Investigation and Comparison of JavaScript Visualisation Libraries for Deloitte Analytics
DELOITTE ANALYTICS
Investigation and Comparison of JavaScript Visualisation Libraries

March 2014

Prepared by: Coleman Conmey  Supervisor: Aideen Keaney
DECLARATION

I declare that the work described in this dissertation has been carried out in full compliance with the ethical research requirements of the School of Computer Science and Statistics.

Signed: ___________________

Coleman Conmey
24th March 2014
ABSTRACT

The aim of this project was to carry out an evaluation and comparison of a number of JavaScript Visualisation Libraries (JVLs) for the Deloitte Analytics Department. JVLs produce interactive web orientated visualisations which can be presented through the use of dashboards. There are a large number of existing libraries, each with their own advantages and disadvantages. It was decided that the best approach for the project was to develop an evaluation framework. Five JVLs were chosen for evaluation; amCharts, CanvasJS, Google Charts, Highcharts and NVD3.js. The evaluation framework provides the client with a standard process for evaluating JVLs.
PREFACE

Deloitte Analytics (hereafter referred to as the “client”) in Ireland deliver solutions to data, technology, and business challenges. The Analytics Department have produced a variety of dashboards for specific clients and demonstration purposes. The client was interested in carrying out an evaluation of a number of JVLs used for dashboard design.

The final evaluation meets the terms of reference specified by the client by providing an evaluation of a number of JVLs. The project has also provided a structured approach to implementing JVLs, and a systematic approach for comparing them to one another.

Development of the evaluation framework was not without its difficulties however. Determining what aspects of a JVL were necessary to include in the final evaluation was challenging and involved the comparison and implementation of many different JVLs. To add to this, every JVL is extensible and modifiable, making it difficult to make contrasts between them.

I would like to thank Mr. Ciarán Tobin and Mr. David Horn for their encouragement and help throughout the project. In particular, I would like to thank Professor Aideen Keaney for her invaluable guidance over the course of the project.
# DELOITTE ANALYTICS

Investigation and Comparison of JavaScript Visualisation Libraries

*March 2014*

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>NO.</th>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION AND SUMMARY</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>The Client</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>The Project Background</td>
<td>1</td>
</tr>
<tr>
<td>1.3</td>
<td>Terms of Reference</td>
<td>2</td>
</tr>
<tr>
<td>1.4</td>
<td>Summary</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>PROJECT OVERVIEW</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Description of Terms</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Project Objectives</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>Project Description</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>Overview Diagram</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>Technical Environment</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>DESCRIPTION OF WORK DONE</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Library Investigation</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>Dashboard Design</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>Evaluation Framework</td>
<td>14</td>
</tr>
<tr>
<td>3.4</td>
<td>JVL Implementation</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>CONCLUSIONS AND RECOMMENDATIONS</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Conclusions</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Recommendations</td>
<td>25</td>
</tr>
</tbody>
</table>
## APPENDICIES

<table>
<thead>
<tr>
<th>NO.</th>
<th>CONTENT</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Original Project Outline</td>
<td>A.1</td>
</tr>
<tr>
<td>B</td>
<td>Interim Report</td>
<td>B.1</td>
</tr>
<tr>
<td>C</td>
<td>Dashboard Requirements</td>
<td>C.1</td>
</tr>
<tr>
<td>D</td>
<td>Screenshots</td>
<td>D.1</td>
</tr>
<tr>
<td>E</td>
<td>Sample Source Code</td>
<td>E.1</td>
</tr>
<tr>
<td>F</td>
<td>Evaluation Findings</td>
<td>F.1</td>
</tr>
<tr>
<td>G</td>
<td>Technical Report</td>
<td>G.1</td>
</tr>
<tr>
<td>H</td>
<td>Glossary and Relevant Terms</td>
<td>H.1</td>
</tr>
<tr>
<td>I</td>
<td>User Guide</td>
<td>I.1</td>
</tr>
</tbody>
</table>

### REFERENCES
1. INTRODUCTION AND SUMMARY

This chapter introduces the project by providing an overview of the client and project background. It sets out the terms of reference and the structure of subsequent report chapters. A Glossary can be found in Appendix H.

1.1. The Client

Deloitte provides audit, consulting, financial advisory, risk management, and tax services to selected clients. Deloitte Analytics in Ireland specialises in capturing, managing and analysing data.

1.2. The Project Background

Dashboards are a valuable tool for analysing and understanding information. A dashboard is a web interface that organises and presents information in a way that is easy to read. They may be tailored for specific demonstration purposes, depending on the information provided by the developer or required for the client. For example, a Business Information (BI) dashboard displays the current status of metrics and Key Performance Indicators (KPIs) of an organisation (see Appendix H.1). An example of a dashboard can be seen in Figure 1.2.1 below.

**FIGURE 1.2.1:** Apple, Inc. Profit and Sales Dashboard. The dashboard displays Apple’s revenue versus income, product sales and smartphone market shares versus Samsung.
The client creates dashboards to present at conferences (i.e. to win new business) and client meetings. Each dashboard is custom built depending on the situation and its purpose. Dashboards are also built at the end of projects as a way visualising results.

The interactive charts within a dashboard are produced using a JavaScript Visualisation Library (JVL). There are now a large number of JVLs, each with their own advantages and disadvantages. The aim of this project was to select a number of JVLs and build a simple dashboard for each in order to compare them on a number of factors: ease of use, appearance, performance, interactivity, support and cost.

