User Evaluation of Polymetric Views Using a Large Visualization Wall

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

Ewan Tempero
University of Auckland, New Zealand
ewan@cs.auckland.ac.nz

Robert Biddle
Carleton University, Canada
robert_biddle@carleton.ca

ABSTRACT

There are few visualization techniques for displaying complex software systems with large numbers of packages and classes. One visualization technique is the System Hotspots View, whose effectiveness has yet to be validated by any empirical studies. We have conducted a user evaluation to see whether participants of our modified System Hotspots View using a large visualization wall can accurately identify key measurements and comparisons in the underlying software system. The results of our user evaluation indicate that participants were able to effectively use our modified System Hotspots View to explore the example domain: version 1.6 of the Java API. Our observations also indicate that there are issues around interacting with the visualization wall.

Categories and Subject Descriptors

H.1.2 [**User/Machine Systems**]: Human Factors; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Evaluation/methodology*

General Terms

Design, Human Factors

Keywords

Large displays, software visualization, user evaluation, visualization wall

1. INTRODUCTION

Effective software visualization techniques help users understand one or several fundamental aspects of complex software systems [6]. One aspect of software systems that would benefit from visualization support is software metrics, such as those that capture the static information around package, class or method size and interdependency [7]. Polymetric Views are techniques that can visualize these software metrics data [9].

Our contribution is a user evaluation of a modified version of one of the Polymetric Views: the System Hotspots View, when applied to software metrics and when displayed on a large visualization wall. Our results identify the accuracy of the users' mental model of the underlying software metrics data, as constructed by interacting with the visualization. As well as this, our results and discussion identify how the large visualization wall impacted the users' behaviour and experience.

Our contribution is novel and useful. To our knowledge there have been no published user evaluations for any of the Polymetric Views. The specific case study we use in our evaluation involves exploring the structure and size of packages and classes from version 1.6 of the Java Standard API on a large visualization wall.

The rest of this paper is organised as follows. Section 2 describes Polymetric Views. Section 3 outlines our user evaluation, Section 4 presents the results, and Section 5 discusses the limitations. Section 6 gives an overview of related work. In Section 7 we present our conclusions and discuss directions for future work.

2. POLYMETRIC VIEWS

Lanza and Ducasse [9] describe Polymetric Views as visualization techniques to help understand the structure and detect problems of a software system in the initial phases of a reverse engineering process. The visualizations use metrics data about the size of classes, packages, and their dependencies. One of the views is the System Hotspots View.

2.1 System Hotspots View

The System Hotspots View uses the following software metrics: Number of Instance Variables (NIV), number of Weighted Methods per Class (WMC), and number of Lines of Code (LOC) [7]. Classes from a software system are grouped in the X-Y axis according to the size of the class. For each class the width indicates the NIV and the height indicates the number of WMC. Colour indicates the number of LOC, the darker the class the more LOC it contains.

We modified the System Hotspots View as it is hard to determine how many packages there are in a system, how many classes are in packages, and the differences between types of classes. Figure 1 illustrates our modified System Hotspots View technique. We grouped classes by packages

SOFTVIS'10, October 25–26, 2010, Salt Lake City, Utah, USA.

down the Y axis and ordered the classes in the packages alphabetically along the X axis. We added text labels for packages. We coloured borders for classes to represent the different kinds of classes: concrete classes have blue borders, interfaces red borders, and abstract classes green borders.

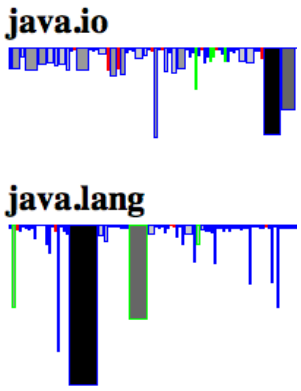

Figure 1: Modified System Hotspots View. Extensions: packages ordered in Y axis, text labels for packages, classes grouped in packages, and coloured borders for different kinds of classes.

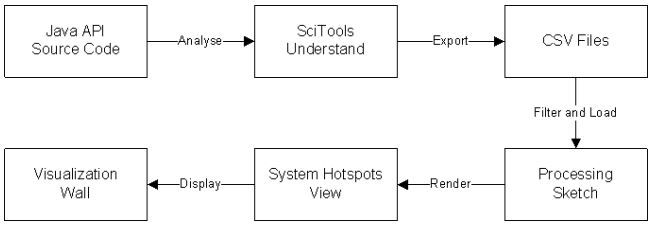

Figure 2: Software visualization pipeline.

2.2 Software Visualization Pipeline

Figure 2 shows the pipeline of tools we used to create our software visualizations. We first downloaded the Java Standard API 1.6 source code from the Sun Microsystems web site[1]. Next we loaded the source code into a SciTools Understand Project and executed the metrics analyser feature which exported the selected software metrics into a CSV file. We filtered the files and then loaded them into a Processing [13] sketch where we implemented our modified System Hotspots View technique. Upon execution of the sketch large static images are generated which can then be viewed on a large visualization wall.

