

Interactive Multi-touch Surfaces for Software Visualization

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

Robert Biddle
School of Computer Science
Carleton University
Ottawa, Canada
robert_biddle@carleton.ca

ABSTRACT

Most software systems are developed by teams of people. The tools used to develop and maintain these systems are primarily designed from a single-user perspective and are bound to Integrated Development Environments (IDEs). These design decisions do not allow users to collaboratively navigate through software visualizations or to analyse software easily. We are investigating whether multi-touch table interaction techniques are more effective for co-located collaborative software visualization than existing single-user desktop interaction techniques. The implications of our research will help inform developers how to design better visualization applications for interactive multi-touch surfaces.

ACM Classification: H1.2 [User/Machine Systems]: Human Factors; H5.2 [Information interfaces and presentation]: User Interfaces. - Multi-touch user interfaces.

General terms: Experimentation, Human Factors.

Keywords: Multi-touch, software visualization, user study.

1. INTRODUCTION

Maintaining large software systems requires understanding the underlying structure, which is a hard task. Understanding software is often a social activity and involves teams of software developers. Software visualization aims to help with techniques to visualize the structure, behaviour, and evolution of software [4]. Visualization tools designed for a single user perspective make it hard for developers to analyse software when working together in a co-located environment (within the same room) using the same interface.

Multi-touch table user interfaces are an example of a co-located collaborative tool which could be used for software analysis. We are investigating whether multi-touch table interaction techniques are more effective for co-located collaborative software visualization than existing single user desktop interaction techniques. In this paper we discuss past work, our multi-touch software visualization prototype, a qualitative user study, challenges, and future work.

2. PAST

An approach to understand how participants use interactive multi-touch surfaces follows a qualitative research method. This method has been successfully adopted to provide insight into the design of tabletop displays for information visualization [9], visual analytics [10], and collaborative design [16].

Ko et al. have explored how software development tools support collaborative software understanding [11]. Storey et al. have explored collaborative software visualization [18]. These studies and very few tools have explored collaborative interactive multi-touch surfaces for software development or software visualization.

Parnin et al. suggest the addition of peripheral interactive spaces to programming environments for supporting developers in maintaining their concentration using touch based devices ranging from portable to tabletops [15]. Anslow et al. conducted a qualitative user study to find out the effectiveness of an established software visualization technique using a large visualization wall [1]. Boccuzzo et al. have made an extension to their 3D tool for software exploration with multi-touch features [2]. Hardy et al. created 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 [7]. Bott et al. created an interactive CRC card system that uses Wii Remotes for collaborative requirements engineering [3].

3. PRESENT

We are developing a prototype called SourceVis for large multi-touch tables to analyse the structure, evolution, and vocabulary of software systems, see Figure 1. The visualizations adapt existing information and software visualization techniques and modify them to support multi-touch and multi-user interaction. SourceVis is built upon MT4j [13].

3.1 Interaction

We envision developers working in groups (2–3) with our prototype to explore at a high level what parts of a software system are large, problematic, and need to be refactored.

Users first load a system by tapping on the menu to select a name of a system before any visualizations can be displayed. Users start a visualization by tapping on the icon of the visualization. Each visualization is displayed in a rotatable and scalable window with options to display at full screen or close. Multiple visualizations can be displayed at once.



Figure 1: SourceVis. Users interacting with visualizations: A. Metrics Explorer, B. Class Blueprint, C. System Hotspots View, D. Wordle Vocabulary.

Users can interact with individual elements in the visualizations. To select an element users tap, drag with one finger, and rotate and resize with two fingers. Double tapping an element displays properties about that element. Multiple elements can be grouped by drawing a shape around them using a lasso gesture and then subsequently move them around the display. A tap and hold gesture displays a new visualization type. Zooming uses a pinch gesture with one or two hands and navigation by scrolling with two fingers.

For a multi-user scenario one developer might be looking at the overview of a system while another developer is looking at the details of a class. This allows users to orient visualizations to where they are standing and make some visualizations larger than others depending on the size of the system.

