On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and a Framework

Margaret-Anne D. Storey Davor Čubranić Daniel M. German *
Software Engineering Group, Dept. of Computer Science
University of Victoria

Abstract

This paper proposes a framework for describing, comparing and understanding visualization tools that provide awareness of human activities in software development. The framework has several purposes – it can act as a *formative evaluation* mechanism for tool designers; as an assessment tool for potential *tool users*; and as a *comparison tool* so that tool researchers can compare and understand the differences between various tools and identify potential *new research areas*. We use this framework to structure a survey of visualization tools for activity awareness in software development. Based on this survey we suggest directions for future research.

CR Categories: D.2.2 [Software Engineering] Tools and Techniques–Computer-aided software engineering (CASE); H.5.2 [Information Interfaces and Presentation] User Interfaces–Evaluation/methodology.
General terms: Human factors, management.
Keywords: Visualization, Computer Supported Collaborative Work, Software Development, Awareness.

1 Introduction

Software engineering is a human-driven and human-intensive activity. Most medium- to large-scale projects involve multiple software developers that may or may not be co-located. Recently, there has been much work in developing collaborative development environments that provide support for coordination and communication during software development [Hupfer et al. 2004]. A key issue in any collaborative activity is *awareness*, or "knowing what is going on" [Endsley 1995, p. 36]. More precisely, awareness is "an understanding of the activities of others, which provides a context for [one's] own activity" [Dourish and Bellotti 1992]. Awareness encompasses knowing who else is working on the project, what they are doing, which artifacts they are or were manipulating, and how their work may impact other work.

In distributed collaborative work, maintaining awareness is considerably more difficult. Various tools and novel techniques have been developed by the software engineering research community to provide awareness during software development. Several of these tools rely on visualization techniques to either augment existing views in the development environments or to provide specialized views of human activities combined with software artifact information. The reverse engineering community also investigates if visualization can assist in design recovery and evolution reconstruction by analyzing how human activities impact code, data, and the system's architecture.

Despite the prolific development of these tools by the research community, it is difficult to understand how the various tools compare or differ, and which potential research areas remain unexplored. In an attempt to gain better understanding of this research and future opportunities, we present a framework of design issues for software visualization tools that provide awareness of human activities in software development. In Section 2 we provide background on awareness in computer-supported collaborative work in general and distributed software development in particular. The framework is introduced and elaborated in Section 3. Using the main dimensions of the framework, a survey of selected awareness tools for software development is provided in Section 4. In this section, the framework is used to compare the characteristics and features of the surveyed tools. Several future research directions are proposed in the Section 5. Finally, in Section 6, we discuss the validity, limitations and applications of the proposed framework.

2 Background and related work

Maintaining awareness becomes particularly difficult in distributed collaborative work because one of the primary awareness mechanisms in face-to-face collaboration—observing colleagues' posture and movement—is not possible when the team is not collocated. Supporting this *consequential communication* [Segal 1995, cited in [Gutwin and Greenberg 2002]] has been the focus of activity support in real-time groupware, with mechanisms such as radar views [Gutwin and Greenberg 2002] or shared feedback [Dourish and Bellotti 1992].

These solutions mimic existing face-to-face awareness mechanisms in their target activities (e.g. collaborative drawing in radar views). However, they do not transfer to distributed software development in a straightforward way, if at all. Software development does not involve the manipulation of physical objects; we cannot readily tell on which artifact a developer is working just from where he or she stands or from the lever that is being pulled. Writing a piece of software is also slower than, for example, drawing a picture, and development teams are often too large to observe all members. Arguably, consequential communication is not the primary awareness mechanism even in collocated software development and is unlikely to be in the distributed case either.

An alternative source of awareness information in software development are the artifacts themselves. Seeing how they change, or *feedthrough* [Dix 1994], makes it possible to get a sense of the ongoing activity in the workspace. Feedthrough awareness has been an unintended benefit of tools used in distributed software development. Configuration management (CM) tools are designed to support concurrent development by multiple developers and provide mechanisms to back out of unwanted changes, but, as Grinter [Grinter 1995] reported, they also create visibility into the development process by providing a history and authorship of all changes to the system. In effect, the code repository becomes an organizational

*e-mail: {mstorey, cubranic, dmg}@uvic.ca

memory that can be accessed to find out what other developers have done. However, as Grinter pointed out, because CM tools provide low-level visibility into the actions of others, it is difficult to get from them a system-level overview or a sense of historical trends. A number of approaches have been developed that use data from the CM system to create a visual representation of a software's evolution and improve awareness, starting with Gulla [Gulla 1992] and Eick et al. [Eick et al. 1992] in 1992. We will visit some of these representative and historical examples in the following sections.

3 Framework

The framework we propose for describing visualizations of human activities in software engineering has five key dimensions: Intent, Information, Presentation, Interaction and Effectiveness. These dimensions were derived from an in-depth analysis of the papers that describe research and tools in this area. Some of the dimensions overlap the dimensions proposed by Price et al. in their well-known software visualization paper from 1992 [Price et al. 1992], but as our framework is specific to human activities we have tailored and refined the framework accordingly. Along the lines of the Cognitive Dimensions Framework proposed by Green [Green 2000], the role of our framework is also to act primarily as a *discussion tool*. It can serve several purposes: 1) as a *formative evaluation* tool to guide tool designers as they develop a new tool; 2) for potential *tool users* to be able to assess the value and application of a particular tool; and 3) as a *comparison tool* so that tool researchers can compare and understand the differences between various tools and identify potential *new research areas*. In this section, we provide an overview of the key dimensions in the framework.

3.1 Intent

Intent captures the general purpose and the motivation that led to the design of the visualization:

Role. The most important aspect of this dimension is identifying who will use the tool. One key role involves the ***developers*** in a software ***team***. Important attributes of the developers include whether they are newcomers to the project, their level of involvement and whether they are co-located with other team members. Some tools are designed for small tight-knit teams whereas others may be designed to support large distributed teams. ***Maintainers, reverse engineers*** and ***reengineers*** will also want to explore human activities that have occurred in the past to guide present and future tasks. Another key role is the ***managers*** who need to gain an understanding of the human activities in the project. ***Testers*** and ***documenters*** may also find it relevant to know who has been developing the project and which changes were made since the last release. Finally, ***researchers*** may wish to investigate human activities and processes across the lifespan of multiple projects.

Time. Another characteristic to contemplate is whether the tool provides awareness of activities occurring in the ***present*** or in the ***recent past*** and whether the tools provide information about the ***history*** of the project. The level of granularity at which a tool explores the past varies according to the software engineering activity. For forward engineering, tools may need to communicate changes that happen on a daily basis (such as commits to the version control system), while for maintenance or reverse engineering support, tools may need to reveal changes over longer periods (such as monthly releases of the software).