3. USER EVALUATION

We wanted to find out how effective our modified System Hotspots View technique is for visualizing large packages and classes using a large visualization wall.

3.1 Procedure

We asked participants to complete some tasks with our modified System Hotspots Views by answering questions about the size of packages and classes in the Java Standard API 1.6. The questions were aimed at discovering how accurately participants perceived the absolute size and relative differences of elements in the System Hotspots View. These questions were not designed to be representative software comprehension tasks. The focus was on how effectively the System Hotspots View represented the underlying data rather than on how useful the data was.

The Java Standard API 1.6 was selected as a case study as it is a large commonly used software system. There was one System Hotspots View for each of the three top level packages in the API defined as `java`, `javax`, and `org`, see Figure 4. For some of the questions we displayed an additional zoomed in view of a package. The Java Standard API 1.6 contains over 200 packages, nearly 5800 classes, 9500 variables, nearly 50,000 methods, and 523,500 lines of code.

The System Hotspots Views were displayed on a large visualization wall. Figure 3 illustrates the visualization wall which has 12 screens arranged 4 (width) x 3 (height). Each individual display is 2560 x 1600 pixels for a total display of 10240 x 4800 pixels. The width of the visualization wall is 2800mm and the height is 1400mm. The bottom edge of the wall is 650mm off the ground. The visualization cluster software is controlled by Rocks[2] and the user interface by the Scalable Adaptive Graphics Environment (SAGE)[3].

Figure 3: Visualization wall.

[1]http://download.java.net/jdk6/source/

[2]http://www.rocksclusters.org

[3]http://www.sagecommons.org

Each participant was given an information sheet about the evaluation, consent form to sign if they agreed, and completed a pre-study questionnaire to find out their Java software development experience. With each participant's consent we video recorded their actions when completing the tasks. We asked participants to think aloud when answering the questions so that we could capture their thoughts about their actions, perceptions, and expectations regarding the application's interface and functionality. Getting the users to talk about their actions and thoughts enabled us to gain insight into how each user views the computer system, identification of their misconceptions, and what parts of the interface caused the most problems. Measurement of the user tasks was done by recording the time taken to answer the total number of questions and any errors.

At the end each participant completed a post study questionnaire. The questionnaire asked for participants opinion on the strengths and weaknesses of the System Hotspots View and the visualization wall. We asked participants how effective they thought the System Hotspots View was for completing the user tasks on a scale of 1–10 where 1 was "very ineffective" and 10 was "very effective".

3.2 Participants

We conducted evaluations with 14 participants who were a convenient sample of academic, graduate, and undergraduate students from our department. 11 participants were male, three were female. Most participants were aged in the range 25–29. All participants except one had a bachelors degree, two had PhDs, the rest were studying towards masters or PhD degrees. 10 participants were undergraduate or post-graduate students, two participants were professional software developers and the other two participants were a university professor and a research fellow.

3.3 User Tasks

Each participant completed all the tasks using the same set of visualizations. The participants recorded their answers in a web form and we recorded the time it took to complete the tasks. The first six questions related to packages and the remaining five to classes.

1. How many packages are there in there in the Java API?
2. What is the biggest package in the Java API?
3. How many classes are there in the java.math package?
4. What package(s) in the Java API contain only interfaces?
5. What package(s) in the Java API contain no classes, abstract classes, or interfaces?
6. What packages contain classes that have the same number of attributes as methods?
7. What is the biggest class (including abstract classes and interfaces) in the Java API?
8. What is the biggest interface in the Java API?
9. What is the biggest abstract class in the Java API?
10. How much bigger is the java.awt.Component class than the java.awt.Container class?
11. How much bigger is the java.awt.Component class than the java.awt.Window class?

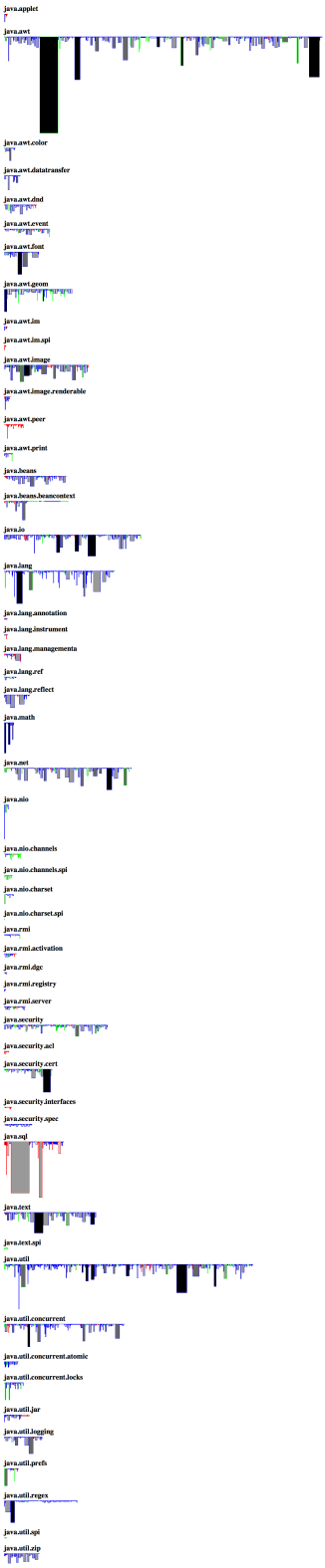

Figure 4: System Hotspots View of the top level java package of the Java Standard API 1.6.

