CorpusVis – Visualizing Software Metrics at Scale

Jack Slater*, Craig Anslow*, Jens Dietrich*, Leonel Merino†,
*Victoria University of Wellington, New Zealand
Email: {slaterjack,craig,jens}@ecs.vuw.ac.nz
†University of Stuttgart, Germany
Email: leonel.merino@visus.uni-stuttgart.de

***Abstract*—We do not know fully understand how software violates metrics based principles, particularly in large systems. Systems are restricted by structural and static deficiencies that we can aim to reduce by providing developers with effective visualizations of their code. We developed CorpusVis a widget-based application to explore software metrics of Java software systems from the Qualitas Corpus. Through an evaluation of the visualization techniques we identified what visualizations were effective and which ones did not scale well for large software systems. Our application helps to reduce the structural and static deficiencies in developers code which enables developers to spend less time maintaining legacy systems and learn to develop more effective code for future systems.**

I. Introduction

Java is one of the most commonly used programming languages, yet we still know surprisingly little about how Java programs are structured in practice particularly in large-scale systems [1]. Measuring software allows us to determine the inner quality of a software system. We know that software systems coded with a poor internal quality will be harder to maintain and extend [2]. Understanding what measures identify internal quality will help improve the maintainability and extendibility of future software systems.

Software visualization aims to help understand the structure, behaviour, evolution, and quality of systems [3]. However, we don't know what types of visualizations are most effective for displaying metrics of large systems even though many exist [4]. The goal of visualizing software and what makes an effective visualization is providing a better comprehension of software artefacts than we can gain from seeing the source files alone [5].

This paper presents *CorpusVis* a widget-based dashboard web application with multiple visualizations of different software metrics to allow developers the ability to create composite visualizations from the provided widgets. The visualizations help illustrate how different systems violate metrics-based principles and what trends occur in the results. By displaying software metrics visualizations, developers can see the quality of a system, where the pain points of their system occurs, and can identify the best action points for future development. We have used the Qualitas Corpus as the dataset for our case study of Java software [6]. We conducted a user study of CorpusVis with 31 software developers to determine the effectiveness of each visualization for displaying metric data and the developed prototype.

II. Related Work

Software visualization is the development and investigation of methods and use of computer graphical representations of many software aspects [3]. These include the structure of a program, the behaviour, and the evolution. The structure refers to the static parts and relations of a software system, the behaviour refers to the process execution, and the evolution refers to how software changes over a period of time period. Diehl claims that developers rarely use visualization as a tool for maintaining software [3].

Software metrics measure a property of software, an example could be the number of lines of code, or the number of methods [7]. Software metrics are used to determine the quality of software. Static analysis computes properties of a program which hold true for all executions [8]. Properties which can't be captured before the runtime are referred to as being "dynamic" and exempt from static analysis. McCabe refers to some typical software analysis metrics for static structure as being, lines of code, number of functions or classes, or complexity regarding graph-theoretical measures, such as cyclomatic complexity [9].

Polymetric Views implements and evaluates the use of polymetric visualizations [10]. A polymetric view is built using the attributes of a square, with respect to height, width, colour, and position, to represent an aspect of a system. CodeCrawler is used for the reverse engineering of software and implements Polymetric Views [11].

SourceVis provides a collaborative software visualization tool for large multi-touch tables using the Qualitas Corpus [12]. SourceVis supports multiple visualization types (including some Polymetric Views) in order to give user's the ability to visualize aspects of a system from different views. SourceVis was designed for multiple users by providing a single collaborative system. SourceVis provides three distinct categories of visualization, exploration, structure, and evolution.

Explora is a tool with the goal of scaling visualization to the Corpora level [13]. Explora focuses on the process of visualizing Corpora size data, this provides us with examples of visualization techniques that can effectively be applied to Corpora, such as the polymetric views that they have implemented. Explora found that using polymetric view charts was an effective way to communicate static metrics within corpora.

978-1-7281-4939-4/19/$31.00 ©2019 IEEE
DOI 10.1109/VISSOFT.2019.00020

Figure 1: CorpusVis: the user interface displaying a variety of widgets and visualizations that show the Ant 1.1 system for usage context. Visualizations include (L to R): Dependency Chart, TreeMap, HotSpot View, ScatterPlot, & Toxicity Chart.

III. CorpusVis

CorpusVis is a novel widget-based dashboard web application that supports multiple visualizations of different metrics to allow developers the ability to create composite visualizations from the provided widgets. The design of the widget-based system was achieved through analysing requirements and iterative progression. The visualizations implemented as the system's widgets were all based on existing implementations but with a number of changes to support the composite visualization structure of CorpusVis. Based on the goals of our project we designed the following requirements:

Req1: The application needs to support the composite viewing of more than one visualization at any given time to allow comparison.

Req2: The application must support the ability to change between systems, and system versions.

Req3: The user must be able to change the layout and visualizations.

A. User Interface

This section covers the core decisions made while designing the overall interface that users would interact with as well as the decisions surrounding which visualizations to implement while linking back to the requirements defined. One of the core ideas of the UI design was to support extensibility and allow visualizations to be easily integrated.

One of the core requirements was to provide an application for the composite viewing of visualizations (Req1). Having the ability to view two visualizations both giving information on a given system lets the user explore more than one aspect of the system at any given time and potentially find trends across the different visualization sources. To allow for composite viewing we decided to build a widget-based dashboard interface. Having a widget interface means that we can let the user determine the level of composition they want to use at any given time. For example in Figure 1 there is an interface with a number of widgets being shown compositely, however another option would be to use one widget to take up the entire screen allowing for viewing a particular chart more intensely.

We designed CorpusVis to include a way for developers to select systems and version numbers for a given system (Req2). The design was to have a simple drop down selection for the different systems, and then to let the developer navigate through the versions with forward and backward arrow buttons to show how the system changes over time (see Figure 2).

