A Reverse Engineering Approach to Support Software Maintenance: Version Control Knowledge Extraction

Xiaomin Wu
University of Victoria
xwu@cs.uvic.ca

Adam Murray
University of Ottawa
amurray@site.uottawa.ca

Margaret-Anne Storey
University of Victoria
mstorey@uvic.ca

Rob Lintern
University of Victoria
rlintern@uvic.ca

Abstract

Most traditional reverse engineering tools focus on abstraction and analysis of source code, presenting a visual representation of the software architecture. This approach can be both helpful and cost effective in software maintenance tasks. However, where large software teams are concerned, with moderate levels of employee turnover, traditional reverse engineering tools can be inadequate. To address this issue, we examine the use of software process data, such as software artifact change history and developer activities. We propose the application of this data confers additional information developers need to better understand, maintain and develop software in large team settings. To explore this hypothesis, we evaluate the use of a tool, Xia, in the navigation of both software artifacts and their version history. This paper introduces Xia, reveals the results of our evaluation and proposes directions for future research in this area.

1. Introduction

Reverse engineering is concerned with the analysis of existing software systems, with the aim of supporting software understanding, maintenance, reengineering and evolution activities through improved program comprehension. To facilitate understanding, traditional reverse engineering tools extract knowledge from source code and software documentation. However, this approach is rather limiting as often information concerning how the code was developed and the rationale for its design are lacking. Moreover, a piece of source code may be cryptic due to a lack of developer comments.

The traditional reverse engineering approach involves analysis of source code and related software artifacts; yet, this approach is both time consuming and may not be able to solve the problem without communication between maintainer and original developer. However, it is difficult to tell who the original developer was through source code analysis alone. To find out who last worked on the code the maintainer must turn to the version log of the software. Therefore, we believe the vast information generated in the software development process, which is usually stored in an associated version control tool, would provide useful information to support program understanding and maintenance.

In this paper, we introduce a reverse engineering approach that abstracts information from both the source code and version logs. A tool, called Xia, was developed to analyze and browse software artifacts and associated versioning information. Xia has been tightly integrated with a full-featured IDE, the Eclipse platform[6]. In Xia, advanced visual user interface techniques are used for browsing and interactively exploring software artifacts as well as the data in a CVS repository. A preliminary user study was conducted to evaluate both the usability and functionality of this tool.

Section 2 gives some background on version control systems, and elicits our problems, which are described in Section 3. Section 4 introduces our approach to the design and implementation of Xia. The details and results of our user study are described in Section 5. In Section 6, we outline our improvements to Xia as a result of lessons learnt from the user study. Finally, Section 7 concludes the paper.

2. Background on version control tools

Presently, most medium to large-scale software projects are developed and maintained in association with a version control tool. Version control tools contribute to software projects in the following ways: *software artifact management*, *change management* and *team work support*. *Software artifact management* involves definition and control of software artifacts (including source files, additional resources, and documentation). *Change management* keeps a record of artifact changes, and allows distribution of software versions. *Teamwork support* allows concurrent development and records team members' activities.

Some common features of version control systems can be summarized as follows:

- Data Repository: A version control system typically includes a repository, which is a centralized library of software artifacts. Clients or users of the version

Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)
1095-1350/04 $20.00 © 2004 **IEEE**

control system can get information about any of the software artifacts 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 commit, the version control tool assigns a new revision number to the changed artifacts, and records the related information of the commitment, such as time, author, the author's annotation, etc.
- Diff operation: Users may interpret comparisons between two revisions of selected artifacts using this operation.
- Get operation: This mechanism is provided for users to retrieve any revision of an artifact or any version of a project.
- Report operation: This operation is used to generate various useful reports about artifacts, such as history of a file, annotations, etc.

3. Problem statement

As a result of operations we discussed in Section 2, a large amount of information is generated and stored in the repository. The following questions are of interest, and are central to our discussion in this paper:

- What can this information mean to the software understanding and maintenance process?
- Can this information be used in a meaningful way to help with teamwork in software maintenance?
- If so, how can the presentation of this information assist software understanding and maintenance?

To investigate these questions, we conducted a survey of five version control systems. In this survey, we posed questions related to the functionality and ease of use of version control systems, as well as users' concerns related to programming in a team. Our results highlighted that although the features of version control systems are considered adequate to support the maintenance and development; the interfaces of these systems are not satisfactory.