Table 1: User tasks results. Numbers (rows 1–11) are percentages. Expertise and effectiveness are self assessed by participants. Ranking is the mean percentage of user task questions.

Participant id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Mean
1. Number packages	89	90	89	74	26	90	62	91	90	64	90	90	90	95	80
2. Biggest package	55	55	100	100	55	100	100	55	100	55	55	55	55	100	74
3. Classes in Math package	100	78	90	89	89	89	89	78	100	89	89	67	78	89	87
4. Packages only contain interfaces	69/2	77/0	69/0	38/0	15/0	69/3	38/0	54/1	69/0	46/2	69/0	69/5	62/1	62/0	58/1
5. Packages no classes	50/1	100/0	0/0	0/0	0/2	0/0	50/0	50/3	100/0	0/0	100/0	50/0	50/0	100/0	46/0
6. Same number of attributes & methods	44/4	33/11	28/5	28/0	6/0	72/2	22/1	28/7	44/5	33/1	28/1	17/1	22/3	56/11	33/4
7. Biggest class	100	100	100	34	100	100	100	100	100	100	100	100	100	100	95
8. Biggest interface	72	82	82	82	82	82	82	100	82	82	82	82	82	100	84
9. Biggest abstract class	57	100	100	100	100	100	100	100	100	100	18	100	100	100	91
10. Comp. vs. Cont.	44	0	28	88	91	55	31	55	37	37	22	73	91	91	53
11. Comp. vs. Wind.	73	0	73	86	69	97	48	86	73	73	53	69	69	69	67
Time taken (mins)	32	25	23	22	18	20	22	20	22	10	26	21	18	30	22
Expertise	7	3	2	7	9	9	6	5	4	6	7	4	3	9	5.8
Effectiveness	5	6	7	7	7	8	8	8	4	7	5	3	8	7	6.4
Ranking	68	65	69	66	58	78	66	73	82	62	73	64	74	89	71

4. RESULTS

We first report the results from the user tasks, then discuss the strengths and weaknesses of our modified System Hotspots View and the visualization wall.

4.1 User Tasks

Table 1 and Figure 5 present the results from the user tasks. The table shows the percentage correct for each participant. The percentage is calculated by the total number of correct answers for that question (e.g. X packages correct out of a total of Y * 100), total metric values of one entity divided by the total metric values of the biggest entity (e.g. NIV, WMC, LOC of package X divided by NIV, WMC, LOC of the biggest package Y * 100), or the difference in size of entities (e.g. biggest class Y divided by the smaller class X * 100). For questions 4–6 there is an additional number which shows the number of incorrect answers the participant gave (e.g. participant 1, Q4, 69/2, 69% correct and 2 answers incorrect). The boxplot shows distributions for each task where the mean is represented by a heavy horizontal bar, the inner quartiles as a box, the outer quartiles as whiskers, and any outliers as small circles.

4.1.1 User Tasks Results

1. Number of packages. There are 203 packages in the Java Standard API 1.6. The closest participant (14) answered with 193. Most of the participants gave an answer between 180-183. Some gave the following: 150, 125, and 52.

2. Biggest Package. The biggest package is javax.swing. Eight participants said that java.awt was the biggest and six participants said javax.swing was the biggest.

3. Number of classes in java.Math package. There are nine classes in the java.Math package. One participant answered with 10. Two correctly said nine classes, seven said eight, three said seven, and one said six classes.

4. Packages that only contain interfaces. There are 13 packages that contained only interfaces. Nine of the participants gave an answer between eight and 10. About 50% of participants gave incorrect additional packages.

5. Packages that contain no classes. There were only two packages that contained no classes, javax.lang.model and javax.rmi. Four participants got both packages, five participants identified one of the packages, and the remainder participants did not identify any.

6. Packages with same number of attributes as methods. We classified 18 packages that had classes with the same number of attributes as methods. We disregarded classes with less than 10 attributes and methods as they are hard to identify. All participants recognized that they should be looking for square boxes. Only two participants got more than 50% of the packages. The rest of the participants identified only some of the most obvious packages that met this criteria.

7. Biggest class in the Java API. The biggest class is Component from the java.awt package. 13 of the participants said Component while the remaining participant said Window also from the java.awt package.

8. Biggest interface in the Java API. The biggest interface is CSS2Properties from the org.w3c.dom.bootstrap package. Two participants got this correct. 11 participants said that the second biggest interface, DatabaseMetaData from the java.sql package was the biggest interface. One participant said that the third biggest interface, ResultSet from java.sql was the biggest. The DatabaseMetadata class has 61 attributes while CSS2Properties has zero. This made DatabaseMetadata take up significantly more area on the screen while CSS2Properties is just a single long red line.