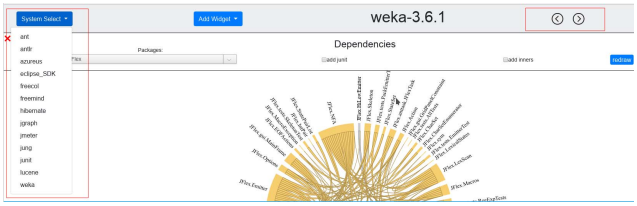

Figure 2: System Selection: shows how a system can be selected in the left hand drop down menu.

To customize the layout of the user interface we adopted a movable widget-based approach (Req3). This was done as it allows for developers to change the size of each widget to the level they require it, and also to arrange the widgets in the best way for comparing. Each widget can be resized by dragging from the bottom right corner, and there is an 'x' place in the top left to allow removal of the widget. Both of these decisions were made based on them being similar to a windowing application. All widget types were able to be moved via drag and drop and had an indicator showing where the grid would snap the widget to.

B. Visualizations

One of the core goals of CorpusVis is to evaluate a variety of existing visualizations when used to analyse large Java systems. This goal requires that visualizations are both selected and implemented that give a holistic view of the current state of software visualization for large systems.

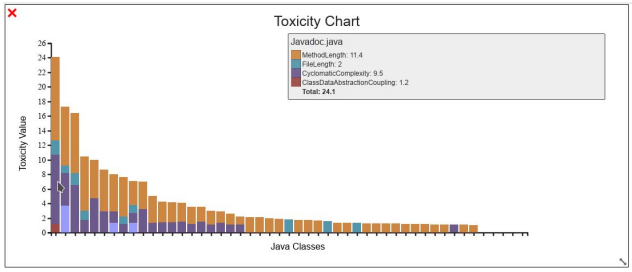

Figure 3: Toxicity Chart: showing the Javadoc system and metrics the classes in the system violates.

1) Toxicity Chart Visualization: The Toxicity Chart was built based upon an implementation which we extended and modified (Figure 3) [2], [14]. The reason for selecting this visualization is because it was effective at displaying lots of metrics at once using basic charts. The technique uses a stacked bar chart which are well understood. To allow for the toxicity chart to function in a widget-based dashboard we added a scrollbar for the X-axis as in the majority of systems there are too many classes to be shown on the widget without shrinking the individual bar width.

2) Dependency Chart Visualization: The Dependency Chart was implemented with a chord diagram showing dependency relations in the selected system (Figure 4). The core reason for selecting this visualization was to explore another area of visualization type, we already had a static analysis metric based visualization, and deemed that it would be useful to also have a visualization which gave a view of the program's class level relationships and system structure. This visualization was designed to give the same features of the Dependency Chart in SourceVis [12].

The Dependency Chart visualization was based on the Dependency Wheel application built for visualizing PHP packages [15]. This visualization in its first iteration did not work due to the large amount of classes in most of the Corpus systems. To account for this we implemented

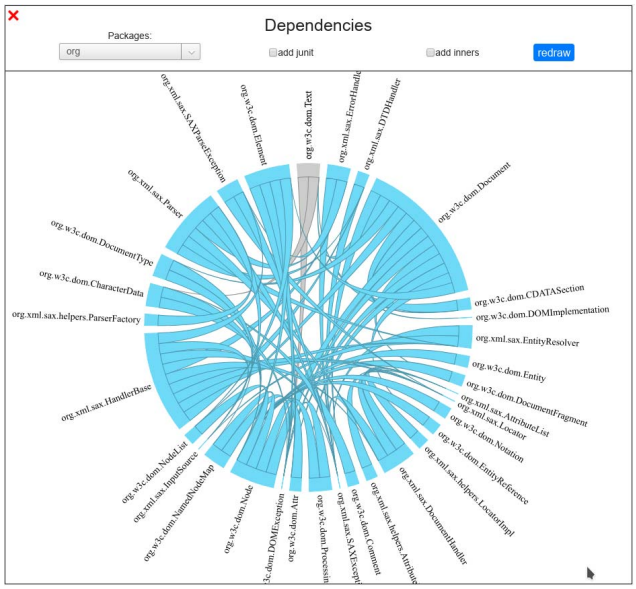

Figure 4: Dependency Chart Visualization: illustrated as a Chord Diagram.

a threshold the number of classes must be under in order to produce a visualization. The thresh hold number was made to be 150 classes, this number was the largest possible number to show while still being able to read each class name. After the threshold we also added in simple filtering so a user can select a particular class in order to get under the thresh hold in larger systems.

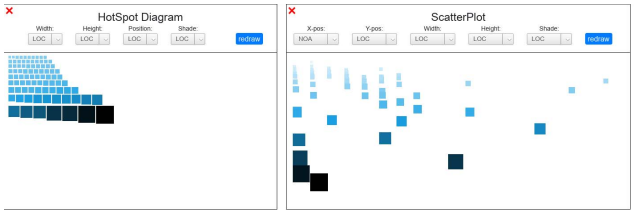

Figure 5: Polymetric Views: HotSpot View and ScatterPlot.

3) Polymetric Views: The final three visualization we implemented were Polymetric Views. These visualizations were chosen as they have already been evaluated and found to work for smaller scale systems [10], [16]. Knowing that these visualizations were effective allows us to compare how effective they are at a larger scale. We chose to implement the HotSpot View, ScatterPlot, and TreeMap diagram as they are three different techniques that potentially will scale differently. The HotSpot View has no overlaps of rectangles suggests that it won't scale as well as we increase the number of classes in a system, similar to the TreeMap, however the ScatterPlot may support more room for scalability as having items overlap is built in by-design. We extended and improved an existing implementation of these Polymetric Views for CorpusVis [17].

C. Implementation

CorpusVis utilized a client-server architecture in order to provide functionality over many devices at once [18]. The architecture for the overall system was designed to represent the structure of the client-server architecture (Figure 6). This diagram describes the high-level architecture for the system, showing how we used technologies in order to fulfill our client-server architecture model.

