Web Software Visualization Via Google's Visualization API

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

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

ABSTRACT

There exist very few toolkits and libraries that make it easy for developers to create visualizations of empirical software metrics data. For this reason the use of software visualization tools is not wide-spread within the software development industry. We are exploring creating visualizations of software metrics using the Google Visualization API for the purposes of visual software analytics. We present examples illustrating what the API is capable of and give some indication as to how it might be used for software visualization in the hope it will help inform developers.

Categories and Subject Descriptors

I.6.9 [**Visualization**]: [Visualization techniques and methodologies]

General Terms

Software Design

Keywords

Visual Analytics, Software Visualization, Metrics, Java

1. INTRODUCTION

Since the inception of Java, a large amount of software has been written in the language and surprisingly little is known about the structure of Java programs in the wild [4]. We are using a tool called mete-tools to generate software metrics data, which analyses Java byte code in different ways over a corpus of Java software[1]. The tool generates a lot of data and we need better methods for comprehending this data.

Visual analytics could be used which is the science of analytical reasoning facilitated by interactive visual interfaces [9]. We are interested in visual software analytics which uses software and information visualization techniques to confirm the expected and expose the unexpected of software.

[1]`http://www.cs.auckland.ac.nz/~ewan/corpus`

This paper explores creating visualizations of software metric data using the Google Visualization API [7] to support visual software analytics. The API is a new JavaScript library released in early 2008 which allows users to create visualizations and reporting applications over structured data and integrate them into a web page, Google Spreadsheet, or Google Gadget. We now evaluate the API for use in software visualization and present some of our software visualizations.

2. EVALUATION OF THE API FOR USE IN SOFTWARE VISUALIZATION

We want to see how effective the Google Visualization API is for use in visual software analytics. We evaluate the API against our experience of creating software visualizations and a framework for evaluating graphics technologies for use in software visualization [1]. In detail we want to experiment with creating visualizations over the web, evaluate the animation and interactivity aspects, examine the text, layout, and extensibility features, test the integration capabilities, and analyze the performance display capabilities of the visualizations created from the API.

What is the design of the API? The aim is to support visualizations over the web with any compliant server-side data source. The API is implemented as a JavaScript library. Developers can use the API so long as they agree to the terms of service, but no modifications can be made to the core of the library, nor can it used offline. Error handling in the API is limited but third-party browser plugins such as Firebug or Microsoft Script Debugger can be used for debugging. Developers proficient in JavaScript can learn the API within a couple of days to become very competent. We are not aware of the API being used in production for software visualization nor of any empirical evaluations.

How are visualizations created and viewed? The visualizations can be created in any text editor. Figure 1 shows the basic code to create a visualization. Lines 1–2 references where the JavaScript API is located. Lines 4–5 load the visualization API, the correct version, and the packages to be used in the visualization. Once the API is loaded the initialize function is called, Line 6. Lines 8–12 get the data source located inside a sheet of a Google spreadsheet, create a query, and then execute that query. The results of the query are handled by the handleQueryResponse function, lines 14–23. This function first checks whether there are any errors in the query response. If there is no errors the data is assigned to a data table. The visualization is next created for the appropriate package name and the data is then drawn in the visualization with the display options.

```
<script type="text/javascript"
  src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
 google.load('visualization', '1',
  {packages:['packagename']});
 google.setOnLoadCallback(initialize);

function initialize() {
 var query = new google.visualization.Query(url);
 query.setQuery('query string>');
 query.send(handleQueryResponse);
}

function handleQueryResponse(response){
 if (response.isError()){
  alert('Error in query');
  return;
 }
 var data = response.getDataTable();
 var vis =
  new google.visualization.PackageName(container);
 vis.draw(data, options);
}
```

Figure 1: Basic code to create a visualization.

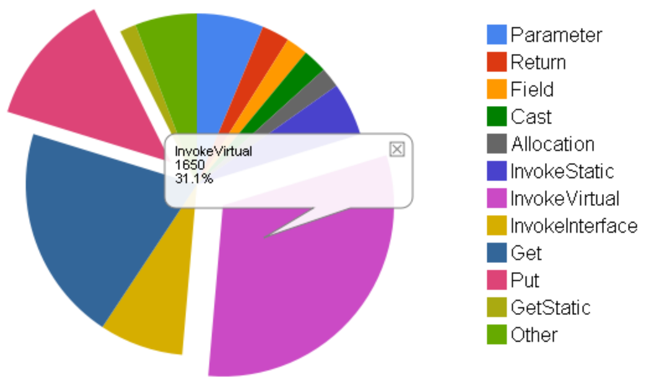
The visualizations are viewed in a web browser and we displayed them in the following browsers: Microsoft Internet Explorer and Mozilla Firefox on Windows, Firefox on NetBSD, and Safari on MacOSX.

