SourceVis: Collaborative Software Visualization for Co-located Environments

Craig Anslow, Stuart Marshall, James Noble
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email: {craig,stuart,kjx}@ecs.vuw.ac.nz

Robert Biddle
Department of Computer Science
Carleton University
Ottawa, Canada
Email: robert_biddle@carleton.ca

***Abstract*—Most software development tools and applications are designed from a single-user perspective and are bound to the desktop and Integrated Development Environments (IDEs). These tools and applications make it hard for developers to analyse and interact with software artifacts collaboratively. We present *SourceVis* – a multi-user collaborative software visualization application for use on large multi-touch tables. We describe the design and visualization features of SourceVis, present findings from a user study, and discuss the implications for building collaborative software visualization applications.**

***Index Terms*—Collaborative Software Visualization; Multi-touch Tables; Multi-user; Co-located Environments**

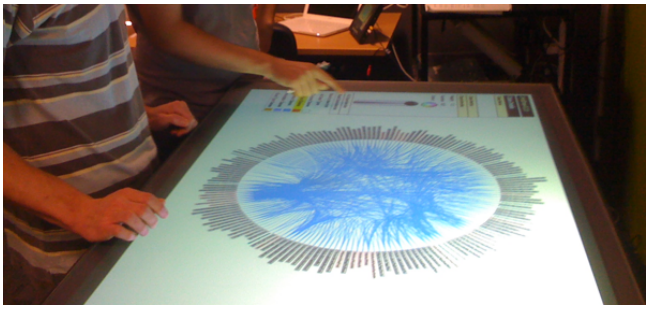
I. Introduction

Understanding software for maintenance is often a social activity and involves pairs of developers working within co-located environments (same room and time). Developers typically work in pairs within a larger team to carry out tasks including: programming, code reviews, refactoring, and visualization of work flow. Most software development tools and applications that support these tasks often involve analysis and visualization features. We focus on three design considerations for building collaborative software visualization applications in co-located environments [11].

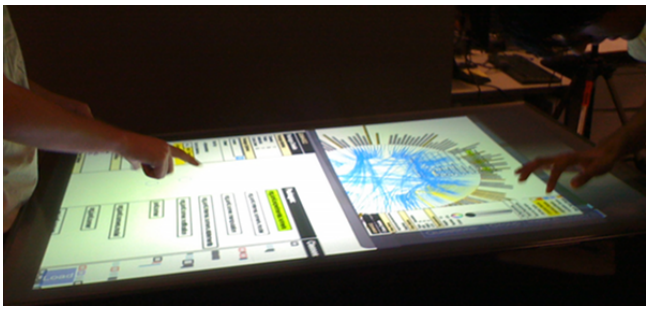
1) Design for multiple users: most tools are designed from a single-user perspective and are bound to the desktop and IDEs (e.g. Eclipse). These tools make it hard for users to analyse and interact with software artifacts collaboratively using the same tool. For example, in pair programming there is only one keyboard and mouse for input which is controlled by the driver. When the observer wants to interact with the tool they have to either swap positions or obtain the keyboard and mouse from the driver.

2) Support multiple visualization types: most analysis applications contain a small range of visualization techniques. If applications supported multiple visualizations then this would allow developers to visualize aspects of systems from different views and reduce the overload for installing lots of applications. For example, in code reviews developers often have to use different applications to explore test coverage, class dependencies, and class diagrams.

3) Display visualizations on large shared interactive surfaces: most applications are viewed on vertical computer displays. Developers quite often have dual vertical displays with a large number of pixels, but when working in co-located teams it is hard to interact and collaborate with information across digital and physical devices. For example, in visualizing work flows many teams use white boards and physical artifacts like post-it notes. Since whiteboards are vertical it makes it hard for developers to orient the information to where they are standing. Instead they have to physically move to manipulate the notes and draw at certain positions on the white board.



(a) System dependency visualization displayed at full screen.



(b) Different visualizations displayed in separate windows at opposite ends of the table.

Fig. 1. SourceVis - multi-user collaborative software visualization.

In this paper we present *SourceVis* – a collaborative software visualization application for use on large multi-touch tables within co-located environments based on these design considerations. We describe the features of SourceVis (§II) and present findings from a user study (§III). Figure 1 illustrates users interacting with a visualization at full screen and different visualizations in separate windows with users interacting at opposite ends of the table.