Currently, the client does not have a department standard JVL for constructing dashboards. To add to this, developers don’t have a process for evaluating new or existing JVLs. This can lead to a number of issues;

- Developers may not be aware of new or improved JVLs;
- Developers become reliant on one JVL. This JVL may not be suitable for specific dashboard designs and its purpose.

1.3 Terms of Reference

The terms of reference are to:

- Study the template dashboard and sample data provided by the client.
- Investigate how dashboards are constructed.
- Select a number of JavaScript Visualisation Libraries to be evaluated.
- Design a framework for library evaluation.
- Develop a number of test dashboards with accompanying data.
- Implement each JavaScript Visualisation Library, with the test dashboards.
- Using the framework, evaluate and compare each dashboard.

1.4 Summary

Chapter 2 gives a general overview of the project and its objectives. It also includes a brief description of the technical environment. More detailed technical information can be found in Appendix G.1 and to G.2.

Chapter 3 describes the work carried out throughout the project’s duration, including; library investigation, dashboard design and the evaluation framework.

Chapter 4 outlines the conclusions and the recommendations of the report.
2.1. PROJECT OVERVIEW

This chapter provides an overview of the project and the evaluation framework. It details the objectives of the project, provides a detailed description of how the evaluation framework was derived, and gives an overview of the technical environment.

2.1. Description of Terms

Prior to establishing the project objectives, it is useful to give a general overview of the components found in this project. The project is composed of two main components; the JavaScript Visualisation Libraries (JVLs) and the Dashboards. References are made to both components throughout the remainder of the project.

*JavaScript Visualisation Libraries (JVLs):*

JavaScript is a computer programming language commonly used to create interactive effects within web browsers. A simple example would be when a user places a cursor over a tab and the cursor highlights the tab (see Appendix D.2). JavaScript programming is difficult and time-consuming to work with. Fortunately, a lot of JavaScript libraries have been developed and shared online. A JVL is a library of pre-written JavaScript code built for producing web orientated visualisations; such as charts, plots and maps. This code library consists of many JavaScript visualisation functions, which make creating interactive visualisations considerably easier. Built-in interactive elements of a JVL include features such as legends and tooltips. See Appendix H.2 for an explanation of these, along with a host of other interactive features.

Every JVL provides its own unique JavaScript API for writing visualisation functions. A JavaScript Application Programming Interface (API) provides a list of classes and functions that describes how to implement a specific library. API documentation is generally found on the JVLs webpage under “documentation”.

*Dashboards:*

A dashboard provides a web interface through which JavaScript visualisations can be supported. In this project, the dashboard component also includes accompanying data, which is located in a separate JavaScript file. The data is expressed through the dashboards as different JavaScript visualisations. Dashboards can be designed to structure and consolidate many JavaScript visualisations on one web interface. Please see Figure 2.1.1 for an example of a simple dashboard structure.
FIGURE 2.1.1: Dashboard Design Example. A JavaScript Visualisation could be implemented into each of these chart areas.

2.2 Project Objectives

Architects of JVLs (e.g. amCharts, Google, Highcharts etc.) are constantly updating to newer versions of their libraries and improving them with new features, chart types and bug fixes. The client would like to improve their understanding of the different JVLs that are available, and their potential features. The aim of this project is to compile a list of JVLs with different features suitable for an array of dashboard designs.

Secondly, the client does not have a standard process for choosing a JVL. In the past, this has caused difficulties for some developers. For example, a developer chose a JVL that did not support a legend feature, which was a necessity for the chart. This issue was not identified until after dashboard had been developed. Without it, trying to interpret the resulting chart and its information was difficult. As a result, the developer had to use another computer programming language to produce the legend. This was time consuming and the added complexity could have resulted in more errors. This project will provide an evaluation framework that will allow developers to systematically assess each JVL prior to the design of a dashboard; removing issues similar to the one outlined previously.

The first objective of the project is to create a compilation of JVLs (see Section 3.1). Next, each JVL will be used to create individual interactive dashboards, using a selection of test dashboards which have been created as part of the evaluation process (see Section 3.2). The dashboards will then be evaluated using a standard framework (see Section 3.3). In general, the project will provide the client with a list of evaluated JVLs which can be used apart of future dashboard designs; and will allow developers to evaluate libraries systematically for each project, saving time and effort, while at the same time, meeting the basic requirements of the client.
2.3. **Project Description**

The client provided a template dashboard and sample data, so that the focus may be solely on implementing JVLs. The client also provided a developed dashboard; implemented with the xCharts JVL (see Figure 2.3.1). At first, time was spent studying the template dashboard design and the developed xCharts dashboard.

![FIGURE 2.3.1: Client Dashboard Template Implemented using the xCharts JVL. The dashboard displays Ireland’s unemployment (2010-2013), births in five Irish towns/cities (2010 – 2012) and CPI versus GDP (1997 – 2012).](image)

The next step was to replicate what the client had created in Figure 2.3.1 using Google Charts (a JVL suggested by the client). This involved implementing three visualisations (i.e. Area, Bar and Line Chart) similarly to Figure 2.3.1 onto the template dashboard using the Google Charts JVL.