Cognitive support. Cognitive support is a term used to capture how a tool or artifact can make human cognition easier or better [Walenstein 2003]. Tools provide cognitive support through "redistribution" when cognitive resources or processes are moved outside the head and onto an artifact or tool. Redistribution occurs when a complex or cognitively tedious task is transformed into a task that can be done more quickly or easily. Awareness can often be achieved by providing answers to particular questions that a user may have. Walenstein describes how cognitive support can be provided by visualization tools when parts of the problem space may be reified in the visual representation. Consequently, many awareness tools do provide visualizations which are geared at answering specific questions about human activities and the artifacts manipulated.

Wu et al., through a survey with developers and managers, identified specific questions that they need to be answered using awareness tools [Wu et al. 2004b]. These questions were also cited as important elements of workspace awareness in [Gutwin and Greenberg 2002]. These questions include: *Who* is or has been working on the artifacts and *who* is the person responsible for or expert in a particular part of the system? *What* happened since a developer last worked on the project (types of events, such as new file added, modified, difference between files etc.)? *Where* did this take place (location of the new file, deletion, etc.)? *When* did this happen? *Why* were these changes made? *How* has a file changed and is there a relationship with other files? Tools can also provide cognitive support when the user does not have specific questions but is trying to gain *insight* about the human activities in the project. The many questions that the various user roles may ask and the reasons for wishing to gain insight about the human activities can be roughly classified into four categories: ***authorship, rationale, time*** and ***artifacts.*** Consequently we consider how tools can provide cognitive support for discovering information according to these four categories.

3.2 Information

This dimension refers to the data sources that a tool uses to extract relevant awareness information:

Change management. Most medium to large-scale software projects are developed and maintained in association with a version control or a configuration management tool. *Configuration management* tools provide support for building systems by selecting specific versions of software artifacts [Grundy 2001]. *Version control* tools contribute to software projects in the following ways: software artifact management, change management and team work support [Wu et al. 2004b]. 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 ***branching*** activities, and allows distribution of software ***revisions*** and ***releases.*** Some environments also use information from a developer's ***local history*** to provide awareness to other members.

Defect tracking. Many larger software projects rely on tracking tools to help with the management of ***defects*** and ***change requests.*** Such systems often store metadata on who is assigned a task and when it has been or should be completed.

Program code. Many tools present awareness information concerning ***files***, ***modules*** and ***components***. Others give more detailed views of ***syntactic units*** such as packages, classes, methods, functions, data types etc. ***Semantic analysis*** may be used to give relevant information to the developer and manager of the impact of changes to other parts of the program code.

Documentation. ***Design*** and ***requirements*** documentation are sometimes used to store projected and actual human roles in a project. ***Architecture*** level documentation and ***test cases*** may also describe human involvment. However, many projects do not specify human involvement in these documents and hence they are seldom used by awareness systems.

Informal communication. Informal communications, such as ***emails*** and ***instant messages,*** can help support awareness during collaboration and coordination tasks. Analysis of this information can also help uncover the intent behind past human activities.

Derived data. Awareness tools may present derived data either from a ***single data source*** or from ***multiple data sources***. For example, a tool may calculate which files were changed during a single transaction or which files are related. If CVS is used, the notion of a transaction may not be explicitly recorded but a transaction can be inferred by analyzing the documentation and file check-in times. Also, events that are similar may be aggregated and displayed as barcharts and histograms. An analysis of the available data sources can improve the awareness tool's ability to answer specific questions for the different user roles.

3.3 Presentation

Presentation refers to how the tool or proposed tool presents the extracted and derived information to the various user roles:

Form. The tool may present awareness information using a combination of ***text***, ***hypertext*** or ***graphics***. It is important to consider which of these three different visual electronic forms are most appropriate.

Kinds of views. Many tools provide awareness information in the form of ***annotations on existing views*** in a software environment. They may use visual variables or icons to emphasize the owner, state or history of a software artifact. ***Statistical views,*** such as barcharts and histograms, can provide comparison and analysis of human activity information. ***Graph views*** can also be used to display relationships between human and software artifacts. Nodes may represent software artifacts, and edges may represent semantic relationships or relationships to various versions and releases. In some cases, humans may be represented explicitly as nodes in the graph but more often human activity is shown as an attribute of the software artifacts and displayed with a visual variable such as color or shape. Some tools may also provide customized ***special views*** to provide cognitive support for particular information seeking or understanding tasks. An example of a special view is a matrix view which may be used to show trends and evolution patterns.

Techniques. Whether the tool provides annotations on existing views or specialized views, they will both use some ***visual variables*** such as colour, position, size, transparency, and map those to appropriate human activity attributes. It is important that the mapping from the visual variables to the attribute data types is appropriate and meaningful given the human visual perception system [Ware 2000]. ***Animation*** or motion can also be used effectively—for example to animate the evolution of a software project. Finally, we consider tools which rely on either user- or tool-generated ***abstractions*** in communicating awareness information. A tool or user may decide to group a set of developers into a subteam or group a set of software files into a module for the purposes of simplifying the visualization.

3.4 Interaction

This dimension refers to the interactivity and liveness of the tools:

Batch/Live. An important consideration is whether the tool operates ***offline*** or ***online*** [Froehlich and Dourish 2004]. Some offline tools require that the user write scripts to batch queries on a repository of information. The tool then displays the queried information using static graphs. Other tools are online and provide updated displays of the information to the users on demand.

Customization. Effective interaction to suit particular user needs will normally require a high degree of customization. This characteristic addresses whether the available views can be further manipulated and to what extent they can be customized. ***Saving customizations*** and ***sharing customizations*** across team members may also be important.

Query mechanism. Some tools require special purpose ***languages*** to specify queries. Others allow the user to visually specify the queries through the use of specialized ***filter widgets*** (such as double sliders, or checkboxes) or by interacting with the visualization directly (such as selection or brushing).

View navigation. How the user navigates the displayed information is important—especially for tools with specialized views. Successful navigation requires that the user maintain orientation so that they know where they are and can decide where to go next. The use of an ***overview*** for ***detailed views*** can be used to provide orientation and to directly support navigation in the information space. Navigation can alternatively be supported by a ***zoomable user interface*** and ***hypertext***. Another important consideration is that the user may need to compare two views ***side-by-side***. The facility to see multiple views at once provides cognitive support [Walenstein 2003] as it reduces the memory load on the user and redistributes some of the required cognition from the user to the tool. To improve the usefulness of multiple views, views should be ***coupled***.

3.5 Effectiveness

This dimension captures the feasibility of the proposed approach, whether it has been evaluated and whether it has been deployed:

Status. Some researchers propose approaches that have not yet been implemented. This characteristic captures whether the system has been partially or fully ***implemented*** and the robustness of the tool. Tool ***availability*** is also important—so that other tool designers and researchers can evaluate it. Froehlich and Dourish also mention the importance of ***interoperability*** [Froehlich and Dourish 2004]. The assumption is that an awareness system will have to work in conjunction with some other tool, be it a software development or reverse engineering environment. A typical restriction of many tools is that they only work with, for example, CVS or Bugzilla. For ***scalability*** we must consider if the tool supports large software projects. If the technique does not appear to scale, it may be the implementation which does not scale rather than the technique.