9. Biggest Abstract Class in the Java API. The biggest abstract class is java.awt.Component. 12 participants said Component, one said Window from java.awt, and the remaining said AbstractButton from javax.swing.

10. Component versus Container. For this question we provided participants a scaled up image of the java.awt package. When considering NIV, WMC, and LOC, the Component class is 2.2 times bigger than the Container class. Participants gave a very wide range of answers from 2–10 times bigger as evidenced by the large box and whisker plot for question 10 in Figure 5.

11. Component versus Window. Again we provided a scaled up image of the java.awt package. When considering NIV, WMC, and LOC, the Component class is 2.9 times bigger than the Container class. Participants were more accurate for this question compared with 10 and answers ranged from 2.5–6 times bigger as evidenced by a condensed box and whisker plot for question 11 in Figure 5.

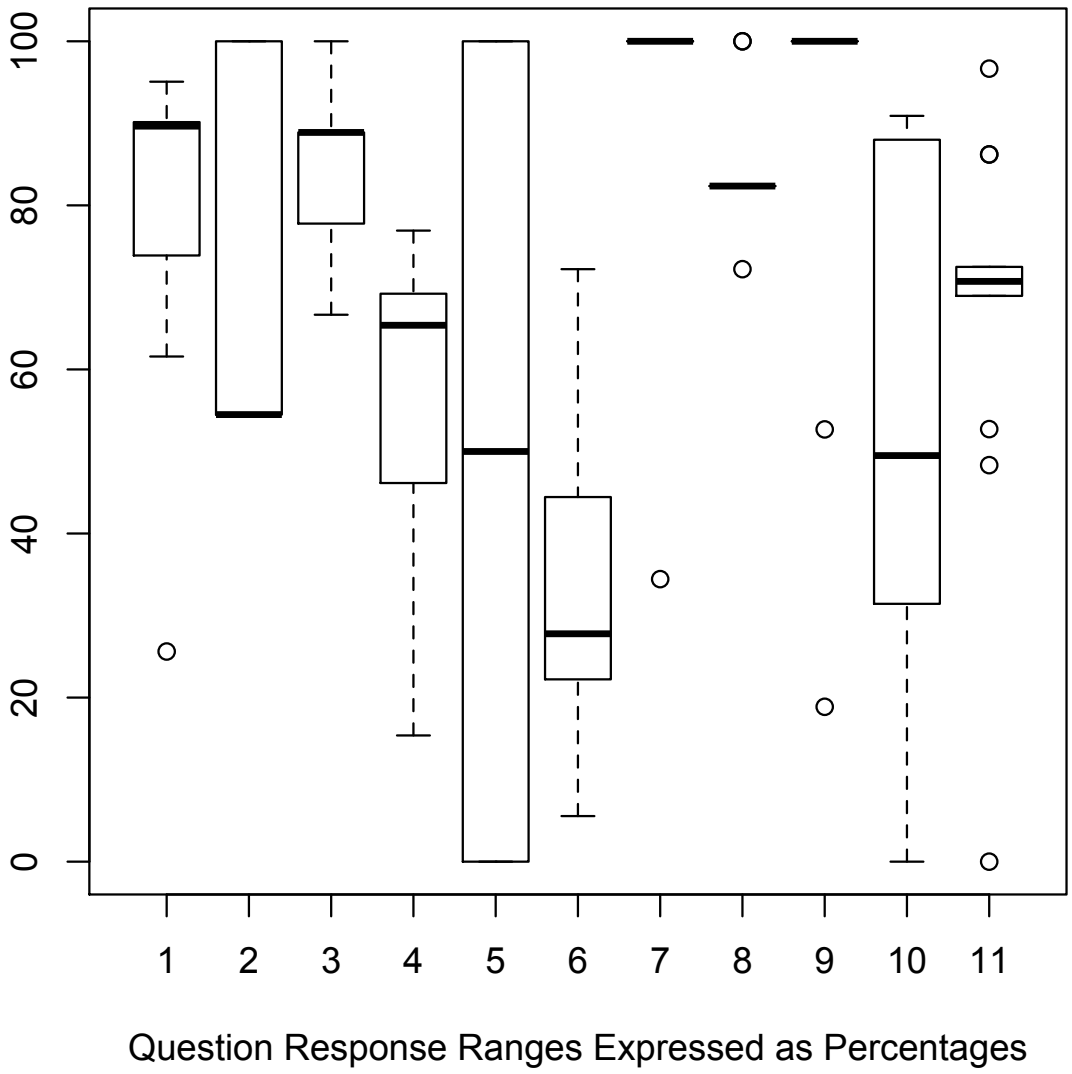

Figure 5: User tasks results, boxplot. 11 questions and 14 participants.