We discovered that users find it difficult to understand and explore the information space of version control systems. 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-based GUI[6][14][25][27]. Such interfaces enable most of the core functionality by simple mouse-click and menu use; however, the resulting output is still unsatisfactory.

Besides the complaint of weak representation of version control systems, users also indicated their interests on certain information contained in version control systems. Our survey demonstrated that the most prominent concerns related to a maintenance or development task include:

- What happened since a developer last worked on the project (types of events, such as new file added, file modified, etc.)?
- Who made this happen?
- Where did this take place (location of the new file, change, deletion, etc.)?
- When did this happen?
- Why were these changes made (what is the rationale of the designer(s) who made the change)?
- How has a file changed (exact details of the change, as well as relationship to other files)?

These questions lead to the problem of understanding and exploring the version control information. We name this problem the "5W+H" problem for brevity, referring to the 5W's, what, who, where, when, why, and H, how, above. The presence of additional version control information assists program understanding, e.g. the comments committed with a change (answer to "Why"). Further information is also accessible, e.g. whom should I talk to about this method (answer to "Who" and "When").

We believe that reverse engineering should combine traditional source code analysis with version control knowledge extraction to further support software maintenance. In this paper, we elucidate evidence that version control knowledge improves software understanding for maintenance, and discuss how to enhance reverse engineering through this method.

We conjectured that applying information visualization techniques to a version control system might resolve problems regarding the understanding and exploration of version control information. Consequently, we investigated related research (tools for visualizing and exploring version control data) and found the following applications: Seesoft[7], Beagle[24], CVS Activity Viewer[5], and others[1][8][9]. However, these tools have rarely been evaluated and hence we are unable to tell how successful these visualization techniques can be for understanding and exploring version control information. In addition, 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.

4. Approach

In our approach, we elected to focus our tool on the version control system known as CVS[4]. CVS is freely available open-source software that is widely used. We believe the widespread user base will make it easier to evaluate the effectiveness of our tool, as users will be easier to find. Our previous experience[10][16] of plugging a visualization tool, SHriMP[19], into the Eclipse platform, encouraged us towards an approach of using the Eclipse platform as a framework for the integration of:

(1) The Eclipse CVS Plug-in, a CVS interface through which the CVS repository information could be accessed and retrieved;

(2) The Eclipse JDT (Java Development Tools) Plug-in, which provides the workspace information for a particular Java project and;

(3) The SHriMP visualization engine

Xia is the result of an integration and customization of these components.

The Eclipse CVS Plug-in[11] and the JDT Plug-in serve as data backends for Xia. The SHriMP visualization tool[20] is customized and used by Xia as a visual front-end for the back-end data. Figure 1 illustrates the architecture of Xia.

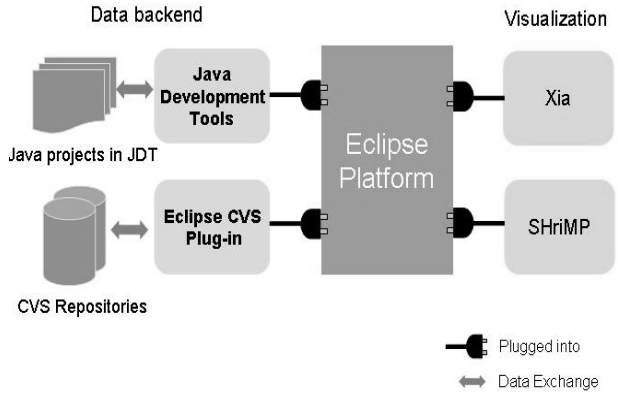

Figure 1. The architecture of Xia

The benefits of building Xia on these existing environments are three-fold. First, using the Eclipse platform allows us to share our results with people in the community. Second, robust visualization techniques of the existing tool, SHriMP, an achievement of our previous research in information visualization, could be re-used. And third, implementation was dramatically reduced as the developer could focus on improving visualization rather than struggling with data extraction. Data in a CVS repository could be retrieved through the API of the Eclipse CVS Plug-in, rather than direct interaction with the CVS repository.

In the following subsections, we first give some background of the supporting tools; then we look into the data we obtained from a CVS repository. Finally, we describe how we design the visualization in our tool to help answer the questions we raised before.