What data can be used as input? Google Spreadsheets, HTML pages with data tables, JSON or CSV file formats, or web accessible databases can be used as data source inputs. The API can freely interoperate with Google Spreadsheets and there exists a Python helper library to work with JSON encoded data that was created by an external developer. All other alternative data sources require custom code. Alternatively data can be encoded in a data table within a JavaScript function. A SQL-like query language exists which allows developers to perform various data manipulations with a query to the data source and is independent to the implementation of any specific data source.

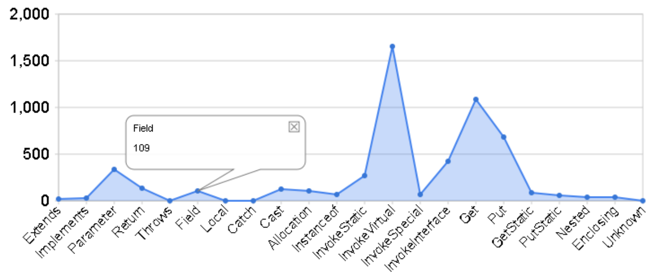
What is the graphical capability of the API? Visualizations displayed in Firefox are rendered in Scalable Vector Graphics (SVG) and Vector Markup Language (VML) for Microsoft Internet Explorer. Visualizations that support animation use Flash. 2D is supported, while some of the charts can display 2 1/2 dimensions. Distortion oriented techniques are possible including focus+context.

What visualization techniques exist? 17 visualization types exist including: tables, spreadsheet like charts, hierarchical charts, maps, and more sophisticated temporal displays. Most of the focus of these visualizations are on spreadsheet charts since the underlying data source is structured data. No specific software visualization techniques [5] or graphs/node-link diagrams are supported. Techniques such as tree maps could be implemented in the future.

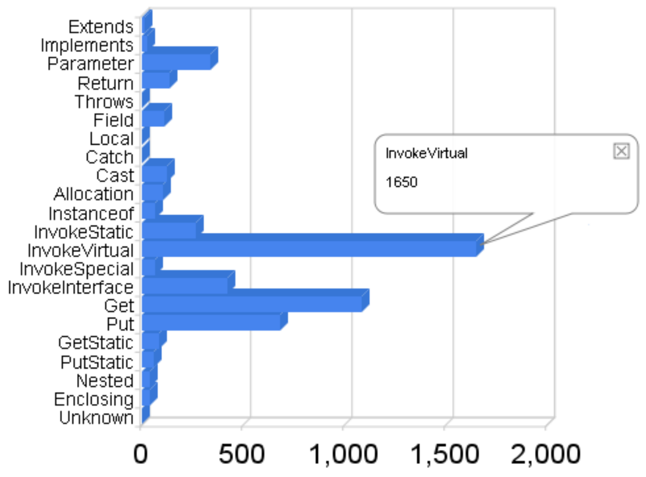
We illustrate some of the possible uses of the API with a data-set showing the frequency of dependencies (see Table 1) of JGraph version 5.10.2.0 (graph drawing application). Figure 2 shows a pie, area, and bar charts all using data from the same Google Spreadsheet. A dependency exists from one type (classes and interfaces) to another if the other is referred to in the byte-code of the first. The kind of dependency, such as InvokeVirtual, Get, etc, indicates what kind of instruction the dependency is part of, or how it is being



(a) Pie Chart



(b) Area Chart



(c) Bar Chart

Figure 2: Basic chart visualization types.

referred to (e.g., Parameter, Return, Field, etc).

Figure 3 shows some of the more specialized visualization types. The motion chart implemented in Flash allows users to explore several indicators or trends over time. The chart shows the version release history of JGraph against Jung another graph drawing application. The chart allows a user to press play to animate the data points evolving over time. It shows that JGraph was created before Jung and by the end of 2008 JGraph had reached version five while Jung was still in version two. It is also evident that JGraph has more development iteration cycles than Jung, but this doesn't measure the quality of code between the applications.

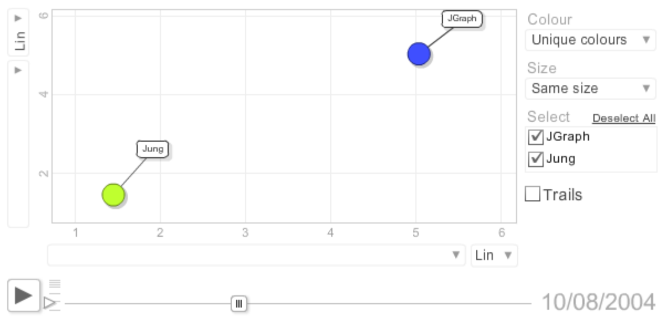
The annotated timeline chart uses the same data set as