![FIGURE 2.3.2: Client Dashboard Template Implemented using the Google Charts JVL.](image)
A comparison of the two dashboards was then performed. The client suggested that the dashboards should be evaluated based on a variety factors such as: *ease of use, appearance, performance, interactivity, support* and cost. Determining a way to evaluate these factors was more complicated than first anticipated. Two issues arose as a result of the initial comparison:

1. A lack of clarity.
2. An unstructured approach.

**Lack of Clarity:**

Without a well defined definition for factors such as ease of use, performance or interactivity in the context of JVLs, it is not possible to evaluate either dashboard assuredly. Similarly, what comparisons can be made between Figure 2.3.1 and Figure 2.3.2 on their appearances? They are almost identical, which makes it difficult to make any meaningful interpretations between the two. These factors have been determined too difficult to interpret. As a result, it was decided that an evaluation must be conducted at a lower level of granularity.

This lower level of granularity would be presented in the form of an evaluation framework (see Section 3.3). The evaluation framework was designed to analyse the underlying components of each stage of a JVL implementation. The evaluation framework is made up of four new factors deemed to provide a concrete platform for evaluation; *Data and JavaScript API Preparation, Customisation, Visualisation* and *Support and Licensing*. Each factor is made up of a set of subsequent components for evaluation. For example, Data and JavaScript API Preparation is described as the preparation of data and the formation of JavaScript functions used to render a chart to a dashboard at its most primary level.

**Rendering** is the process of generating a visualisation (chart) on the internet from a combination of computer programming languages.

Data and JavaScript API Preparation is made up of three underlying components; *Library Source Code, JavaScript API Format and Data Format*. The library source code contains all of the functions and visualisation capabilities of that JVL. The JavaScript API is a reference manual which details the collection of classes and subsequent class functions for producing visualisations. Finally each library has specific formats for data input. These are all necessary components of the Data and JavaScript API Preparation factor.

The factors of the evaluation framework and their subsequent components and can be seen in Figure 2.3.3. See Section 2.4 for a detailed description of the JVL implementation process.
FIGURE 2.3.3: Evaluation Relationship Diagram. This diagram provides an overview of the evaluation framework.

The diagram outlines the following components:

- **Evaluation**
  - Data and JavaScript API Preparation
  - Library Source Code
  - JavaScript API Format
  - Data Format

- **Support and Licensing**
  - Documentation
  - Community Support
  - Personalised Support
  - Licensing

- **Customisation**
  - Default Appearance
  - Custom Appearance
  - Animation
  - Chart Rendering

- **Visualisation**
  - Titles/Labels
  - Tooltips
  - Legend

- **Axes**

- **Tools**
  - Titles/Labels
  - Tooltips
  - Legend

- **Legend**

- **Factors**

- **Document**
**Unstructured approach:**

Without a clear set of objectives to be achieved by either dashboard, it would be difficult to make any comparisons. For example, the xCharts JVL could have been tedious to implement, but may meet a significant amount of the requirements desired by the client. For this reason, it was decided to create two separate dashboards, the Default Dashboard and the Custom Dashboard, with corresponding lists of requirements/objectives. An example of the Default Dashboard versus the Custom Dashboard can be seen in Figure 2.3.4.

Requirements for either dashboard included features (e.g. tool-tips, legends, animation etc), chart types and styling requirements that each JVL must adhere to (See Appendix C for a description of the dashboard requirements). This way, JVLs can be compared on; the difficulty of implementing these requirements through the use of their corresponding JavaScript API's; and their ability to meet these requirements (see Section 3.3).

**FIGURE 2.3.4:** Example of the Default Dashboard (Top) versus the Custom Dashboard (Bottom) Implemented using the Highcharts JVL.
The example demonstrates the default capabilities of the Highcharts JVL versus a custom design applied through the customisation of its JavaScript API functions. The rationale behind each dashboard was as follows:

- Default Dashboard; used to evaluate the difficulty of implementing a specific JVL. The dashboard also provides a way of determining a JVL’s default capabilities with regards to features and appearance.
- Custom Dashboard; used to evaluate the customising capabilities of a specific JVL (i.e. to change a charts appearance and add non-default features), and the difficulty in customising the JavaScript API functions to achieve this visualisation.

As outlined in the project background in Section 1.2, the client required a number of JVLs to be selected and compared. This is an important part of the project and is described in further detail in Section 3.1. The aim was to create a list of ten of the most popular JVLs, including their chart types and important features (see Table 3.1.1 and Table 3.1.2). The list was compiled in discussion with the client and by referring to internet sources (see References). Five JVLs were then chosen to be evaluated in detail using the evaluation framework; amCharts, CanvasJS, Google Charts, Highcharts and NVD3.js. They were chosen based on their collection of chart types and key features. Furthermore, their selection was aided by the ease at which the information on chart types and features could be found using their corresponding web pages.

The five JVLs chosen to be evaluated were implemented using the two dashboards (i.e. the Default Dashboard and the Custom Dashboard). This in effect means that ten dashboards were created in total. See Appendix D.3 through to D.12 for a screenshot of each dashboard and its corresponding JVL. The client will be provided with the particular source code used for implementing each library to the two dashboards. This code can be beneficial to the client for future projects as it represents a library of worked and documented examples using five different JVLs.

2.4. Overview Diagram

Figure 2.4.1 gives a high-level overview of the how the implementation of a JVL is performed, and how the evaluation framework relates to each aspect of this process. There are two main processes in a JVL implementation; Data and JavaScript API Preparation and Rendering.

- Data and JavaScript API Preparation; this process involves the preparation of data and the formation of JavaScript API functions used to render a chart to a dashboard.
- Rendering; rendering is the process of generating a visualisation (chart) on the internet from a combination of computer programming languages (HTML, CSS, JavaScript and jQuery).