4.1. Supporting tools

4.1.1. The Eclipse CVS Plug-in. The Eclipse Platform provides a framework that allows third party tools to be developed as plug-ins to the platform. For example, the Eclipse CVS Plug-in is one such implementation. The Eclipse CVS Plug-in contains the most common CVS features and has a non-public API for accessing the repository data. The data integration of Xia was implemented using the CVS data retrieval methods of this API.

4.1.2. The Eclipse JDT Plug-in. The Eclipse JDT Plug-in provides a well-documented API, which gives us access to information about the software artifacts in a Java project. Software artifacts are organized into an easily understood hierarchy. For example, the "Children" of a Java class are fields, methods, and other classes. The relationships between these artifacts could also be found through the JDT search engine. This makes it quite easy to rebuild the software structure without parsing the source code. Integrating this component with Xia involved calling the JDT API.

4.1.3. The SHriMP visualization tool. SHriMP is a domain-independent information visualization tool developed at the University of Victoria. SHriMP employs advanced visualization techniques to assist browsing and exploration of large and complex information spaces. The component-based architecture of SHriMP enables easy integration of SHriMP with other tools[2]. As Xia reads data from CVS repositories, via the Eclipse CVS Plug-in, SHriMP components provide support for the visualization.

SHriMP provides browsing of data 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 useful for representing parent-child relationships in a hierarchical structure, as it makes efficient use of space. When a nested graph is combined with zooming techniques, the screen space is unlimited to the user; hence, large information spaces become more manageable for users. Zoomable nested graphs cover multiple levels of abstraction, providing a further benefit. By zooming in or out, users are able to see the highest level of structure as well as the lowest level of details. Arcs between nodes in the graph represent relationships between these nodes. For example, an arc between two "file" nodes may represent

the fact that a method in one file calls a method in another file.

4.2. Data acquisition

The CVS repository is a good resource of information for helping to answer the 5W+H questions. We believe that pertinent information can be obtained and visualized. For instance, the log message in CVS contains the record of each commitment, including:

- The author *who* made the commitment;
- The comments made by the author of *what* was changed and hopefully *why* it was changed; and
- The time and date *when* the file revision was created.

To understand *how* a change occurs, we propose using the *diff* function of CVS. The location of the changed file in the repository hierarchy helps determine *where* the change takes place. In our tool, the information we required was retrieved directly from both the CVS repository via the Eclipse CVS Plug-in, and the JDT in Eclipse, or the information was calculated from the retrieved data.

4.3. Data analysis and visualization

Data retrieved from a CVS repository and the JDT is visually represented in Xia, using the visualization techniques found in SHriMP. In the following subsections, we describe the visual representation of software artifacts, and associated revision details. Following this, we outline our method of interactively exploring this information. Finally, we summarize how our visualization techniques, specialized for the CVS domain, can help answer the 5W+H questions for a software maintainer.

4.3.1. Artifacts and attributes. By analyzing data from the JDT and the CVS repositories, we classified data into two categories, namely the software artifacts, and associated revision details. The software artifacts include file revisions, folders, and other code-level entities (such as methods) and are represented as nodes in our visualization. Revision details are handled by associating revision attributes with "file revision" nodes. (see Table 1.) These attributes reflect human activities that concern people in answering the 5W+H questions.

Attributes defined in Table 1 have been classified into two categories according to their data types, nominal and ordinal. Nominal attributes are strings whereas ordinal attributes have numeric or ordinal values. In the following sections, we discuss the visual representation of CVS artifacts and attributes.

Table 1. File Revision Node Attributes

Attribute Name	Data Resource	Data Type
File revision number	Retrieved	Ordinal
File revision tags	Retrieved	Nominal
Date of last commitment (check-in) of a file revision	Retrieved	Ordinal
Author who changed the file most recently	Retrieved	Nominal
Author who changes the file most times in a particular time period	Calculated	Nominal
Comments associated with each commitment	Retrieved	Nominal
Number of changes associated with a file revision	Calculated	Ordinal

4.3.2. Visual representation of software artifacts. A single file revision in the CVS repository is mapped to a single node in SHriMP. Likewise, a folder containing file revisions is mapped to a parent node of file revision nodes. Right clicking on a node brings up a menu with options to display the source code, attributes, and documentation for that node.