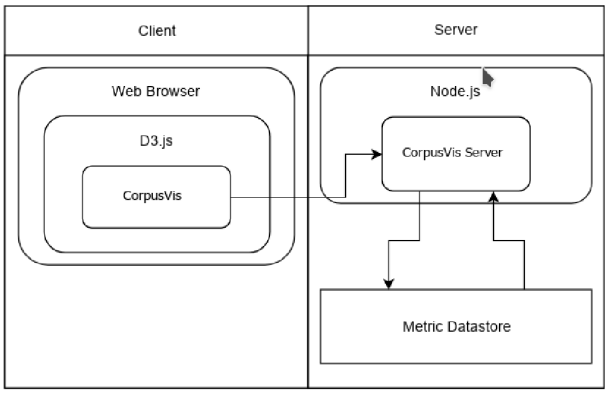

Figure 6: CorpusVis Web-based Architecture, consisting of a client server model and different components.

We used Node.js as the core server technology [19] due to it being written in Javascript, making it easy to extend and only requiring knowledge of one JavaScript framework. The server was used for two core purposes, the first was to serve the CorpusVis public facing application. The other functionality of the server was to host an API for the generated CorpusVis data, including the JSON files storing all of the generated metrics and dependency information. The API hosted by the Node.js server allows for these files to be accessed via REST requests and parsed into data objects at runtime.

For the widget based framework we used JavaScript for inserting and moving objects around the screen and the Gridstack framework [20]. We created features for dynamically resizing, dragging, and deleting widgets. The visualizations were added as their own elements within a widget. For each widget data was loaded from the server only once and stored in the front-end, each widget was then given an id, and would parse the stored data each time it needed updating. We also used JQuery and Bootstrap for user interface components and styling.

For the front-end we used D3 to build the visualizations [21]. D3 was chosen based on two reasons. The first was that there were already some software visualizations implemented in D3 such as the Toxicity Chart visualization which we extended. The second core reason for using D3 was the lack of competitors offering graphics libraries in the web space. One alternative for the front-end visualizations that was considered was JavaFX, however, Oracle has begun deprecating web deployment technologies [22].

D. Data Generation

For the Toxicity Chart date was generated using Checkstyle which is a tool for describing coding standards [23]. By default Checkstyle will use Java style guides, similar to Google [24]. We have used custom guides for CheckStyle to gather information on those classes which violate metrics. For the Toxicity Chart, we used predefined metric values set as inputs to our Checkstyle script (see Table I), which then checks for any classes that break the given metric and generates an XML document including all of the violations. Each metric has a threshold value, which when violated by any given class, the amount gone over by becomes the toxicity value for that metric. For example if a given class file had a file length of 550 lines, the toxicity value for 'File Length' would be 50. In order to generate a toxicity score for each file we developed a script which would run the above Checkstyle process over every version of every system.

Metric	Level	Threshold
File Length	File	500
Class Fan-Out Complexity	Class	30
Class Data Abstraction Coupling	Class	10
Anon Inner Length	Inner Class	35
Method Length	Method	30
Parameter Number	Method	6
Cyclomatic Complexity	Method	10
Nested If Depth	Statement	3
Nested Try Depth	Statement	2
Boolean Expression Complexity	Statement	3
Missing Switch Default	Statement	1

Table I: Software metrics for the Toxicity Chart visualization.

The dependency relationship generation was done similar to the Toxicity scores where we used DependencyFinder [25]. The process generated an XML files with an element listed for every other class in the system with a value stating whether or not the given class has a dependency or not. For the Chord Layout we built dependency matrices, every class name would be stored in an ordered array, then a matrix would be built specifying which classes had dependencies on each other.

The HotSpot View, and ScatterPlot chart used the same data format as in the original [10]. This data format was a JSON object for each class in a system, with the object containing the specified metric values and class identifying information. We used Checkstyle to generate the data, but instead of checking for violations we set the violations for the metrics we wanted to output to 0 in order to ensure data would always appear rather than only when passed the thresholds (like in Toxicity scores).

The data for the TreeMap visualization is very similar to the that of the HotSpot and ScatterPlot. The difference is that the JSON objects are contained within a nested hierarchy based on packages. So each packages will hold an array of JSON class objects, as well as any packages that are nested within. Using these we can use the same metrics as above, but also visualize the system's class hierarchy.

IV. User Study

To evaluate CorpusVis we conducted a user study to assess the effectiveness in providing a dashboard widget based application for visualization, and to evaluate the effectiveness of the individual visualization techniques.

A. Design

One fundamental goal of performing user studies is to seek insight into how effective a particular technique is [26]. The aim of our user study is to seek insight into how effective each of the chosen visualization techniques performs with large systems based on qualitative user experience feedback. User studies can be used to evaluate the strengths and weaknesses of techniques [26].

To enable comparison of software visualization evaluations, we adopted the benchmarks framework introduce by Maletic and Markus [27] and extended by Merino *et al.* [28] to describe the context of the evaluation: *(i)* Medium. The user study was conducted through Google Forms. For the 10 user studies performed in the lab under observation the study was conducted on a Dell Optiplex 9010 Intel i7-3770 CPU @ 3.40GHz computer with a quad core processor and 8 GB of RAM, with Arch Linux and Google Chrome web-browser. The hardware and browser choice was optional for remote participants. *(ii)* Technique and Interaction. The details of the implemented techniques are described in Section III. *(iii)* Tasks. Table II shows the user study tasks.

B. Participants

The study was conducted with 31 participants. The participants consisted of students with at least 3 years computer science experience, and ex-students or academics contacted through person connections. 10 experimental sessions of the study were performed under observation and the remaining 21 were performed remotely. All participants had basic knowledge of Java system structure.

