we don't deal with *branches* since we want to start with a relatively simple scenario in our research. Therefore, the Number of Changes could be calculated by counting the number of commitments, or log messages, in a file history. When branches are taken into consideration, the calculation would be more complicated.
- Number of Changes associated with a particular file revision during a specific time period. This measurement is useful when the attention on the program is drawn to a specific time period.
- History of a file. This information provides an overview of related activities to a file.

4.2.3 Data Structure

After extracting and abstracting the CVS data, we then defined the data structures to hold this data. Besides inheriting the generic characteristics from the SHriMP interfaces, the CVS Data Bean, the CVS Artifact, and the CVS Relationship also have their own features that are specific to the CVS domain.

The CVS Artifact defines the structure of the artifacts in the CVS domain. We treated each revision of each file in the CVS repository as an artifact. Furthermore, a project under version control is created and maintained as a folder, under which there are subfolders and files that make up the contents of the project. So we also consider these folders, project folder and subfolders, as artifacts since they constitute

indispensable parts of the project hierarchy. In the Eclipse CVS Perspective, the project version structure was displayed in the CVS Repositories View (see Fig. 4.3).

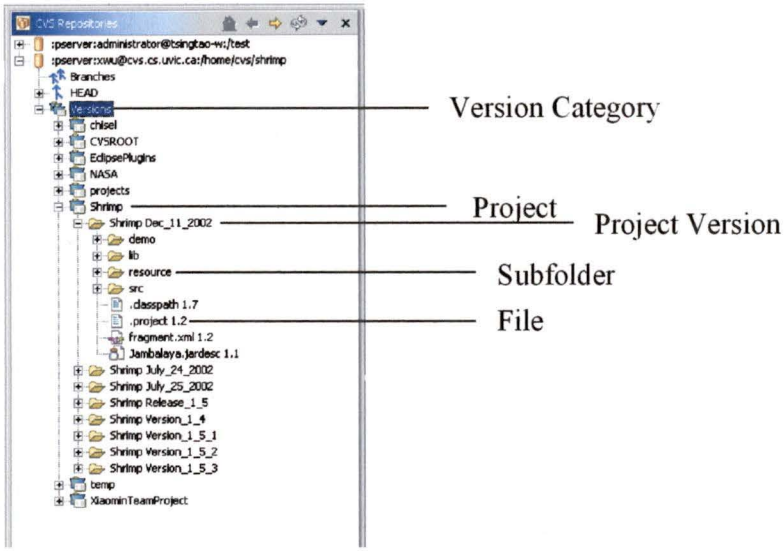
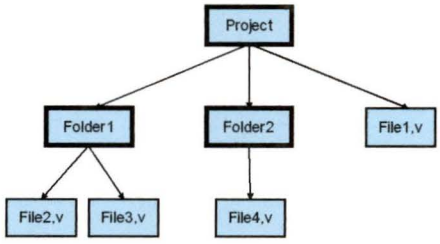


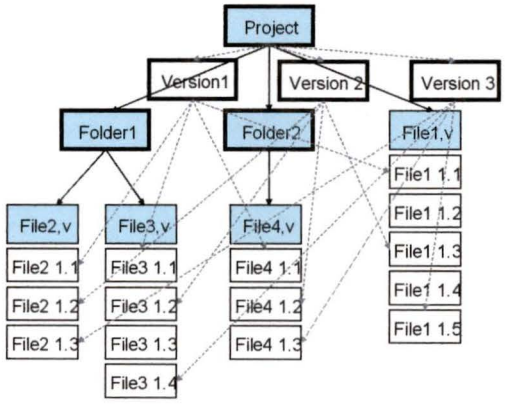
Figure 4.3: A screenshot of the Eclipse CVS repositories view

In this view, we can see that a project that has multiple versions is put under the Version Category folder, and the project preserves a hierarchical structure which is comprised of a project folder, folders for each project version, subfolders and file revisions from top to bottom. However, the data is not stored in the repository in this kind of hierarchy. As mentioned above, each project maps to exactly one folder in the CVS repository, and subfolders and files are put under it. In the repository, all revisions of a file are actually contained in one file which has ",v" appended to the end. This is called the RCS format, in which only the first revision of the file and file deltas are stored. To recreate a particular file revision, file deltas are added to the original revision of the file. Likewise, all the project versions are maintained in the repository in the same and only one folder, the project module. There is not such a "project version" folder in the repository which encapsulates the corresponding file

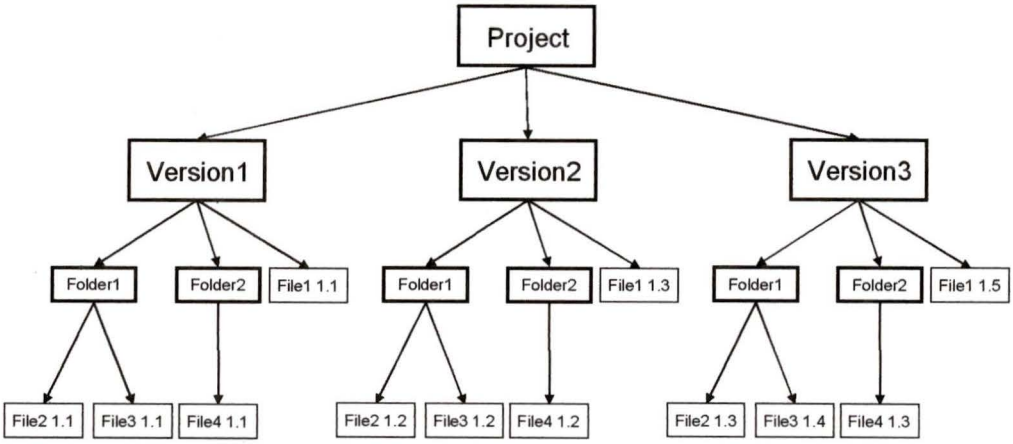
revisions. Figure 4.4 shows the difference between the real repository project hierarchy and an organized hierarchy for display.



(a) Project hierarchy in a CVS Repository



(b) Project versions and file revisions that are "hidden" in the repository



(c) The versioning information is abstracted from the project hierarchy

Figure 4.4: The CVS artifacts hierarchy

To meet our requirements, we mimicked the CVS Repositories View's way of organizing the repository data. As a consequence, we defined the project version, file revision, and folder as artifacts (see Fig. 4.4 c).

To define the structure of the artifacts, we investigated the Eclipse CVS source code to find out their data types. We found that both the project version and folder are of the same type in the Eclipse CVS plug-in, which is RemoteFolder, and the files are of type RemoteFile. As discussed above, different versions of a project physically appear in the CVS repository as the same folder, and different file revisions physically appear in the CVS repository as the same file. However, we need to tell apart different project versions as well as file revisions to support a clear version structure.

To resolve this problem, firstly we introduce the notion of artifact ID in the CVS domain. The artifact ID is used to uniquely identify a CVS artifact. Secondly we assign different IDs to different project versions and file revisions. The artifact ID consists of the network location of the repository in which the project resides, the relative path of the folder or file to the repository, the relative path of the file or folder to its parents, the tag of a folder or file, the revision of a file, and the type of the CVS element which would be a folder or a file.

By doing this, a project version could be identified using a combination of the project folder and the version tag. A *tag* is a label assigned to the collection of revisions in a particular moment of the version history. A version of the project could be announced by tagging the most up-to-date and comparatively mature collection of file revisions. Hence in the CVS ID class, the combination of the network location of the repository, the relative path of the folder to the root of the repository, the relative path of the file or folder to its parents, the tag of the folder and the type of the element (which would be folder) identifies a unique version of the subject project. Again note, these project version artifacts do not physically exist, but were created for the purpose of a clear history hierarchy. Similarly, a unique ID for the file revision artifact is a

combination of the physical file and the revision number, which consists of the repository location, the relative path to the root of the repository, the relative path to the parents, the revision number, and the element type.

In addition to assigning a unique ID for each of the artifacts, attributes attached to each artifact also need to be considered. As mentioned in Section 4.2.2, the log message and derived statistical data of each artifact are treated as attributes of the artifact. These attributes are defined in the CVS Artifact and their values are obtained directly from the repository or derived based on the obtained raw data.

The relationships among the CVS artifacts are quite simple, the same as those among the CVS folders and files. Basically, there is only one type of relationship we need to consider for now. Since the smallest element under the CVS control is a source file, the smallest artifact is the file artifact in our data bean. The relationship among file artifacts and folder artifacts is the "contains" relationship. However, this one relationship is not sufficient for understanding software. A more fine-grained configuration management is required to manipulate on fine-grained configuration items, such as classes, methods, variables, etc. With this information, the relationships among the artifacts could be more sophisticated.

4.3 Visualization

This section introduces some generic SHriMP features, how we achieved the presentation integration, and features of the prototyped visualization.

4.3.1 Presentation Integration

The visualization part of Xia is accomplished by integrating the SHriMP visualization tool with the Eclipse CVS repository view. The whole SHriMP visualization environment was put into an embedded frame inside an Eclipse view, operating on the data produced in the previous data integration process.

In this integration, SHriMP is plugged into the Eclipse Platform by extending two extension points: view and popup menu. Figure 4.5 is a screen shot showing how Xia is activated through the popup menu in the Eclipse CVS repository view. The invocation brings up a new window which contains the SHriMP view.

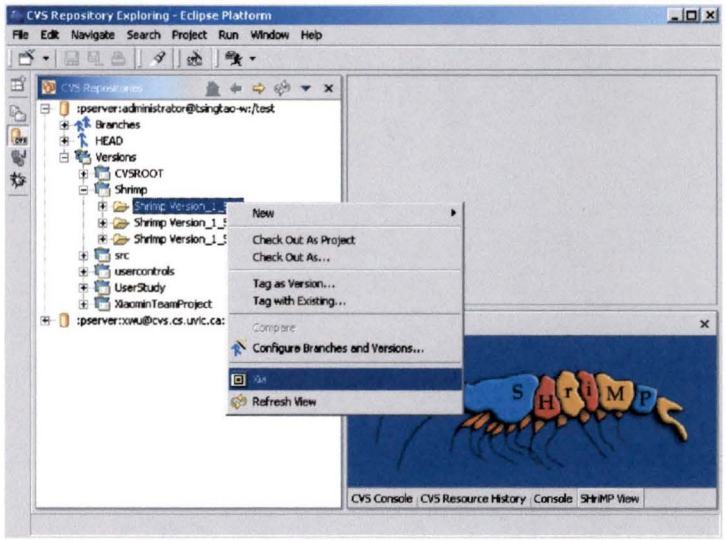


Figure 4.5: Xia popup menu in the Eclipse CVS repository view

4.3.2 Visualization Techniques in SHriMP

SHriMP's component-based framework provides good extensibility and easy integration with other tools. SHriMP has become a sophisticated visualization

environment as more and more visualization techniques are adopted and implemented within it.