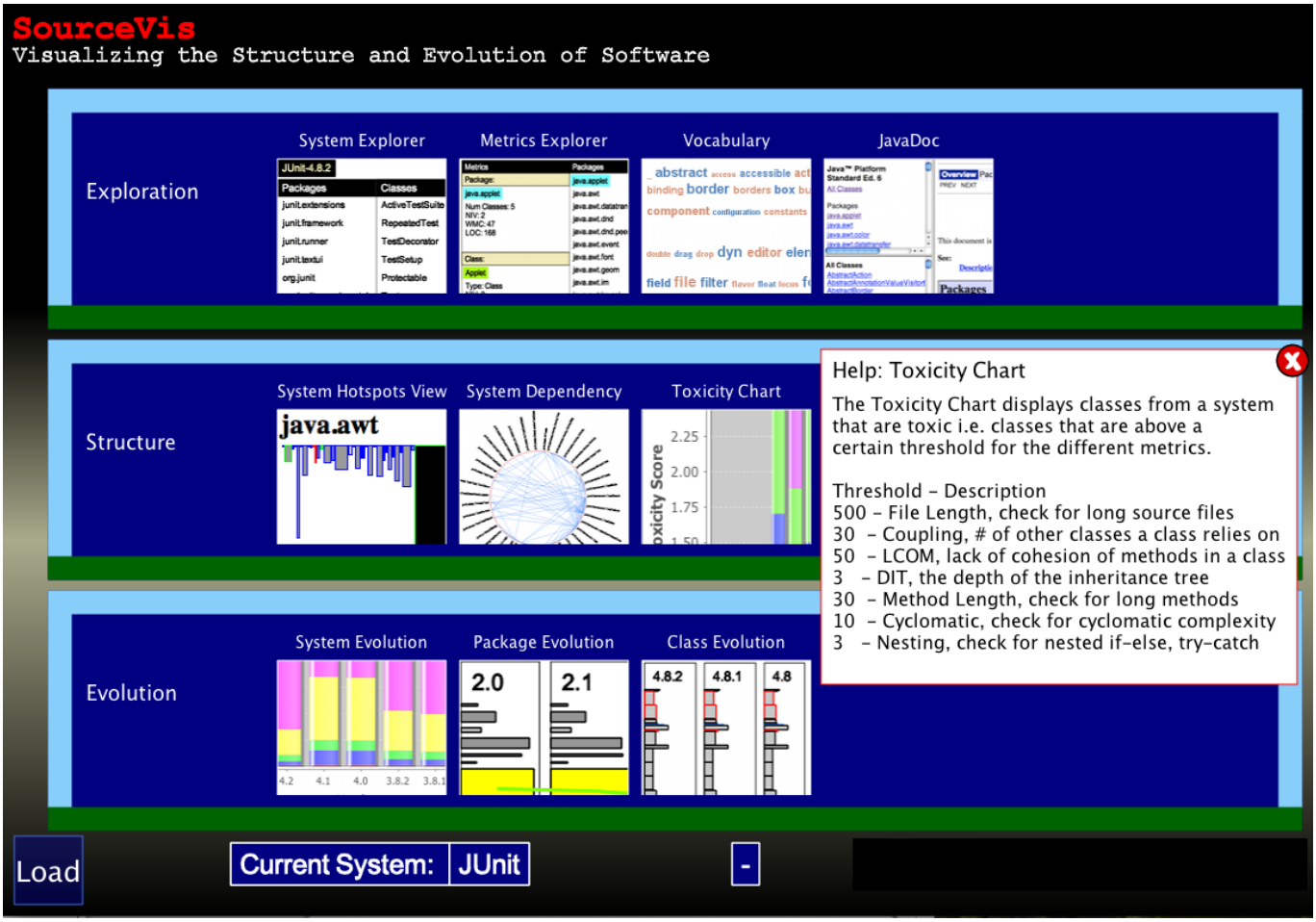


Fig. 2. SourceVis startup screen. Selecting load displays available systems. Visualizations are grouped into three categories: exploration, structure, and evolution. Tapping an icon displays a new visualization. Tapping and holding an icon displays help information. Tapping cyan coloured borders on categories orients icons and text to that direction.

II. SourceVis

SourceVis is a collaborative application for visualizing the structure and evolution of software systems. The aim is to help developers working in co-located teams to explore how a system has been structured by viewing metric visualizations and evolution visualizations. These steps can help identify what parts are large and likely need to be refactored.

SourceVis is designed for multiple users to interact synchronously or asynchronously on a multi-touch table, and supports multiple visualization techniques. We selected a representative sample of existing techniques from the information visualization and software visualization literature [8], from our visualization wall user study [2], and our preliminary candidate multi-touch software visualizations [1]. SourceVis is displayed on a large custom-made horizontal interactive multi-touch table (48 inch projected image display), that we built following some existing technical guides [20]. Visualizations can be displayed at full screen, within scalable windows, or at any orientation on the horizontal plane. Multiple visualizations can also be displayed at once.

A. User Interface

Users first load a system by selecting a system from the load menu at the bottom left of the startup screen (see Figure 2). Only one system and all versions from that system can be loaded at any one time. Visualizations are launched by tapping a visualization icon. They are grouped into three categories: exploration, structure, and evolution. The visualizations in the startup screen are overview visualizations and users can select individual entity visualizations from these overviews to get more specific details about different entities (e.g. classes).

Each visualization is displayed in a separate window which allows multiple visualizations to be displayed at once; either next to each other, overlapping one another, or within side another window. The right hand side border of the window has options to maximize or close the visualization. Once a visualization is displayed at full screen there is an option in the bottom right in Figure 3(a) that allows closing (denoted as cross inside circle) or making the visualization smaller (denoted as windows overlapping).

Users interact with elements in the visualizations by tapping for select, dragging elements with one finger, and rotating and resizing elements with two fingers of either the same hand or different hands. Tapping and holding an element (e.g. icon, entity, metric label) displays properties about that element in help message boxes (see Figure 2), and for entities (e.g. packages, classes) displays a pie menu. Multiple entities can be grouped by drawing a shape around them by using a lasso gesture. The grouped elements can be moved around. Zooming uses a pinch gesture with one or two hands. Panning uses two fingers in any direction on the canvas of the visualization.

As SourceVis is designed for a horizontal display, it is important for users to be able to move around the table and orient visualizations to where they are physically standing. The visualization windows can be moved, rotated or scaled in size which allows the window to be oriented in any direction. On the start screen the categories enclosing the visualization icons and text can be oriented to any of the four directions of the table. The direction is represented by a green border. Figure 2 shows all categories oriented to the bottom of the image.

For a multi-user scenario a visualization could be displayed at full screen where one user is focused on the overview of a system while another is focused on a certain aspect of the same visualization (Figure 1(a)). Another scenario users could be viewing two separate visualizations at the same time. One visualization could be at full screen while the other inside of the larger one. Alternatively two separate visualizations could be displayed at opposite ends of the table (Figure 1(b)).

There are menus for each visualization including: system version, options, and pie. The system version menu displays the current version in the top left of the visualization. The options menu displays information about entities (e.g. name and metrics) and has features for sorting (e.g. alphabetical and descending by all metrics), filtering (e.g. by class type and by slider threshold values), and searching (e.g. by entity name using a keyboard) entities. The options menu can be collapsed to allow for more screen real estate. A pie menu is displayed with a tap and hold gesture on an entity. Figure 3(a) shows a pie menu for a class with options to display metrics properties in a dialog box or a new visualization type.

B. Software Visualizations

SourceVis contains 13 visualizations grouped into three categories and are aimed at supporting answering the kinds of questions developers ask within industry [22]. The visualizations allow users to explore, see the structure, and evolution structure of a system. The exploration category contains visualizations that show a list of entities, metrics about systems, packages, and classes, and vocabulary employed in entities. The structure category adapts Polymetric Views [14] to multi-touch including the System Hotspots View and Class Blueprint, and graph based visualizations to show dependencies between entities. The evolution category shows how a system has evolved over time focusing on structural changes including versions, packages, and classes of a system using Polymetric View encodings and charts.

1) Exploration: The System Explorer (SE) shows all packages (left hand side) and classes (right hand side) in scrollable lists where each entity is selectable. Tapping a package highlights the name and classes from that package are displayed. All classes in the system can be displayed by tapping the classes heading label. In Figure 3(a) junit.framework package and junit.framework.TestSuite class have been selected, and metrics properties for TestSuite, and a pie menu are on display.