C. Procedure

Participants were directed to a short web tutorial to read through and were encouraged to figure out the basics of the system before commencing the study. The evaluation for CorpusVis used a questionnaire that contains four types of questions that relate to: *(i)* performing small-tasks with each visualization and followed a task-orientated evaluation theory [3], *(ii)* Likert scale questions in order to evaluate the effectiveness of the different visualizations by the user's opinion, *(iii)* gaining feedback on the design of the application by asking open ended questions regarding the positive and negative aspects of each visualization and the application, and *(iv)* The System Usability Score, which we added as a way to evaluate the usability of the system without requiring a laborious process or extending the user study length [29]. The following are the steps that were carried out during each session in the user study:

1) The participant was given a copy of the information sheet and consent form (*i.e.*, hard copy for observed studies, electronic copy for remote).
2) The participant was directed to the CorpusVis introduction page containing links to the CorpusVis tutorial, application, and user study.
3) The participant was encouraged to read through the tutorial to gain an understanding of how to use the system and get a brief overview of each visualization.
4) The participant had each of the different sections of the user study explained, including each visualization type, overall system, and the questionnaires.
5) The participant then completed the questionnaire while using CorpusVis.

D. Data Collection

Table II shows the 3 types of questions asked: 5-step Likert scale questions, small-task questions, and short-answer questions. For each visualization we first collected their response to a small-task question aimed at ensuring they had some experience with using the visualization before evaluating the effectiveness. We then collected a Likert scale rating on the effectiveness on each visualization, then finally offered them an opportunity to share any further thoughts on the visualization. For the overall application Likert scales and short answers were collected to capture the effectiveness of the application, and the participants thoughts on how it could be improved. Last we performed a System Usability Scoring (SUS) questionnaire [29] using a 5-step Likert scale. The System Usability Scale is a Likert scale method, that allows for us to perform a usability assessment at a low cost and with little effort [29]. We have used the SUS to evaluate the usability for CorpusVis as it is a proven method of evaluating the appropriateness of computer systems [29].

V. Results

Now we present and discuss the results from the user study. We present the results by the evaluation of the visualization and then the application.

A. Software Visualization Evaluation

The following sections describes the different questions asked for each of the visualizations and the results attained.

Toxicity Chart.

Q2: Using the toxicity chart visualization, what would you say is the most prevalent negative metric in the `Weka` *system?* This question was a small task to answer "does the system solve the user's problem" and to ensure the participant had a basic understanding of the visualizations functionality. Among the answers 25 were correct, 3 incorrect, and 3 were invalid.

Q3 & Q4: Likert Scales for Toxicity Effectiveness. To evaluate the effectiveness we wanted to look at two aspects, the effectiveness of showing metrics for individual classes, and then the effectiveness of evaluating an

#	Question	Question Type
Q1	Do you agree to the collection and analysis of your given responses?	Consent
Q2	Using the toxicity chart visualization, what would you say is the most prevalent negative metric in the system Weka in its first version?	Small-Task
Q3	Please rate how well the visualization shows negative metrics at a system level	Likert Scale
Q4	Please rate how effective the visualization displays class-level metrics	Likert Scale
Q5	Please list any other thoughts you have regarding this visualization	Short-Answer
Q6	Using the Dependency Chart, for the system Ant-1.1, in the com package, which class has the greatest number of dependencies?	Small-Task
Q7	Please rate how effective this visualization is for viewing system dependencies	Likert Scale
Q8	Please list any other thoughts you have regarding this visualization	Short-Answer
Q9	Using the HotSpot View, for the system freemind-0.0.2, which class has the highest number of public methods (NOPUBM)?	Small-Task
Q10	Please rate how effective this visualization is for viewing class information	Likert Scale
Q11	Using the scatter chart, for the system Weka-3.6.8, which class has the highest number of attributes/fields (NOA)?	Small-Task
Q12	Please rate how effective this visualization is for viewing class information	Likert Scale
Q13	Please list any other thoughts you have regarding these visualizations	Short-Answer
Q14	Which class in the TreeMap for jgraph-5.10.0.0 has the highest lines of code (LOC)?	Small-Task
Q15	Please rate how effective this visualization is for viewing class information	Likert Scale
Q16	Please list any other thoughts you have regarding this visualization	Short-Answer
Q17	As an overall assessment, how effective is the application for understanding and visualising software metrics in a large codebase?	Likert Scale
Q18	Please list any reasons for your answer to the previous question	Short-Answer
Q19	As an overall assessment, how effective is the system for analysing changes to a system over time?	Likert Scale
Q20	Please list any improvements or changes you would like to see in the system?	Short-Answer
Q21	Please list any observations you had regarding how the visualizations handled large scale systems	Short-Answer
Q22	Do you think a tool like this would be better suited for small or large teams, why?	Short-Answer
Q23	I think that I would like to use this system frequently	Likert Scale
Q24	I found the system unnecessarily complex	Likert Scale
Q25	I thought the system was easy to use	Likert Scale
Q26	I think that I would need the support of a technical person to be able to use this system	Likert Scale
Q27	I found the various functions in this system were well integrated	Likert Scale
Q28	I thought there was too much inconsistency in this system	Likert Scale
Q29	I would imagine that most people would learn to use this system very quickly	Likert Scale
Q30	I found the system very cumbersome to use	Likert Scale
Q31	I felt very confident using the system	Likert Scale
Q32	I needed to learn a lot of things before I could get going within this system	Likert Scale

Table II: User study tasks with the questions and question type for answering by participants.

entire system with the chart. In Figure 7 we present the results of the Likert ratings for the toxicity chart. The results for the effectiveness at the class level shows a higher effectiveness on the scale. For the class level response, one participant did not answer the question.