Multiple Views

SHriMP provides multiple views at different levels of abstraction to present hierarchically structured information. The **Main SHriMP View** is a nested graph with a zoomable interface. A nested graph is fairly useful for representing the parent-child relationships in a hierarchical structure with efficient space utilization. Combined with zooming techniques, the screen space becomes unlimited to the user, which is extremely beneficial for large information spaces. In the Eclipse CVS plug-in, software in the repository is displayed in a tree-like hierarchical structure of folders and file revisions. This structure corresponds well to nested graph in SHriMP. Xia adopted the nested graph as the main view. In Xia, project version nodes encompass folder nodes that belong to this version, and folder nodes encompass the contained file revision nodes and/or subfolder nodes. The **Hierarchical View** provides an overview of the whole dataset. A default relationship is chosen to show the hierarchy. Users are able to choose the level of hierarchy to display. For example, the user can choose to display only the highest level of the hierarchy because she or he may want to have some abstract understanding of the program; or the user can expand the hierarchy to the very bottom if she or he is also interested in the low level details. The **Thumbnail View** is another way of maintaining the context while navigating. The Thumbnail View is used as a complement to the Main SHriMP View. Users can bring up the Thumbnail View which shows a thumbnail of the entire nested graph with the current working view highlighted in a red rectangle. Operations on this red rectangle simultaneously change the viewpoint of the nested view.

Various Layouts

The Grid layout [64] is the default layout for each node. Nodes are arranged in a rectangular grid within the container, or the parent node. Every node is of equal size. Several options are provided for grid layout to allow the grids to be arranged in certain order. In Xia, versions of a software project are displayed in a grid layout, with each of the versions represented by a node. Contents of the version were contained in this node in a nested graph.

The Spring Layout [64] uses a spring drawing algorithm which clusters highly related node and pushes away more unrelated ones. The Spring Layout is useful at revealing node correlations and decreasing arc crossings.

The Tree Layout [64] is a widely used visual representation for hierarchically structured information. An acyclic graph with multiple layers is produced in this layout. Each node is assigned a layer number. The nodes in the same layer are arranged in a special order to minimize the edge crossings between two consecutive layers.

The Radial Layout [52] is a variant of the tree layout. Layers are represented as circles with the same centre but different radius. Nodes are arranged in corresponding circles according to their layer numbers. The root node is put in the centre of these circles. The radial layout is advantageous at utilizing the screen space.

The Treemap layout uses the Treemap algorithm [50, 51] developed at University of Maryland. Treemap is a 2-d space-filling representation designed for traditional tree structures. Besides the benefit of saving screen spaces, Treemap is also useful in that the size and order of the nodes can be mapped to selected data, which is effective in showing attributes of nodes.

4.3.3 Interactive Exploring

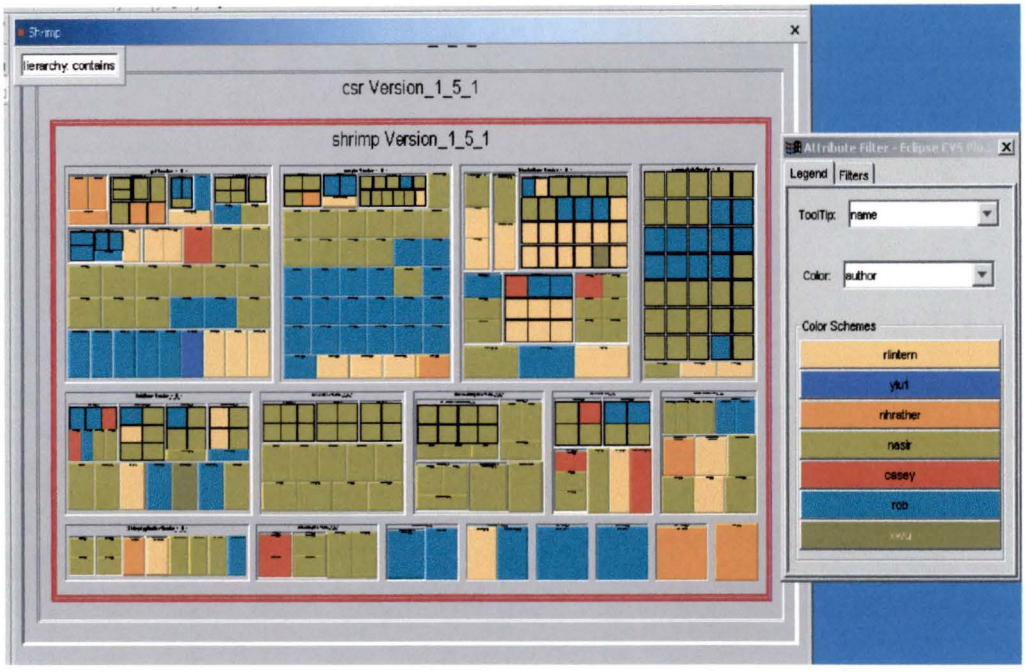
Besides the existing views and layouts provided within SHriMP, we also used several other visual metaphors and interactive techniques for Xia, which included color schemes, tooltips, and dynamic filters. These visual techniques were implemented in an Attribute Panel.

Color Schemes

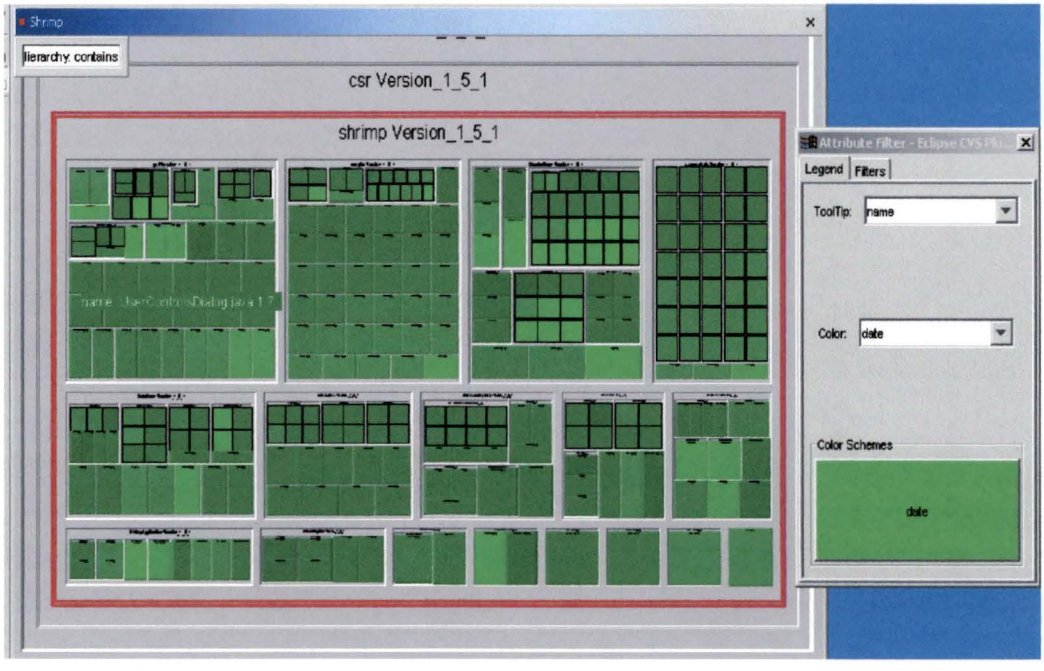
As discussed in Chapter 2, colors could be used for both nominal and ordinal data by different means. In Xia, different colors were assigned to different nominal attribute values. For example, each of the authors can be represented by a color. With ordinal data, we use color intensity to represent an attribute's value. For example, the date of commitment for each file revision is ordinal data. In Xia, green was used as a default color for date, and the intensity of green was employed to represent the value of the date attribute. For example, a more recent date is assigned a brighter green color.

Currently, the attributes of author, change type and node type (node type is SHriMP's default color schemes) are assigned nominal colors, which means each value of the attribute is assigned a distinct color. Also, a color chooser dialog is provided to each of the attributes allowing users to choose their favorite colors.

When assigning a color or intensity to the value of an attribute, the nodes in the SHriMP view change their color according to the attribute values. Figure 4.6 shows two screenshots of coloring the nodes according to their nominal and ordinal attributes.



(a) The color of each node is determined by the author who committed the latest change



(b) The color and intensity of each node is determined by the date of the latest commitment

Figure 4.6: Color schemes for nominal data (a) and ordinal data (b)

Tooltips

Tooltips are especially useful in providing instant information. In SHriMP, tooltips display and disappear instantly with mouse movement. That is to say, there is no delay between the mouse motion and a tooltip's appearance. We adopted this instant tooltip technology in Xia and made extensive utilization of it to display attribute information in a textual format.

In the attribute panel, we developed a combo box for choosing any of the attributes to display in the tooltip. In the CVS domain, the attributes include file name, revision, author, date, changes, size, etc. Once an attribute is chosen, the text of the tooltips show the value of the attribute. Another advantage of using tooltips in Xia, whose domain has many attributes, is that tooltips can be combined with other attribute visualization techniques to provide redundant but useful visualization.

Dynamic Filtering

Filtering is an important technique in information exploration. The obvious benefits of filtering are reducing people's cognitive overhead and putting more focus on useful information. In Xia, two filtering techniques, checkbox filter and double slider filter, are provided for filtering nominal data and ordinal data respectively (see Fig. 4.7).

A checkbox filter is a set of checkboxes automatically generated for each nominal attribute. In a checkbox filter, each of the checkboxes is associated with a value of the nominal attribute. Once a value is checked, the nodes with this attribute value will be unfiltered and will show on the screen. If it is unchecked, nodes with this attribute value will be filtered and disappear from the screen.