Customisation was another important part of the implementation process and was dependent on dashboard selection (Custom Dashboard). This can include customising chart colours, adding additional features such as animation and tooltips or even doing something as basic as adding a title to a chart.
Customisation was only relevant for the Custom Dashboard because it required the change of appearance from the defaults provided by a specific JVL. This is an important aspect of the project, as the client relies heavily on the customisation of charts for specific clients and demonstrations.

A JavaScript Visualisation is a product of Data and JavaScript API Preparation and rendering a JVL to a dashboard with or without customisation. The visualisation produced is then evaluated on a collection of components depending on their relevance to that specific dashboard. An example would be an evaluation of the Default Appearance component, which is only relevant to the Default Dashboard. It would be evaluated to determine what default styling format and default features the JVL provides. An example would be that without any customisation, the Highcharts JVL provides animation, legends and tooltips by default. This is useful information for the client. This particular JVL would be suitable for cases when the client does not have a sufficient amount of time to begin customising a chart.

The evaluation framework is continually updated throughout every stage of the JVL implementation process. This helped to develop an effective framework which evaluates the most common factors associated with each JVL. The evaluation process is explained in detail in Section 3.3.

**FIGURE 2.4.1:** Overview Diagram. This diagram represents a flowchart of the various processes and decisions taken throughout the implementation of a JVL.
2.5. Technical Environment

Dashboards are created using a combination of web technologies i.e. HTML, JavaScript/jQuery and CSS. This collection of technologies for creating interactive and animated websites such as dashboards is generally described under the umbrella term Dynamic HTML (DHTML) (see Appendix G.2).

HyberText Markup Language (HTML) creates web pages that can be used to support and display JavaScript visualisations. HTML is used to create chart areas, tabs, dropdown boxes and text elements. Cascading Style Sheets (CSS) formats HTML. CSS are used to structure dashboards (i.e. chart, tabs, dropdown boxes and text positioning) and format dashboard designs (i.e. colours and fonts).

The appearance of dashboards is important for demonstration and visualisation purposes. For this reason, all modern browsers (e.g. Mozilla Firefox 27, Google Chrome 33, and Internet Explorer 10 etc) support Scalable Vector Graphics (SVG). SVG is a language for describing two-dimensional vector graphics in Extensible Markup Language (XML). JavaScript visualisations which use SVG do not lose any quality if they are zoomed or resized (see Appendix G.5). Not all JVLs support SVG. JVLs that do not support VML create two-dimensional vector graphics (visualisations) using Vector Markup Language (VML), which is also an XML based file format. VML has been largely deprecated in favour of SVG. VML is not supported in many modern browsers anymore (e.g. IE 10 or above), although some libraries support both file formats.

JVLs use JavaScript (JS) to allow client-side scripts to interact with the user through the dashboard visualisations and web browser. Client-side scripting describes the actions performed on clients’ computers. JS is an object orientated computer programming language commonly used to create interactive effects within web browsers. Interactions between the user and the dashboard are primarily controlled by jQuery. jQuery is a JS library designed to simplify the client-side scripting of HTML (i.e. tabs and dropdown box interactions). An example of a jQuery interaction can be seen in Appendix D.1. jQuery is used to initiate JVL rendering of charts within the dashboard when the web page is called. jQuery is continually updated, and the latest version (jQuery 2.0) was downloaded in March 2014.

Data which is used to provide the information to display charts is stored in JavaScript Object Notation (JSON). JSON is a type of syntax in JavaScript used for storing and exchanging text information. JSON is language independent, lightweight in terms of code, fast and human readable which makes it easy to work with.

All HTML, JavaScript, CSS and JVL files will be available on a CD accompanying this document. More detailed technical information and an explanation of the terms above can be found in Appendix G.
3. DESCRIPTION OF WORK DONE

This chapter outlines the work carried out throughout the duration of the project. The chapter is made up of four sections:

- Library investigation;
- Dashboard design;
- Evaluation process;
- JVL implementation.

3.1. Library Investigation

The client was interested in the evaluation and comparison of a number of JVLs. A compilation of JVLs needed to be defined prior to dashboard design and implementation. Ten popular charting libraries were chosen using a combination of internet sources (see References) and discussions with the client. The client outlined the important features and chart types generally used within the Analytics Department. These included; charts types such as scatter plots, line charts, bar charts and column charts, to name a few (see Table 3.1.1); and features such as legends, tool-tips and a host of other interactive features (see Table 3.1.2). An explanation for each of these features can be found in Appendix H.2. Each library was evaluated based on their chart types and list of features.

It was decided that there would be no added benefit from including the xCharts library in the list, as the client is already familiar with that library. Five JVLs were chosen for implementation from the tables below based on their ability to meet the requirements of the client. The five JVLs were; amCharts, CanvasJS, Google Charts, Highcharts and NVD3.js.

Evidently from the tables below, a lot of the libraries appear to have similar capabilities. Another differentiating factor was the ease at which the information on chart types and features could be found using their websites. Finding the information from certain JVL’s websites was made more tedious by poor documentation and simply a lack of recognition on the internet. It was easy to establish the more popular libraries through the use of internet websites such as Google and Wikipedia. An example of this would be Rickshaw, an open source library (see Appendix H.1) that provides very little in terms of documentation and examples. In comparison to this library, Highcharts provides extensive documentation and easy to access examples with JavaScript code available.