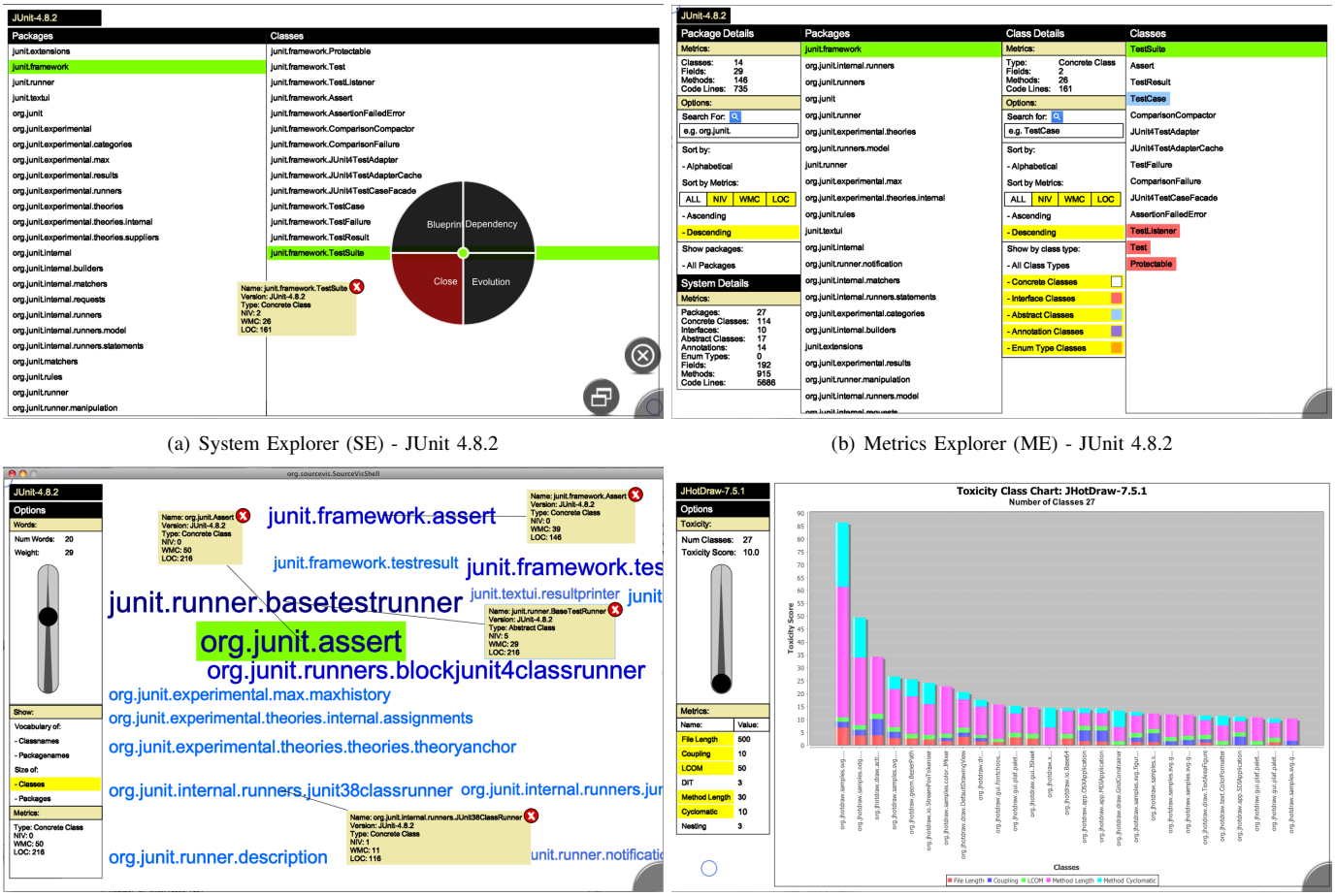
The Metrics Explorer (ME) shows metrics about packages and classes. The metrics for a package are the total number of classes, number of variables (NIV), number of methods (WMC), and number of lines of code (LOC). The metrics for a class are NIV, WMC, LOC [9]. Tapping a package (left hand side) displays metrics about a package and also displays the classes from the package (right hand side). Likewise tapping a class displays metrics about a class. In Figure 3(b) the packages are sorted in descending order by all metrics. The junit.framework package is selected and shows that there are 14 classes of which 10 classes are concrete, three interfaces, and one abstract class. The classes have been sorted in descending order and TestSuite is selected.

The Vocabulary View (VV) shows the vocabulary used in the names of entities (e.g. packages, classes) to understand the coding standards employed. The visualization uses a word cloud representation. Packages and classes can also be visualized separately that use metrics (e.g. NIV, WMC, LOC) to determine the font size for the name of the entity. In Figure 3(c) the largest class org.junit.Assert has been selected. Four class metrics properties are being displayed and some classes have been filtered using the slider leaving a total of 20 classes.

The Toxicity Chart[1] (TC) (Figure 3(d)) shows which classes are toxic in a system and how the problems are distributed. A class in the chart is represented as a bar and the height shows the toxicity score. The score is based on several metrics and the higher the score the more toxic a class is. In our adaptation we use the following metrics: file length, coupling, lack of cohesion, depth of inheritance tree, method length, cyclomatic complexity, and method nesting. The individual metrics of the score are colour coded. Classes that score zero points are not included in the chart. The data in the chart can be filtered by selecting all or some of the metrics and using a slider to show classes greater than the toxicity score threshold. Individual elements in the chart can not be tapped. The chart can be resized or displayed as another chart.

2) Structure: The System Hotspots View (SHV) highlights large classes in a system [14]. In our adaptation packages are displayed down the Y axis in the packages pane and classes from each package along the X axis in the classes pane. Packages and classes can be sorted by metrics. Classes are represented using Polymetric View encoding where width indicates NIV, height WMC, and shading LOC. A darker shading means more lines of code. Coloured borders represent the type of class (e.g. black a concrete class, red interface, blue abstract class). In Figure 4(a) the packages have been

[1] http://erik.doernenburg.com/2008/11/how-toxic-is-your-code/



(a) System Explorer (SE) - JUnit 4.8.2

(b) Metrics Explorer (ME) - JUnit 4.8.2

(c) Vocabulary View (VV) - JUnit 4.8.2

(d) Toxicity Chart (TC) - JHotDraw 7.5.1

Fig. 3. Exploration Visualizations.

sorted, junit.framework and junit.framework.Assert have been selected, and the keyboard is available to search for classes.