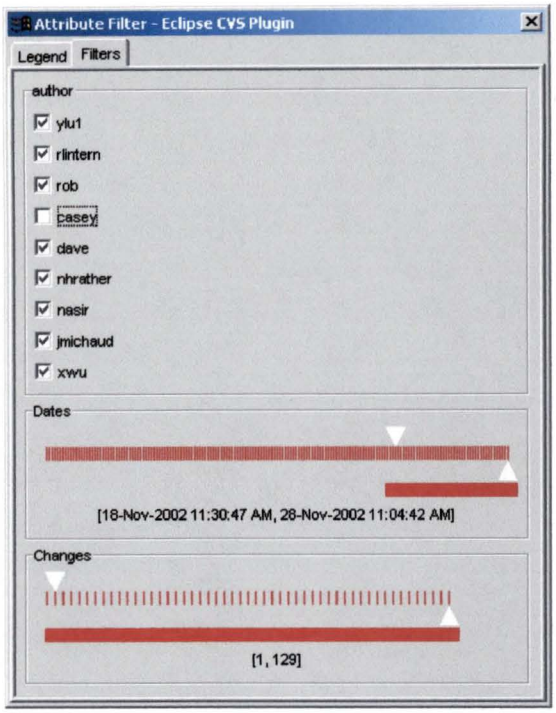


Figure 4.7: Dynamic filters

A double slider filter is an effective dynamic query mechanism that is especially useful for visualizing ordinal attributes. The double slider [49] was originally developed at the University of Maryland, and combined with the Treemaps visualization tool. Unlike the traditional single slider which can only be used to select a single value, a double slider allows the user to select a range of values for query by adjusting the minimum and maximum value of the slider, and can also be used to choose a single value by setting the minimum and maximum value of the slider to the same value. We implemented the double slider in Xia to filter two ordinal attribute values: dates and changes. In Fig. 4.7, two double sliders are created for date and changes. The values of the date attribute and changes attribute in the subject domain have already been extracted from the CVS repository and sorted in order. Each value is represented as a red vertical line, similar to a ruler or scale, and two white triangles are used to adjust the range. The left triangle is used to set the minimum value and the

right one is used to set the maximum value of the range. This adjustment is accomplished by easy mouse dragging operations on these two triangles. The label below the slider shows a textual presentation of the adjusted range, and is updated immediately with the adjustment of the slider. When a range is set, nodes with attribute values outside the range are immediately filtered.

In our approach, attributes of artifacts are visualized and explored by using color schemes, tooltips, and dynamic filters. These techniques can be used individually as we described above or in combination to support more sophisticated information exploration. For example, we can formulize queries such as: What are the files that have been changed more than 10 times within the last month by Fred? To find out the answer, we set the minimum value of the change slider at 10, set the range of the date slider from a month ago to the most recent date, and uncheck all other authors in the author checkbox filter. The nodes showing on the screen represent the answer we seek.

4.3.4 Overview + Detail Visualization

The visualization illustrated above presents an overview of software artifacts in a program. In addition, Xia also provides visualization techniques for the detailed information of each of the artifacts. This information is displayed within an embedded panel in the artifact node, and can be accessed by zooming on the node and selecting corresponding options from the context menu.

support this integration. Our workload in building Xia was greatly reduced by leveraging these existing tools.

The robust visualization techniques of SHriMP were applied in Xia to visualize and explore the software history and associated human activities. Multiple views and various layouts provide different levels of detail of the information. An attribute panel was developed to support interactive exploration and visualization of attributes as well as statistics for each file. In the next chapter, we introduce a tool called Creole, and how we integrate Xia with Creole to gain more benefits.

Chapter 5 An Integration with Creole

As described in previous chapters, information from the CVS repository was visualized by Xia. However, another important information resource, data in a programmer's workspace, has not been dealt with. To resolve this problem, we tried an interesting integration: Xia and Creole [34, 52]. This chapter introduces Creole; then describes our integration process; and finally discusses the benefits of this approach.

5.1 Creole

Creole refers to the integration of SHriMP with the Eclipse JDT (Java Development Tools) plug-in [16]. It is used to browse the software structure of a Java program developed in the Eclipse JDT environment. The architecture of Creole is very similar to that of Xia, in which the only difference is the back-end data resource. Back-end data was provided by the Eclipse JDT plug-in in Creole, instead of the Eclipse CVS plug-in in Xia. Figure 5.1 shows the architecture of Creole.

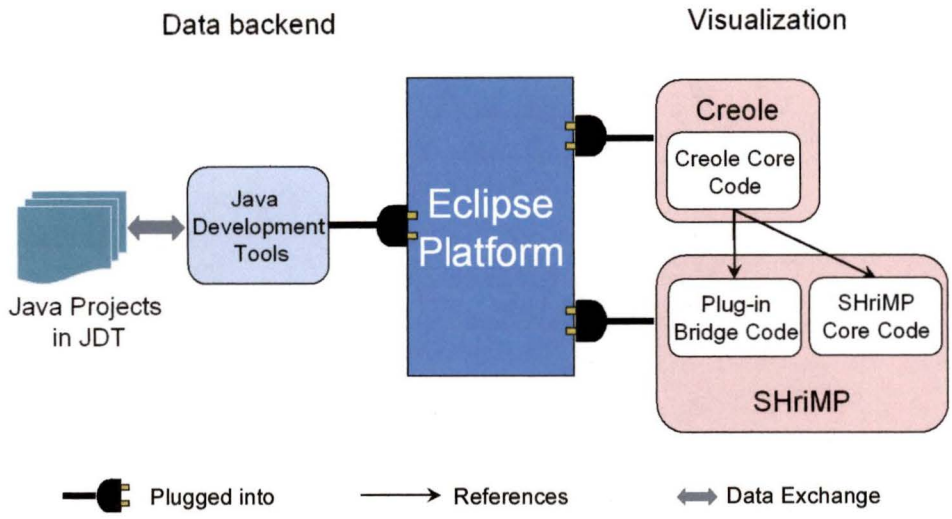


Figure 5.1: The architecture of Creole

As a subproject of Eclipse, JDT contributes a set of plug-ins that provide a fully functional Java Integrated Development Environment (IDE) to the Eclipse Platform. In Creole, software artifacts from Java programs are extracted through the JDT interfaces. A typical Java program is broken down into packages, classes, methods, variables, etc. These components are treated as software artifacts, being represented by nodes in the view, and interactions among these components are treated as relationships, being represented by arcs in the view.

Through Creole, every Java project that uses Eclipse JDT as the development environment could be browsed using SHriMP. Creole adds two views and two actions to the Eclipse platform. The views are SHriMP Main View and SHriMP Hierarchical View. The two respective actions are "Navigate to In SHriMP Main View" and "Highlight in SHriMP Hierarchical View". These actions can be invoked from the context menu of any Java project, from the Package Explorer's toolbar and menu, and from the Java Editor's context menu by highlighting and right-clicking on any valid Java identifier in the editor. Figure 5.2 shows a screenshot of Creole.

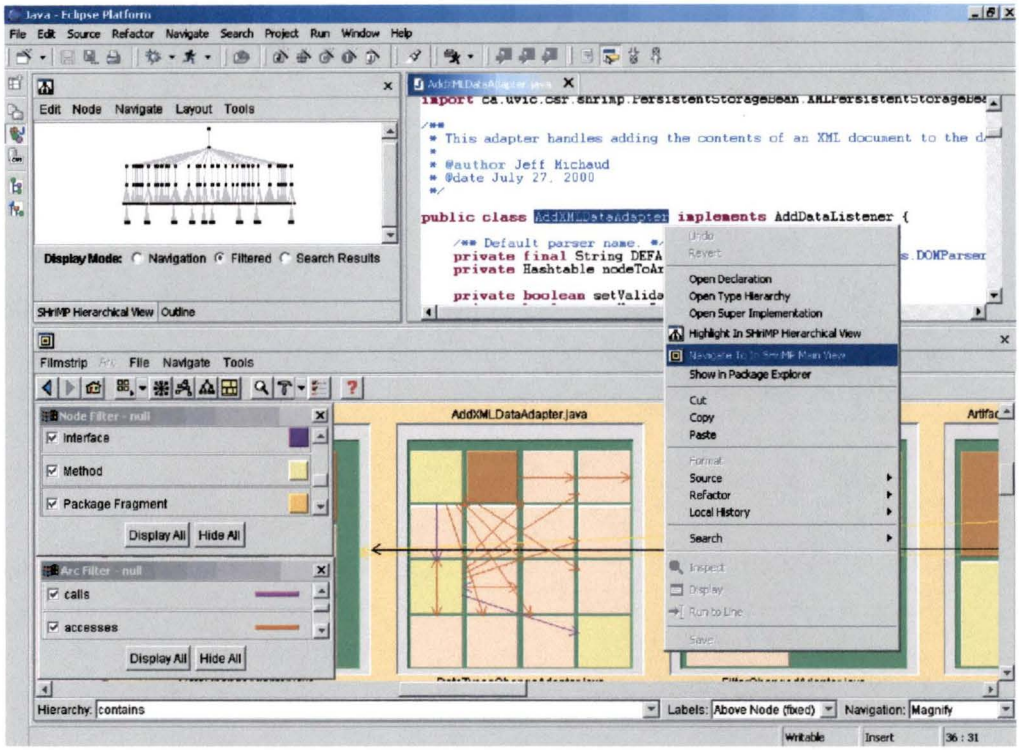


Figure 5.2: A screenshot of Creole. *SHriMP Hierarchical View* is shown in top left pane, *SHriMP Main View* shown is shown in the bottom pane and source code shown in top left pane, all of which are synchronized. The right mouse menu activated through the source code editor contains the invocation of Creole.

5.2 The Integration

As mentioned above, a programmer's typical working environment includes the source code in the workspace and in the repository. Xia is able to browse the information in the repository while Creole is used to visualize the software project in a programmer's workspace. Therefore, the integration of Xia and Creole would be able to support the visualization of a programmer's entire work environment. To find out a way to integrate Xia and Creole, source code investigation was conducted manually. By analyzing the Eclipse API and performing development tasks in the Eclipse environment, we conjectured that there must be some kind of "bridge" between the CVS element and JDT element that allows them to map to each other. As

a result of investigating the Eclipse source code, we found this "bridge" -- several function calls in the API, and incorporated it into our code. These methods made any Java element in the JDT able to access the CVS repository and retrieve related CVS information.

Following the building of the "bridge" of these two data resources was the integration of visualization. This phase was quite straightforward and was accomplished in a very short period of time. As both Creole and Xia use SHriMP as the visualization engine, the only differences between them are the customized features that were unique to the CVS domain. The attribute panel which is used to explore the CVS attributes in Xia was integrated with Creole at the code level (i.e., packages were added to Creole source code). Figure 5.3 illustrates the architecture of Xia and Creole integrated with each other.