It must also be noted that some of the charts below have extensive charting and map types to choose from. Not all chart types were considered for this project but it would be a consideration for further study.
### TABLE 3.1.1: JVL Chart Types Table.

<table>
<thead>
<tr>
<th>Chart Types</th>
<th>amCharts</th>
<th>Arcadia Charts</th>
<th>Chart.js</th>
<th>CanvasJS</th>
<th>gRaphaël</th>
<th>Google Charts</th>
<th>Highcharts</th>
<th>NVD3.js</th>
<th>jqPlot</th>
<th>RGraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Bar</td>
<td>✔️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Column</td>
<td>✔️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Scatter</td>
<td>✔️</td>
<td>☑️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Line</td>
<td>✔️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Time Series</td>
<td>☐</td>
<td>☑️</td>
<td>☐</td>
<td>☑️</td>
<td>☐</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
</tbody>
</table>

### TABLE 3.1.2: JVL Features Table.

<table>
<thead>
<tr>
<th>Chart Types</th>
<th>amCharts</th>
<th>Arcadia Charts</th>
<th>Chart.js</th>
<th>CanvasJS</th>
<th>gRaphaël</th>
<th>Google Charts</th>
<th>Highcharts</th>
<th>NVD3.js</th>
<th>jqPlot</th>
<th>RGraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legend</td>
<td>✔️</td>
<td>☑️</td>
<td>☐</td>
<td>☑️</td>
<td>☐</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Drilldown</td>
<td>✔️</td>
<td>☑️</td>
<td>☐</td>
<td>☑️</td>
<td>☐</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Tooltips</td>
<td>✔️</td>
<td>☑️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Scrolling/Panning</td>
<td>✔️</td>
<td>☑️</td>
<td>☐</td>
<td>☑️</td>
<td>☐</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
<tr>
<td>Zooming</td>
<td>✔️</td>
<td>☑️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>✔️</td>
<td>☐</td>
<td>☑️</td>
<td>☑️</td>
<td>☐</td>
</tr>
</tbody>
</table>
3.2. Dashboard Design

The dashboards were developed using a combination of Microsoft Expression Web 4 and a free Mozilla Firefox extension, Firebug. Microsoft Expression Web 4 was primarily used for dashboard design and formatting (HTML and CSS). It did support JavaScript but was relatively poor for debugging (finding errors) in the JavaScript code. As a result, Firebug was used when implementing JavaScript code from the JVLs. Rendering times were also found using Google Developer Tools.

3.3. Evaluation Framework

A framework was designed to look at every aspect of a JVL; from studying the online documentation, all the way to creating the interactive visualisation. The evaluation framework is built around the processes used for data and JavaScript API preparation, customising and rendering a visualisation. Four main factors were defined for the evaluation framework; Data and JavaScript API Preparation, Customisation, Visualisation, and Support and Licensing. In this section, each factor is described in detail and its subsequent components. Refer back to Figure 2.3.3 for an overview of the evaluation framework and its components.

To demonstrate the evaluation framework, a worked example will be presented throughout this section. The selected JVL for this evaluation is Google Charts. For a full list of the evaluations and their corresponding findings see Appendix F.1 through to F.8.

Data and JavaScript API Preparation:

This factor is described as the preparation of data and the formation of JavaScript API functions used to render a chart to a dashboard. This factor at a basic level, analyses the minimal amount of coding required from a JVL to produce a visualisation after rendering. It is applied to both the Default and Custom Dashboard. The reason for this being that both dashboards require different chart types, and therefore contributes to a more comprehensive evaluation of a JVL. This factor requires the analysis of three components: the Library Source Code, a JVL’s API Format and a JVL’s Data Format.

FIGURE 3.3.1: Data and JavaScript API Preparation Relationship Diagram.
• Library Source Code; the library source code contains all of the functions and visualisation capabilities of a JVL. This can either be downloaded from the specific charting website to the user’s platform, or a web link can be referenced directly in the dashboard HTML (see sample code in Appendix E.1). There are benefits to both approaches; if the library is downloaded directly, it is possible to work offline (i.e. not directly connected to the internet). Open source libraries may be customised by anyone and readily adapted to meet specific user needs. Most library source code is minified, which is the process of removing all unnecessary characters from the library. JVLs normally provided both minified and non-minified source code. See Appendix G.6 for advantages and disadvantages of minification in JavaScript.

If a link is used to implement a JVL, the library is restricted to online use (i.e. the charts must have access to the internet to produce visualisations). Although there is a restriction in having access to the internet, it simplifies the process considerably.

Some JVLs also requires the developer to download their corresponding Cascading Style Sheets (CSS) as their charts will not work without them. For example NVD3.js requires a specific style sheet, and without it the charts become disfigured (see Appendix D.13).

• JavaScript API Format; A JavaScript API is a reference manual which details the collection of classes and subsequent class functions for producing visualisations. It is a guide on how make the source code produce visualisations.

Every JVL provides an individual JavaScript application programming interface (API). This component is measured on how difficult it was found to understand the class structures within a JVL API, and how difficult it was found to implement their corresponding functions.

• Data Format; each JVL requires specific formats for data input. Three formats have been found through this evaluation: JSON, Two-Dimensional Arrays, and DataTables.