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 graphs in SHriMP, as illustrated in a screen shot of the CVS data in Figure 2. In Figure 2, parent folder nodes (shown in dark grey) encompass file revision nodes (shown in white). The outmost light grey nodes represent two versions of the same project. Node size relates to the size of the content (number of children) within the node, so the version on the left (which is graphically larger) contains more sub-nodes than the version on the right. Also, a node's screen location is based on the size of its contents – larger nodes to represent larger file revisions are placed in the left upper-most corner of a folder node, and smaller nodes are found as you move to the right and down.

4.3.3. Visual representation of attributes. In addition to the pre-existing SHriMP visualization techniques, we developed an **Attribute Panel** for showing and querying attributes associated with file revisions (see Fig. 3). The attribute panel concept, and the widgets it supports, was originally developed at the University of Maryland, and combined with the Treemap visualization tool[17]. The attributes reflect human activities that concern people in answering the 5W+H questions.

File revision attributes, and their values, are browsed by two means. The first is by placing a simple table within each node displaying this node's attributes

(accessed by right-clicking on the node). The alternative method involves using appropriate visual variables for the attributes[26]. For instance, using color, intensity, tool tips, size, and position to highlight nodes and accentuate their differences.

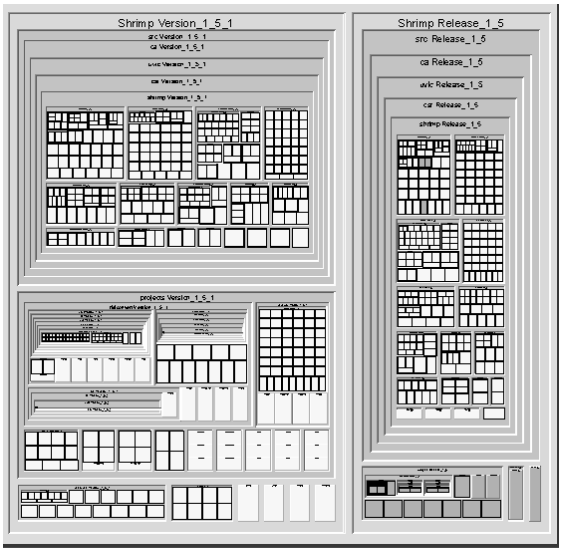

Figure 2. A nested graph showing two versions (light grey) of a software project. Versions contain folders (dark grey), which in turn encompass files (white) and other folders.

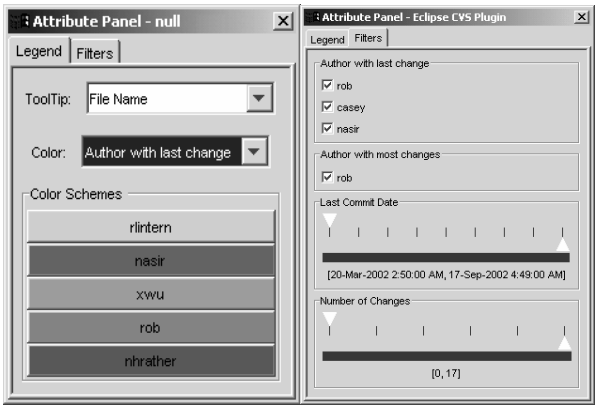

Figure 3. The Attribute Panel in Xia: The user can change colors associated with developers and how attributes are mapped to tool tips and can filter nodes

Tool tips provide instant messages that are easily perceived during browsing. In Xia, all attributes in the CVS domain can be viewed with tool tips. The user selects which of the attributes to show in the tool tips.

Colors may be used for both nominal and ordinal attributes, though different color schemes are necessary for each type[3]. For nominal attributes, each of the values may be assigned a distinct color; whereas ordinal attributes may use color intensity instead of different colors (see Figure 4). For example, in Xia, the date of commitment and number of changes are the two ordinal attributes that can be visualized using color intensity. These ordinal attributes are sorted in an old-to-new and few-to-more order respectively, and then each value is assigned an intensity of green (the default color). In our example, a more recent date is assigned a brighter green color. When assigning a color or intensity to the value of an attribute, nodes in Xia will change their color according to the attribute values. Figure 4 shows a screen shot of coloring nodes according to their ordinal attributes. The arcs between nodes enable people to focus on a specific task and keep track of its relationship to other files in the project. This kind of awareness is very important for teams to collaborate on maintenance and development work effectively, especially if there are many dependencies between the different artifacts that are being worked on.