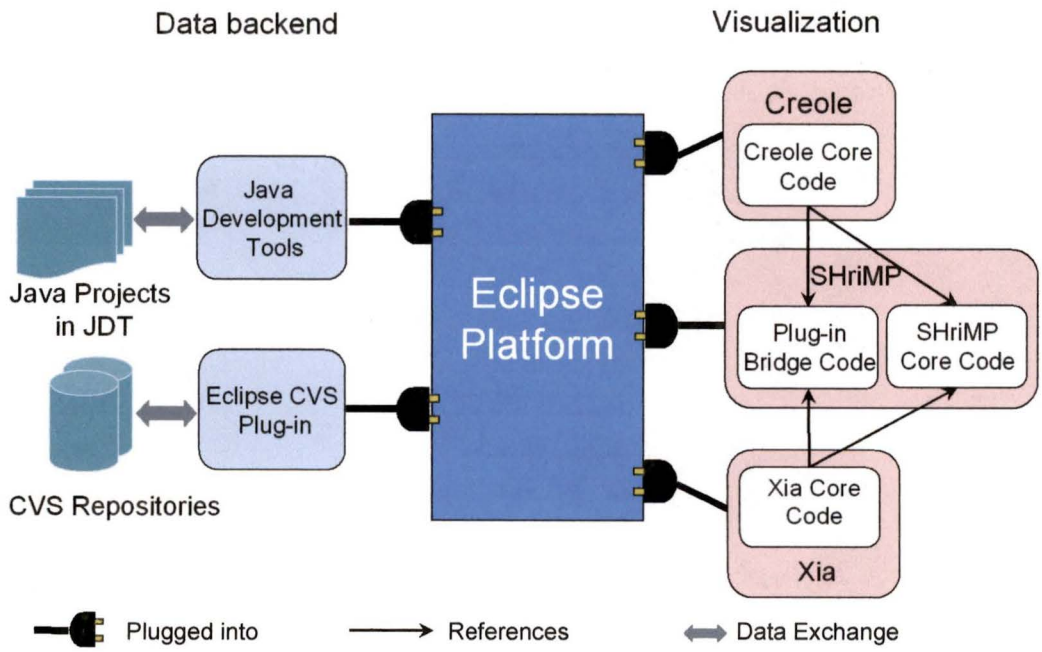


Figure 5.3: The architecture of Xia and Creole integration

5.3 Benefits of the Integration

Integrating Xia and Creole is beneficial for creating a visualization of the working data of a programmer in a team. The benefits can be summarized as follows:

- The data resource from Xia and Creole constitutes a programmer's real world working data in a team environment. In particular, Xia represents data in the CVS repository, while Creole represents data in the workspace.
- We have the ability to produce visualizations that show information relevant to the software's structure and architecture as well as information relevant to the version control history.
- The relationships between software artifacts are able to be displayed, hence providing more meaningful information and answering more sophisticated questions. For example, Fig. 5.4 shows a Java program in which nodes are colored according to the author who made the latest change. Purple arcs represent the "reference" relationship. The arcs between nodes enable people to focus on a specific task and keep track of its relationship to other files in the project, as well as other people who are working on those files. This kind of awareness is important for teams to collaborate effectively, and provides a path for people to track the impact of changes.

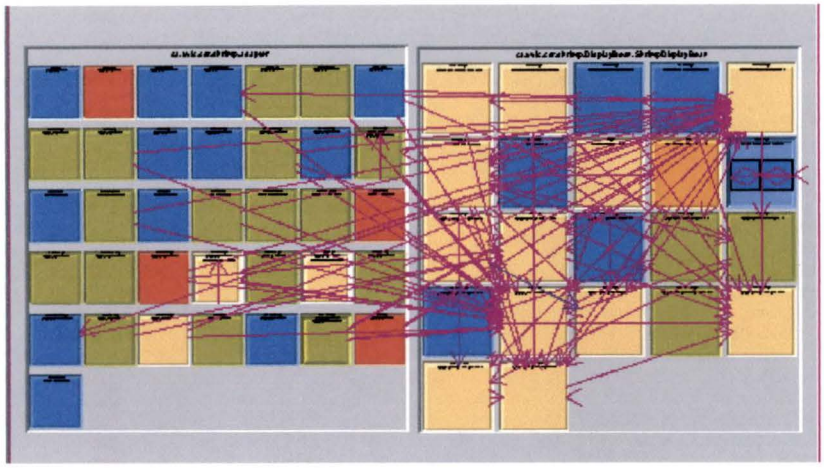


Figure 5.4: The color of each node is determined by the author who made the latest change. The purple arcs represent the "reference" relationship.

5.4 Summary

This chapter first introduced a tool called Creole, which is implemented by integrating SHriMP with the Eclipse JDT plug-in; then described our approach to integrate Xia and Creole. The approach provides visualizations of information from a programmer's workspace as well as the CVS repository to support software development in a team environment. Benefits of integrating Xia and Creole were also discussed.

Chapter 6 Evaluation

We conducted a preliminary user study to test both the functionality and usability of Xia. Before our study, our hypothesis was that Xia could be used to improve the collaboration of team work by answering the 5W+2H questions. We also expected the tool would properly reflect an overview of the version history and hence help people to better understand the software history. In addition, the effectiveness of various visualization techniques for answering these questions was also of interest.

A Java project with four versions was chosen as the dataset for the study. The most recent version of the project contained approximately 1000 lines of code. Participants' background information with version control tools was gathered in a pre-questionnaire [see Appendix C]. Tasks were created based on plausible real world scenarios adapted from a real world project. The following sections describe in detail the participants, the procedure of the study, the tasks and the results of the study.

6.1 Subjects

Five graduate students from the Department of Computer Science at the University of Victoria participated in the study. Each participant had programming

experience on a team software project, working with at least one version control tool. In addition, each of them had experience with the Eclipse development environment and CVS, which is important to our study as participants can focus on specified tasks without any effects caused by the unfamiliarity of the environment. However, a deficiency in our subject selection correlates to our users not being part of the development team working on the source code under study, and hence not being familiar with this code. We attempted to offset this shortcoming by introducing the study as a scenario in which the user was part of an active development team. We also provided a detailed explanation of the source code to offset any unfamiliarity with the project domain. Since our users already had experience with other CVS tools we could collect their opinions on how the tools compared based on their previous experiences.

6.2 Procedures

Following the pre-study questionnaire (i.e., to determine their previous programming and version control experience), a fifteen-minute orientation on Xia was provided to each participant. In this orientation, the participants were introduced to the basic tool operations, and the tool's core features. Following the orientation, a task list was administered to participants. No time limit was set for the participants to resolve the tasks, since we were more interested in observations of how the given tool is used in practice. As a result of this, the actual time spent to complete the tasks varied among different participants, ranging from 30 to 90 minutes. Users were encouraged to think aloud so we could verify our interpretation of their actions [40]. Their evaluation was recorded using the screen and audio recording software

Camtasia by which the details of users' interaction with the tool were traced and reviewed afterwards. Following completion of the tasks, further inquiry into the user's opinion of the tool was gathered through a post-study questionnaire.

6.3 Tasks

Two sets of similar tasks [see Appendix D] were assigned to participants corresponding to two different data resources: the data in the CVS repository and the data in the programmer's own workspace. These two sets of data constitute a programmer's working data in the real world. The tasks involved exploring the information space and answering questions related to team work and software history.

For example, one of the tasks asked the participant to name all programmers who have been working on the project. Another task asked the participant to find out who was the last person working on a particular file. These two tasks correspond to the "Who" question on the project and file levels. Also, we asked the user to determine what kind of changes to a particular file had been made, which answered the "What" question. As per the "When" question, we encouraged the user to establish which file was changed most recently. With respect to the "How" question, participants were asked to discover how a particular file was changed in the latest commitment. The "Why" question was explored by asking for the rational behind a particular change, and the "History" of a file was explored by request too. In addition, we also posed some tasks that we believe are of interest to both project managers and programmers. For example, we asked the user to find the file that has changed most often in the project. We hypothesize that this measurement be useful for people looking for stable or active file(s) in a project.

6.4 Results

We collected both positive and negative feedback from participants through observations and questionnaires. On the positive side, participants successfully resolved most tasks in a very short time, despite the learning curve they must overcome. Some general observations showing the benefits of the tool were summarized as follows:

- The visualization and exploration techniques provided by the Attribute Panel (see Fig. 6.1, as described in Section 4.3.3) were used frequently to resolve the tasks. Also, participants pointed out in their post-study questionnaires that they would like to use features of the Attribute Panel in their everyday work.

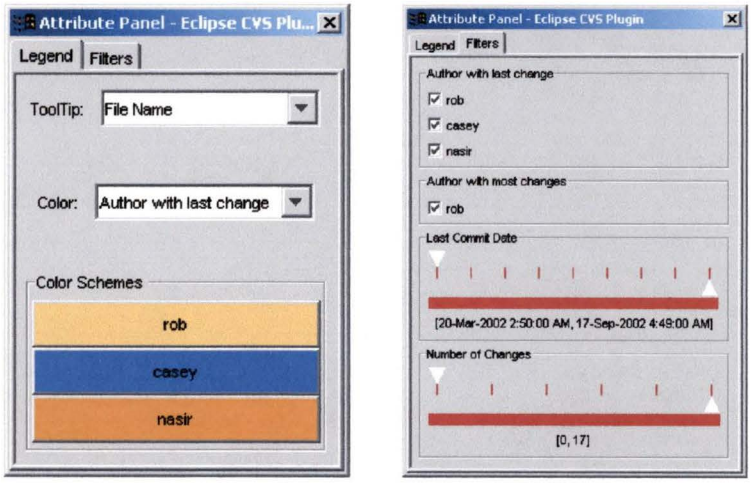


Figure 6.1: The Attribute Panel showing the user study data.

- The tool appeared to be easy to learn and use. Although only fifteen minutes of orientation was provided, participants used the tool effectively to perform the tasks. They were aware of the possible ways to use the tool to solve problems and did not require additional assistance or note any significant difficulties.