Cost. The adoption of any tool has a cost associated with it. ***Economic cost*** is a key concern, in addition to other costs such as the cost of ***installing*** the tool, ***learning*** how to use it, and the costs incurred during its ***usage***.

Evaluation. A tool that has been formally evaluated and compared to other approaches will more likely be adopted than one that has not. It is very common for these tools to be evaluated by the designers through informal ***case studies***. The complexity and size of the software in the case study is very important to consider. When a new tool has been evaluated with users other than the tool designers (i.e. in ***user studies***), confidence in the tool's benefits will be further increased. If the tool has been deployed and subsequently ***adopted***, then the tool has been evaluated through its usage. The rate of adoption can be an important indicator of the usefulness of a tool. However, lack of adoption does not necessarily imply that the tool is not effective as adoption is affected by many forces.

4 A survey of tools

In this section we describe several awareness tools for software engineering using our proposed framework as a template. The framework highlights not only what the tools do but also what they do not do well. The tools we review do not encompass an exhaustive list, however, they are representative of the kinds of tools that are developed in industry and academia to support awareness during software development. Moreover, we only focus on tools that specifically provide awareness of human activities.

4.1 Seesoft

Intent: Seesoft is one of the earliest tools that visualized program history from version control system's data [Eick et al. 1992]. It is a generic tool for visualizing statistics associated with lines in text

Dimensions	Characteristics	Features
Intent	Role	Team, Developer, Maintainer, Reengineer, Manager, Tester, Documenter, Researcher
	Time	Present, Recent Past, Historical
	Cognitive Support	Authorship, Rationale, Time, Artifacts
Information	Change management	Local History, Releases, Branching, Revisions
	Program code	Modules/components, Syntactic units (e.g., files), Semantic analysis
	Defect tracking	Defects, Changes
	Documentation	Requirements, Test cases, Design, Architecture
	Informal communication	Email, Instant messages
	Derived	Single source, multiple source
Presentation	Form	Text, Hypertext, Graphical
	Kinds of views	Annotated views, Statistical views, Graph views, Special views
	Techniques	Visual variables (hue, transparency, position etc), Animation, Abstractions
Interaction	Batch/Live	Offline, Online
	Customization	Level of customization, sharing and saving customizations
	Queries	Query language, Filter widgets
	View navigation	Overview+detail, Zoomable views, Coupled views
Effectiveness	Status	Implemented, Availability, Scalability, Interoperability
	Cost	Economic cost, Installation, Learning, Usage
	Evaluation	Adopted, Case study, User study

Table 1: Summary of the framework.

files, whose goal was to develop techniques for visual representation of large amounts of code for the purposes of code exploration and project management.

Information: For activity awareness, SeeSoft uses a version control system to provide data about authorship, age, and description of revisions. It also uses the contents of the files to create a visual map of the software.

Presentation: The key characteristic of Seesoft is its line-based visualization that maps each line of source code into a thin row (as small as one pixel high) on the screen, with files comprising the system arranged in columns across the screen. The colour of each row represents a value of the attribute that is being visualized, such as age or developer who authored it (see Fig. 1).

Figure 1: A screenshot of Seesoft (from [Ball and Eick 1996])

Interaction: The visualization is interactive and a user can easily select with a mouse only a subset of information to be displayed (for example, to colour only the lines, across all files, that were created together). The technique employed is "brushing": simply moving the mouse over an entity in the view (for example, a line of a file or a developer in the colour scale) selects it and updates the rest of the view. Clicking in the view serves as a zoom function and shows the text of the corresponding lines in the files.

Effectiveness: Seesoft's authors report informal field use in their organization, but no details of evaluation are reported.

4.2 VRCS

Intent: VRCS [Koike and Chu 1997] was designed to facilitate version control and module management. For developers, it provides a visual representation of file versions to facilitate building an entire system. The implementation they describe lacks support for multiple authors.

Information: Versions are explicitly modeled, and relationships between file versions are shown to facilitate compilation builds. Compilation file dependencies are extracted from make operations.

Presentation: In VRCS, each version of the history is represented as a 3D tree showing module and file relationships in the x and y dimensions and time in the z-axis. Files that are in the same module are placed close to one another in XY plane. Each major release is represented as a sphere in the center of the version tree. Links between version nodes show which ones should be compiled together (see Figure 2).

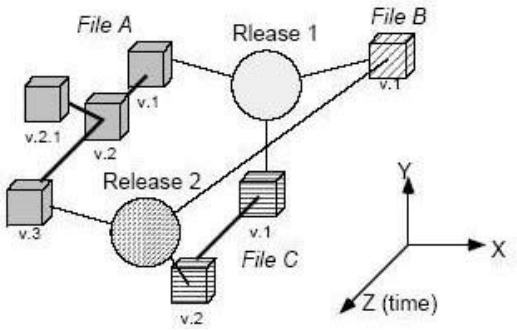

Figure 2: A graph created with VRCS (from [Koike and Chu 1997])

Interaction: For the most part the views are static, but the graphical representation of files and versions can be used for selecting which files to check in and out.

Effectiveness: VRCS was evaluated with 10 graduate students. The main result was that the interface facilitated a faster check-in. It interoperates with the RCS version control system.

4.3 Tukan

Intent: Tukan [Schümmer and Haake 2001] provides orientation and activity awareness. It is particularly useful at helping developers find knowledgeable people to work with and at avoiding conflicts.

Information: Tukan does program analysis to determine which program artifacts are semantically related. It also extracts version information from the ENVY version control system. This information is aggregated to determine the severity of potential conflicts.

Presentation: This tool presents a graph of artifacts (e.g. methods, instance variables) and weighted relationships (e.g. composition, inheritance, creation, task). Artifacts that are related are placed next to each other in the graph. An awareness browser uses weather symbols to show the severity of the conflicts (see Fig. 3.) In the resource browser, artifacts are annotated with a history of their state to indicate how "close" they are to the user's current focus.

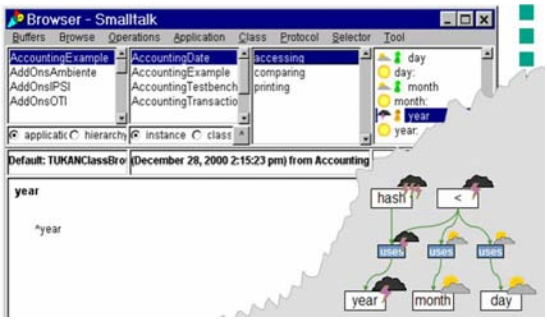

Figure 3: A screenshot of Tukan (from [Schümmer and Haake 2001])

Interaction: Tukan is an online system and has a synchronous cooperative code editor which uses colour coded cursors to provide context for other navigation. The system adapts the visual cues as the users browse and update the code.