Type	Definition
Extends	The to module appears in the extends clause
Implements	” ” ” ” in the implements clause
Parameter	” ” ” ” as the type of a parameter
Return	” ” ” ” as a return type
Throws	” ” ” ” in a throws clause
Field	” ” ” ” as the type of a field
Local	” ” ” ” as the type of a local variable
Catch	” ” ” ” as the type of an exception in a catch clause
Cast	” ” ” ” in a cast expression
Allocation	” ” ” ” in a allocation (”new”) expression
Instanceof	” ” ” ” in an instanceof expression
InvokeStatic	A static method is invoked on the to module
InvokeVirtual	An instance method is invoked on the to module (which is a class)
InvokeSpecial	A constructor or private method is invoked on the to module
InvokeInterface	An instance method is invoked on the to module (which is an interface)
Get	An instance field of to module is read from
Put	An instance field of the to module is written to
GetStatic	A static field of the to module is read from
PutStatic	A static field of to module is written to
Nested	The to module is nested within the from module
Enclosing	The to module encloses the from module
Unknown	Somehow a dependency to the to module was detected but the type could not be determined

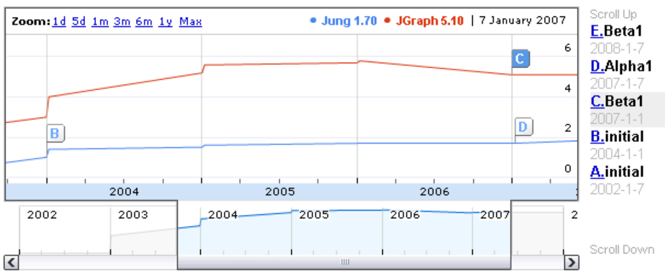
Table 1: Dependency Type

the motion chart. The chart allows data points to be annotated such as the date and version of a Beta release or when different versions of a language were used such as Java 1.4 and 1.6. The bottom part of the chart has features to zoom-in and display segments of time and then slide the segment over the whole time series.

The API can be extended by developers to create their own visualizations other than the predefined ones, see Figure 4. Currently there are six visualization types created

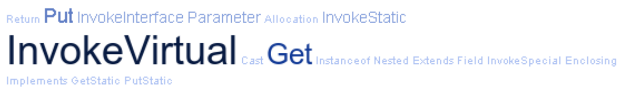


(a) Motion Chart

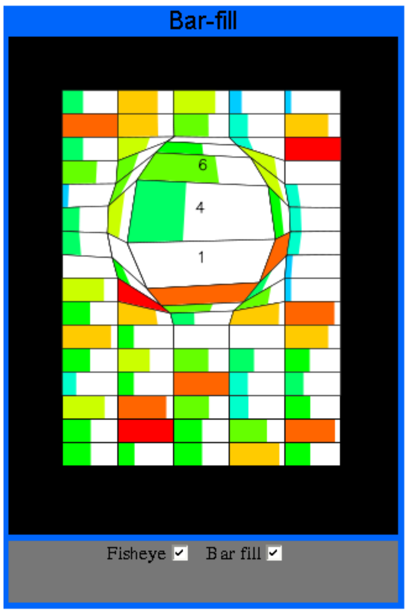


(b) Annotated Time Line Chart

Figure 3: Google specialized visualization types.



(a) Tag Cloud



(b) Magic Table

Figure 4: External developer created visualization types.

by non-Google developers which range from basic counting to more domain specific areas such as heat maps. The tag cloud displays in alphabetical order text at different font sizes depending on the frequency of the words in the data set. Quite clearly the most frequently occurring dependency types of JGraph are InvokeVirtual, Get, and then Put. The magic table displays the data source that is used as input represented as a table. If a cell has a numerical value then filled bars can be used to represent the values using a colour ordering. A graphical fish eye view can be enabled which displays zoomed-in values of cells when the mouse hovers over a selection of cells and the neighbouring cells are distorted.

How well does the API perform? There is no perceptual differences with rendering the visualizations on the various browsers. However, the motion chart had some latency lagging issues when the trail of items option was selected. Google Spreadsheets can be used as input to the visualizations which has a limit of 200K cells and importing of spreadsheets from other formats to Google Spreadsheets have a limit of approximately 1MB. We have not used any large data sets stored in external data sources yet. It takes approximately five seconds[2] for one visualization to render. We embedded 12 visualizations on one web page and it took approximately 20 seconds for the visualizations to display.