- Participants considered the tool informative, from both the project manager and programmer perspective. Candidates indicated they believe the Xia tool could prove helpful in helping them solve many problems they encounter in a work environment.
- The tool was also used to answer more sophisticated questions by making use of a combination of features. For example, one of the tasks asked the participants to find out which file is most stable and which file is most active. The participants defined "stable" and "active" in a similar way: a file that has not been changed for a long time and to which very few changes were made was considered stable; the opposite held true for an active file. To answer this question, participants chose both the last commit date double slider and number of change double slider to narrow down the range of candidate nodes, and analyzed the candidate nodes.
- The visualization features in Xia helped the users gain more awareness of their teammates' activities.
- The participants were impressed by the immediate feedback the visualizations provided when they posed a new query. Some of them had special interests in color schemes while others used filters more often.

Though the positive feedback is encouraging, we also noticed some deficiencies of the tool, which is of more interest to us. These deficiencies are related to both the functionality and usability of Xia. We summarized our findings from these two perspectives. First, regarding the deficiencies of the functionality, we found:

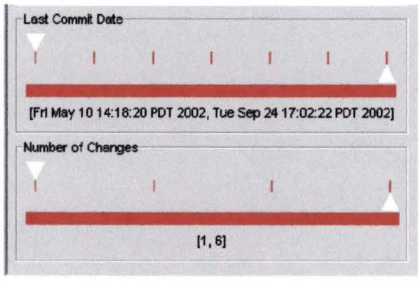
- Some participants were confused when working with different revisions of the same file. They suggested that some kind of mapping between different revisions of the same file would be helpful.

- The file revision organization requires a more elegant display. Currently, the file revisions are organized by software versions. However, revisions not belonging to a particular version are not considered or displayed in the tool. This may lead to a loss of information.
- Some participants also suggested a time-line arrangement of project versions.
- Visualization of other attributes was also anticipated by some of the users. For example, one of the users was interested in who originally created a particular file.
- Participants considered the "diff" function—a comparison of two different file revisions important in their everyday work. We considered displaying the CVS plug-in's diff view within Xia, however, Xia does not currently support this feature due to difficulties embedding Eclipse's SWT GUI inside of Xia's Java Swing Component (a problem discussed by Rayside *et al.* [44]). Further investigation is required for this technical issue. However, as Xia is a plug-in for the Eclipse platform, we can invoke the CVS plug-in's "diff" view in a separate SWT window as an alternative. The ability to use and integrate existing functionalities with our tool demonstrated another benefit of integrating our visualization tool with a full-featured IDE.

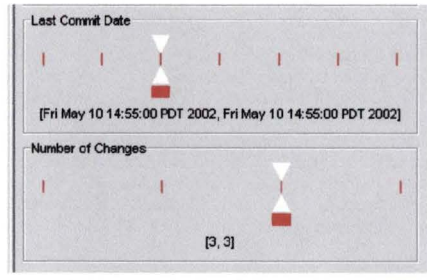
Second, regarding the deficiencies with respect to usability, we found:

- Although the participants thought that the use of color intensity to interpret ordinal attribute values is very useful, they also pointed out that it is hard to tell which is brighter or darker when two intensities are close. In this case, they used tooltips for assistance.
- The interface of double slider is somewhat unintuitive. The following picture shows the interface of the double slider (see Fig. 6.2 (a)). There are two red

triangles used to set the value range of the slider. The one above the scale is used to set the minimum value of the slider, and can not be dragged over the right hand triangle below the scale. Similarly, the one below the scale is used to set the maximum value of the slider, and can not be dragged over the left hand triangle. There was no problem for people to operate on these triangles until the triangles meet together representing a single value (see Fig. 6.2 (b)). We noticed people tried to drag the upper triangle to the right side of the lower triangle and drag the lower triangle to the left of the upper triangle when they wanted to expand the value range of the double slider. They changed their ways of operation until they found the previous dragging didn't work.



(a) The ranges of double sliders are expanded to maximum.



(b) The minimum and maximum values of the range are set to the same representing a single value.

Figure 6.2: GUI of double slider

In retrospect, "below" is a better match to "minimum", while "above" is a better match to "maximum". We consider a switch of these two triangles might be a solution to this problem: using the triangle below the scale to represent the minimum value, and the triangle above to represent the maximum value; the triangle below should always be on the left side of the triangle above.

We also have some trivial findings about the user interface design in addition to those more important ones we described above. These trivial issues were not even considered when we designed the interface, but were recognized and pointed out by users in the study.

- Improper wording in the interface is somewhat misleading. For example, our original text displays for some of the attributes were: name, revision, date, changes. We referred "name" to the file name, "revision" to the revision number of a file revision, "date" to the last commit date of a file revision, and "changes" to the number of changes made to a file. However, some users have a different understanding of these terms. For example, one of them thought "name" was the name of a person, and got confused with the author; he also thought "changes" was the exact changes that a file revision has, not the number of changes. We learnt a lesson from this, and realized that even a small design defect can cause big problems. Consequently, we replaced the terms with more descriptive wording.
- A "reset" button in the Attribute Panel was suggested by participants. In performing the tasks, users need to adjust the settings of the Attribute Panel now and then. It is a little bit annoying that users need to set back all the wizards to their default values before performing a new setting, and it is likely that some of the old settings are left behind since the user might forget or even not notice them. Therefore, a reset button is considered helpful in setting all wizards to default values before a new setting is configured.
- In Xia, ordinal attributes are highlighted using color intensity. We have two ordinal attributes in the CVS domain: last commit date and number of changes. We assigned a brighter intensity to a newer date and a bigger number of

changes, and vice versa. However, this assumption was not interpreted in the GUI; therefore users can't understand it and have to use other visual mechanisms for assistance (e.g. the tooltips). Consequently, we put a text description below the color schemes panel, explaining how we assign intensities to different values.

- The place to invoke the Attribute Panel in Eclipse is hard to locate. Users always tried to locate it through the Eclipse main menu or toolbar, while the actual location is within the SHriMP toolbar. This revealed a problem of how to make effective presentation integrations with an IDE.

6.5 Themes

We abstracted several themes from the data collected in the study, which included the questionnaires and observations.

Theme 1: The approach of version control visualization is novel for most users.

Although visualization of version control is being researched by several groups, every day users are not aware. In our study, some of the users didn't know or use any version control visualization tool before; some of them considered the GUI of a version control tool as so-called visualization. People are familiar with various version control tools but are not yet exposed to version control visualizations.

Theme 2: Basic version control features should be supported in the visualization.

Version control activities and visualizations should not be separated. The separation of version control features and the visualization of version information may bring the inconvenience of switching between tools. Users always prefer to stay

in a tool and resolve all tasks in this tool rather than switch between tools. One of the important version control features that should be incorporated with visualization is the Diff function (compare two revisions of a software artifact). An advanced support could be check-in/out from the visualization.

Theme 3: Version metrics—potential measurements in the version control domain.

More attributes should be associated with each software artifact (e.g., who created the file?). More metrics at the project level should also be considered. Further investigations and/or surveys are needed to collect opinions and define version metrics.

Table 6.1 summarizes the version metrics of different tools discussed in Section 2.3. In the table, the "Tool" column indicates the name of the tool; the "Version Metrics" column lists metrics related to version control in this tool; the "Visualization" is the visual metaphor used for each of the version metrics. The "Associated with" column explains the abstraction level of the metrics. Different tools define an *artifact* in different ways. In Palantir, an artifact is a file as Palantir uses RCS and CVS as version control systems; in Beagle, an entity could be a file, a module or a subsystem.

Tool	Version Metrics	Visualization	Associated with
SeeSoft	code age	color (rainbow color scale)	each line of code
	file size (LOC)	height of bar	each file
	the amount of new code (LOC)	height of bar	each file
	programmer	color	each line of code
	bug-fixing	color, length of line	each line of code
	change type (deleted, added, changed, unchanged)	color	each line of code
CVS activity Viewer	programmer	color	each line of code
	code age	color	each line of code
Palantir	severity of the change	vertical bar	each artifact
	impact of the change	vertical bar	each artifact
	author information (name, email, URL)	color, text display	each artifact
	change comments	text display	each artifact
	artifact name	text display	each artifact
	artifact version	text display	each artifact
	date checked out	text display	each artifact
	event type (change type)	color	each artifact
	pair-wise conflicts	color	each workspace
Beagle	change type	color	each entity
	age	intensity	each entity
	version number	text display	each entity
	release date	text display	each entity
	name of entity	text display	each entity
Eick's approach	date when a change was made	color	each change
	author	color	each change
	type of change	color	each change
	size of change	width of bar	each change
	effort: how many developer-hours devoted	text display	each change
	interval: how long the change took in calendar time	text display	each change
	number of changes by a developer	color, height of vertical towers, bar chart	each module
Gevol	the time a file was first created	color	each file
	the author of each change of each file	color	each file
	the time of each change to each file	color	each file

Table 6.1: Summary of version metrics

Theme 4: File revisions should be organized in different ways, producing different layouts from different perspectives.

In Xia, file revisions is organized by version tags. However, different organization of file revisions is needed to display different aspects of the version information. For example, time is an important factor in version control systems. File revisions organized in a time line view is essential for people to understand the version history of the software.

Theme 5: Color is always good to communicate differences.

Different colors are easy to tell apart by humans; therefore it is appropriate to use different colors to represent different nominal values. However, these colors should not be similar hues, in which case they are hard to tell apart from each other, especially on a busy screen. In addition, the number of colors that humans can perceive at the same time is limited; therefore in the case of a big value set, we need to seek another solution. For example, this approach would not be suitable if there were more than 12 developers. To scale in this case, we would require both filtering and the use of colors.

Theme 6: Intensity is good for distinguishing between the maximum and minimum of a set of values, but is not helpful when comparing two adjacent values.

When a set of values is sorted and assigned different intensities, it is always easy to tell which is the brightest and which is the darkest. However, for two adjacent values whose intensities are always close, it is a little hard for people to tell which one

is brighter and which one is darker, especially when the dataset is large and the granularity of intensities is small.

6.6 Lessons Learnt from the User Study

In Section 6.4, we discussed the results of the user study, and limitations of our tool. In this section, limitations of the user study are discussed.

First, although the number of users is sufficient to find many of the usability problems in our approach, we believe that more users would provide more feedback, especially for performing statistical analyses.

Second, the size of the project used in the study was small. We believed a larger project would be able to create a more mature testing environment; however, the study required users to gain some knowledge of the code in a relatively short time, and thus confined us to a small project.

Third, the user and the project were kind of "separated". Participants were not the real people working on the project; hence they were not familiar with the code. This is not a realistic setting and may bring barriers for users performing the tasks.