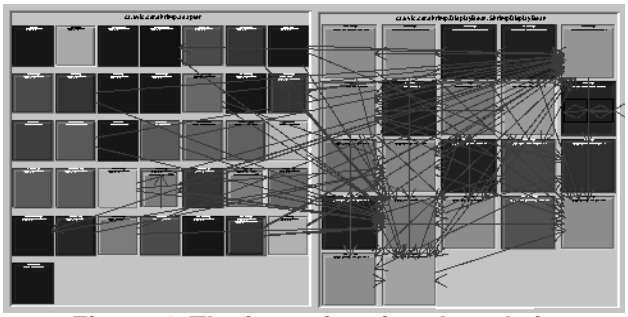

Figure 4. The intensity of each node is determined by the date of the latest commitment.

4.3.4. Interactive Exploration. The **Attribute Panel** also supports dynamic exploration using filters. Two kinds of filtering widgets have been developed for different attribute types. A checkbox filter, as illustrated on the right hand of Figure 3 is created for each of the nominal attributes. A checkbox filter consists of a set of checkboxes associated with each of the attribute values in the domain. An unchecked checkbox results in the corresponding nodes having equal attribute values being filtered from the screen. The other filtering widget, a double slider, is designed for ordinal attributes and is especially useful for dynamic queries. 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 select 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: Date of last commitment and Number of Changes. These two sliders could be used together to

perform a multi-variable query. For example, if a programmer wants to look at the file that changed most frequently in the past week, he/she would be able to get the result by setting the Date of Last Commitment slider to the corresponding range, and setting the Number of Change slider to its maximum value.

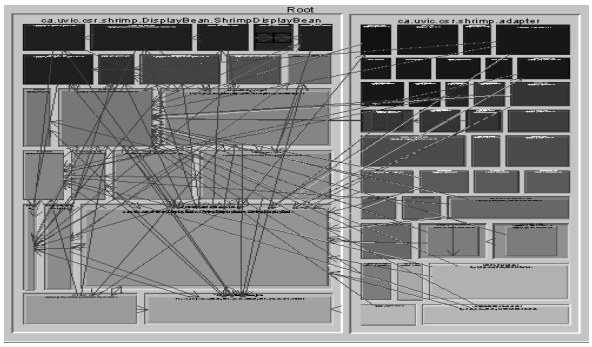

Figure 5. An ordered Treemap layout in which the number of changes determines the size of nodes and the position of nodes are ordered by their last commit date.

4.3.5. Ordered Treemap Layout. To provide a view that addresses the 5W+H problem at the entire project level, we adopted the Ordered Treemap algorithm created by the HCI lab at the University of Maryland[18]. Treemap is a 2-d space-filling representation designed for traditional tree structures. Besides the benefit of saving screen space, Treemap is also useful in that the size and order of the nodes can be mapped to selected data, an effective means of showing node attribute. By using the Ordered Treemap algorithm, we produced a Treemap layout in the SHriMP visualization environment. The Treemap layout was used in Xia for the purpose of visually interpreting the attributes of software artifacts. Two distinct features of the Treemap layout are the variation of the node size according to the associated numerical attribute and the repositioning of nodes according to their associated ordinal value. Figure 5 demonstrates a screen shot in which node size was adjusted according to the number of changes and the position of nodes are ordered by their last commit date. This ordering feature provides a comparable view for files in a project, hence answering the *When* question at the project level.

4.3.6. Summary of visualization features for CVS. Xia provides various ways to visualize data or derived data from the CVS repository and the JDT. In addition, relationships between files can be determined using information extracted from the Eclipse JDT.

The 5W+H questions can be answered by interacting with the features in Xia which includes the double slider filters, the different layout algorithms (such as the Treemap layout) effecting size and order of the nodes, the checkbox filters, tool tips, color and intensity. In addition to these features, Xia provides easy access to the source code, documentation (Javadoc) and comments in the CVS repository. The user can zoom into a node representing a file revision and switch between these different views. In Table 2 we summarize how these different features can be used to answer the 5W+H questions.