Q5: Qualitative Responses. The final question for the Toxicity Chart was to request any other thoughts on the toxicity chart from the user. 24 participants responded to this question with a range of answers. *(i)* Overview of System. 5 participants stated that the visualization does not currently give a view of the system overall, or it is hard to capture the entire system's "level of toxicity" when the entire data set does not fit on a single widget in most of the systems. A suggested improvement for this issue was to display another bar showing the total value for the metrics in the system overall (aggregated values for each class). *(ii)* Filtering & Comparison. 5 participants made a reference to the inability to compare classes easily or filter the visualization in order to select specific classes. One improvement suggested was to implement an "on-click" function that would save a classes information when clicking on a bar in the chart that would then be shown when hovering over other class bars. Another suggestion was to implement a "topn" filtering to show only a selection of classes. *(iii)* Other Responses. The rest of the responses were compliments on the visualization, small implementation errors, and have a tutorial for this visualization.

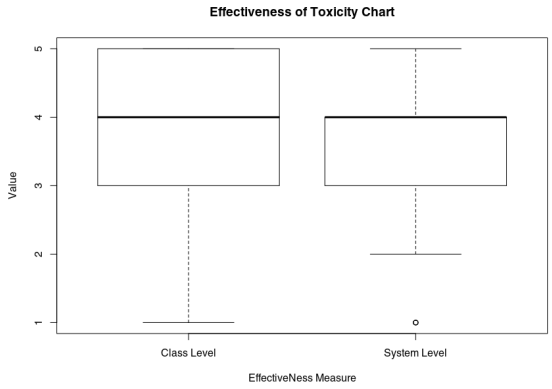

Figure 7: Results from questions 3 & 4, rating the effectiveness of the Toxicity Chart for system level and class level metrics respectively.

Dependency Chart.

Q6: Using the Dependency Chart, for the system `Ant-1.1`*, in the com package, which class has the greatest number of dependencies?* This small task question aimed at evaluating how easy the visualization was to use for the basic tasks it is intended. This question required the user to determine which class in a system had the most dependencies in a system. Among the answers 28 participants were correct, and 3 gave a wrong class name or no class name as an answer when

XmlDocument was the class with the greatest number of dependencies.

Q7: Please rate how effective this visualization is for viewing system dependencies. This question uses a Likert scale to evaluate the effectiveness of the dependency visualization for viewing system dependencies. Figure 8 shows the distribution of responses for the Likert scale rating the effectiveness of the Dependency Chart. Half of the scores are between 3.5 and 4.5, which seems very high considering the high level of negative responses to the open-ended questions that we present next.

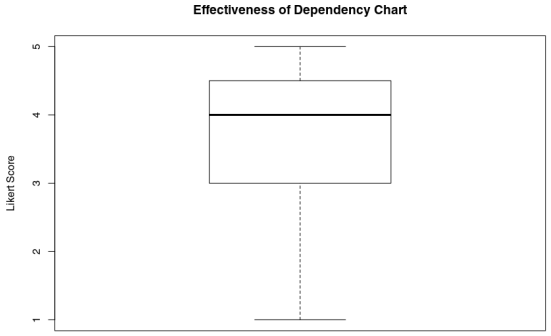

Figure 8: Results from question 7 which used a Likert scale in order to rate the effectiveness of the Dependency Chart visualization.

Q8: Qualitative Responses. 23 participants responded to this question.
(i) Scalability. 10 of the participants cited scalability as an issue for this chart. One participant says "Too many things within the screen at once", with others referring to the overlapping of text when looking at the larger systems in the Corpus. *(ii)* Other Responses. The remaining responses had a large spread, some participants requested that the filtering be more selective having the ability to step further down than the top level package. A number of participants had questions around how colors were chosen. 3 participants also commented on the hover-ability of the chart stating that the chart was a lot simpler when they noticed it.

HotSpot View & ScatterPlot.

Q9 & Q11: Small-Task Questions. These questions were a small task questions aimed at assuring a participant could use the visualizations to a basic degree. In the HotSpot View participants were required to perform a basic filtering by the Number of Public Methods (NOPUBM). All 31 participants answered correctly. In the ScatterPlot visualization participants were required to perform a filtering by the Number of Attributes/-Fields (NOA). We later found there were two possible correct answers therefore in the chart both answers have been considered as correct. Among the answers 25 participants were correct, and 6 gave an incorrect answer to the task.

Q10 & 12: Effectiveness Questions. The results of these questions are shown in Figure 9. We observe that the HotSpot View scored much higher than the ScatterPlot. HotSpot scores concentrate between 3 and 5, compared to the more spread and lower middle 50% between 2 and 4.5 for the ScatterPlot. It is interesting to see the significantly large difference in the effectiveness considering the perceived similarity in the design of these visualization types.

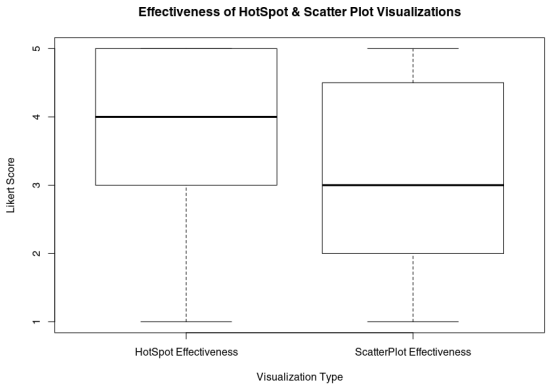

Figure 9: Results from question 10 & 12 which used a Likert scales in order to rate the effectiveness of the HotSpot View & ScatterPlot visualization.

Q13: Qualitative Responses. For the HotSpot View and ScatterPlot the responses to the question regarding any further thoughts were much more concise. *(i)* Outliers. There were 3 responses that complimented these visualization types, one stated that they were good due to being able to "target attributes and make them the target classes with negative metrics stand out". *(ii)* Other responses. These included, complaints around the labelling of parameters (NOA, NOPUBM) and claims of the number of available parameters being redundant due to only needing to use 1 to answer the given questions.

TreeMap.