Effectiveness: Tukan was developed as a plug-in for the Smalltalk system. It was evaluated by the designer's development group only.

4.4 ADVIZOR

Intent: Eick et al. demonstrated a series of visualization techniques to help managers understand and manage the software change process [Eick et al. 2002]. These visualizations are built on top of the general visualization framework called Advizor^{TM}, a commercial product.

Information: The visualizations use data drawn from the version control system, bug tracking system, and source code files.

Presentation: A wide range of views are used to present the data, from matrix displays, to 2D and 3D bar charts, pie charts, zoomable text displays, and graphs (see Fig. 4).

Interaction: All views are interactive and linked together, so that selection in one view causes updates in all other views.

Effectiveness: The authors reported the tool's application on the full history of a 15-year-old software system for telephone switches, comprising approximately 100 million lines of code.

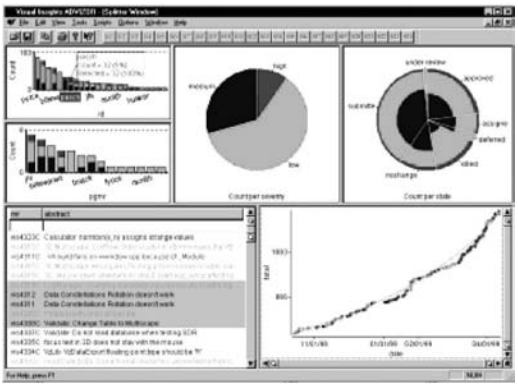

Figure 4: A screenshot of ADVIZOR (from [Eick et al. 2002])

4.5 Xia/Creole

Intent: The goal of the Xia tool is to give visual insight into version control activities that have occurred in the past [Wu et al. 2004b]. It provides an overview of human activities in the context of architectural views of the system and displays architectural differences between two versions. It was specifically designed to answer questions concerning the authorship, time and location of changes made in code. Its visualizations have been integrated into the Creole visualization plug-in for Eclipse [Lintern et al. 2003]. Using Creole, it is possible to generate custom views of the human activities within the context of the software architecture in addition to call graph and data flow views.

Information: Creole extracts information from the CVS version control system and combines it with semantic analysis on the program code to infer dependencies in the software architecture.

Presentation: Xia uses hierarchical graph views to display differences between the architecture and artifacts of two versions, see the left hand side of Fig. 5. The right hand side of Fig. 5 shows an example of the graph-based views in Creole where visual attributes, colour and position, are mapped to attributes of the information extracted from CVS. In this view, colour is used to distinguish new, changed, deleted and unchanged code with previous versions.

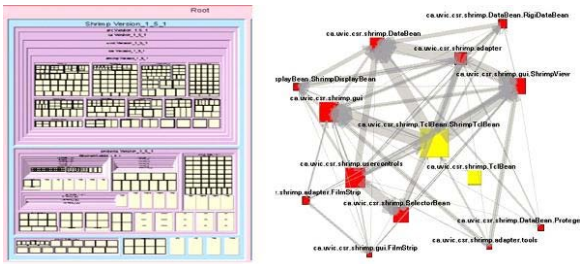

Figure 5: Two views from Xia and Creole respectively (from [Wu et al. 2004b; Lintern et al. 2003])

Interaction: Creole and Xia are both interactive and customizable, however, the views do not update according to current activities in the development environment. This tool provides a number of query widgets to support exploration of the data. For example, check boxes can be used to show data according to selected authors, double sliders are used to display data according to a range of dates and number of check-ins. Tooltips can also be customized to show different attributes such as author, file creation date, check in time, and number of check-ins. Creole provides overviews, a zoomable

user interface and hypertext to support navigation.

Effectiveness: The design of Xia was preceded by a survey to identify requirements for the tool features. A small user study was conducted with 5 subjects. Although the results were anecdotal they were useful in suggesting improvements for the tool. Creole and Xia have both been integrated with Eclipse as plugins. Creole is available for download from *www.thechiselgroup.org/creole*.

4.6 Palantír

Intent: The goal of the Palantír [Sarma et al. 2003] tool is to provide awareness for distributed software development teams. The key issue they explore is how their tool can be used to avoid coding conflicts.

Information: Palantír extracts information from a number of popular version control tools. It also extracts the difference between files by comparing the number of lines changed.

Presentation: Palantír uses color annotations on file names in the resource view to show the current editor of files. The resource tree view can also be enhanced with horizontal bars indicating the severity of ongoing and committed changes. Another view is the hierarchical view which shows pairwise differences according to conflicts between each possible pair of authors (see Fig. 6).

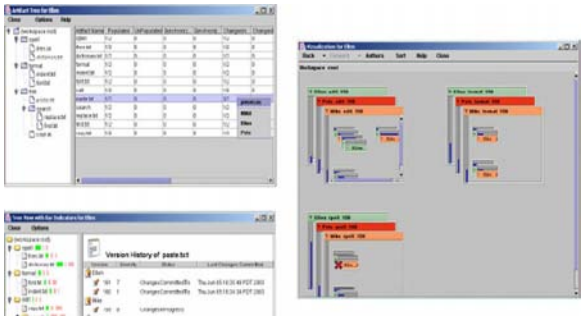

Figure 6: A screenshot of Palantir (from [Sarma et al. 2003])

Interaction: The tool is interactive and provides information as it is needed by using a "push" mechanism rather than a "pull" approach. Cues will inform the user about completed or in progress changes such as add, delete, remove and move activities. The tool supports zooming within the chat view and the user can compare two versions in the hierarchical view.

Effectiveness: Palantír has not been formally evaluated in a user study. It is available as an Eclipse plug-in and as a standalone tool from *http://www.ics.uci.edu/~asarma/Palantir*.

4.7 Jazz

Intent: Jazz [Hupfer et al. 2004] is a "collaborative development environment" to enhance and enrich collaboration in small, informal software development teams. Jazz has several features to support awareness of team member activities in addition to screen sharing.

Information: Jazz extracts activities from the environment's user interface and the local history to monitor how the source code is manipulated. It also relies on informal information such as who is online and their status.

Presentation: The key visual feature of Jazz is the *Jazz band* which provides peripheral awareness of the status and activities of other team members. The file names in the resource view in Eclipse are enhanced using color and icons to show the status of the resources (whether they are being edited, checked-in etc). Tooltips also provide responsibility information. Chats are visibly anchored in the code, thereby providing collaboration in context (see Fig. 7).