Table 2. Map of visualization techniques to questions of interest when working with CVS

Question	Visualization techniques
What	The name of the changed file can be shown using labels on the nodes (which are visible when users zoom in), or they can be shown using tool tips when the user brushes over nodes in the graph with a mouse.
Who	Can be distinguished using different node colors; filter by name using a checkbox. Tool tips could also be used to show the author's name.
Where	Nested within relevant folders in the layouts
When	Date can be shown using color intensity, tool tips. File revisions can also be filtered by date
Why	Rapid access to the code, CVS comments (in the attribute table) and documentation (by right clicking on the nodes, or by zooming in to an embedded view)
How	Access to the code, Javadoc and CVS comments by zooming on a file revision node. Tool tips, intensity, size and location could also be used to show number of changes to a file. Relationships between files are shown using arcs, which could be used to trace the impact of changes.

5. Evaluation

We conducted a preliminary user study to test both the functionality and usability of Xia. Before our study, our conjecture was that Xia could be used to resolve the 5W+H problem at both the file and project levels. 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 visualization techniques for answering these questions was also of much 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. Tasks were created based on plausible real world scenarios adapted from a real world project.

We did not compare Xia to other tools in this study. Although the literature contains references to several tools 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.

The following subsections describe in detail the participants, the procedure of the study, the tasks and general observations.

5.1. Subjects

Five graduate students in 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, one subject had experience as a software project manager. As is the case with maintainers performing real-life reverse engineering tasks, the subjects were not part of the original development team that created the source code used for the study, and hence were not familiar with this code. We did however provide an overview explanation of the source code as the participants may also have lacked knowledge 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.

5.2. Procedures

Following the pre-study questionnaire (to determine their previous programming and version control experience etc.), 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 would be used in a more realistic setting without time constraints. As a result of this, the actual time spent to complete the tasks varied among different participants, in the range of 30 to 90 minutes. Users were encouraged to think aloud so we could verify our interpretation of their actions[12]. Following completion of the tasks, further inquiry into the user's opinion of the tool was gathered through a post-study questionnaire.

5.3. Tasks

Two sets of similar tasks were assigned to participants corresponding to two different data resources: the data in the CVS repository and a "working copy" of this data in the programmer's own workspace. These two sets of data constitute a programmer's data in the real world. The tasks involved exploring the information space and answering questions related to teamwork and software history, including the 5W+H questions.

For example, one of the tasks asked the participant to name all programmers that 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. In regards to the "What" question, we asked the user to determine what kind of changes to a particular file have been made. 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 rationale behind a particular change.

In addition, we also posed some tasks that we believe are of interest to both project managers and programmers alike. For example, we asked the user to find the file that has changed the most times in the project. We hypothesized this measurement is useful for people looking for stable or active files in a project.

5.4. General observations

Participants successfully resolved most tasks. Some general observations were as follows:

- The visualization and exploration techniques provided by the **Attribute Panel** 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.
- 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 help 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 all defined "stable" and "active" in similar ways: 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 and to learn which programmers were working on files in the past and who was currently working on files.
- The participants were impressed by the immediate feedback the visualizations provided when they posed a new query, which was thought not possible in a command-line environment. For example, some queries could not even be posed using commands, e.g. sort the files by their file ages. As to the visualization techniques, some of the users had special interests in color schemes while others used filters more often.
- When we asked the participants to explain the rationale behind a particular change, one of the participants chose to browse the source code first, as most programmers usually do. He compared the two revisions of the source code, read the source code line by line, and tried to understand why the change was made. He gave up on the source code exploration after a few minutes, as he could not find the answer in the code. Then, he took another approach to look at the programmer's comments committed along with the file changes, and found the answer in the comments where the programmer who checked in the changes explicitly stated, "interface changed, more methods to implement".

Though the positive feedback is encouraging, we also noticed some deficiencies of the tool:

- Although the participants thought that the use of color intensity to interpret ordinal attribute values as being very useful, they also pointed out that it was hard to tell which was brighter or darker when two intensities were very close. In this case, they used tool tips for assistance.
- 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.
- 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.
- 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 will not be considered or displayed in the tool. This may lead to the loss of information.
- Participants considered the "diff" function – a comparison of two different file revisions very 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 on account of difficulties embedding Xia's Java Swing[23] GUI inside of Eclipse's SWT[13] GUI (a problem discussed by Rayside *et al.*[16]). 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.

5.5. Limitations of the Study

The study we conducted is a preliminary step to provide feedback on the use of a tool such as Xia. The number of users was sufficient to find many of the usability problems in our approach as well as helped us refine the design of the tool for future iterations in our work. The size of the code studied was small but the study required that the users gain some knowledge of the code in a relatively short time. In the next section we discuss some improvements we made according to the results of our user study.