Q14: Which class in the TreeMap for `jgraph-5.10.0.0` *has the highest Lines Of Code (LOC)?* This question was used as a small task question in order to ensure the participant has a basic understanding of how to use the TreeMap visualization. All 31 participants gave the correct answer: `EdgeRenderer.java`.

Q15: TreeMap Visualization Effectiveness rating. This question was evaluated using a Likert scale to determine the effectiveness. Figure 10 shows the TreeMap effectiveness results where there was a large spread across the data range, however the median value was 4, and the middle 50% of the data was between 3 and 5.

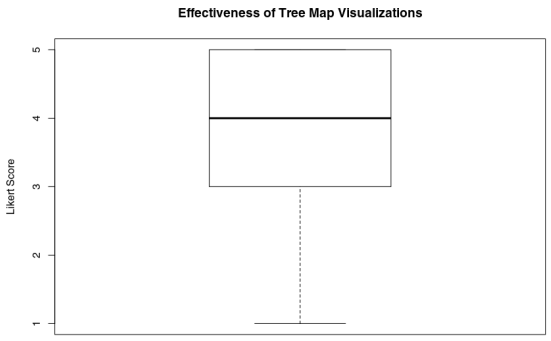

Figure 10: Results from question 15 which used Likert scales in order to rate effectiveness of the TreeMap visualization.

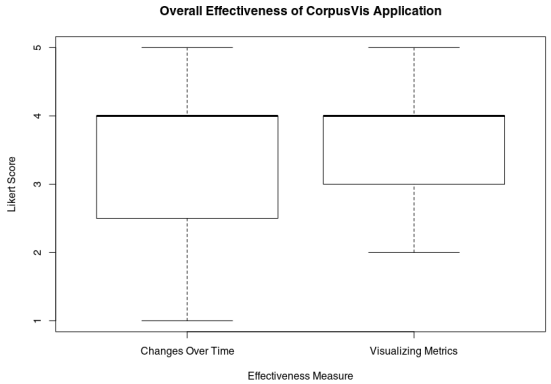

Figure 11: Results from questions 17 & 19 evaluating the CorpusVis application's overall effectiveness.

Q16: Qualitative Results for TreeMap. We asked participants to elaborate.
(i) Scaling. The most commonly mentioned response (6 responses) for the TreeMap was when using larger systems it becomes impossible to compare the rectangles. For scaling in terms of the package boundaries one user stated, "The grey lines are useful for showing the boundaries of packages, but they can be distracting at times, especially when there are a large number of packages. In this case they can take up a large portion of the graph." *(ii)* Parameter Values. Another trend in responses was regarding the available parameters. 3 participants mentioned that either they only found use from the shade parameter as is was easy to notice the difference it produced, and the ordering pattern was not intuitive.

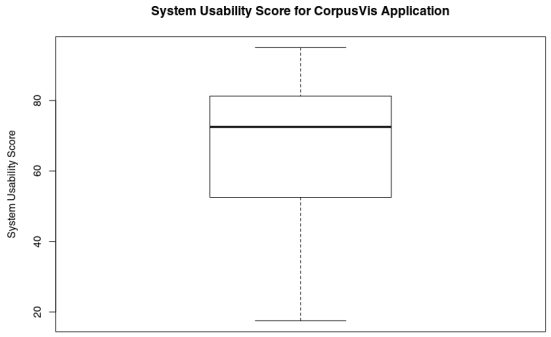

Figure 12: Results from questions 23–32, fitted to the System Usability Score (SUS) model in order to produce an overall score of 72.5 for the CorpusVis [29].

B. CorpusVis Application Evaluation

Effectiveness. Participants were asked to rate the overall effectiveness of CorpusVis when viewing metric based visualizations for large code bases as well as the effectiveness of showing changes in a system over time. In Figure 11 we present the resulting scores of both visualizing changes in a system over time and visualizing metrics in a system (median of 4.0). The Likert ratings for visualizing metrics, however, is much more concentrated towards the upper end of the scale. All responses were between 2 and 5, and the middle 50% of the data was situated between 3 and 4 on the scale. In contrast, for changes over the time the middle 50% of the data was between 2.5 and 4, and the minimum value was 1. Figure 12 shows the median value for the System Usability Scores, which was 72.5. According to a previous study [29], this score can be interpreted as *Good*. What is interesting in the System Usability Score chart is the minimum value, CorpusVis has a 'Good' median SUS value, however, the minimum score given is 7 which would suggest an *Awful* rating from that participant.

Qualitative Results. Following the effectiveness ratings a few open-ended questions (Q20-Q22) were asked regarding the participants opinions on the system and it's potential uses. These questions were all intended to find out what the participants thought the core areas for future work are for this application. The final question regarding team size was also included to test the hypothesis that large teams have larger scale codebases meaning this tool is better in smaller teams.

Q20: Improvements. The first area of improvement recommended for CorpusVis was with regard to changes over time and the switching between data sets. Participants asked for the ability to have individual widget datasets, so instead of the system being set for all of the widgets, being able to individually chose the system and version level for each widget. They also requested that it be clearer where the differences are when changing between versions. In fact, 3 participants suggested that a good way to do this would be through the usage of

animations to get a more fluid image of which metrics values have changed.

Q21: Handling of Large Systems. In this section the core observations were as follows:

- The application suffers from a noticeable lag when there are a number of visualizations on the screen and when changing between versions.
- The tutorial does not give a clear enough guide on using the system and UX design is the major restricting factor.
- Being able to scroll is good in most of the widgets, but it would be nicer to see the whole widget.

While this question did not intend for user's to give opinions on individual visualizations, a number of times they were cited as examples. Only one visualization received more negative comments than positive, 3 participants cited the Dependency Chart as being hard to use due to losing the ability to discern classes.

Q22: Small Teams vs. Large Teams. This question asked participants whether they thought the tool was better suited for small or large teams. There was a mixed response to this question. 11 participants stated large teams would be best with the main reason being that visualizing smaller data sets is not as useful and this tool helps with breaking down large systems due to being metric based. 6 participants stated that the tool would be useful for any size team without any reasons cited. Finally, 8 participants thought the tool would be best suited to small teams due to the limitations in the individual visualizations currently implemented.