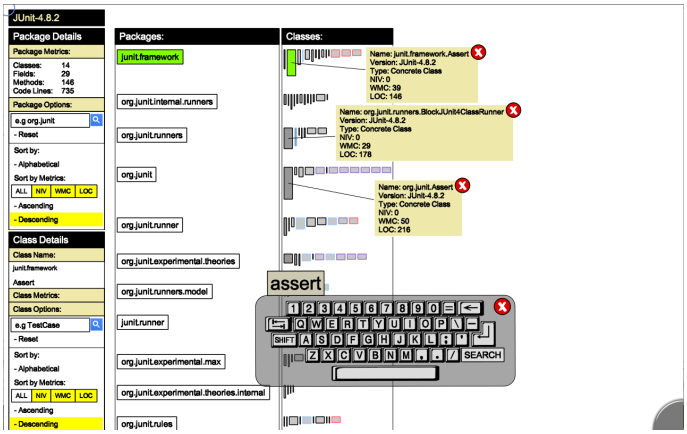
The System Dependency View (SDV) (Figures 1(a) and 4(b)) shows all the classes in a system displayed around a circle with curved edges to represent dependencies between classes. Tapping a class displays the class name, metric values, and highlights the class dependency edges. A slider allows filtering dependencies according to the weight value. Classes can be filtered according to class type. Classes that have no dependencies and or no references can be displayed which are potentially redundant classes. All the classes around the circle can be rotated by doing a rotation gesture in the middle.

The Class Dependency View (CDV) shows what classes a class depends on. The class is coloured yellow with blue edges to dependent classes. Dependent classes are coloured according to class type and can be filtered by the slider. Edges are weighted according to the number of dependencies between classes. The stronger the dependency the thicker the edge. Figure 4(c) shows org.jhotdraw.draw.DefaultDrawingView has 17 dependent classes.

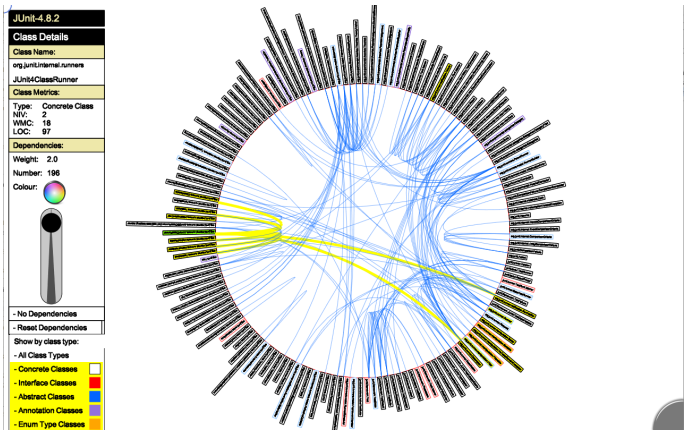
The Class Blueprint View (CBV) (Figure 4(d)) shows the references between methods and attributes within a class [14]. The visualization has five layers from left to right. The first four layers relate to the methods and the final layer to the attributes. The initialization layer displays initialization methods (e.g. constructors), interface layer public methods, implementation layer private methods, and accessor layer accessor and mutator methods (e.g. get()). The attribute layer displays the attributes. The methods and attributes have different fill colours depending on which layer they belong to, likewise the edges for the references. Edge weight can be adjusted by the slider to highlight coloring. Multiple methods or attributes can be selected at once to highlight edges.

3) Evolution: The System Evolution View (SEV) shows how a system has evolved using a chart similar to the TC. The metrics are the total number of packages, classes, methods, fields, and lines of code. The chart can be updated by selecting different metrics. All versions or major versions (ending in zero in the version number) of a system can be displayed.

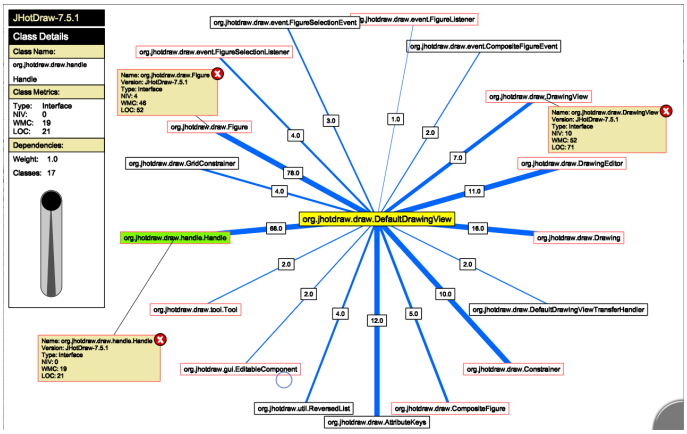
The System Package Evolution View (SPEV) (Figure 5(a)) shows all packages and all versions of a system in a Polymetric View encoding. The width of a package represents the number of methods, height number of fields and shading the number of lines of code. Each package is grouped inside a version



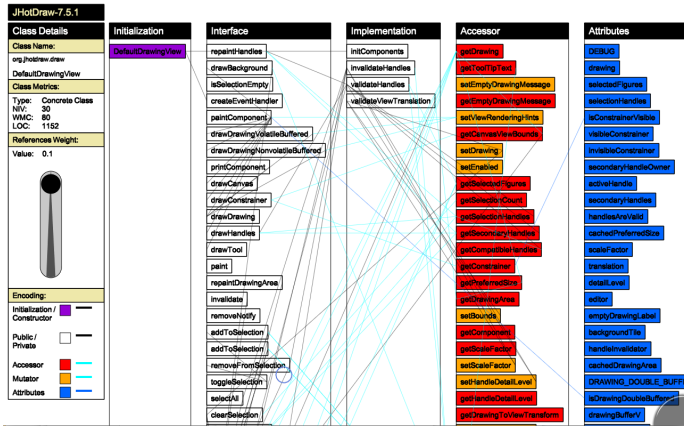
(a) System Hotspots View (SHV) - JUnit 4.8.2



(b) System Dependency View (SDV) - JUnit 4.8.2

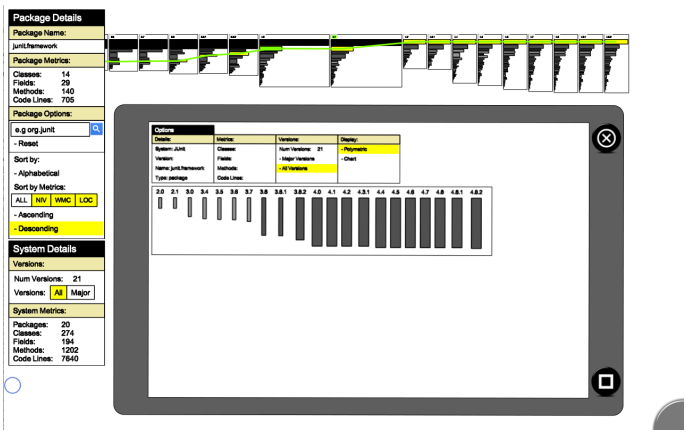


(c) Class Dependency View (CDV)- JHotDraw 7.5.1 org.jhotdraw.draw.DefaultDrawingView