6. Improvements

Based on our observations from the user study and from suggestions made by the users, we have subsequently made significant changes to the visualizations we provide, in particular with respect to graph layouts. We have an early prototype of this new layout. We describe how these new layout address some of the issues revealed in the user study:

- From our observations, we noticed that the organization of file revisions needs improvement. All file revisions should appear in the view regardless of whether they belong to a software version or not. Revisions that belong to the same file should be

clustered. In Figure 6, we show a clustering of revision of files.

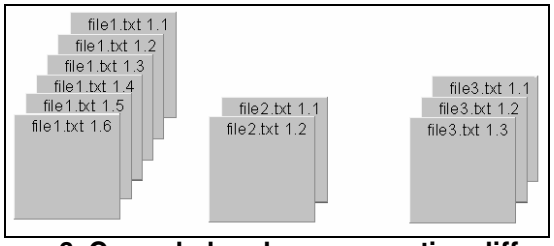

Figure 6. Cascaded nodes representing different revisions of the same file

- The arrangement in Figure 6 uses screen space effectively, but version information is occluded. Sometimes the user wants to be able to see which versions are available. The animation expands the cascaded nodes to a histogram so that a column presents each file. Taller columns therefore have more revisions. Furthermore, to gain an overall picture of software versions, file revisions that belong to a particular version need to be related. We achieve this by connecting files related to a particular version using colored lines as shown in Figure 7. Alternatively, the histogram can be morphed to the nesting approach we showed in Section 2.

This new graph layout is still under development and we will evaluate it in our next round of user studies.

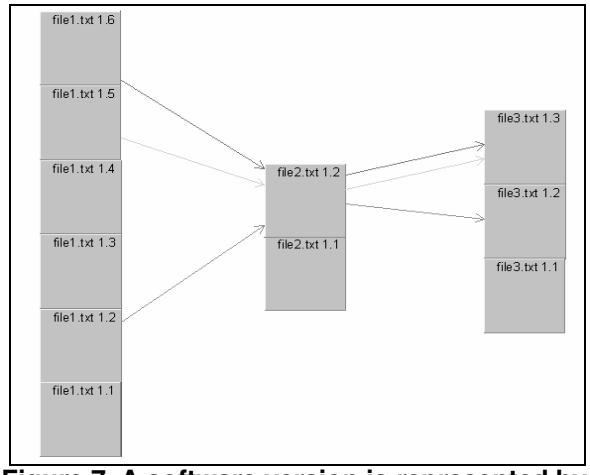

Figure 7. A software version is represented by connecting the file revisions belonging to this version. For example, version 2 contains revision 1.5 of "file1.txt," revision 1.2 of "file2.txt" and revision 1.3 of "file3.txt." Light grey arcs connect them.

Future work also involves implementing our own "diff" view, as the users indicated it was very important for their work. Eclipse has a useful design for browsing comparison results, which can potentially be adapted to our tool design.

Another version control tool, Subversion[21] has drawn our attention recently. Subversion is considered a "compelling replacement" for CVS and has more features such as supporting atomic transactions and versioning directory information. Subversion has also been integrated with the Eclipse platform in a plug-in called Subclipse[22]. We plan to apply visualization techniques to Subversion in the future, and the integration of Subversion and Eclipse will reduce our workload in this effort. The technical architecture of this approach is similar to that of Xia, using the Eclipse platform as the framework.

We also observed that individuals were interested in discovering information related to a certain modification request. A modification request always results in logically related changes to several artifacts - the rationale of these changes is identical. Xia has a limited ability to mine for artifacts belonging to a certain modification request in current stage. One possible solution is using the time slider to discover the group of files submitted at the same time, assuming changes related to a certain modification request were submitted together. However, practical situations are complex and our assumption may not be tenable. We believe that analyzing the documentation – in which the modification requests are documented – as well as code pieces would better solve this problem. Also, other version control tools, such as the Rational ClearCase[15], provide the management of modification requests and associated changes. Integrating Xia with Rational ClearCase is feasible, as ClearCase has already been integrated with Eclipse.