6.7 Summary

This chapter provided a preliminary evaluation on Xia. Details of the procedure and general observations were described. Positive and negative feedback were summarized, providing good directions for further improvements of the tool, and leading to more research questions.

Chapter 7 Conclusions

We designed and developed a version control visualization tool, called Xia, to browse the information generated in a team work environment. The primary goal of building this tool was using visualization to help people share information and bring awareness in team work environments, display version history to help people better understand the software development process, and improve the work efficiency.

In Xia, versions of a software project are displayed in a grid layout, with each of the versions represented by a node. Contents of the version were contained in this node in a nested graph. CVS attributes of each file were attached to the file revision node and displayed as visual variables. Visual metaphors were applied to highlight these attributes, and query techniques were applied to explore the CVS information space.

A user study was conducted to evaluate the tool based on its functionality and usability. Positive and negative feedback were collected, which showed the advantages of the design of the tool as well as pointed out the deficiencies, leading to future improvements of the tool.

7.1 Contributions

In this thesis, we investigated and discussed related work in the field of version control visualization, which is a novel field to be further explored. The background information and related work summarized in this thesis should provide a good resource and set a starting point for people in future related research.

We have proposed an approach to display version control information. An advanced visualization tool, SHriMP, was integrated with a widely used version control system, CVS, through an integrated software development environment. This integration alleviated us from the burden of creating visualizations from scratch and communicating with CVS servers. The integration with a full-featured IDE also increases the possibility of Xia being adopted in the real world, and hence providing us with more feedback.

The employment of visual metaphors and interactive filtering mechanisms provide effective means for understanding and exploring the version control information, which contains the version information of software artifacts and associated human activities, hence bringing awareness among team members in collaborative software development.

We also conducted a user study on our version control visualization tool, which has been rarely done in related research [62]. The results of the user study bring new insights into this area and could be a good starting point for other researchers' work.

7.2 Future Work

7.2.1 Future Research Questions

In our survey and user study, we found unanswered questions in our approach. For example, one of the participants wants to know who the creator of a particular file is. We believe there are more measurements or attributes, and version control metrics that we need to explore. Future research on this could be conducted through further surveys on project managers and software developers, as they are our target users of the tool.

Finer granularity of version control is still desirable. Although we were able to show finer detailed software artifacts by Creole, the version control is still at the file basis; hence we can't tell directly who changed a particular method. Future work includes applying visualizations on version control tools with finer granularity (e.g., Stellation [54]).

7.2.2 Improvements of Visualizations

Different layouts could interpret a concept from different angles. We have several ideas of new layouts in presenting the version control information.

Regarding the problems discovered from the user study, in particular the organization and layout of file revisions, we have an early prototype of a proposed solution. First, we extracted all revisions from the CVS repository regardless of whether they belong to a particular software version or not. Revisions that belong to the same file are then clustered in a cascaded view (see Fig. 7.1).

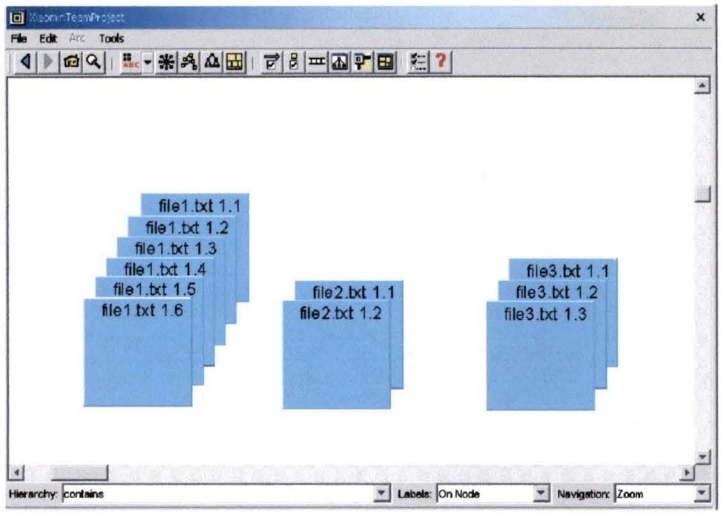


Figure 7.1: Cascaded nodes representing different revisions of the same file

The arrangement in Fig. 7.1 uses screen space effectively, but version tags are excluded. Sometimes the user wants to be able to see which project versions are available. Our graph layout animates to the following view in Fig. 7.2. The animation expands the cascaded nodes to a histogram so that each file is presented by a column. Taller columns therefore have more revisions.

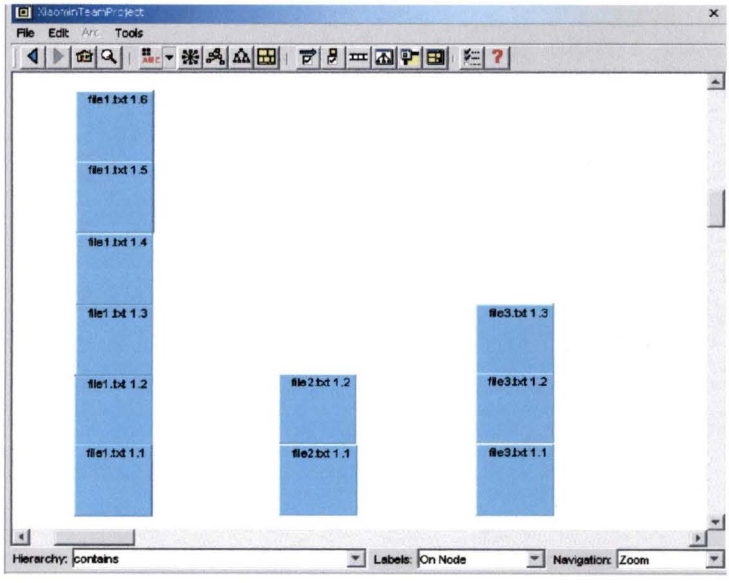


Figure 7.2: Cascaded nodes are expanded to a histogram

Furthermore, to gain an overall picture of software versions, file revisions that belong to a particular project version need to be related. We achieve this by connecting files related to a particular project version using colored lines (see Fig. 7.3). This layout is named "Tagged layout" as file revisions are connected by version tags.

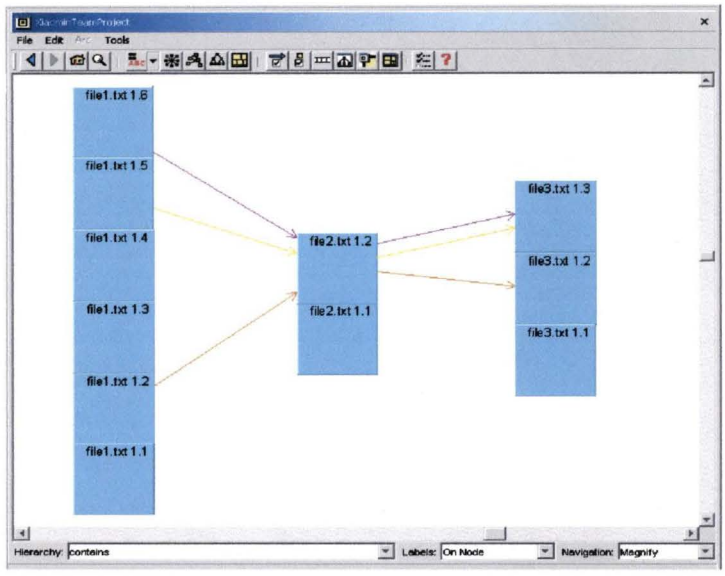


Figure 7.3: A project version is represented by connecting the file revisions that belong to this version. For example, Version 3 contains Revision 1.6 of file 1, Revision 1.2 of file 2 and Revision 1.3 of file 3. They are connected by purple arcs. Different versions are connected using lines with different colors.

Besides the Tagged layout, a spring layout that organizes file revisions by authors is also of interest. In this layout, an author node is created for each of the developers. File revision nodes that represent file revisions being modified by a particular person are gathered around the author node representing that person. This layout should be able to present an obvious answer to "Who touched what" question. In contrast, we can present another view that has file nodes in the centre, surrounded by author nodes representing persons that have been working on this file.

7.2.3 Future User Studies

We did not compare Xia with other tools in our study. Although the literature contains references to several tools [3, 10, 15, 18, 19, 21, 47, 59] that do visualization of CVS information, we were not able to obtain a mature prototype of any of these tools that was integrated with a CVS system in an IDE. We did not compare to the command line interface, as the users we recruited for our study were already familiar with the command line interface and could provide us feedback on it already. In the future, we may conduct user studies to compare Xia with other mature version control visualization tools, and test the new layouts we prototyped after the preliminary user study and the redesign of the GUI.

7.2.4 Other Issues

There are also some other related issues we may deal with in the future. First, the Attribute Panel implemented in Xia has been considered a good exploration widget for finding the attributes of artifacts by participants of the user study and researchers in the CHISEL group at the University of Victoria, to whom the Attribute Panel has been first introduced. Based on the request of researchers in the CHISEL group, the Attribute Panel has been slightly modified at the structure level and more tightly integrated with SHriMP to suit all information domains, not only the CVS domain. Therefore, any information spaces that adopted SHriMP as the visualization front-end would be able to use the Attribute Panel as an attribute exploring tool.

Second, we have some technical problems to resolve in the future. For example, the speed of the program is fairly slow due to the time spent querying the CVS server when collecting the data resource. We consider speed essential since our tool will never be adopted if it takes too long to produce outputs.

7.3 Summary

Techniques of information visualization have been tightly integrated with different disciplines, and have proved to be a valuable and effective methodology to help understand and explore all kinds of domain information. Furthermore, team work has become a dominant style of software development, and many version control systems have been developed and adopted in this process, to support team work and manage historical information related to the development process. To better understand and explore the vast information contained in version control systems, hence better support team work and programmers' understanding of software version history, visualization techniques can be applied to these systems.

In this thesis, a version control visualization tool, Xia, was designed and implemented for this purpose. Xia integrated an advanced visualization tool, SHriMP, with a widely-used version control tool CVS. Information contained in CVS was represented using various layouts and visual metaphors, and explored through interactive querying mechanisms. A user study was conducted to evaluate the tool, and valuable feedback was collected from participants. The feedback led to our future design of the tool and more questions in this line of research.

Bibliography

[1] ADELE team, "Reverse-Engineering and Configuration Management: Concepts and Perspectives," presented at the Software Conf'96, Paris, France, 1996.