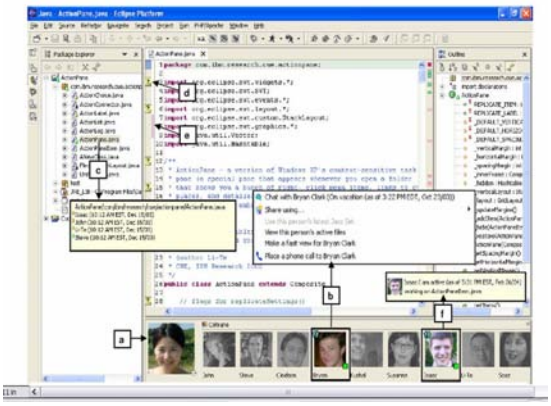

Figure 7: A screenshot of Jazz (from [Hupfer et al. 2004])

Interaction: Jazz provides up-to-the-minute awareness information by monitoring and displaying information as activities occur. Team members can add or drop members from the Jazz Band on demand. Jazz provides the user with limited controls for determining what is displayed but does not provide support for running queries or other filters. Views in Jazz are coupled and a developer can navigate large chats using hypertext-like links.

Effectiveness: Since it is a fairly recent tool, it has not been evaluated in a formal user study nor has it been deployed. Although it is a plug-in for Eclipse, it is not currently available.

4.8 softChange

Intent: The main goal of softChange is to help programmers, their management and software evolution researchers in understanding how a software product has evolved since its conception [German et al. 2004].

Information: softChange extracts the metadata from a version control system (CVS) and its corresponding defect tracking system (Bugzilla) and correlates both; it also extracts the different revisions of each of the files and does syntactic and semantic analysis (extraction and comparison of semantic units and comments between two revisions of a given file). It then tries to classify changes (defect fixes, changes in documentation, addition of new features, etc) based on the available information.

Presentation: softChange is composed of a hypertext component and a graphical component. The hypertext component allows the user to navigate, search and inspect, for a given change, who made it and when, the files modified, why the change occurred, and when applicable, the defect that was fixed. The graphical component provides two types of views: first, it calculates statistics and presents them in histograms where the horizontal axis is usually time, and therefore provides an overview of the evolution of the project; and second, it provides graphs that show files, authors and their interrelationships (such as which files have been modified together, or which authors modify which files—see Fig. 8).

Interaction: softChange's hypertext interface allows the user to freely navigate and search the information space. The graphical views in softChange are generated in batch mode and the user is allowed to specify some parameters for their creation.

Effectiveness: softChange has been used by its authors in studies of software evolution and in the analysis of global software development practices in large open source projects. No formal user testing has been performed. It is available on request.

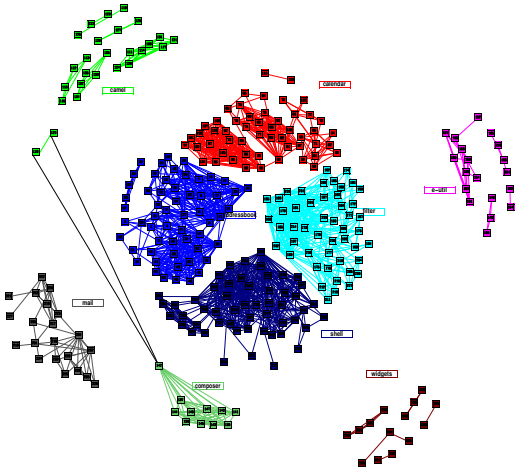

Figure 8: A graph created by softChange (from [German 2004])

4.9 Evolution Matrix

Intent: Evolution Matrix is an approach to visualizating software evolution [Lanza 2001]. It is a specialized view within CodeCrawler, a tool for object-oriented reverse engineering.

Information: Evolution Matrix uses program analysis to calculate various metrics based on a set of releases of the software.

Presentation: The key characteristic of Evolution matrix is its presentation of the software as a 2D matrix with classes arranged on the Y-axis and time-ordered releases on the X-axis. Cells of the matrix contain rectangles that encode values of two of the class's metrics, one for each dimension. The tool displays the number of methods and number of instance variables in the class, and identifies several typical patterns of class evolution (see Fig. 9).

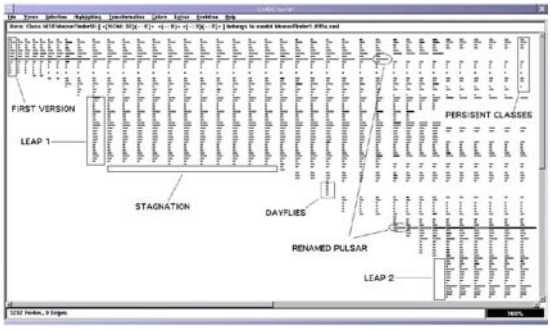

Figure 9: A screenshot of Evolution Matrix (from [Lanza 2001])

Interactivity: The view is interactive: a user can change the metrics that are being visualized, as well as presentation parameters such as the mapping of colours to data values.

Effectiveness: This tool was evaluated through two case studies of unspecified size.

4.10 Augur

Intent: Augur is a visualization tool designed "for the developers participating in [distributed] software development" [Froehlich and Dourish 2004]. Its focus is on monitoring activity in the project and exploring it in detail both in time and in the context of artifacts where activities occurred.

Information: Data used in Augur's visualizations comes from the metadata recorded in the version control system and the contents of the revisions. Program analysis is conducted on the revisions to detect syntactic units that are modified in each revision. This data can then be combined with age information and displayed in the primary view. Version control data are also aggregated to display cumulative views of activity in a given time period.

Presentation: Augur builds on Seesoft's approach to line-based visualization of code artifacts, but encodes two additional attributes next to the line information. It also provides secondary views that complement the line-based visualization by showing various cumulative graphs and statistics (see Fig. 10). Augur also has a zoomable user interface.

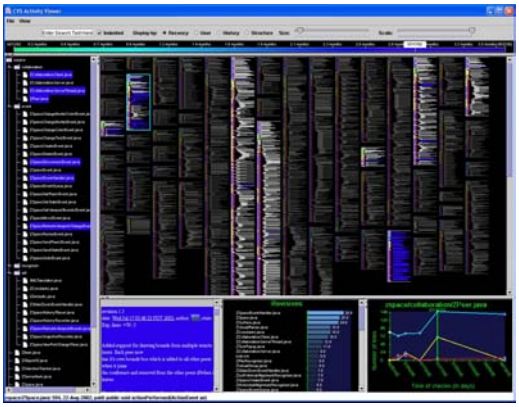

Figure 10: A screenshot of Augur. (Adapted from [Froehlich and Dourish 2004])

Interaction: Augur is interactive and its views are coupled – a selection in one view updates the presentation in all views. Information can be selectively displayed according to authors and according to time ranges.

Effectiveness: Augur has been applied to several medium-to-large open source projects. Members of the open source projects reported that they could interpret Augur's presentations and found them useful. It can be downloaded from *drzaius.ics.uci.edu/~jfroehli/augur*.