[2]Performance timings were done on a Dell Optiplex GX745 workstation running NetBSD with 2.8GHz Core Duo processor, 2GB RAM, and ATI x1300 Graphics, and a MacBook Pro running MacOSX with 2.4GHz Core Duo processor, 4GB RAM, and NVIDIA GeForce 8600M GT

How can the visualizations be presented? The data is independent of the display. Data in the visualizations can still be referenced. Basic layout features are supported but no advanced techniques such as force directed. The motion chart is the only visualization that supports animation. Displaying multiple views of different parts of the data set is possible. No video or sound capabilities are supported, but the YouTube API could be integrated.

How can users interact or customize a visualization? Users can select a point or value in the visualization which creates a pop up box listing the label type and the value. For example the pie chart allows users to select pieces of the pie (either on the pie or from the legend) which pop out from the main core of the pie. Users can't rotate or resize visualizations since these properties are all defined when first created. Users can't manipulate objects within the visualization to move them to different parts of the screen. Users can only interact with the visualization using a mouse. The API provides listeners that can respond to events that are triggered in the visualization such as when a user clicks an item in a visualization an alert window will appear.

What user navigation options are supported? Navigation within a visualization is not supported. If a visualization is larger than the current screen size then web browser scroll bar navigation is required. Viewpoints of specific locations in a visualization is not supported either.

What user tasks [8] are supported? All the visualizations provide an overview and show details on demand for the user, and some support being able to zoom into items of interest such as Figure 3. Only one externally developed visualization allows filtering out uninteresting items in a visualization. Showing relationships among items in a visualization is not supported, a history of user actions can't be saved, and it is not possible to extract sub-collections of information from a visualization.

3. RELATED WORK

Duignan et al. [6] evaluated SVG and found that it can meet the requirements of software visualization only adequately. They found that to do anything more than the basic tasks required scripting, and that SVG lacks an in-built layout constraints system and the ability for data display independence.

Anslow et al. [2] evaluated X3D and concluded that the major advantages of X3D are rich graphics, extensibility, and XML integration. The major disadvantages of X3D are lack of software visualization user controls, a primitive animation model, and weak support for filtering and layout. Nonetheless they encourage software visualization developers to adopt X3D if they need 3D for the web.

Anslow et al. [3] explored creating visualizations with Many Eyes [10]. All of the visualizations in Many Eyes are rendered in Flash. Users have to upload their data to the application and then select an appropriate visualization which matches the data set. There exist 16 pre-defined visualization types ranging from spreadsheet charts, maps, text, network diagrams, and tree maps. No user generated visualization types can be uploaded to the application, developers can't create visualizations using external data sources, and data sets are publicly accessible. The Google Visualization API allows for more flexibility than Many Eyes, but provides less powerful visualization techniques.

4. CONCLUSIONS

We are interested in understanding Java software through visual software analytics which uses software and information visualization techniques to confirm the expected and expose the unexpected of software. In this paper we have explored creating visualizations from software metrics data using the Google Visualization API in order to support visual software analytics. We found that the *weaknesses* of the API is that only visualization chart like types are really supported, no specific software visualization techniques are supported, visualizations only operate on structured data, and support for animation and user tasks is limited. The *strengths* are that no additional plugins are required to display visualizations, all code and data are processed and rendered in the browser, multiple visualizations can be embedded on the same web page, and that the API can be extended to create domain specific visualizations. In the future, we plan to integrate some of these Google Visualization API techniques to build a complete visual software analytics tool for developers to understand Java software.

Acknowledgments

This work is supported by the New Zealand Foundation for Research Science and Technology for the Software Process and Product Improvement project, and Telstra Clear.

5. REFERENCES

[1] Craig Anslow. Evaluating extensible 3D (X3D) graphics for use in software visualisation. Master's thesis, Victoria University of Wellington, 2008.

[2] Craig Anslow, James Noble, Stuart Marshall, and Robert Biddle. Web software visualization using extensible 3D (X3D) graphics. In *SoftVis*, pages 213–214. ACM, 2008.

[3] Craig Anslow, James Noble, Stuart Marshall, and Ewan Tempero. Visualizing the word structure of java class names. In *OOPSLA Companion*, pages 777–778. ACM, 2008.

[4] Gareth Baxter, Marcus Frean, James Noble, Mark Rickerby, Hayden Smith, Matt Visser, Hayden Melton, and Ewan Tempero. Understanding the shape of java software. In *OOPSLA*, pages 397–412. ACM, 2006.

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

[6] Matthew Duignan, Robert Biddle, and Ewan Tempero. Evaluating scalable vector graphics for use in software visualisation. In *APVis*, pages 127–136. Australian Computer Society, Inc., 2003.

[7] Google. Google visualization API, 2008. `http://code.google.com/apis/visualization`.

[8] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *VL*, pages 336–343, 1996.

[9] James J. Thomas and Kristin A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics.* IEEE, 2005.

[10] Fernanda B. Viegas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. Manyeyes: a site for visualization at internet scale. *Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007.