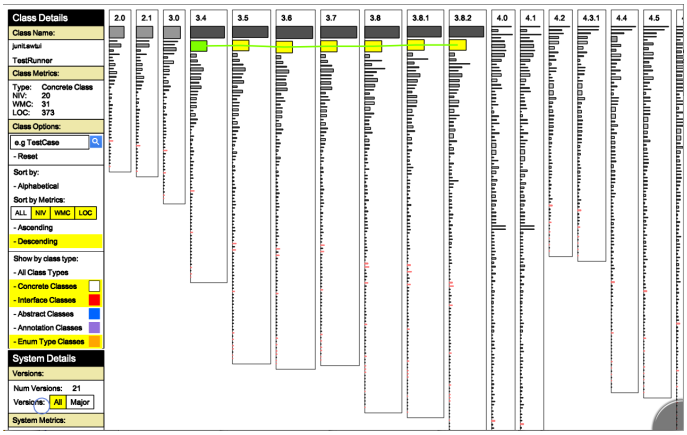


(d) Class Blueprint View (CBV) - JHotDraw 7.5.1 org.jhotdraw.draw.DefaultDrawingView

Fig. 4. Structure Visualizations.



(a) System Package Evolution View (SPEV) - JUnit, Package Evolution View (PEV) - junit.framework



(b) System Class Evolution (SCEV) - JUnit and junit.awtui.TestRunner selected

Fig. 5. Evolution Visualizations.

pane. Packages can be sorted across the whole visualization or within each version. A package can be highlighted across all versions it appears in. We do not cater for packages changing name between versions. All versions or major versions of a system can be displayed. The Package Evolution View (PEV) shows the evolution of one package over time, displayed as a

Polymetric View encoding or as a chart, and can be launched from other views. In Figure 5(a) the Package Evolution View is displayed above the System Package Evolution View.

The System Class Evolution View (SCEV) is similar to System Package Evolution View but shows classes which can be sorted, filtered by class type, and searched for. In Figure 5(b) junit.awtui.TestRunner appears in seven versions. The Class Evolution View (CEV) shows the evolution of a class and is similar to the Package Evolution View.

C. Implementation

SourceVis has been used with sample systems from the Qualitas Corpus [25] ranging in size from small to very large based on lines of code and number of classes, see Table I. The systems were loaded into a static analysis tool called Understand[2] to generate metrics data for the visualizations.

SourceVis is implemented upon MT4j which is an open source toolkit for multi-touch [15]. We integrated third party libraries for the vocabulary[3] and chart[4] visualizations.

We created classes to represent entities from Java for systems, packages, classes, methods, and fields. These classes all implement an interface which has generic behaviour such as getEntityName(). These classes extend the MT4j MTTextArea class so text can be displayed or represented as a rectangle with no text. Some specialized classes were required to represent words and dependencies. The classes have attributes for the following aspects: if properties are being displayed, dependency weight of an edge, what layer an entity belongs to, and what colour an entity is. We created a comparator class to compare entity names of the same type.

We extended the MT4j main applet to a SourceVisShell class which launches SourceVis. Once launched the startup screen is displayed and represented by the SourceVisCategoriesScene which contains all the loaded systems as VisSystems. A VisSystem contains the system name, list of all versions, current version, and associated data metrics files. The SourceVisCategoriesScene has three kinds of categories: exploration, structure, and evolution. Each Category has many VisScenes which are images and other properties. A VisScene contains one SourceVisScene which represents a visualization in SourceVis and extends the MT4j AbstractScene.

TABLE I
SOFTWARE SYSTEMS VISUALIZED BY SOURCEVIS.

System (latest)	Line of code	Classes	Versions	Size
Azureus 4.5.0.4	453433	7249	51	V Large
Weka 3.7.2	224356	2099	49	V Large
ArgoUML 0.34	194859	2905	10	Large
FindBugs 1.3.9	109096	1744	2	Large
JHotDraw 7.5.1	75958	1070	6	Medium
GanttProject 2.0.9	47051	1058	2	Medium
JUnit 4.8.2	6164	209	21	Small
SquirrelSQL 3.1.2	6944	2211	2	Small

[2]http://www.scitools.com/
[3]http://opencloud.mcavallo.org/
[4]http://www.jfree.org/jfreechart/

III. USER STUDY

The aim of our study was to collect data about how effective our visualization techniques are for software understanding in order to meet our design challenges following a qualitative approach. Other researchers have used a similar research approach to understand how groups of participants collaboratively use a multi-touch table in a co-located environment. The findings have provided insight into the design of multi-touch tables for information visualization [11], visual analytics [12], and collaborative design [21].

A. Participants

There were six male participants who were computer science graduate students. All had a degree in computer science: one a diploma, three with an honours degree, one masters degree, and a PhD. The participants conducted the user study in pairs (e.g. Participants ID 1 and 2). All pairs chose their fellow participant. The first pair had known each other for four years, second pair 10 years, and third pair two years. None had previously used any software visualizations tools before. All had some form of professional development experience. When working on previous projects four participants regularly programmed with other developers daily, weekly, or monthly. All participants except one claimed that they did regular code reviews either daily or weekly. They have used a range of tools for code reviews including: source revision control, unit tests, project management, bug tracking, and IDEs. Four of the participants have touch mobile phones, while one of these also has an iPad. Two of the participants have used touch screens for work previously.

B. Procedure

Participants were welcomed, given an information sheet, consent form, and a pre-study questionnaire to complete. The questionnaire asked participants about their demographics and background experience. Participants then explored the example applications from MT4j for 10 minutes, followed by a demonstration of SourceVis by the session instructor, then an additional 10 minutes to explore other systems in SourceVis on their own. Figure 6 shows the user tasks which involved answering 16 questions using 11 of the visualization techniques and is similar to the kinds of questions software developers ask within industry [22]. Each pair of participants completed all the tasks using the same set of visualizations. The System Package Evolution View and Package Evolution View were not used as the System Class Evolution View and Class Evolution View were of similar design. The example systems used in the study were JUnit and JHotDraw. The participants recorded their answers to the questions on a sheet provided. We recorded the time it took participants to complete the user tasks. With each participant's consent we video recorded their actions and asked them to think aloud. Participants completed a post-study questionnaire which asked for their opinion on the effectiveness, strengths, and weaknesses of the interaction capabilities and the visualizations. We allowed up to 90 minutes to complete the study.