3.2 Visualizations

The visualizations are grouped into three categories: exploration, structure, and evolution. The exploration category contains visualizations that show metrics about a system, vocabulary employed in entities, and a touch enabled web browser for documentation. The structure category adapts Polymetric Views [12] to multi-touch. The evolution category shows how a system has evolved over time focusing on

structural changes and developer revision histories. We now describe some of the visualizations from the exploration and structure categories.

Metrics Explorer (Annotated A. in Figure 1). Shows metrics about the different modules in a system such as the number of packages, classes, methods, and variables [6]. All the packages in a system are first displayed alphabetically. Tapping a package displays the metrics about the package and the classes it contains. Likewise tapping a class displays metrics, methods, and variables. In the figure a user has done a tap and hold gesture on one of the classes which has displayed the associated Class Blueprint which has partially obscured the Metrics Explorer.

Class Blueprint (B. in Figure 1). Shows the dependencies and references between methods and attributes within a class and adapted from Lanza et al. [12]. The visualization is broken into five layers. The first four layers relate to methods and the final layer to attributes. The methods and attributes have a different fill colour depending on which layer they belong to. Likewise the edges for the dependencies and references. The weight of an edge can be adjusted by moving a slider up (for thickest) or down (for invisible). In the figure a user has selected one of the interface methods by tapping and holding which has highlighted in green one accessor method that it calls and one attribute it accesses. With their other hand they have selected one of the attributes which has highlighted a method that references that attribute which is the same accessor method the interface method calls.

System Hotspots View (C. in Figure 1). Shows large packages and classes in a system and adapted from Lanza et al. [12]. Packages are displayed down the Y axis and classes from each package along the X axis. In the figure a user has double tapped on the package label which has displayed properties about the package in a linked yellow box. The properties include the total metrics, and options for visually sorting the classes in the package alphabetically, ascending, or descending by individual metrics or groups of the metrics. Each class is represented as a rectangle where the width indicates the number of variables and height number of methods. The colour of a class is represented as the number of lines of code. The darker the rectangle the more lines of code the class contains. Different border colours represent the type of class (e.g. red is interface, blue abstract class, no border a concrete class). In the figure a user has also double tapped on the large black class `java.awt.Component` which has displayed the class properties including metrics. Classes can be moved around the visualization to be compared with other classes and can be grouped together. A tap and hold gesture on a class displays the associated Class Blueprint.

Vocabulary (D. in Figure 1). The modified Word Cloud and Wordle provide a quick overview of the vocabulary used in the entities (e.g. packages, classes, methods) of a software system to understand the coding standards employed. In the figure a user has moved some words around, grouped some of the larger words together, and filtered out some of the smaller words. We intend to link this visualization with others so that selecting or grouping words highlights entities in other displayed visualizations that use these words.

3.3 User Study

We conducted a user study with 10 participants using Source-Vis. The aim of the study was to collect data about how effective our software visualization techniques are for program comprehension in order to validate our interactive and visualization design decisions following a qualitative approach [9].

Procedure. Participants were given an information sheet, consent form, and a pre-study questionnaire. The questionnaire asked about their demographics and background experience. With each participant's consent we recorded their actions and asked them to think aloud. The study was conducted with a 48 inch low cost rear diffuse illuminated multi-touch table that we built, based on some existing designs [5]. Following the pre-study questionnaire, participants were given a warm up exercise by experimenting with the example applications from MT4j for five minutes. For the user tasks participants were asked 14 program comprehension questions similar to the types of questions software developers ask within industry [17]. The questions asked participants to identify, count, and find information within the same set of visualizations. The sample data set was the Java Standard API. Participants recorded their answers to the questions on a sheet attached to a clipboard. We recorded the time it took participants to complete the user tasks. Participants completed a post-study questionnaire which asked for their opinion on the effectiveness, strengths, and weaknesses of the interaction capabilities and the visualizations.