4.1.2 Discussion

Time, Expertise, Effectiveness, and Ranking

The time for the user tasks ranged from 10 to 32 minutes with a mean of 22 minutes. 12 of the 14 participants program in Java. Eight of the participants use version 1.6, two use 1.5, and one was unsure what version they used. The mean expertise of the participants with the Java API on a scale of 1–10 where 1 is "novice" and 10 is "expert" was 5.8. The participants' most frequently used packages were: java.util, java.lang, java.io, java.math, java.awt, and javax.swing. The most frequently used classes were: String, Scanner, Collections, and ArrayList. Similarly the mean effectiveness of the System Hotspots View was 6.4 which shows that the participants thought that the System Hotspots View was "effective" to answer the questions but not "very effective". Figure 6 shows the distributions of the scores of the participants in completing the user tasks. All participants scored over 50% accuracy and there was a wide divergence in the rankings.

Identifying and Counting. The System Hotspots View was very good for users to identify, count, and estimate numbers of entities in the Java API. The boxplot in Figure 5 shows small plots that have a mean greater than 80% for questions 1, 3, 7, 8, 9. For question 1 there is one outlier

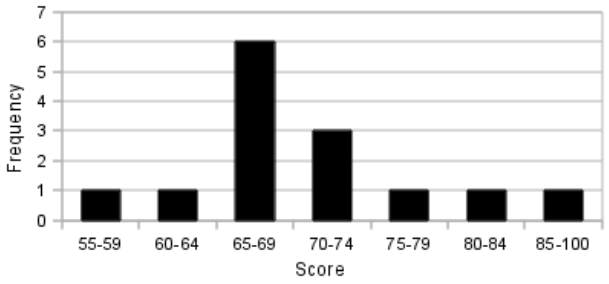

Figure 6: Distribution of the participant's scores.

where participant 5 has just guessed the answer. For question 3 nearly 90% of the participants got this correct.

Questions 7–9 have condensed box and whisker plots which shows that the participants got the same answer for the question with a couple of outliers each. For question 7, 95% of the participants answered this correctly. Participant 4 said that Window was the biggest class, hence the outlier below 40%, but perhaps that was a typographical error when the user was entering the data into the web form. For question 8 there is one outlier at the top which represents the two participants who got the answer correct. Since the DatabaseMetadata class took up more area than the CSS2Properties class participants perceived that DatabaseMetadata was bigger. For question 9, 91% got this answer correct. The two participants who got it wrong failed to see the green border for the Component class.

We believe participants would have done much better on question 2 if we had of had the images in Figure 4 to the same scale as the javax.swing package is significantly bigger than the java.awt package.

Filtering Information. The System Hotspots View is less good for filtering information or finding a subset of information. The main problem with answering questions 4 and 5 is that some of the information the participants had to find in the visualizations was very small.

For question 4 users had to list packages that contained only interfaces and they often included additional packages which were all concrete classes. The concrete classes in these packages were really small and hard to see. The most amount of incorrect packages were five by participant 12.

For question 5 the participants had to find packages that contained only a package label with no classes. This was a hard task as these packages were both small. There were only two packages, javax.lang.model and javax.rmi. Participants either got both, one, or none of them. This is evident in the boxplot were there is a value at 100, 50, and 0. In retrospect the underlying data for this question was incorrect. The javax.lang.model package does not have any classes but has one enum and one annotation which we don't represent in our System Hotspots View, but it is something we could easily remedy. Likewise javax.rmi actually has three classes but the version of the visualizations we were using were an earlier version which didn't include this information. Nonetheless we believe these two minor aspects did not have an affect on the ability of the participants for this task as they did not need to know about the discrepancies with the underlying data to answer the question.

Comparing. The System Hotspots View is worst for comparing numbers of filtered entities. Question 6 is the task that participants did the worst on where the boxplot shows the mean about 30%. Only two participants, 14 and 6 got more than 50% of the packages. Of the correct packages participants identified the most obvious ones, but when it came to some of the smaller packages they could not compare the number of attributes and methods very well as the images were too small for them.

When comparing classes the participants performed worse for question 10 than 11. The boxplot shows that question 10 has a much larger range of answers than question 11. In question 10 eight of the participants and in question 11 seven said that the Component class was much bigger than it actually was. One participant said that there was no difference in size between the classes, but they must have interpreted the question incorrectly. For these questions some of the participants raised the point that the LOC for each class was hard to determine as these classes were all in the same range so they could not tell how many more LOC one class had than another.

4.2 System Hotspots View

4.2.1 Strengths of the System Hotspots View

Overview. Many participants stated that the visualizations created a large overview of the system which enabled them to get a quick understanding of the size of the system as a whole and the characteristics of packages and classes.

> "Quick understanding of gross characteristics plus ability to focus in." - Participant 2

> "Good overview and overall understanding of packages."- 4

> "Quick to read not cluttered with data." - 7

> "Gives a good summary of the information in an easily accessible way does not require any knowledge of Java to use." - 13

> "Good at showing multiple dimensions, and good organisation of those mediums; number of attributes and methods are more important." - 14

Identifying. Participants thought that the System Hotspots View easily helped them to identify large packages and classes, interfaces, and outliers. There is evidence for this as participant answers to questions 2 and 7-9 were mostly correct. Questions 7-9 show that participants got over 80%. While for question 2 most would have got this answer correct but the images were not to scale.