(SE) How many classes are there in org.jhotdraw.Geom?
(ME) What is the largest package in JHotDraw version 6.0.1?
(ME) How many interfaces does this package contain?
(ME) From that package what is the largest class?
(TC) In JHotDraw version 7.5.1 how many classes have a toxicity score greater than 5.0 for file Length?
(VV) In JHotDraw version 7.5.1 what are the four largest words used in class names?
(VV) What are the two largest classes?
(SHV) In JHotDraw version 7.5.1 what is the largest package?
(SHV) From the same package what is the largest class?
(CB) In this class how many accessor methods are called by the public method repaintHandles?
(CD) How many interfaces does this class depend on?
(CE) How many versions does this class appear in?
(SD) In JUnit 4.8.2 how many classes have no dependencies?
(SCEV) In JUnit what major version contains the most classes?
(SCEV) How many versions contain class junit.swingui.TestRunner?
(SCEV) How many versions contain annotation classes?

Fig. 6. User tasks by software visualization, 16 questions in total.

C. Qualitative Findings

We now present the qualitative findings from the study as feedback given by participants and summarized in Table II.

1) Strengths: SourceVis allowed multiple users to interact at the same time which encouraged participants to collaborate, learn from each other on how to use SourceVis, and work as a team.

> "Working with someone cooperatively helped me to better understand how to manipulate the information (which settings to toggle for instance). I was constantly communicating with my partner and we were always assisting each other. We were able to easily take turns manipulating the interface." PID2.

SourceVis supported multiple visualizations and displaying many visualizations at once. Users can launch new visualizations from the start screen or using a pie menu from a currently displayed visualization to show more detailed information about an entity (e.g. Class Blueprint, Class Evolution).

> "It is easy to follow a particular class around the different visualizations, and are often linked directly via the tap and hold pie menu." PID1.

SourceVis was designed so that many elements in the visualizations could be manipulated, in a visually consistent manner, such as sorting, filtering, searching, and moving elements.

TABLE II
SUMMARY OF QUALITATIVE FINDINGS.

	Strengths	**Weaknesses**
Visualizations	Multi-user Interaction Multiple Visualizations Data Manipulation Overviews Details on Demand Software Metrics and Trends Outliers and Relationships	Visualization Context Navigation UI Consistency Menus Pie Menu Search
Multi-touch Table	Large Shared Display Multi-touch Interaction	Screen Resolution Touch Detection

> "I liked how I could manipulate everything. I appreciated being able to zoom and rotate individual items away from my partner so I could get a better look." PID2.

The visualizations provided an overview of a system and the ability to drill down to get more details on demand about specific entities including packages and classes.

> "There were lots of information on one screen which allowed me to quickly understand the data on a high-level before manipulating the components to get individual details." PID2.

The metrics data made several trends and information easily discoverable that are hard to get at otherwise, especially concerning evolution and dependencies.

> "The visualizations did a great job of showing me metrics, trends, and dependencies more easily than I have experienced with existing IDEs and version control tools." PID2.

The visualizations were designed to help identify entities that are outliers such as large and small classes, and relationships between entities.

> "Being able to quickly see problem classes is a great feature." PID3.
> "The visualizations helped when trying to look at the relationship between different classes/methods/packages." PID 4.

Participants liked the large size of the table as it easily allowed them to see and share lots of things about the visualizations at the same time.

> "The size makes it very easy for people to be looking at different things at the same time." PID1.

The multi-touch capabilities allowed multiple participants to interact which removed the barriers from a single person being in control.

> "Having a large screen that removes the need for a singular keyboard and mouse allows all team members to contribute to the task at hand. It felt quite good." PID3.

2) Weaknesses: Given the participants were novice users of SourceVis some were confused as to what visualization they were currently looking at and got lost through the linking of some of the visualizations especially if there was deep nesting. Adding breadcrumbs and labels would help users to remember the context of the visualizations. Some participants got lost in terms of navigating within a visualization.

> "I found that there were too many visualizations which made it difficult to remember which one does what. Some visualizations looked very similar. I sometimes forgot which visualization I was looking at or where I came from." PID2.

All visualizations supported navigation by two finger or hand zooming and panning gestures on the background canvas. Occasionally, when multiple participants tried to zoom at the

same time, SourceVis was confused as to what gesture to perform. Navigating (zooming and panning) in the visualizations was really only effective when one user was in control.

> "With multiple users there were issues with two people zooming by accident." PID4.

Many elements in the visualizations were designed to behave in a similar way so that users can manipulate elements (e.g. drag, tap and hold) and perform interaction gestures (e.g. two finger zoom in and out, and pan for navigation) in a consistent manner. Occasionally there were some unexpected behaviour where some elements could be moved such as the submenus in the options menu and touching some text labels had different effects across different visualizations.

> "It was not always obvious what part of the interface could be touched and what it would do, and there were some inconsistencies as to how to close specific popups." PID5.

We located the main menus primarily on the left hand side of visualizations which made it difficult for participants to interact with them if they were standing on the other side of the table. If a participant wanted to display a new visualization from the start screen and one visualization was being displayed at full screen, it required closing or minimizing the current displayed visualization. Creating flexible free floating menus for the options and visualizations on the start screen to open anywhere in SourceVis will make it easier for participants to launch new visualizations.

> "It was a bit cumbersome if you wanted to for example click on something but the buttons were on the other side, then you had to ask your team mate to do it." PID5.

The pie menus caused some issues when displayed, especially if the canvas was zoomed in or out at a very short or long way. Either the pie menu was too small, or too big and off screen. Most of the time the pie menu worked as expected and participants could read the sub menus. This could easily be rectified by displaying the pie menu at the same zoom level.

> "Pie menu needs to be independent of zoom, as it becomes a little clunky and hard to read when half of it disappears off the bottom of the screen." PID3.

When a search query was issued the entities that matched the query were displayed while other entities were hidden. Any subsequent search would search upon entities that were currently visible and not the hidden entities. To make a subsequent search on all the original entities a user would have to tap the reset option, which redraws the visualization in the original state. Another way to implement this would have been to highlight the entities found and fade the colour of other entities.

> "The search option should reset the list at the beginning of a new search." PID1.

The resolution of the table was 1280x800 pixels which is lower than most contemporary desktop computers. This resolution occasionally made some text hard to read especially when the current view was zoomed out a long way.

> "This particular table does suffer from low resolution, making reading small text hard." PID1.

Detecting touch points was inaccurate sometimes which caused some participants to be cautious when they touched the table while their colleague was also touching. This led one pair to take turns when interacting with the table.

> "I would have liked to manipulate the interface at the same time as my partner, but didn't due to accuracy issues with the table. Sometimes we both touched the same object at the same time, which lead to surprising changes in size and position of objects." PID2.

D. Quantitative Findings

All pairs answered all the questions correctly. The first pair took 22 minutes, second pair 24 minutes, and third pair 29 minutes; for an average of 25 minutes.