4.11 Beagle

Intent: The goal of Beagle tool is support for exploring software changes, particularly how releases of a project evolve [Tu and Godfrey 2002].

Information: Beagle uses the releases of a software project, and it extracts its functions/methods and creates call graphs. It attempts to track functions/methods in case they are merged or renamed from one version to the next.

Presentation: The main visualization is the call graph for every given release. Beagle also generates some static graphs: tree views show how a given function/method evolves, and "scatter plots" which are 2 dimensional plots that show the structural changes of a file or group of files (it shows if functions are deleted, moved, renamed or merged between two different releases). Beagle also supports zoomable views.

Interaction: It is an interactive application that allows the user to track the call graph of a given file or function/method.

Effectiveness: Beagle was demonstrated by applying it on gcc (the GNU C/C++ compiler) and Postgres DBMS.

4.12 Spectrograph

Intent: Spectrograph is a specialized visualization that shows where and when changes occur in a system [Wu et al. 2004a].

Information: It uses data from software releases and CVS history.

Presentation: Spectrographs are very similar to Evolution Matrix. The X-axis is usually time related (releases or revisions of files) and the Y-axis varies: files, authors, directories. A metric determines how a given point is displayed (see Fig. 11).

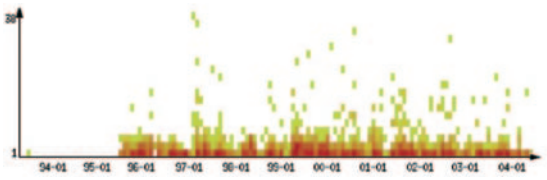

Figure 11: A spectrograph (from [Wu et al. 2004a])

Interaction: Data is collected in a batch process, but a user can interact with the view to zoom into the view, filter the data, change presentation parameters such as mapping of colours to data values, or issue queries.

Effectiveness: Spectrograph was demonstrated by applying it on large open source projects but with no user evaluation.

4.13 Summary

Many of the surveyed awareness tools were developed in the past few years and are still under active development. Given the recency of the work, researchers are not always aware of parallel work. These parallel efforts may be advantageous as they may lead to more innovation. However, it is also important to have some common context for presenting and comparing research results. In Table 2 we attempt to provide an overview of the different characteristics and features of the various tools we surveyed. Our interpretation is based on our use of the surveyed tools (when possible, from reading the papers that describe them and from discussions with the tool developers). Not all of the characteristics and features are included in this table, particularly those features which are difficult to summarize in this fashion. For a more detailed description of the tools and for better figures please refer to the original papers.

5 Research agenda

From our detailed analysis of visualization tools that provide awareness of human activities in software development, we have been able to make a number of observations. These observations are ordered according to the dimensions in the framework and they point to some possible directions for future research.

Intent: Need for requirements. There is a lack of empirical work that sheds light on the desirable features that should be provided by an awareness tool (notable exceptions include [Wu et al. 2004b; Gutwin et al. 2004; Souza et al. 2004]). Without a principled description of these requirements, it is difficult to compare and evaluate these tools, and it is hard to assess how different tools could be combined to solve particular development problems. More work is needed to better define who the tool users are and which types of questions the tools should help answer.

Information: Fact extraction is key. A visualization will only be useful if the underlying data serves some purpose. Many of the questions that software developers have can be answered by deriving and aggregating facts from various data sources. Some tools lack the facility to do more detailed analyses of the data extracted. Moreover, few of the tools we surveyed consider potentially useful sources of information such as defects, documentation and informal communications. We also noticed that most of the tools use different methods for fact extraction, and in some cases are reinventing the wheel. It would be advantageous to have a common data model and a standardized way to extract and share these facts.

Presentation: Combine views. There does not appear to be a lack of visualization techniques that can be applied to providing activity awareness in software development. What is lacking is how to integrate the various techniques so that they can be effectively used in combination to answer the questions the users will have. Few of the tools discuss how various textual, hypertextual and graphical views can be best combined to provide awareness.

Interaction: Need for improved queries and online tools. Many of the surveyed tools were not interactive and operated in an offline mode—this is, in some cases, due to the early nature of the research. The tools support researchers as they explore which visualizations may be useful. We also conjecture that more powerful query techniques, such as the use of a formal query language, may be beneficial when developers and reverse engineers have specific queries about past human activities.

Effectiveness: Need for more evaluation and benchmarks. The most striking observation from this research is that not only is there a lack of evaluation for these tools, but there is also a lack of well-defined benchmarks and evaluation criteria. Most tools have only been evaluated through case studies with the only users being the designers of the tools. Again, this is not surprising given the recency of much of this research. Tools that can be downloaded are amenable to a more unbiased evaluation by other researchers. Tools that interoperate with other software development tools also increase the likelihood that they will be used and can provide feedback on how they work in a more realistic context. An important criteria for evaluation is a cost-benefit analysis–although a tool may have a high cost of installation, learning, or usage, if it provides significant benefits then it will be more likely be used.

6 Discussion

Surveys in any research discipline play an important role as they help researchers synthesize key research results and expose new areas for research. The framework we propose is an initial but well-thought out attempt at providing a mechanism for describing, comparing and understanding awareness tools to support software development tasks. It follows in the tradition of Price et al.'s taxonomy [Price et al. 1992] and there is some overlap between the dimensions of our framework and the top-level categories of their taxonomy. We believe building on this earlier taxonomy gives our approach a solid grounding in the existing theoretical foundation of the field. The divergences are mainly the result of our focus on visualizing human activity during the development of a software system, rather than visualizing the software itself. We also give greater attention to people that will use the tools and the tasks for which the tools will be used, influenced by an existing framework of workspace awareness in groupware [Gutwin and Greenberg 2002]. We also prefer the term framework rather than taxonomy, as framework captures the qualitative aspects we emphasize rather than the need to classify particular tools.

The dimensions we propose are not pairwise-independent (just as the Cognitive Dimensions proposed by Green are not [Green 2000]). Indeed, a tool designer would not be able to provide many of the features suggested by the framework without making some compromises on other features. We also do not suggest that a single tool should provide all of these features as this could lead to a very cumbersome tool that is difficult to use and may lack purpose.