Participants. There were eight males and two females, who worked in pairs. One group of participants had known each other for 18 months while the other pairs knew each other for 12 months, 6 months, and 2 months. The other pair did not know each other. The age of participants was in the range of 25-29. All participants had a bachelors degree in computer science and three had a masters degree. Of the participants; one was currently an honours student (4th year undergraduate), three masters students, and six PhD students. Four participants had used some software visualization tools before but not on a frequent basis. Seven of the participants had used desktop touch screens or touch tables before. All had experience in programming using the Java Standard API.

Limitations. The small number of participants, who were a convenience sample, of graduate computer science students. The warm up exercise was some example applications not our visualizations. This was the first time participants had used our prototype before. The questions we asked 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 on the sheet. This may have led to a learning bias. When measuring how long it took for participants to answer the questions we had to account for the time participants spent thinking aloud and recording their answers on the sheet attached to the clipboard.

3.4 Results

Perceived Effectiveness. In the post-survey the Word Cloud ranked as the most effective technique followed by Metrics Explorer and Wordle. The System Hotspot Views and Class Blueprint ranked the same and slightly below the others.

Time and Errors. The first pair took 20 minutes to complete the user tasks, second pair 28 minutes, third pair 22 minutes, fourth pair 24 minutes, and fifth pair 21 minutes, for a mean average of 23 minutes. Pairs one, two, and five answered all the questions correctly, for a total of 36 (100%). Pair three received 34 (94%), and pair four 33 (92%).

Metrics Explorer. This visualization provided an overview and gave participants a clear summary of the metrics about a system. The white background made it easier to read the names of packages and classes, and colour to highlight the selected entities made them stand out. One participant would have liked to have seen inheritance information about classes. A couple of participants were not sure what to expect when they tapped on the name of an entity such as where information was going to be displayed. Nor did some of them know the best location for the metrics information, some suggested putting it next to the name of the entity instead of the left hand side of the visualization. Some packages contain many classes which required lots of scrolling to find a class, and some participants suggested adding a text-based search.

Class Blueprint. Participants liked how this visualization showed what methods and attributes were connected to each other. All participants commented that the highlighting of edges made it an effective way to answer questions when using this visualization. They also liked how multiple users could highlight more than one edge at a time. The slider that adjusted the weight of the edges was a welcomed addition, but some participants were not aware of it. Since the edges cross each other and methods this made it confusing for some participants to be able to read the names of methods. One participant suggested that if different methods were selected then only show the intersection if one exists.

System Hotspots View. This visualization made it easy for participants to compare the different entities in the system as packages were initially laid out in alphabetical order and classes grouped in packages, plus the ability to move entities around the visualization. Once the participants remembered the information cues (metrics and colour encoding) it was easy to identify certain aspects of a system such as the types of classes and large classes. The properties windows made it easy to determine precise information about a package or a class. The sorting provided a quick way to answer some of the identify and count questions. The example system used was large, participants found that it was hard to get an overview of all the information because when the visualization started it was half zoomed in. This meant lots of scrolling to find what they were looking for. Some classes in the visualization were small and required zooming in to validate their colour.

Vocabulary. All participants found the word font size and background colours made it easy to understand these visualizations. Long words made it slightly harder to interpret. It was hard to compare words if they were not next to each other as the absolute size is not easy to see and nor is it clear what metric was being used. Separating grouped words with a gesture was not easy to do. Adding colours to the different words helped to distinguish between them. Words can be overlapped which helped when comparing the size of two

words together, but sometimes there was too much overlapping. A slider was added to filter out words, but it confused some participants as they suggested it was not intuitive.

3.5 Lessons Learnt

Support Collaboration. Seven of the participants stated that it was easier to work in groups than individuals because you can each look at several parts of the same visualization at once, divide the tasks up, and discuss the answers.

Display Multiple Visualizations. Most of the participants did not display multiple visualizations at once, even though they knew they could. They only did this when they navigated to a Class Blueprint. Some commented having different ways of looking at a system with multiple visualizations would be beneficial and suggested adding synchronization features.

Provide Visualization Help. Participants were not familiar with our visualizations, so providing help documentation such as how to interact with the visualization and what the encoding means would help improve usability.