Figure 7 shows the average perceived effectiveness of the techniques each participant stated in the post-survey, with 0 being least effective and 10 being most effective. There is no rating for the System Package Evolution and Package Evolution visualizations as participants did not use these when answering the questions. Most of the visualizations ranked between 7 and just over 8 on average. The overall perceived effectiveness ranked between 7 and 8, with 7.8 on average.

The Metrics Explorer was perceived as the most effective technique at 8.2 on average. The Exploration visualizations ranked the highest between 7.7 and 8.2 on average. The chart based visualizations ranked between 7.5 and 8 on average, with the Toxicity Chart perceived as the most effective chart visualization. The System Hotspots View and Class Evolution visualizations both use Polymetric encodings and ranked quite similar approximately 7.5 on average. The dependency visualizations were perceived least effective and ranked between

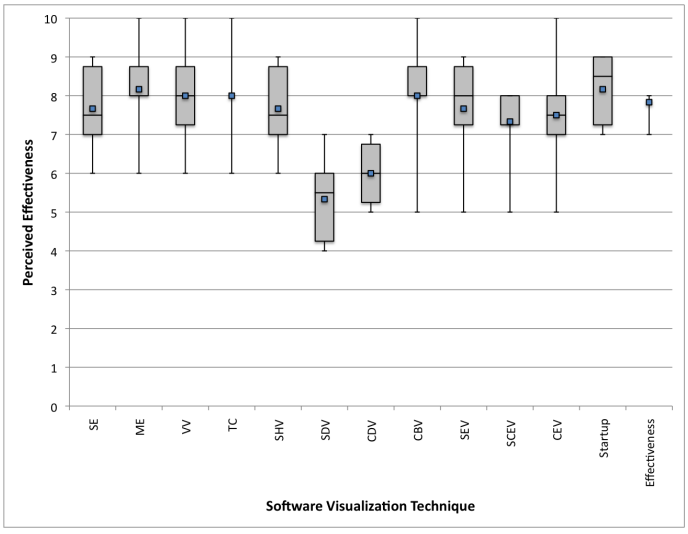


Fig. 7. Perceived effectiveness of the software visualization techniques by participants, 0 being least effective and 10 most effective.

5 and 6 on average. No participants ranked the dependency visualizations higher than 7. At least one participant ranked the Metrics Explorer, Vocabulary, Toxicity Chart, Class Blueprint, and Class Evolution View techniques the most effective at 10. At least one participant ranked the Class Blueprint, System Evolution, System Class Evolution, and Class Evolution techniques the least effective at 5. The Startup Screen ranked between 7 and 9, with 8.2 on average.

We asked participants if they had access to a large multi-touch table what software development activities they would use it for. The activities were: planning team development tasks, meetings that were either face-face or video conferences, designing the architecture of a system using modelling techniques, implementing code by pair programming, analyzing code or systems and conducting code reviews, testing and debugging code, not at all, or something else. 35% stated design, 29% planning, 24% analysis, 6% both implementation and testing, and 0% not at all and something else.

E. Limitations

Our study was conducted with a small number of participants, who were a convenience sample of computer science graduate students. This was the first time participants had used SourceVis, hence they were novice users. This was a study on understanding existing software systems they may not have been familiar with and had made no contributions to. The questions were not numbered on the sheet provided nor did we vary the order, but all participants answered the questions in the order they were listed. This may have led to a learning bias on the visualization techniques. As this was a qualitative user study we were less concerned with how long it took participants to answer the questions, instead we wanted to have a general understanding of how long it would take to conduct the study. As we asked participants to think aloud we accounted for this time when measuring how long it took them to complete the tasks.

IV. Discussion

Developing upon MT4j required a significant amount of effort to build something substantial. Every object that was displayed had to be rendered as a MT4j geometric shape so all classes had to extend one of the existing built in shapes. There existed very few user interface controls and menus, hence it was time consuming to build our own widgets. The visualizations relied heavily on text so we tried to make the text as easy to read as possible and allowed text objects to be scaled, rotated, and oriented in different positions. Nonetheless some participants found the text was hard to read when viewing a visualization in an individual window as the text became blurred. SourceVis currently only supports mouse, finger, and hand input but we would like to support others forms including: fiducial markers, digital pens, and tablets.

Adding some advanced techniques could help the usability of some visualizations. The System Dependency displays many classes which are hard to read when displaying all classes. Adding a technique that would enhance the class names such as a focus + context technique like a fish eye lens could make it easier to read a class name. The Class Dependency displays one class' dependencies at a time and new visualizations need to be launched to view a dependent class' dependencies. Instead of launching a new visualization having an option to expand the dependencies of a dependent class would form a much larger graph visualization and nodes could also be collapsed.

Extending SourceVis to support other visualizations is possible. As we have a representation of programming language constructs, any visualization that utilizes these Java entities can make use of these classes. The generic options menus can be added to new visualizations, likewise the gestures and custom pie menu. If visualizations require charts they can use the chart integration class. We have integrated two open source libraries, therefore integrating other libraries should be possible too.

The table has a large display but the resolution is rather low at 1280x800 pixels. Ideally we would like the table to have a much higher resolution when compared with current desktops as this will allow greater precision for work place tasks. Using LCD screens instead of projectors would create a much higher resolution. Anything that was put onto the table generated a blob which affected SourceVis be it fingers, paper, or some other physical object. Putting objects such as paper on the table minimizes the available display space for viewing and interacting. The only way to prevent participants from being able to put paper on the table would have been to remove the paper altogether or create physical frame borders around the table which paper or other objects could be placed upon. Instead we chose to use nearby tables for participants to place paper. We also tried having separate windows displayed on the screen which contained the questions, but we felt that this would effect participants switching between the visualizations and the questions and take up too much screen real estate.

The table is a prototype and the participants experienced a range of hardware performance issues from slow rendering, fake touches from hovering fingers just above the surface, and unexpected behaviour. The hardware we used was a Dell Optiplex 760 Intel Core 2 Duo, 3.0 GHz, 8GB Ram, ATI Radeon HD 3400 Series. Some visualizations had issues with displaying lots of data on the screen, which meant participants had to wait until the system had completed the rendering before they could move onto the next interaction. SourceVis worked very well for small to medium sized data sets on all of the visualization techniques. The table employs a diffused illumination setup which makes it hard to prevent hover touch points and caused some fake touches to be detected slightly above the surface. Sometimes participants were touching the same object on the screen at the same time and occasionally the object would all of a sudden move or change size unexpectedly due to unintended touches which caused confusion and frustration. We would like to explore commercial multi-touch tables such as the Microsoft PixelSense[5] table to see if it would alleviate the low resolution and performance issues.