7. Conclusions

This paper describes a visualization tool, Xia, used to browse and explore the software with its version history and associated human activities. Xia is integrated with the Eclipse platform, which includes a modern IDE as well as an interface to the version control tool, CVS. This integration makes the reverse engineering process easier than traditional methods, and provides additional information, such as version logs, to better support program understanding. Integration also increases the possibility of Xia being widely adopted in the real world, and hence providing us with more feedback on the effectiveness of using visualization of version control information to support reverse engineering and program understanding.

We also discussed the evaluation of Xia, and the resulting observations. Observations from our user study led us to further design improvements for the tool. Through the study, we observed that the visualization techniques applied to version control show great potential

for assisting in the information exploration process, hence improving the efficiency of program understanding of the whole team in software maintenance and development.

8. References

[1] Ball, T. A. and Eick, S. G. 1996. Software visualization in the large. *IEEE Computer*, vol. 29, no. 4, pp. 33-43.

[2] Best, C., Storey, M.-A. and Michaud, J. 2002. Designing a Component-Based Framework for Visualization in Software Engineering and Knowledge Engineering. In *Proceedings of the 14th international conference on software engineering and knowledge engineering*, pp323-322.

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

[4] CVS 2004. The CVS website: http://www.cvshome.org/

[5] Dourish, P. 2002. "Visualizing Software Development Activity": http://www.isr.uci.edu/projects/augur/

[6] Eclipse Platform, 2004. The Eclipse Platform Subproject Webpage: http://www.eclipse.org/platform/index.html

[7] Eick, S. G., Steffen, J. L., and Summer, E. E. 1992. Seesoft – A tool for visualizing line oriented software statistics. *IEEE Trans. Software Engineering*, vol 18(11), pp. 957-968.

[8] Fisher, M., and Gall, H., 2003. MDS-Views: Visualizing problem report data of large scale software using multidimensional scaling, in *Proceedings of the International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA)*, Netherlands.

[9] German, D., Hindle, A., and Jordan N., 2004. Visualizing the evolution of software using softChange, In *Proceedings of Software Engineering Knowledge Engineering (SEKE'04)*, Banff.

[10] Lintern, R., Michaud, J., Storey, M.-A., and Wu, X. 2003. Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse. To appear in *Proceedings of Software Visualization 2003*.

[11] McGuire, K. 2002. VCM 2.0 Story (article in Eclipse website: http://dev.eclipse.org/viewcvs/index.cgi/platform-vcm-home/docs/online/vcm_story2.0/vcm2story.html)

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

[13] Northover, S. 2001. SWT: The Standard Widget Toolkit, http://www.Eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html

[14] Perforce 2004. Perforce website: http://www.perforce.com/

[15] Rational ClearCase 2004. Rational ClearCase website: http://www-306.ibm.com/software/awdtools/clearcase/

[16] Rayside, D., Litoiu, M., Storey, M.-A., Best, C. and Lintern, R. 2002. Visualizing Flow Diagrams in Websphere Studio Using SHriMP Views (Visualizing Flow Diagrams). *Information Systems Frontiers: A Journal of Research and Innovation* (Kluwer, ISSN 1387-3326), vol 4 (4)

[17] Shneiderman, B. 1992. Tree Visualization with Tree-maps: A 2-d space-filling approach. *ACM Transactions on Graphics*, vol 11(1), pp. 92-99.

[18] Shneiderman, B. and Wattenberg, M. 2001. Ordered Treemap Layouts. In *Proc. IEEE Symposium on Information Visualization 2001*, 73-78. IEEE Press, Los Alamitos, CA

[19] SHriMP 2004. SHriMP Website: http://shrimp.cs.uvic.ca/

[20] Storey, M.-A., Best, C., Michaud, J., Rayside, D., Litoiu, M. and Musen, M. 2002. SHriMP views: an interactive environment for information visualization and navigation. In *Proceedings of CHI 2002 Conference, Extended Abstracts on Human Factors in Computer Systems,* Minneapolis, Minnesota, USA, pp. 520-521.

[21] Subversion website: http://subversion.tigris.org/

[22] Subclipse: A Subversion Eclipse Plug-in. Project homepage: http://subclipse.tigris.org/

[23] Swing 2003. The Swing Connection, http://java.sun.com/products/jfc/tsc/

[24] Tu, Qiang and Godfrey, Michael 2002. An Integrated Approach for Studying Software Architectural Evolution. In *Proc. of 2002 International. Workshop on Program Comprehension (IWPC-02).*

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

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

[27] WinCVS 2004. WinCVS website: http://www.wincvs.org/