> "The visualizations are also helpful to spot relative sizes and anomalies (very large or oddly shaped classes)." - 3

> "Outliers are easy to spot (only interfaces, many attributes, many methods)." - 4

> "Interfaces are easy to spot." - 5

> "Easy to spot excessively large classes." - 11

Size. Participants thought that the characteristics of the System Hotspots View made it easy to understand the size of packages and classes. All participants quickly identified the largest packages and classes for questions 2, and 7-9.

> "It's easy to see which packages have big packages or lots of packages."- 1

> "Height and width give a quick feel for the number of members and hence size." - 5

> "They showed comparative sizes, and the types of the major classes in the Java API." - 6

> "Gives you an overall feeling on the size of Java packages." - 11

> "Good for looking at the sizes of class." - 12

Colour. Some participants commented on the use of colour to identify entities in the system. The border colours for classes made it easy for participants to identify the appropriate classes for questions 7-9.

> "The use of colors helped me to quickly spot the different types of classes." - 3

> "Had color." - 7

> "Diagrams and colours are better than words and big diagrams are easier to see." - 8

4.2.2 *Weaknesses of the System Hotspots View*

Comparing. Participants thought that it was hard to compare the relative size of packages and classes. Participants struggled with comparing the size of entities. Questions 10 and 11 are evidence of this as participants gave a wide range of answers.

> "One can tell comparatively which class has more methods or which has more attributes, but they couldn't exactly tell how many more of these methods or attributes they have." - 1

> "The relative sizes between the three main visualizations were not correct. So comparing between these three images was difficult. It was also difficult that the order chosen was the names of the packages by alphabet, this did not help with any of the questions." - 3

Large Packages. Participants thought that the System Hotspots View was good for counting classes in small packages but for large packages it was hard to tell how many classes it contained and what kinds of shapes some classes were. Participants struggled with identifying small classes that were boxes as required for question 6.

> "The visualization seems less useful when there are too many classes in a package, as you can't actually tell the shape of a class." - 11

> "Hard to 'query' the diagrams to count the number of objects of a certain type." - 12

> "You cannot tell how many classes there were in large packages." 14

Data Manipulation. Participants wanted interaction features for manipulating the data in the System Hotspots Views such as querying, sorting, and filtering. These features would have helped participants to answer all the questions more effectively.

> "Make it easier to see the packages with small classes, have some kind of a scale so you can tell the actual number in each entity." - 1

> "Interaction with the metrics (such as being able to sort) would be very nice!" - 3

> "Interaction should provide more detail, e.g. entity names, LOC, and there should be ways to query and sort, e.g. by color or height." - 5

Scale. All the participants struggled with the non-uniform scale of the three visualizations. This affected the answer to question 2 where participants would have almost certainly all the got the correct answer.

> "Make the images equal size." - 3

> "All packages have to be scaled relative to the others, packages in the middle section are smaller scaled than others." - 4

> "Some of the images were too small to make out the shapes and kinds of classes." - 6

Colour. A few participants raised issues about the colour encoding and fonts used for the text. They thought that the green and blue borders for abstract classes and concrete classes were hard to identify when the classes were large and had a black shading. This affected participant answers to question 9 which had the same answer to question 7. However, the red borders for interfaces clearly stood out for question 8.

> "Don't use bold font, don't use a serif font on a screen." - 3

> "Some of the classes are tiny so it is hard to see colours and borders at that resolution." - 5

> "Borders hard to see (green, blue)." - 10

One participant raised the issue that he was red-green colour blind.

> "Relied on red / green one-pixel borders, which were hard for me to look at." - 12

Metric Shading. The shading of the LOC was also confusing for some participants when determining the size of a class as it was not clear what the actual value for this metric was for the class they were inspecting. Having a larger scale of the LOC shading would have helped participants to answer questions 10 and 11 better.

> "Shading is capped (1000 LOC or 100,000 LOC hard to tell) absolute number of things hard to get, relative numbers works better." - 4

"The shading as an indicator for size is hard to evaluate, and it conflicts with my desire for size to be indicated by area." - 5

Representation and Encoding. A couple of participants quite often referred back to the technique description to remember the meaning of the dimensions and colour encoding.

"Had to keep referring to the key to remember what dimension or color represented what aspect. Have the key visible on the screen and be able to hide the smaller classes below a threshold." - 7

"It is easy to confuse the width of a package to be the number of classes in it. The width instead shows the number of attributes in a package." - 14

4.3 Visualization Wall

4.3.1 Strengths of the Visualization Wall

Display Size. All participants enjoyed the large display size and the ability to display large amounts of information. This resonates with the findings by Andrews et al. [1].

" Allows all the data to be seen at once, makes it easier to find the classes that stand out easier." - 1

"It was big!" - 9

Physical Size. Several participants commented that it was useful to physically walk around to use the wall. This resonates with the findings by Ball et al. [2].