A question that could be raised is whether the framework we propose has validity. To quote Price et al.: if our framework "provides a meaningful way of describing software visualization technology, then it should facilitate a clear and concise statement of the essential features" of specific tools [Price et al. 1992, p. 602]. From our initial tool survey experiences, our framework fulfills this role. While working on this paper, we found that the framework greatly helped our thinking and guided our writing. We also observed that the

		See Soft	VRCS	Tukan	Advizor	Xia/ Creole	Palantir	Jazz	Soft change	Augur	Beagle	Spectro graph	Evo- Matrix
Intent													
Role	Team size	Any	1	Any	Any	Any	Any	Small	Any	Any	Any	Any	Any
	Developer	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Manager	Yes			Yes				Yes			Yes	Yes
	Tester/Documenter	Yes								Yes			
	Maintainer/Reengineer					Yes			Yes	Yes	Yes	Yes	Yes
	Researcher	Yes			Yes	Yes			Yes	Yes	Yes	Yes	Yes
Time	Present		Yes	Yes			Yes	Yes					
	Recent past	Yes	Yes	Yes		Yes	Yes	Yes		Yes			
	Historical	Yes			Yes	Yes			Yes	Yes	Yes	Yes	Yes
Cognitive support	Authorship	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	
	Rationale							Partial	Yes	Yes			
	Time	Yes			Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes
	Artifacts	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Information													
Change management	Local history							Yes					
	Releases		Yes						Yes		Yes	Yes	Yes
	Branching		Yes					Yes	Yes				
	Revisions	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes			
Tracking	Defects/Changes	Yes			Yes			Yes	Yes				
Program Code	Syntactic units			Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Semantic analysis			Yes		Yes			Yes		Yes	Yes	
Document- ation	Requirements, tests												
	Design, Architecture												
Informal comms.	Email												
	Instant messages							Yes					
Derived	Single source				Yes		Some			Yes		Yes	
	Multiple source			Yes				Yes	Yes		Yes		Yes
Presentation													
Form	Hypertext				Yes			Yes	Yes		Yes		
	Graphical	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes
Kinds of views	Annotates other views			Yes			Yes	Yes					
	Graph views		Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes		
	Statistical views				Yes				Yes	Yes			
	Other views	Map						Yes	Yes	Map	Yes	Matrix	Matrix
Techniques	Visual variables	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Animation												
	Abstractions		Yes		Yes			Yes	Yes		Yes		
Interaction													
Batch/ Live	Offline					Yes			Yes		Yes	Yes	
	Online	Yes	Yes	Yes	Yes		Yes	Yes		Yes	Yes		Yes
Customizable	Customization level		Lo		Hi	Hi	Lo	Lo					
Queries	Query language											Yes	
	Filter widgets	Yes			Yes	Yes	Yes	Yes		Yes	Yes		Yes
Navigation, Orientation	Overview+detail	Yes				Yes				Yes	Yes		
	Zoomable views				Yes	Yes				Yes	Yes		
	Coupled views			Yes	Yes	Partial	Yes	Yes		Yes	Yes		
Effectiveness													
Status	Availability					Yes	Yes		Yes	Yes	Yes		
	Interoperability		Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes		Yes
Evaluation	Adopted				Yes								
	Case study	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	User study		Yes			Yes							

Table 2: Summary of the tools

framework helped us identify similar tools as well as unique tools for the awareness of human activities in software development. For example, we were able to determine that the Spectrograph approach may be a special case of the Evolution-Matrix approach. We anticipate that other researchers will apply the framework to evaluate more tools, and that their experiences will further refine the framework.

We see our framework primarily as a discussion tool rather than a formal representation of the characteristics and features of an awareness tool. Furthermore, we do not propose that it should be used for directly comparing tools, but rather that it can be used to discern what are the important questions to ask about these tools. The ultimate test of its validity will be whether it resonates with developers, tool designers and researchers. Our own initial experiences are promising, and we look forward to the discussion and exchange of ideas that we hope our framework will initiate in the software visualization and software engineering communities.

References

BALL, T., AND EICK, S. G. 1996. Software visualization in the large. *IEEE Computer 29*, 4, 33–43.

DIX, A. J. 1994. Computer-supported cooperative work—a framework. In *Design Issues in CSCW*, D. Rosenburg and C. Hutchison, Eds. Springer Verlag, 23–37.

DOURISH, P., AND BELLOTTI, V. 1992. Awareness and coordination in shared workspaces. In *Proc. of the ACM Conference on Computer-Supported Cooperative Work*, 107–114.

EICK, S. G., STEFFEN, J. L., AND SUMMNER JR., E. E. 1992. Seesoft—a tool for visualizing line oriented software statistics. *IEEE Trans. on Software Engineering 18*, 11, 957–968.

EICK, S. G., GRAVES, T. L., KARR, A. F., MOCKUS, A., AND SCHUSTER, P. 2002. Visualizing software changes. *IEEE Transaction on Software Engineering 28*, 4, 396–412.

ENDSLEY, M. 1995. Toward a theory of situation awareness in dynamic systems. *Human Factors 37*, 1, 32–64.

FROEHLICH, J., AND DOURISH, P. 2004. Unifying artifacts and activities in a visual tool for distributed software development teams. In *Proc. of the 26th International Conference on Software Engineering (ICSE'04)*, 387–396.

GERMAN, D., HINDLE, A., AND JORDAN, N. 2004. Visualizing the evolution of software using softChange. In *Proc. of the 16th Internation Conference on Software Engineering and Knowledge Engineering (SEKE 2004)*, 336–341.

GERMAN, D. 2004. An empirical study of fine-grained software modifications. In *20th IEEE International Conference on Software Maintenance (ICSM'04)*, 316–325.

GREEN, T. 2000. Instructions and descriptions: some cognitive aspects of programming and similar activities. In *Working Conference on Advanced Visual Interfaces (AVI 2000)*, 21–28.

GRINTER, R. E. 1995. Using a configuration management tool to coordinate software development. In *Conference on Organizational Computing Systems*, 168–177.

GRUNDY, J. C. 2001. Software architecture modeling, analysis and implementation with SoftArch. In *Hawaii International Conference on System Sciences*, 9051.

GULLA, B. 1992. Improved maintenance support by multi-version visualizations. In *Proc. of the International Conference on Software Maintenance*, 376–383.

GUTWIN, C., AND GREENBERG, S. 2002. A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work 11*, 3/4, 411–446.

GUTWIN, C., PENNER, R., AND SCHNEIDER, K. 2004. Group awareness in distributed software development. In *Proc. of the 2004 ACM Conference on Computer Supported Cooperative Work*, 72–81.

HUPFER, S., CHENG, L.-T., ROSS, S., AND PATTERSON, J. 2004. Introducing collaboration into an application development environment. In *Proc. of the ACM 2004 Conference on Computer Supported Cooperative Work*, 444–454.

KOIKE, H., AND CHU, H.-C. 1997. VRCS: Integrating version control and module management using interactive three-dimensional graphics. In *Visual Languages VL'97*, 170–175.

LANZA, M. 2001. The Evolution Matrix: recovering software evolution using software visualization techniques. In *Proc. of the 4th International Workshop on Principles of Software Evolution*, 37–42.

LINTERN, R., MICHAUD, J., STOREY, M.-A., AND WU, X. 2003. Plugging-in visualization: experiences integrating a visualization tool with eclipse. In *Proc. of the 2003 ACM symposium on Software visualization*, 47–56.