JavaScript Object Notation; or JSON, is a type of syntax in JavaScript that is used for storing and exchanging text information. This means that the data is stored in a human-readable format that is easy to interpret. A JSON object can be passed into a JVL function as a parameter. This makes the code much lighter as it is text based, and generally easier to implement. See sample code in Appendix E.2 for an example of a chart rendered using the JSON format.

A Two-Dimensional Array is simply a group of two lists, or arrays that are combined into one dataset. A Two-Dimensional Array can be passed into a JVL function as a parameter.
• The last option for the component Data Format is a DataTable. A DataTable is a specific data format required for Google Charts. A DataTable is a basic two-dimensional table. All data in each column must have the same data type. Each column has a descriptor that includes its data type, a label for that column (which might be displayed by a visualization), and an ID, which can be used to refer to a specific column (as an alternative to using column indexes). Each cell in the table holds a value. Cells can have a null value, or a value of the type specified by its column.

**TABLE 3.3.1:** Google Charts Data and JavaScript API Preparation Evaluation Table.

<table>
<thead>
<tr>
<th>Library Source Code</th>
<th>JavaScript API Format</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>☑ Hyperlink</td>
<td>☐ Easy</td>
<td>☐ JSON</td>
</tr>
<tr>
<td>☑ Downloadable</td>
<td>☑ Moderate&lt;sup&gt;1&lt;/sup&gt;</td>
<td>☑ Two-Dimensional Array</td>
</tr>
<tr>
<td>☑ CSS</td>
<td>☐ Hard</td>
<td>☑ DataTable</td>
</tr>
</tbody>
</table>

**Comments**

1. The API format is easy to understand, but having to use DataTables made the process allot more difficult when compared with other charting libraries. Further explanation can bee seen in Appendix F.4.

**Customisation:**

Customisation can be performed using a JVL’s API functions to change the chart style from the defaults of a JVL. Depending on a JVL’s API format and extensiveness, customising can be challenging. This is most evident when switching between different libraries and having to learn new formats for accomplishing the same task. An example would be adding a chart title, which requires a different process for each individual JVL. The majority of time spent on implementing a library is in the customisation process. Customisation can be carried out on the following properties and features: *Axes*, *Titles/Labels*, *Tooltips* and *Legends*. The Customisation factor is primarily concerned with the Custom Dashboard. It is concerned with determining whether a JVL can provide dynamic customisation functions within its library, and without having to result to other forms of programming code. Using the Custom Dashboard, an evaluation of the customisation ability of a JVLs features is possible. See Appendix C for a list of the features assigned to the Custom Dashboard and their corresponding requirements.

**FIGURE 3.3.2:** Customisation Relationship Diagram.
• Axes; an axis has many qualities that are necessary for successful JVL implementation. First of all, it is important that developers have the option to change the range on an axis or axes, to provide a better visualisation of the data. Axes should also be able to work with time series data, providing the option to format the output on an axis. Another option is the management of gridlines, which manages the number of points represented on an axis.

• Titles/Labels; chart and axes titles are an important feature of any chart to help explain the information displayed. Labels help to identify data points along an axis. Titles and labels can be formatted in many different ways i.e. colour, font size, font family etc. Labels can be customised by adding prefixes such as dollar signs or suffixes such as percentages. Other important aspects to label customisation are; their ability to show dates effectively and their ability to rotate labels. Rotation allows a chart to display more labels if the data set is cluttered and has many different categories (see Appendix D.11). The customisation of titles/labels allows the developer to produce effective charts for better visualisations. The ability to remove labels or limit the number shown can be helpful if a chart is overwhelmed with unnecessary labels. An example of this would be in a time series chart. It may be appropriate to show labels only at certain intervals.

• Tooltips; tooltips are an interactive part of the chart, and display information on a data point when the user places a cursor over it. Depending on the requirements, tooltips information can be manipulated to present the information from the chart in different ways. Tooltips have many different aspects that can be altered. These include the tooltip indicator itself and the format of the text within the tooltip. Tooltips can also be customised to show HTML text and images, creating interesting interactive capabilities. Another important aspect is the ability of a chart to show multiple series on one tooltip; this occurs in combo charts when it would be convenient to see the two corresponding data points. See Appendix D.14 for an example of multiple series on one tooltip.

• Legend; there are two types of legends recognised in JVLs; Static legends and Interactive legends. Static legends allow users to select data series, highlighting them within the chart; this is a form of basic interactivity. Interactive legends allow users to hide certain data series and focus on the other remaining series. This allows the user to have complete control over the data they want to analyse (see Appendix D.15). It may also be the case that a JVL does not support a legend feature. Finally JVLs should allow for the repositioning of legends. If not, charts can be at a disadvantage when legends take up critical space in the chart area of a dashboard. To do something as simple as move a legend underneath a line chart can greatly improve the visualisation. Legends are customisable in the sense that data series descriptors can be can formatted (i.e. font, colour, indicator/marker etc).
TABLE 3.3.2: Google Charts Customisation Evaluation Table.