"No scrolling, just walking about." 3

"It was big enough to walk around and point things out on an eye level, and could compare two sizes quite easily." - 6

"Resolution excitement when walking around the screen." - 9

"Large display estate and can move around." - 11

"Step back to see a bigger picture, step close to see detail. " - 12

Concentration. Two participants felt the visualization wall helped them to concentrate better when answering these kinds of questions.

"It is 'stimulating' to move about while looking at metrics (or any type of documentation for that matter) in the sense that you are more concentrated on the task because you can move freely - are not confined to a single position such as sitting on your chair." - 3

"I could see more data at once, which lessened requirements for me to remember stuff that wasn't visible." - 5

4.3.2 Weaknesses of the Visualization Wall

Interaction. This was the first time for all participants using a large visualization wall. Of biggest concern with the visualization wall was that there was a lack of interaction with the static images. Figure 7 shows some of the techniques (also including hands and paper) participants used to compare data in the System Hotspots View. Most of the participants wanted to be able to drag images around the screen, re-size and scale images, and zoom in on details.

"I would like manual zoom control to focus in on details." - 2

"You can't interact with the visualization." - 11

"Ability to drag images around to compare, click on them while looking at them." - 12

"Provide an easy way to re-size the areas you are looking at." - 13

The monitors in the visualization wall emitted a large amount of heat. Participants were tempted to touch the monitors to interact with the data in the visualizations. We had to make them aware that they were not touch screens.

Physical Height. The height of the wall posed problems for some participants as they were shorter than others so it made it hard for them to read the information on the top row of screens, likewise for very tall participants who had to bend down to see the information on the bottom row, see Figure 7.

"Some diagrams higher than me." - 8

"The height of the bottom screens not good for standing but height of top screens is good." - 13

Interior Bezels. The bezels around the outside of each of the monitors created problems for some participants trying to understand information that spread from one screen to another. Bi et al. [4] claim that interior bezels do not affect visual search nor error rate; however, splitting objects across bezels is detrimental to search accuracy.

"Hard to see diagrams that spread across screens." - 8

"The gaps between the screens caused issues with viewing the information." - 9

"Bezel makes it hard to see the image on the crossover height of the bottom screens." - 13

5. LIMITATIONS

Results. We conducted our user evaluation with a small number of participants. Future studies should involve a greater number of participants and a significant number of industry professionals. We did not vary the order of the questions which might have introduced a learning bias.

Scale of Visualizations. Due to limitations with our implementation and the SAGE software used to drive the visualization wall all the images of the three top level packages were not at the same scale. This made it difficult for participants when comparing the size of packages and classes

(a) Hands (b) Fingers

(c) Pen (d) Bending

(e) Crouching (f) Kneeling

Figure 7: Participants interacting with the System Hotspots Views on the visualization wall.

and counting packages and classes in packages. The height of the largest image was greater than the height of our visualization wall. Once this image was scaled down the image was too small to properly view on the visualization wall.

Interaction. There were no user controls for participants to be able to manipulate the data in the visualization such as the ordering of classes or filtering out any information. The only interaction with the visualizations was controlled by the observer of the user evaluation who moved the images around the screen and at the participant's request. The interaction was controlled on a machine sitting a couple of metres back from the visualization wall. The actions that were available were repositioning and minimal scaling of the images using a mouse. These two important interaction tasks were quite cumbersome. The limited control screen didn't actually show the details of the images, only the outline. Making positional adjustments to the image had a slight delay effect with re-displaying the image. The images were large but not high resolution and the quality of the image was severely affected when the images were scaled larger or smaller.

Visualization Wall. Some of the participants thought that the height of the visualization wall too tall for them or that they had to bend down or crouch to view the information, see Figure 7. To alleviate this issue we provided a chair for tall people to sit on when they required viewing the bottom row of monitors. We didn't provide a stepping block for the shorter participants but this would have been a useful asset.

We alleviated the interior bezel issue by correcting the images with the SAGE software which made an image flow from one screen to another seamlessly. However, some of the participants still had difficulties determining what information was being shown on which display.

Measurement. Time measurements were taken using a manual stop watch once participants were given the question sheet and after they had the System Hotspots View technique explained to them. Some typed their answers into the web form as they went while the others wrote their answers on the sheet provided and then entered their answers into the web form on completion of the tasks. Therefore the time measurement was not as accurate as it could be.

6. RELATED WORK

To our knowledge there have been no published user evaluations for any of the Polymetric Views. We have not seen any System Hotspots Views that contain more than 1,000 classes published in the literature. Numerous tools support Polymetric Views [9].

Code Crawler is a language independent reverse engineering tool which was the first tool to use Polymetric Views and has since been integrated as a plugin to Eclipse [10]. CodeCity is a tool that stems from Code Crawler and uses a 3D city metaphor based on Polymetric Views to display additional kinds of metric information such as disharmony maps which focus on design flaws [17, 18]. The Mondrian toolkit aims to bring the Polymetric Views closer to the code by extending existing programming languages to use embedded scripts in their programs to create visualizations [12]. Bergel et al. [3] extended the Mondrian toolkit by displaying dynamic information about CPU consumption in Class Blueprints. Softwarenaut uses Polymetric Views but focuses on the dependencies between modules [11]. Lagrein is a tool that supports a number of Polymetric Views and augments them with software requirements and change history information [8].