[5]http://www.microsoft.com/en-us/pixelsense/default.aspx

V. Related Work

Some studies have explored how tools support collaborative software understanding [13] and collaborative software visualization [24], but neither focused on interactive surfaces. A number of prototypes have explored using different interactive devices to support collaborative software development. FastDash is an ambient visualization system displayed on a projector for providing awareness about developer activities in software teams [3]. CodeSpace uses shared touch screens, mobile touch devices, and Kinect sensors to share information during developer meetings [6]. CodePad uses peripheral interactive devices ranging from portable tablets to tables to support developers in maintaining their concentration [19]. CoffeeTable is a visual system that uses digital pens and Wii Remotes for interaction to assist with software development processes such as what developers are working on, a summary of the architecture, and work flow activities [10]. CREWW is an interactive CRC card system that uses Wii Remotes for collaborative requirements engineering [5]. Other researchers have also explored using multi-touch tables for activities including: software exploration [4], pair programming [23], code reviews [16], and software modeling [17].

Polymetric Views [14] are a well adopted suite of techniques for visualizing software metrics. A quantitative user experiment was conducted with CodeCity which utilizes Polymetric View techniques and the results validated that CodeCity outperformed in both correctness and completion times of two state of the art exploration tools (Eclipse and a spreadsheet of metrics data) [26]. Another common technique is to visualize the evolution of contributions made by developers [7], [18], whereas SourceVis focuses on how the structure of a software system has evolved.

VI. Conclusions

We presented SourceVis, a collaborative software visualization application for co-located software development teams to use on large multi-touch tables. We designed SourceVis by focusing on three considerations for collaborative visualization applications in co-located environments [11]: design for multiple users, support multiple visualizations, and display visualizations on large shared interactive surfaces. We described the design, visualization features, and implementation details of SourceVis. We presented findings from a small qualitative user study conducted with computer science students.

The implications for collaborative software visualization with multi-touch tables are visualizations should support multiple users and make it easy for them to control the interface. Visualizations should be viewed from different view points and angles. Menus should be adaptable to be displayed from anywhere. Switching between visualizations should be seamless. The display screen should be large and high resolution so that team members can easily share the visualizations.

In the future, we plan to conduct a larger qualitative study of SourceVis with professional software developers. We would also like to conduct a quantitative study to compare SourceVis against state of the art exploration tools (e.g Eclipse).

Acknowledgments

This work is supported by the New Zealand Ministry of Science and Innovation, TelstraClear, Victoria University PhD Completion Scholarship, and the Canadian NSerc SurfNet network.

References

[1] C. Anslow, S. Marshall, J. Noble, and R. Biddle. SourceVis: a tool for multi-touch software visualization. In *Proc. of Interactive Tabletops and Surfaces (ITS)*. ACM, 2011.

[2] C. Anslow, J. Noble, S. Marshall, E. Tempero, and R. Biddle. User evaluation of polymetric views using a large visualization wall. In *Proc. of SoftVis*. ACM, 2010.

[3] J. Biehl, M. Czerwinski, G. Smith, G., and Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *Proc. of CHI*. ACM, 2007.

[4] S. Boccuzzo and H. Gall. Multi-touch collaboration for software exploration. In *Proc. of ICPC*. IEEE, 2010.

[5] F. Bott, S. Diehl, and R. Lutz. CREWW: collaborative requirements engineering with Wii-remotes. In *Proc. of ICSE*. ACM, 2011.

[6] A. Bragdon, R. DeLine, K. Hinckley, and M. Morris. Code space: touch + air gesture hybrid interactions for supporting developer meetings. In *Proc. of Interactive Tabletops and Surfaces (ITS)*. ACM, 2011.

[7] A. Caudwell. Gource: visualizing software version control history. In *Proc. of OOPSLA Companion*. ACM, 2010.

[8] S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Verlag, 2007.

[9] N. Fenton and S. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing, 1998.

[10] J. Hardy, C. Bull, G. Kotonya, and J. Whittle. Digitally annexing desk space for software development. In *Proc. of ICSE*. ACM, 2011.

[11] P. Isenberg. *Collaborative Information Visualization in Co-located Environments*. PhD thesis, University of Calgary, 2009.

[12] P. Isenberg, D. Fisher, M. Morris, K. Inkpen, and M. Czerwinski. An exploratory study of co-located collaborative visual analytics around a tabletop display. In *Proc. of VAST*. IEEE, 2010.

[13] A. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proc. of ICSE*. IEEE, 2007.

[14] M. Lanza and S. Ducasse. Polymetric views-a lightweight visual approach to reverse engineering. *IEEE TSE*, 29(9):782–795, 2003.

[15] U. Laufs, C. Ruff, and J. Zibuschka. MT4j a cross-platform multi-touch development framework. In *Proc. of the Workshop on Engineering Patterns for Multi-Touch Interfaces at EICS*. ACM, 2010.

[16] S. Muller, M. Wursch, T. Fritz, and H. Gall. An approach for collaborative code reviews using multi-touch technology. In *Proc. of Workshop on CHASE*, 2012.

[17] S. Muller, M. Wursch, P. Schoni, G. Ghezzi, E. Giger, and H. Gall. Tangible software modeling with multi-touch technology. In *Proc. of Workshop on CHASE*, 2012.

[18] M. Ogawa and K-L. Ma. code_swarm: A design study in organic software visualization. In *Proc. of InfoVis*. ACM, 2009.

[19] C. Parnin, C. Görg, and S. Rugaber. Codepad: interactive spaces for maintaining concentration in programming environments. In *Proc. of SoftVis*. ACM, 2010.

[20] J. Schöning, J. Hook, T. Bartindale, D. Schmidt, P. Oliver, F. Echtler, N. Motamedi, P. Brandl, and U. von Zadow. *Tabletops - Horizontal Interactive Displays*, chapter Building Interactive Multi-touch Surfaces, pages 27–49. Springer Verlag, 2010.

[21] S. Scott, S. Carpendale, and K. Inkpen. Territoriality in collaborative tabletop workspaces. In *Proc. of CSCW*. ACM, 2004.

[22] J. Sillito, G. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *Proc. of FSE*. ACM, 2006.

[23] A. Soro, S. Iacolina, R. Scateni, and S. Uras. Evaluation of user gestures in multi-touch interaction: a case study in pair-programming. In *Proc. of the International Conference on Multimodal Interfaces (ICMI)*. ACM, 2011.

[24] M-A. Storey, C. Bennett, I. Bull, and D. German. Remixing visualization to support collaboration in software maintenance. In *Proc. of ICSM*. IEEE, 2008.

[25] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. Qualitas corpus: A curated collection of Java code for empirical studies. In *Proc. of APSEC*, 2010.

[26] R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: A controlled experiment. In *Proc. of ICSE*. ACM, 2011.