<table>
<thead>
<tr>
<th>Axes</th>
<th>Titles/Labels</th>
<th>Tooltips</th>
<th>Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Range</td>
<td>✓ Format titles&lt;sup&gt;1&lt;/sup&gt;&lt;br&gt;✓ Format labels&lt;br&gt;✓ Add prefix/suffix to labels&lt;br&gt;✓ Format time series labels&lt;br&gt;✓ Rotate labels&lt;sup&gt;2&lt;/sup&gt;&lt;br&gt;✓ Limit labels&lt;sup&gt;3&lt;/sup&gt;</td>
<td>✓ Format tooltip&lt;sup&gt;4&lt;/sup&gt;&lt;br&gt;✓ Format text&lt;br&gt;✓ Manipulate data&lt;br&gt;✓ HTML&lt;br&gt;✓ Multiple Series</td>
<td>✓ Static&lt;br&gt;☐ Interactive&lt;br&gt;✓ Position&lt;br&gt;☐ Format</td>
</tr>
<tr>
<td>✓ Time series</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓ Gridlines</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comments

1. Titles cannot be centred.
2. Only works for the horizontal axis.
3. Google Charts cannot limit the number dates shown on a time series. It defaults to five or less. In order to work with time series labels effectively, the data must be presented in ‘string’ format in the DataTable, instead of ‘date’.
4. Only through the use of HTML or the addition of another row to the DataTable. It is not possible to manipulate the data shown in the tooltip by adding suffixes or prefixes.

Visualisation:

A JavaScript Visualisation is produced as a result of rendering a JVL with the dashboard. Visualisation is evaluated under five components: Default Appearance; Custom Appearance; Animation and Chart Rendering.

**FIGURE 3.3.3:** Visualisation Relationship Diagram.

- Default Appearance; the Default Dashboard is used to evaluate the layout and appearance of the charts produced without any customisation. This will determine whether a JVL can successfully render the data to a chart without the need for customisation. Axes labels should be clear and easy to read, evenly distributed and formatted correctly. Data should be distributed efficiently; making full use of the chart area and displaying as vivid an image of the data as possible. The default appearance will also indicate default features provided by the JVL. The time saved by a developer without having to customise the appearance of a chart could be considerable.
• Custom Appearance; the Custom Dashboard is evaluated on its appearance capabilities when given a strict set of styling parameters See Appendix C.1 for the requirements of each dashboard. Sometimes a developer will need to create a specific dashboard for a client. This could involve customising the charts using colours, fonts and require having a set list of features. This is significant for the client as they build custom dashboards regularly. The will allow the client to see the custom formatting abilities of the five different JVLs.

• Animation; powerful visualisations can be produced when adding animation to charts. The animation sequence that takes place after the chart has been rendered and in-between interactions with the chart and the user (e.g. tabs and dropdown boxes) can have some effects on the visual performance of a dashboard. Some animation sequences are delayed on initiation and others lag between responses of the interaction with the user. There are three types of animation sequences which have been encountered; flow, bounce and transition. Flow and bounce animation are different visual effects that occur when the chart is initialised or when the user interacts with the chart. Transition animation only occurs when the user interacts with the chart.

• Chart Rendering; charts can be rendered using SVG or VML for older browsers. Some JVLs support both options while others just support SVG and do not cater for older browsers. SVG leads to better visualisations over VML. Another aspect of when charts are rendered is their ability to resize with the browser. Some JVLs are intelligent and resize charts depending on the browser size, while other JVLs charts just compact together and become unreadable. Rendering time is the time it takes for a dashboard to load successfully onto a webpage.

<table>
<thead>
<tr>
<th>Default Appearance</th>
<th>Custom Appearance</th>
<th>Animation</th>
<th>Chart Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Effective axes scaling</td>
<td>✓ Excellent^3</td>
<td>✓ Flow</td>
<td>✓ SVG</td>
</tr>
<tr>
<td>✓ Legend^1</td>
<td>✓ Good</td>
<td>✓ Bounce</td>
<td>✓ VML</td>
</tr>
<tr>
<td>✓ Tooltips</td>
<td>✓ Poor</td>
<td>✓ Transition^4</td>
<td>✓ Resizable</td>
</tr>
<tr>
<td>✓ Animation</td>
<td>✓ Issues^2</td>
<td></td>
<td>Rendering time 160 milliseconds</td>
</tr>
</tbody>
</table>
Comments

1. Legends run out of chart area if they are too long (see Appendix D.5).
2. Labels run out of chart area if they are too long.
3. Legends take their names from corresponding column id’s in the DataTable. Because of this, jQuery cannot be used to manipulate the text that appears in the legend when a user switches between tabs and dropdown boxes.
4. Transition Animation takes away from the initial visual experience that occurs with other JVLs. Trying to integrate animation into Google Charts is extremely tedious (see code in Appendix E.6). A DataTable made the manipulation of datasets on one chart awkward as all data must be combined into one DataTable to allow for the transition sequence to occur.

Support and Licensing:

Support describes the different resources a JVL provides to its users. Many different options are available depending on the JVL. Support and Licensing has been divided into four components; Documentation, Community Support, Personalised Support and Licensing.

![Support and Licensing Diagram](image)

**FIGURE 3.3.4**: Support and Licensing Relationship Diagram.

- Documentation; API documentation is provided for each JVL through their corresponding websites. Depending on the JVL, some JavaScript API’s are more extensive than others. This is especially the case when considering an open source library like NVD3.js versus Google Charts. NVD3.js has a poorly documented JavaScript API which makes it very difficult to implement. Also included under the factor Documentation is tutorials. Many JVLs architects provide online tutorials on how to create charts. Other JVLs go a step further and provide interactive demos on their website where the user can implement and test different chart types and data online without having to download any packages. It is a good way of testing a JVL to see if it suits what the developer requires for a specific dashboard.