Large visualization walls are rare and very little work has been done to investigate how they might be used in the context of information or software visualization. Yost et al. [19] conducted an evaluation to explore the effect of using large visualization walls for information visualization.

Their results showed that performance on most tasks was more efficient and sometimes more accurate because of the additional data that could be displayed, despite the physical navigation. Andrews et al. [1] conducted a study which demonstrated how large displays support sensemaking. Ball et al. [2] identified that physical navigation helped improve user performance with large displays. Bi et al. [4] discuss how interior bezels affect user behaviours, and suggest guidelines for effectively using tiled-monitor large displays and designing user interfaces suited to them. Other researchers have investigated large displays for daily desktop computing tasks [5] such as navigation tasks [16] and window and task management [14].

7. CONCLUSIONS

We have conducted a user evaluation to measure the effectiveness of a modified System Hotspots View technique for software visualization using a large visualization wall. In summary, our modified System Hotspots View technique is very good for users to identify, count, and estimate numbers of entities in a software system. The technique is less good for filtering information or finding a subset of information. The technique is worst for comparing numbers of filtered entities.

Future work is to create a comprehensive and interactive software visualization tool. We will implement other Polymetric Views [9] such as System Complexity and Class Blueprint, provide user controls for tasks like filtering, querying, and sorting, and augment the visualizations with documentation such as Javadoc.

The large visualization wall supported displaying large amounts of data at once, however it lacked effective techniques for user interaction. Some issues that arose from the techniques were the lack of user controls for querying, sorting, and filtering.

Ben Shneiderman [15] claims that gigapixel displays will be useful for some tasks, but innovative interface design is likely to have higher payoffs and wider usage. Given this we are also interested in exploring software visualization in the context of multi-touch tables and walls. We will conduct user evaluations with our multi-touch software visualizations to investigate how collaboration and interactivity affects the usability of Polymetric Views.

Acknowledgments

This work is supported by the Software Process and Product Improvement project through the New Zealand Foundation for Research Science and Technology, as well as a Telstra-Clear scholarship. Thanks to Roger Cliffe for technical assistance with the visualization wall and user evaluation.

8. REFERENCES

[1] C. Andrews, A. Endert, and C. North. Space to think: large high-resolution displays for sensemaking. In *Proceedings of CHI*, pages 55–64. ACM, 2010.

[2] R. Ball, C. North, and D. A. Bowman. Move to improve: promoting physical navigation to increase user performance with large displays. In *Proceedings of CHI*, pages 191–200. ACM, 2007.

[3] A. Bergel, R. Robbes, and W. Binder. Visualizing dynamic metrics with profiling blueprints. In *Proceedings of TOOLS Europe*, 2010.

[4] X. Bi, S.-H. Bae, and R. Balakrishnan. Effects of interior bezels of tiled-monitor large displays on visual search, tunnel steering, and target selection. In *Proceedings of CHI*, pages 65–74, 2010.

[5] X. Bi and R. Balakrishnan. Comparing usage of a large high-resolution display to single or dual desktop displays for daily work. In *Proceedings of CHI*, pages 1005–1014. ACM, 2009.

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

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

[8] A. Jermakovics, M. Scotto, A. Sillitti, and G. Succi. Lagrein: Visualizing user requirements and development effort. In *Proceedings of ICPC*, pages 293–296. IEEE, 2007.

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

[10] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice*. Springer Verlag, 2005.

[11] M. Lungu and M. Lanza. Softwarenaut: cutting edge visualization. In *Proceedings of SoftVis*, pages 179–180. ACM, 2006.

[12] M. Meyer, T. Girba, and M. Lungu. Mondrian: An agile information visualization toolkit. In *Proceedings of SoftVis*, pages 135–144. ACM, 2006.

[13] C. Reas and B. Fry. *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press, 2007.

[14] G. Robertson, M. Czerwinski, P. Baudisch, B. Meyers, D. Robbins, G. Smith, and D. Tan. The large-display user experience. *IEEE Computer Graphics Applications*, 25(4):44–51, 2005.

[15] B. Shneiderman. Extreme visualization: squeezing a billion records into a million pixels. In *Proceedings of SIGMOD*, pages 3–12. ACM, 2008.

[16] D. S. Tan, D. Gergle, P. G. Scupelli, and R. Pausch. Physically large displays improve path integration in 3d virtual navigation tasks. In *Proceedings of CHI*, pages 439–446. ACM, 2004.

[17] R. Wettel and M. Lanza. Visualizing software systems as cities. In *Proceedings of VISSOFT*, pages 92–99. IEEE, 2007.

[18] R. Wettel and M. Lanza. Visually localizing design problems with disharmony maps. In *Proceedings of SoftVis*, pages 155–164. ACM, 2008.

[19] B. Yost, Y. Haciahmetoglu, and C. North. Beyond visual acuity: the perceptual scalability of information visualizations for large displays. In *Proceedings of CHI*, pages 101–110. ACM, 2007.