[2] C. Ahlberg, and S. Truve, "Exploring Terra Incognita in the Design Space of Query Devices," presented at the Engineering for Human-Computer Interaction conference (EHCI '95), Grand Targhee Resort, Wyoming, USA, 1995.

[3] T. A. Ball, and S. G. Eick, "Software Visualization in the Large," *IEEE Computer*, vol. 29, no. 4, pp. 33-43, 1996.

[4] C. Best, *Designing a Component-based Framework for a Domain Independent Visualization Tool*, Masters Thesis, Dept. of Computer Science, University of Victoria, 2002

[5] C. Best, M. A. Storey, and J. Michaud, "Designing a Component-Based Framework for Visualization in Software Engineering and Knowledge Engineering," presented at the 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, 2002.

[6] Bonsai website, http://www.mozilla.org/projects/bonsai/, 2003

[7] R. Bosch, C. Stolte, D. Tang, J. Gerth, M. Rosenblum, and P. Hanrahan, "Rivet: A Flexible Environment for Computer Systems Visualization," *Computer Graphics*, vol. 34, no. 1, pp. 68-73, 2002.

[8] S. K. Card, and J. D. Mackinlay, *Readings in Information Visualization, Using Vision to Think*: Morgan Kaufmann, 1999.

[9] H. B. Christensen, "Experiences with Architectural Software Configuration Management in Ragnarok," presented at the International Symposium on System Configuration Management, Brussels, 1998.

[10] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler, "A System for Graph-Based Visualization of the Evolution of Software," presented at the 2003 ACM symposium on Software visualization, San Diego, USA, 2003.

[11] S. B. Compton, and G. R.Conner, *Configuration Management for Software*: Van Nostrand Reinhold, 1994.

[12] CVS Website, http://www.cvshome.org/, 2003

[13] S. Dart, "Concepts in Configuration Management Systems," presented at the 3rd International workshop on software configuration management, Trondheim, Norway, 1991.

[14] S. Dart, "Spectrum of Functionality in Configuration Management Systems," Carnegie Mellon Software Engineering Institute CMU/SEI-90-TR-11 ESD-90-TR-212, 1990.

[15] P. Dourish, "Visualizing Software Development Activity", http://www.ics.uci.edu/~jpd/research/seesoft.html, 2003

[16] Eclipse Homepage, http://www.Eclipse.org, 2003

[17] Eclipse Platform Subproject Webpage, http://www.eclipse.org/platform/index.html, 2003

[18] S. G. Eick, and J. L. Steffen, "Seesoft - A tool for visualizing line oriented software statistics," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 957-968, 1992.

[19] S. G. Eick, T. L. Graves, A. F. Karr, A. Mockus, and P. Schuster, "Visualizing Software Changes," *IEEE Transactions on Software Engineering*, vol. 28, no. 4, pp. 396-412, 2002.

[20] K. Fogel, and M. Bar, *Open Source Development with CVS*: Electronic copy is available online: http://cvsbook.red-bean.com/, 2000.

[21] H. Gall, and M. Jazayeri, "Visualizing Software Release Histories: the use of color and third dimension," presented at the International Conference on Software Maintenance (ICSM '99), Oxford, England, 1999.

[22] R. Holt, and J. Y. Pak, "GASE: Visualizing Software Evolution-in-the-Large," presented at the 3rd Working Conference on Reverse Engineering (WCRE '96), Monterey, CA, 1996.

[23] J. J. Hunt, and W. F. Tichy, "Compression Comparison: RCE (bdiff) vs. RCS (diff)", http://www.ipd.uka.de/~RCE/compare.ps, 2002

[24] IEEE Computer Society, "IEEE Guide to Software Configuration Management," *ANSI/IEEE Std 1042-1987*: The Institute of Electronical and Electronics Engineers, Inc, 1987.

[25] IEEE Computer Society, "IEEE Standard for Software Configuration Management Plans," *IEEE Std 828-1998*: The Institute of Electronical and Electronics Engineers, Inc., 1998.

[26] IEEE Computer Society, "IEEE Standard for Software Configuration Management Plans," *IEEE Std 828-1990,*: The Institute of Electronical and Electronics Engineers, Inc., 1990.

[27] IEEE Computer Society, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990,*: The Institute of Electronical and Electronics Engineers, Inc., 1990.

[28] IEEE Computer Society, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 729-1983*: The Institute of Electronical and Electronics Engineers, Inc, 1983.

[29] D. F. Jerding, and J. T. Stasko, "The Information Mural: A Technique for Displaying and Navigating Large Information Spaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 3, pp. 257-271, 1998.

[30] J. P. D. Keast, M. G. Adams, and M. Godfrey, "Visualizing Architectural Evolution," presented at ICSE-99 Workshop on Software Change and Evolution (SCE-99), Los Angeles, 1999.

[31] M. Lanza, and S. Ducasse, "Understanding software evolution using a combination of software visualization and software metrics," presented at LMO'2002 (Langages et Modèles à Objets), 2002.

[32] S. W. Liao, "SUIF Explorer: An Interactive and Interprocedural Parallelizer," presented at the 7th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP),, Atlanta, Georgia, 1999.

[33] P. Lindsay, A. MacDonald, and P. Strooper, "A Framework for Subsystem-based Configuration Management," presented at 13th Australian Software Engineering Conference (ASWEC'01), Canberra, Australia, 2001.

[34] R. Lintern, J. Michaud, M. A. Storey, and X. Wu, "Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse," presented at the 2003 ACM symposium on Software visualization, San Diego, USA, 2003.

[35] C. Maidantchik, and A. R. C. da Rocha, "Managing a Worldwide Software Process," presented at the International Workshop on Global Software Development, Orlando, Florida, 2002.

[36] K. McGuire, "VCM 2.0 Storey", http://www.eclipse.org/platform/index.htm, 2003

[37] J. Michaud, M. A. Storey, and H. Muller, "Integrating Information Sources for Visualizing Java Programs," presented at the IEEE Conference on Software Maintenance, 2001.

[38] Microsoft Visual Sourcesafe home page, http://msdn.microsoft.com/ssafe/, 2003

[39] K. L. Mills, "Introduction to the Electronic Symposium on Computer-Supported Cooperative Work," *ACM Computing Surveys*, vol. 31, no. 2, pp. 105-115, 1999.

[40] J. Nielsen, *Usability Engineering*: Academic Press, 1993.

[41] S. Northover, "SWT: The Standard Widget Toolkit", http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html, 2003

[42] Perforce website, http://www.perforce.com/, 2003

[43] Rational ClearCase website, http://www.rational.com/products/clearcase/index.jsp, 2003

[44] D. Rayside, M. Litoiu, M. A. Storey, C. Best, R. Lintern, "Visualizing Flow Diagrams in Websphere Studio Using SHriMP Views," *Information Systems Frontiers: A Journal of Research and Innovation*, vol. 5, no. 2, pp. 161-174, 2003.

[45] Rigi website, http://www.rigi.csc.uvic.ca/index.html, 2003

[46] A. Sarma, A. Noroozi, and A. Hoek, "Palantir: Raising Awareness among Configuration Management Workspace," presented at the 25th International Conference in Software Engineering, Portland, US, 2003.

[47] A. Sarma, and A. Hoek, "Palantir: Coordinating Distributed Workspaces," presented at the 26th Annual International Computer Software and Applications Conference, Oxford, England, 2002.

[48] J. A. Scott, and D. Nisse, "Chapter 7: Software Configuration Management," in *Guide to the Software Engineering Body of Knowledge: SWEBOK*: The Institute of Electronical and Electronics Engineers Inc., 2000.

[49] B. Shneiderman, "Dynamic Queries for Visual Information Seeking," *IEEE Software*, vol. 11, no. 6, pp. 70-77, 1994.

[50] B. Shneiderman, "Tree Visualization with Tree-maps: A 2-d space-filling approach," *ACM Transactions on Graphics*, vol. 11, no. 1, pp. 92-99, 1992.

[51] B. Shneiderman, and M.Wattenberg, "Ordered Treemap Layouts," presented at the 2001 IEEE Symposium on Information Visualization, Los Alamitos, CA, USA, 2001.

[52] SHriMP website, http://www.shrimpviews.com, 2003

[53] R. Spence, *Information Visualization*: Addison-Wesley, 2001.

[54] Stellation project homepage, http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/org.eclipse.stellation/main.html, 2003

[55] M. A. Storey, C. Best, J. Michaud, D. Rayside, M. Litoiu, and M. Musen, "SHriMP views: an interactive environment for information visualization and navigation," presented at CHI 2002 Conference, Extended Abstracts on Human Factors in Computer Systems, Minneapolis, Minnesota, USA, 2002.

[56] W. F. Tichy, "RCS - A System for Version Control," *Software - Practice and Experience*, vol. 15, no. 7, pp. 637-654, 1985.

[57] I. Tifft, "Version Control Integrations - Conquering revision management with JBuilder 5," 2001.

[58] J. E. Tomayko, "Software Configuration Management," Software Engineering Institute, Carnegie Mellon University. SEI Curriculum Module SEI-CM-4-1.4, 1990.

[59] Q. Tu, and M. Godfrey, "Integrated Approach for Studying Software Architectural Evolution," presented at 2002 Intl. Workshop on Program Comprehension (IWPC-02). 2002.

[60] H. Völzer, B. Atchison, P. Strooper, P. Lindsay, and A. MacDonald, "A Tool for Subsystem Configuration Management," presented at the International Conference on Software Maintenance (ICSM'02), Montreal, Quebec, Canada, 2002.

[61] C. Ware, *Information Visualization, perception for design*: Morgan Kaufmann, 2000.

[62] Z. Weinberg, "Novel Methods of Displaying Source History: A Preliminary User Study." http://www.panix.com/~zackw/cs260/novel-methods-of-displaying-source-history.pdf, 2002

[63] WinCVS website, http://www.wincvs.org/, 2003

[64] J. Wu, *Integrating Visualization Techniques to Support Program Comprehension*, Masters Thesis, Dept. of Computer Science, University of Victoria, 2000

Appendix A: Survey – Part A

* This is a survey sent out through email to researchers in the CHISEL Group at the University of Victoria.

We are trying to find the best way to visualize the version history of software systems. We may need your input/inspirations.

Here is a brief description about the project:
Firstly, we have the following tools at hand:

A visualization tool, SHriMP

A repository that stores all the historical data related to a software system. The format of the storage is a common version tree.