PRICE, B. A., SMALL, I. S., AND BAECKER, R. M. 1992. A Taxonomy of Software Visualization. In *Proc. of the 25th Hawaii International Conference on System Sciences (HICSS)*, vol. 2, 597–606.

SARMA, A., NOROOZI, Z., AND VAN DER HOEK, A. 2003. Palantír: raising awareness among configuration management workspaces. In *Proc. of the 25th International Conference on Software Engineering*, 444–454.

SCHÜMMER, T., AND HAAKE, J. M. 2001. Supporting distributed software development by modes of collaboration. In *Proc. of the European Conference on Computer Supported Collaborative Work*, 79–98.

SEGAL, L. 1995. Designing team workstations: the choreography of teamwork. In *Local Applications of the Ecological Approach to Human-Machine Systems*, P. Hancock, J. Flach, J. Caird, and K. Vicente, Eds. 392–415.

SOUZA, C. D., REDMILES, D., CHENG, L.-T., MILLEN, D., AND PATTERSON, J. 2004. Sometimes you need to see through walls a field study of application programming interfaces. In *Proc. of the 2004 ACM Conference on Computer Supported Cooperative Work*, 63–71.

TU, Q., AND GODFREY, M. W. 2002. An integrated approach for studying architectural evolution. In *Proc. of the 10th International Workshop on Program Comprehension (IWPC'02)*, 127–136.

WALENSTEIN, A. 2003. Observing and measuring cognitive support: Steps toward systematic tool evaluation and engineering. In *Proc. of the 11th International Workshop on Program Comprehension (IWPC'03)*, 185–195.

WARE, C. 2000. *Information visualization: perception for design*. Morgan Kaufmann Publishers Inc.

WU, J., HOLT, R. C., AND HASSAN, A. E. 2004. Exploring software evolution using spectrographs. In *Proc. 11th Working Conference on Reverse Engineering*, 80–89.

WU, X., MURRAY, A., STOREY, M.-A., AND LINTERN, R. 2004. A reverse engineering approach to support software maintenance: Version control knowledge extraction. In *Proc. 11th Working Conference on Reverse Engineering*, 90–99.

Figure 1: A screenshot of Seesoft (from [Ball and Eick 1996])

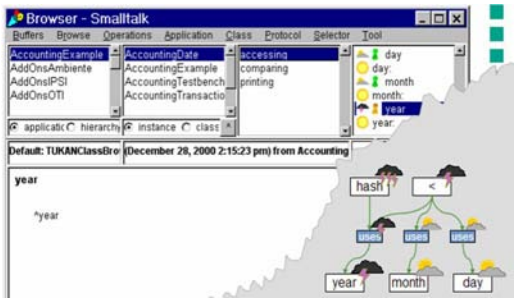

Figure 3: A screenshot of Tukan (from [Schümmer and Haake 2001])

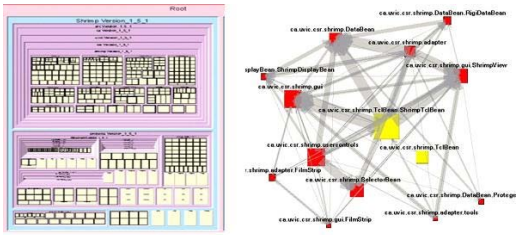

Figure 5: Two views from Xia and Creole respectively (from [Wu et al. 2004b; Lintern et al. 2003])

.

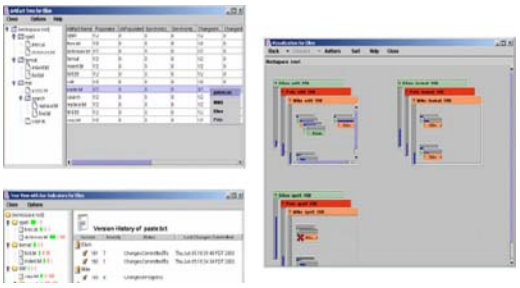

Figure 6: A screenshot of Palantir (from [Sarma et al. 2003])

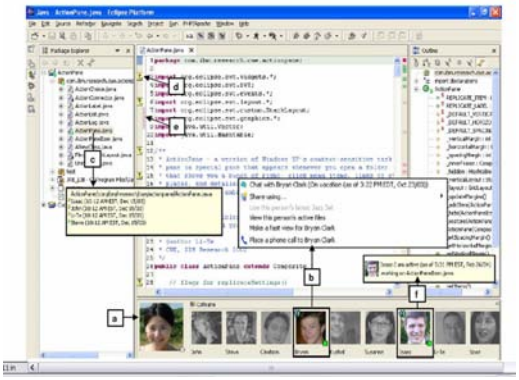

Figure 7: A screenshot of Jazz (from [Hupfer et al. 2004])

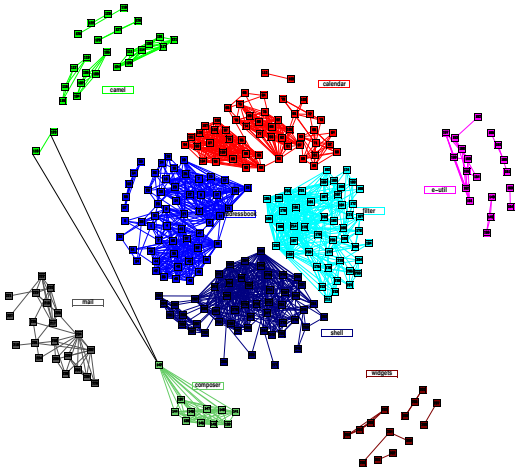

Figure 8: A graph created by softChange (from [German 2004])

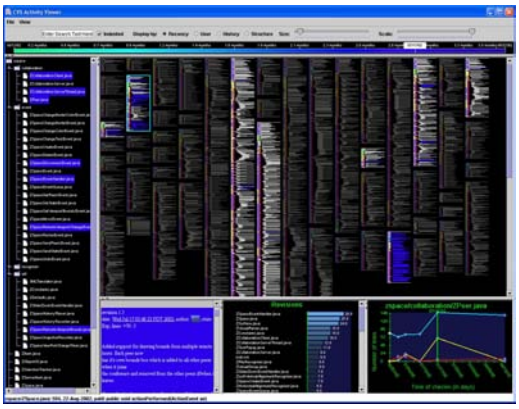

Figure 10: A screenshot of Augur (from [Froehlich and Dourish 2004])

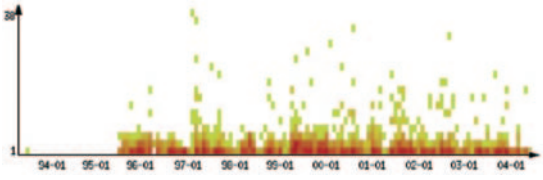

Figure 11: A spectrograph (from [Wu et al. 2004a])