Use Higher Resolution. The resolution of our touch screen was only 1280x800 pixels so having a higher resolution would make it easier to display more visualizations at once.

Display Radar View. It is important for users not to lose navigation context when visualizations are large and require lots of scrolling. Displaying a small radar view of the current context would help with navigation, but would come as an expense by taking up some screen real estate.

User Study. Make questions more subjective which will involve more exploration with the visualizations, use industry professionals as participants, conduct a study over a longer period of time and in a real world setting, and compare our prototype with a control visualization tool.

4. FUTURE

Challenges. Our visualizations are primarily viewed in isolation, we need some way to synchronize them together especially when some of them use the same underlying data source. Since our visualizations of the software are not displayed within an IDE it is important to be able to link the visualizations somehow with the underlying source code. Our focus has been to visualize the source code through exploration and navigation techniques, but some participants expressed that they wanted to enter text to do this and to program using the touch table. In order to support text entry we could adopt an existing text-entry method [8]; or to support programming we could scale a programming model approach for tablets that uses tiles and behaviour constructs [14].

Future Work. We intend to create a more comprehensive prototype and are currently working on evolution visualizations. Once our prototype is more mature we plan to conduct a large collaborative quantitative between subjects user experiment similar to Wettel et. al. involving industry professionals [19].

ACKNOWLEDGMENTS

This work is supported by the New Zealand Ministry of Science and Innovation for the Software Process and Product Improvement project, a Telstra Clear Postgraduate Scholarship, and the Canadian NSERC SurfNet Network.

REFERENCES

1. 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.
2. S. Boccuzzo and H. C. Gall. Multi-touch collaboration for software exploration. In *Proc. of International Conference on Program Comprehension (ICPC)*. IEEE, 2010.
3. F. Bott, S. Diehl, and R. Lutz. CREWW: collaborative requirements engineering with Wii-remotes (NIER track). In *Proc. of ICSE*. ACM, 2011.
4. S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Verlag, 2007.
5. J. Schöning et al. *Tabletops - Horizontal Interactive Displays*, chapter Building Interactive Multi-touch Surfaces, pages 27–49. Springer Verlag, 2010.
6. N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing, 1998.
7. J. Hardy, C. Bull, G. Kotonya, and J. Whittle. Digitally annexing desk space for software development (NIER track). In *Proc. of ICSE*. ACM, 2011.
8. U. Hinrichs, M. Hancock, C. Collins, and S. Carpendale. Examination of text-entry methods for tabletop displays. In *Proc. of the Workshop on Horizontal Interactive Human Computer Systems (TABLETOP)*. IEEE, 2007.
9. P. Isenberg. *Collaborative Information Visualization in Co-located Environments*. PhD thesis, Uni of Calgary, 2009.
10. 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.
11. A. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proc. of ICSE*. IEEE, 2007.
12. M. Lanza and S. Ducasse. Polymetric views—a lightweight visual approach to reverse engineering. *IEEE Trans. on Soft. Eng.*, 29(9):782–795, 2003.
13. 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 Symposium Engineering Interactive Computing Systems (EICS)*. ACM, 2010.
14. S. McDermid. Coding at the speed of touch. In *Proc. of Symposium on New Ideas in Programming and Reflections on Software (Onward!)*. ACM, 2011.
15. C. Parnin, C. Görg, and S. Rugaber. Codepad: interactive spaces for maintaining concentration in programming environments. In *Proc. of SoftVis*. ACM, 2010.
16. S. Scott, S., Carpendale, and K. Inkpen. Territoriality in collaborative tabletop workspaces. In *Proc. of CSCW*. ACM, 2004.
17. J. Sillito, G. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *Proc. of FSE*. ACM, 2006.
18. M-A. Storey, C. Bennett, I. Bull, and D. German. Remixing visualization to support collaboration in software maintenance. In *Proc. of ICSM*. IEEE, 2008.
19. R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: A controlled experiment. In *Proc. of ICSE*. ACM, 2011.