A version control tool that works on the repository

Second, we want to use, or extend, the above tools to visualize the information stored in the repository. In our approach, we may integrate the visualization tool with the version control tool, so we don't need to interact with the repository directly. We may use the API provided by the version control tool to manipulate the repository (e.g. some basic operations: get data, diff, etc). After we get the data we needed from the repository, or the results of the data operation from the version control tool, we will visualize it in some form using SHriMP.

So here are some questions for you:

1. What do you think about the current version control tool you are using? Do you think it provides all the necessary functionality to make a group work efficiently? If not, what is missing?

2. What mostly interests you, besides your own work, when working in a group?

3. What are the important things you think that you need to visualize when you are working as a member of a group using change management tools?
   Some examples are: you may want to see the old versions of files you are working on, or you may want to see what other people are doing, or you may want to see what changes have been made recently, etc. Please provide scenarios, as many as you can.

4. What to compare?

One of the most common functions of version control tools is to compare the differences between two targets. So what could be the target of the comparison? Examples are: different versions of a file, different releases of a program, etc.

5. Basically, the smallest unit in a version control tool is a file. So, given a file, what is your main concern about that file? Examples are: Who wrote it originally? What has been changed compared with the latest version? Why the change happened? Who made such a change? Did the change affect other components?

6. As a summary, changes happening in the software development process can be divided into three categories: add a new file/method/variable, delete a file/method/variable, and modify an existing file/method/variable. So, what could be the criteria of a good method to visualize the changes?

7. Currently, most of the version control tools show the data using text views. However, we may want to provide a more robust visualization using SHriMP. By this means, the historical data would be shown in graphical views. What kind of graph may you want to see, on the basis of SHriMP's visualization techniques?

8. Any other thoughts/ideas?

Appendix B: Survey – Part B

*This is a further survey based on the previous one.

Thank you all for your help with the first survey. Here comes another survey on the basis of the output of the first one. As we concentrated on "what to visualize" in the first survey, our focus is "how to visualize" this time. Below is a list of what to visualize associated with questions about how to visualize. Please use your imagination and don't be limited to SHriMP.

1. *What to visualize*: The overall view of the project.
   Background Information: Most of the existing VCS (version control systems) use a tree structure to display data in the repository/working directory. As the data storage in the repository or the working directory is much like that in a file system, some people think a tree structure is the best way. However, various visualization techniques could make it more robust, and in fact, a lot of effort has been contributed to this area.

   Question: What is your favorite visualization method for the overall view of the project?

2. *What to visualize*: Changes in the overall view.
   Background Information: Changes can be divided into three categories: add a component, delete a component, and modify a component. Actually, add or delete a component is a structure-level activity while modify a component is a lower-level activity. So it's better to tell what kind of change it is. According to the output of last survey, people are interested in viewing what has been changed not only in the content level but also in the structure level. So the preliminary idea is to highlight the components that have been changed in the overall view of the project.

   Questions:
   (1) How to highlight the changed components and tell apart different kind of changes using visualizations? Using colors, size, or something else?
   (2) How to tell the changing frequency of a component using visualizations?

3. *What to visualize*: Changes at low level.
   Background Information: Apparently, changes can only be observed by comparing two different copies of a component. Here we proposed three kinds of comparisons:
   1) Working directory vs. latest version in repository
   2) Working directory vs. any other version in repository
   3) Any two versions in repository

   Currently, changes are displayed by listing the two copies side by side using text views in most Version Control Software.

Question: What other visualizations could be applied?

4. *What to visualize:* Impact of changes

   Question: There should be a way to link the changes to the affected components, and then display the affected components someway. What is your favorite UI to go through this process?

5. *What to visualize*: User management

   Question: People want to see what other group members are doing or what they have done, so the question is: Where to put and how to visualize the team members' information and their related work in the main UI?

6. *What to visualize*: User annotations

   Question: User annotations are good hints to figure out why a change was made. So the question is: Where to put and how to visualize the user annotations in the main UI?

   If you have any other aspects that I missed out, please give a brief description. Thanks.

Appendix C: Pre-study Questionnaire

Pre-Questionnaire for study entitled:
Evaluation of a Version Control Visualization Tool

Please answer the following questions as completely as you can. Make multiple choices if you think they apply to you. If you feel uncomfortable answering any of the questions, please skip it. Your name will not be recorded. In our records you will be referred to as subject ____.

1. Have you ever been working in a team software project?
   - ☐ Yes
   - ☐ No

2. If you answered "Yes" to question 1, what was your position in the team?
   - ☐ Project manager
   - ☐ Programmer
   - ☐ Other. Please specify: _____

3. Have you ever used any of the following version control tools? Check all you have used.
   - ☐ CVS
   - ☐ Perforce
   - ☐ Rational ClearCase
   - ☐ Other. Please specify: _____

4. What features a version control visualization tool you think should have?
   - ☐ Showing the version history of the system
   - ☐ Showing the contents in your own workspace
   - ☐ Showing the changes of each file
   - ☐ Showing other people's work
   - ☐ Other. Please specify: _____

5. Have you ever used any version control visualization tool?
   - ☐ Yes.
   - ☐ No.

6. If you answered "Yes" to question 5, what is the version control visualization tool you used? And what features you like most?

Appendix D: Tasks

Tasks and Interview Questions for study entitled:
Evaluation of a version control visualization tools
Subject: _____

Tasks:
View the version history

1. Try to find out all programmers working on this project.
2. Try to find out the file that has been changed most recently.
3. Try to find the file that has been changed most times.
4. Select the file "CutCopySelectedArtifactAdapter.java", try to find out who is the last person working on this file.
5. Within the file in task 4, try to browse the history of this file.

View your own workspace

6. Try to find out the file that has been changed most recently.
7. Try to find the file that has been changed most times.
8. Randomly select one file, try to find out who is the last person working on this file.
9. Within the file in task 8, try to find out changes the last person made to this file.
10. Using all the mechanisms the tool provided, try to find out which file is the most stable one? And which is the most active one?

Questions:

1. What's the difference of the information you obtained from the repository view and your workspace view? Which one is more helpful to your everyday work?
2. Do you feel difficult when performing the tasks? If so, which is the most difficult one?
3. Are there any tasks you have been always performing in your everyday work while not listed here?

Appendix E: Post-study Questionnaire

Post-study questions for study entitled:
Evaluation of a Version Control Visualization Tool

Please answer the following questions as completely as you can. If you feel uncomfortable answering any of the questions, please skip it. You name will not be recorded. In our records you will be referred to as subject _X Y Z W V.

1. Please rate your satisfaction using a 5-point scale for the following aspects of the tool. In the scale, -2 means Very Dissatisfied, -1 means Dissatisfied, 0 means Neutral, 1 means Satisfied, and 2 means Very Satisfied.
   a. Over all ease of use
      ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2
   b. Ease of learning
      ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2
   c. Intuitive
      ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2
   d. Informative (i.e. does the tool provide enough domain information?)
      ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2

2. What features of the tool did you find useful?

3. What features did you find confusing? Do you have any suggestion to improve them?

4. What features do you think you would use most often?

5. What features did you want that were missing?

6. What features do you think should be removed? And Why?

7. Please draw some use cases that you think it might happen in your work?

8. Other comments:

Appendix F: Post-study Questionnaire with Answers

Post-study questions for study entitled:
Evaluation of a Version Control Visualization Tool

Please answer the following questions as completely as you can. If you feel uncomfortable answering any of the questions, please skip it. You name will not be recorded. In our records you will be referred to as subject _X Y Z W V.

1. Please rate your satisfaction using a 5-point scale for the following aspects of the tool. In the scale, -2 means Very Dissatisfied, -1 means Dissatisfied, 0 means Neutral, 1 means Satisfied, and 2 means Very Satisfied.
   a. Over all ease of use
      ☐ -2 ☐ -1 ☐ 0 ☒ 1 ☐ 2
   b. Ease of learning
      ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☒ 2
   c. Intuitive
      ☐ -2 ☐ -1 ☐ 0 ☒ 1 ☐ 2
   d. Informative (i.e. does the tool provide enough domain information?)
      ☐ -2 ☐ -1 ☐ 0 ☒ 1 ☐ 2

2. What features of the tool did you find useful?
   - **X**: who edit file ever, most recently; how many times updated (number of changes)
   - **Y**: Filtering, node coloring by attribute.
   - **Z**: Coloration; Floating Attribute Panel.
   - **W**: use color to provide a fast view of history and who touched the file, who made the last change, version dates
   - **V**: The attribute panel, particularly the filters.

3. What features did you find confusing? Do you have any suggestion to improve them?
   - **X**: None.
   - **Y**: CVS diff. Legend required (added for following studies)
   - **Z**: author with most changes selection? (since there is only one author, user got confused first and then realized the confusion was caused by the data set); Name should be changed to File name; hard to find Xia and Creole (user thought the proper place could be the Run-External tools in Eclipse)
   - **W**: Changes renamed to Number of Changes; Revision renamed to Revision Number; Name to File Name; Date to Commit Date; where to

find attribute panel, expected right click; Expand descendants should have already happen.

- **V**: Although color scheme is helpful, it takes lots of real state from the screen. This affects other possible info like users, file name, etc from being displayed.

4. What features do you think you would use most often?
   - **X**: Filters (by author)
   - **Y**: Filtering by date
   - **Z**: Colored changes
   - **W**: Code comparison; who changed what when.
   - **V**: Filter by last commit date to see all the files that changed recently.

5. What features did you want that were missing?
   - **X**: Creator of file.
   - **Y**: Clearer view of release tags vs. revisions
   - **Z**: Mapping files between versions.
   - **W**: Finer grain view of data; seeing code.
   - **V**: The tool is based on CVS so it's file centric. I think that source control should be abstracted at a higher level like activities or feature. Typically a user changes many files at once and they are related.

6. What features do you think should be removed? And Why?
   - **X**: None.
   - **Y**: extra package details were unnecessary. Not sure why organize by tags, not sure how relevant it is.
   - **Z**: Expand all descendents in SHriMP, should open it automatically.
   - **W**: No.
   - **V**: None.

7. Please draw some use cases that you think it might happen in your work?

8. Other comments:
   - **X**: too focused on attribute panel, missed dropdown menu items.
   - **Y**: timeline view?

UNIVERSITY OF VICTORIA PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain by the University of Victoria shall not be allowed without my written permission.

Title of Thesis:

Visualization of Version Control Information

Author [redacted]

Xiaomin Wu

August 1, 2003