C. Comparison Between Visualizations

In the previous section we looked individually at the effectiveness of the visualizations implemented in CorpusVis. While we can individually evaluate them, the subjective nature of Likert questions means it will be more effective to compare the visualization scores against each other. Figure 13 shows a comparison of the effectiveness of all visualizations. It is interesting that of all the visualizations the one with the lowest median Likert rating is the ScatterPlot visualization with 3. All of the other visualizations have a median score of 4, however with varying levels of spread in the data range.

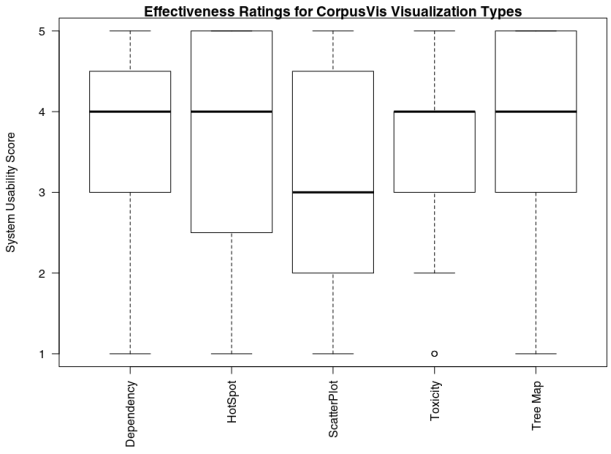

Figure 13: Combined Likert scores for all visualization types by all participants.

VI. Discussion

We now discuss the results of the user study and elaborate on what we can learn from CorpusVis.

A. Software Visualizations Evaluation

The first research question for this project was "How effective are currently used software metrics visualizations when applied to a large code bases?" This section discusses what we have found and how that question has been answered by our study.

Visualization Effectiveness Scores. In Figure 13 we have shown all of the effectiveness scores rated by 31 users for all of the visualizations. What we can see in these scores is that by median value, the ScatterPlot is considerably worse than the other visualizations. This data is interesting because in the questions evaluating the visualizations qualitatively, it was the Dependency Chart that received the highest levels of negative feedback with 10 people sighting scalability as a a major issue for this visualization type.

From the effectiveness scores we have learned that all of the visualization types were capable of performing their given tasks effectively. This is one potential limitation of the study as all of the systems were selected randomly for the tasks and based on ensuring each tasks had the data required to perform the visualization task. So if we had chosen the largest systems for the user study maybe we would have got different results. To more effectively test this aspect it would have been better to get a greater number of tasks performed for each visualization to ensure the scores are representative of the whole Corpus data set.

Based upon the perceived effectiveness scores we have received, the Dependency Chart, HotSpot View, and Toxicity Chart can all be considered equally effective, with the ScatterPlot being the worst. However, without more data it is hard to determine of the other four visualizations which is the most effective technique.

Qualitative Responses. For the qualitative responses, all of the chart types received comments regarding their scalability. Because the tasks involved in the questionnaire did not always use the biggest systems from the Corpus, so the information given may have been different if we had. For all of the charts, there was a one-to-one relationship of chart object to class of the system. What we've learned through the qualitative responses is that for every chart, there is a limit to how far the visualization can be extended while still remaining effective at a granular level. One particpant stated that the application overall is "decent for seeing an holistic view of changes, though it still struggles to see

specific bits", this supports the idea that the visualizations are being extended to a point where we can see an overview of the system but it has become too hard specifically to examine smaller parts of the system.

Taking into account the qualitative responses, it becomes clear that the ScatterPlot, Dependency Chart, and TreeMap were perceived by users to be less scalable. The HotSpot and Toxicity Chart interestingly were the two charts that had scroll bars implemented, when implementing the application, having scroll bars on the individual widgets was deemed as a bad UX decision and it would be better if the visualizations fitted to the widget size, however participant responses suggest that scroll bars were an effective solution and make this a potentially usable for future work.

B. CorpusVis Application Evaluation

The second research question we aimed to answer was "How can we use a web-based application displaying composite visualizations to effectively convey static errors in large Java systems?" This section discusses what we have found and challenge by implementing CorpusVis.

Widget-based Application. One of the core parts of the implementation of CorpusVis was using a widget-based dashboard as the main user interface component. What we found is that using a widget-based dashboard made it easy for users to find the information that they required as they could dynamically change the dashboard at any point to give one widget more or less space. One participant stated that "The widgets really helped to change the visualization to look for the metrics I was interested in. When things looked convoluted, the widgets helped to drill down and find the answer." The qualitative feedback received along with the 'Good' system usability score of 72.5 suggests that we have succeeded in providing a usable and scalable software visualization application.

One learning from the user study was that the quality of the tutorial given and the overall UX design did hinder our application. One participant stated that, "It was not obvious that the mouse could be used to stretch the charts. discovered this feature by accident." This suggests the user interface perhaps could benefit from an on-screen tutorial, tool tips, or a more clear design scheme to ensure participants can pick up the functions intuitively.

Developing the application specifically for the web brought it's own challenges, and it became apparent that once more than 5 widgets were added to the dashboard changing between systems began to show noticeable lag and participants cited this in their responses.

Future Improvements. CorpusVis was found to be least effective when it came to the navigation of system versions and displaying differences in the different versions. A valuable areas of future work would be to extend the application with a better navigation experience for users, this could include changing how navigation is implemented, adding animations when changing systems, and possibly implementing the ability to view multiple versions at once.

Similarly to the multiple versions, another limitation of the application is the inability to view multiple systems at once. It would be a possible area of future work to implement the ability for each dashboard to have its own dataset behind it so then they could all be showing different systems.

We have found that the visualizations struggled to scale due to the one-to-one nature that is commonly seen for classes in software visualizations. One way to try and combat this is to build filtering into the visualizations as we have done for packages in the Dependency Chart. A potential area for future work is to add in filtering frameworks. This work could also involve more research into find, or developing more visualizations which do not have one-to-one mappings so that scalability can be improved.

During the development of CorpusVis, the most challenging part was extracting the static analysis data from the Corpus source files. This was due to the limitations of currently available software, and the sheer volume of data causing static analysis tools to take a long time to run. In CorpusVis the data files are stored as flat JSON files, a potentially useful area for future work would be to build a database of static analysis metrics data for the Qualitas Corpus in order to remove this required step from the process for future developers.

VII. Conclusions

In this paper we presented CorpusVis which visualizes software metrics via an extensible web-based viewing approach. CorpusVis is designed as a widget-based dashboard application whereby users can select, manipulate and interact with added visualizations. To illustrate CorpusVis we used the Qualitas Corpus data which includes a number of Java systems. We conduced a user study of CorpusVis with 31 participants to understand what visualization techniques were effective. We found that the visualizations themselves were all effective for the smaller systems in the Corpus, however in the larger systems they became ineffective due to the sheer volume of data. We found that of the visualizations, the HotSpot View and Toxicity Chart were more effective than the Dependency Chart, ScatterPlot and TreeMap. The differences between these visualizations and the others were that they both kept a static size of the visualization no matter the size of the widget, and used scrolling to view all of the visualization. By developing CorpusVis we have given insight into developing visualizations of software metrics at scale.

Acknowledgments

We thank the participants for participating in the user study. Merino acknowledges funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Projektnummer 251654672 – TRR 161.

REFERENCES

[1] G. Baxter, M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero, "Understanding the shape of java software," *SIGPLAN Not.*, vol. 41, no. 10, pp. 397–412, Oct. 2006.

[2] E. Doernenburg, "How toxic is your code?" https://erik.doernenburg.com/2008/11/how-toxic-is-your-code/, accessed: 2018-10-11.

[3] S. Diehl, *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer, 2007.

[4] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2010.

[5] T. Ball and S. G. Eick, "Software visualization in the large," *Computer*, vol. 29, no. 4, pp. 33–43, Apr. 1996.

[6] 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 Asia Pacific Software Engineering Conference (APSEC)*, 2010, pp. 336–345.

[7] J. Domingue, "Software visualization and education - introduction," in *Revised Lectures on Software Visualization, International Seminar*. Springer, 2002, pp. 205–212.

[8] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Berlin, Heidelberg: Springer-Verlag, 1999.

[9] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. 2, no. 4, pp. 308–320, Jul. 1976.

[10] M. Lanza and S. Ducasse, "Polymetric views – a lightweight visual approach to reverse engineering," *IEEE Trans. Softw. Eng.*, vol. 29, no. 9, pp. 782–795, Sep. 2003.

[11] M. Lanza, "Codecrawler – lessons learned in building a software visualization tool," in *Proc. of European Conference on Software Maintenance and Reengineering (CSMR)*, March 2003, pp. 409–418.

[12] C. Anslow, S. Marshall, J. Noble, and R. Biddle, "Sourcevis: Collaborative software visualization for co-located environments," in *Proc. of Working Conference on Software Visualization (VISSOFT)*. IEEE, 2013, pp. 1–10.

[28] L. Merino, J. Fuchs, M. Blumenschein, C. Anslow, M. Ghafari, O. Nierstrasz, M. Behrisch, and D. Keim, "On the impact of the medium in the effectiveness of 3D software visualization," in *Proc. of VISSOFT*. IEEE, 2017.

[13] L. Merino, M. Lungu, and O. Nierstrasz, "Explora: A visualisation tool for metric analysis of software corpora," in *Proc. of Working Conference on Software Visualization (VISSOFT)*, 2015, pp. 195–199.

[14] "Toxicity reloaded," https://github.com/softvis/toxicity-reloaded, accessed: 2018-10-11.

[15] "Dependencywheel an interactive visualization of package dependencies," http://www.redotheweb.com/DependencyWheel/, accessed: 2018-10-11.

[16] C. Anslow, J. Noble, S. Marshall, E. Tempero, and R. Biddle, "User evaluation of polymetric views using a large visualization wall." in *Proc. of Symposium on Software Visualization (SoftVis)*. ACM, 2010.

[17] "Polymetric views - online version," https://github.com/softvis/polymetric-views, accessed: 2018-10-11.

[18] A. Berson, *Client-server architecture*. McGraw-Hill, 1992, no. IEEE-802.

[19] N. Foundation, "Node.js," https://nodejs.org/en/, accessed: 2018-10-11.

[20] "Gridstack.js," http://gridstackjs.com/, accessed: 2018-10-11.

[21] M. Bostock, "D3.js - data-driven documents," https://d3js.org/, accessed: 2018-10-11.

[22] "Javafx," https://www.oracle.com/technetwork/java/javase/9-deprecated-features-3745636.html, accessed: 2018-10-12.

[23] "Checkstyle," http://checkstyle.sourceforge.net/, accessed: 2018-10-11.

[24] "Google java style guide," http://google.github.io/styleguide/javaguide.html, accessed: 2018-10-11.

[25] D. Finder, "Dependency finder," http://depfind.sourceforge.net/, accessed: 2018-10-11.

[26] R. Kosara, C. Healey, V. Interrante, D. Laidlaw, and C. Ware, "User studies: Why, how, and when?" *IEEE Computer Graphics and Applications*, vol. 23, no. 4, pp. 20–25, 7 2003.

[27] J. I. Maletic and A. Marcus, "CFB: A call for benchmarks-for software visualization," in *Proc. International Workshops on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE, 2003, pp. 113–116.

[29] J. Brooke, "Sus: A quick and dirty usability scale," *Usability Eval. Ind.*, vol. 189, 11 1995.