- Community Support; this component describes the main influences that were used as a resource outside the dynamics of a JVL’s support options. Forums such as Stack Overflow and Google Groups provide a feature for asking questions on technical programming issues such as web development. Stack Overflow in particular is a useful site as it provides many solutions to JVL implementation.
GitHub is a web-based hosting service for software development projects. GitHub offers both paid plans for private repositories, and free accounts for open source projects. An example of an open source JVL found on GitHub is the NVD3.js JVL. Users can access source code, JavaScript API documentation and chart examples directly from GitHub and it provides another channel for queries.

JSFiddle is another free community resource that was used and referred to extensively throughout the implementation of different JVLs. JSFiddle is an online development playground primarily developed for JavaScript implementation. This was a very useful resource for JVL implementation. JSFiddle lets developers play with the three core elements of Web development; HTML, CSS and JavaScript. JSFiddle provides a custom environment to test your JavaScript, HTML, and CSS code right inside the browser. Many examples of different JavaScript Visualisations are posted on jsFiddle. Some JVLs use JSFiddle as part of their support options.

- Personalised Support; JVLs that are not open source are more likely to provide extensive personalised support to its subscribers. This is an effective way of gaining a strong community following. Examples of personalised support include; personalised forums for requesting features and highlighting bugs, email, frequently asked questions section etc.

- Licensing; licenses are either; free for commercial and non-commercial usage; or the proprietor of the JVL will charge a licensing fee if the library is to be used by an organisation for commercial use. This is an important aspect worth considering by the client, as they may not want to pay licensing fees when there are free libraries available. Licenses vary from library to library and in some cases are dependent on the size of the organisation. A description of the terms and conditions for licenses will be described in the comments section, depending on the JVL.
TABLE 3.3.4: Google Charts Support and Licensing Evaluation Table.

<table>
<thead>
<tr>
<th>Documentation</th>
<th>Community Support</th>
<th>Personalised Support</th>
<th>Licensing</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ JavaScript API documentation</td>
<td>✓ Forums¹</td>
<td>✓ Email</td>
<td>✓ Free for all usage⁴</td>
</tr>
<tr>
<td>✓ Tutorials</td>
<td>✓ GitHub</td>
<td>✓ FAQ</td>
<td>✓ Free for non-commercial usage</td>
</tr>
<tr>
<td>❑ Demos</td>
<td>❑ JSFiddle</td>
<td>✓ Features request²</td>
<td></td>
</tr>
<tr>
<td></td>
<td>❑ Bug fixes</td>
<td>✓ Other³</td>
<td></td>
</tr>
</tbody>
</table>

Comments

1. Forums are provided in the form of Google Groups.
2. Provides a catalogue for bug reports and feature requests.
3. Google’s Hall of Fame where users can submit their developed visualisations to be posted on the web page.
4. Although it is free for all usage, it is not open source. It is important to make the distinction between the two. For an explanation of open source see Appendix H.1.

Bugs
1. Cannot use ‘google’ as a variable name when implementing Google Charts.

3.4. JVL Implementation

JVL Implementation involved the preparation of data and the formation of JavaScript API functions used to render a chart to a dashboard. Each JVL contains functions used for producing and customising charts, which is known as the source code. The source code, depending on the JVL, was either downloaded from the specific charting website to a platform, or a link was referenced directly in the dashboard HTML code. If the source code was downloaded to the platform, it must be referenced similarly in the HTML code to its directory/location on the platform (see Appendix E.1). The dashboard now has full access to all of the charting capabilities the JVL has to offer.

Preparation of data was an important part of the implementation process. There were many different format choices for storing data. Extensible Markup Language (XML) is widely used but a decision was made to use JavaScript Object Notation (JSON) for this project. The main reason for choosing JSON as the primary format for data is because many of the JVLs use JSON formatted data as inputs to their functions. Not one of the JVLs implemented throughout this project have used data in XML format. Intuitively, it makes sense that libraries choose to use JSON as it is an extension of JavaScript. It is also widely used and seen as the more popular choice in comparison with XML and not just for JVLs. JSON provides a short and quick way to read and write data. For this project, the data was to be reused for every JVL implementation. The project consisted of ten separate dashboards and two datasets.
Normally, using the JavaScript API functions, a dataset would be hard coded into each individual dashboard. This would mean the replication of the same dataset multiple times. Therefore, to save time and platform memory, the two datasets were stored in separate JavaScript files, which were then referenced by each dashboard. This created difficulties as some JVLs required data to be formatted in a specific way other than JSON. amCharts, CanvasJS and NVD3.js required the data to be in JSON format whereas Google Charts and Highcharts required the data to be in different formats. Using a combination of JavaScript functions and jQuery, the data was manipulated to match the format required by that specific JVL in the dashboard HTML code prior to rendering (see Appendix E.2). Manipulating the data into the right format for each library created some of the greatest difficulties in the project.

The final part of JVL implementation, involves creating a function for rendering a chart to a specific area within the dashboard. First of all, the function requires a method for obtaining the data from its directory. Depending on the JVL, another function could be used to manipulate the data into the right format. A number of functions from the JVL were then used to create the specific chart visualisation. The final part of the process involved referencing the correct chart area to render the visualisation to (see Appendix E.6). See Figure 3.4.1 below; which displays a chart rendered to Chart Area Three using Google Charts.

![Chart Rendering](image)

**FIGURE 3.4.1:** Chart Rendering. Chart Area Three displays a scatter plot of Apple’s revenue versus net income using the Google Charts JVL.
4. **CONCLUSIONS AND RECOMMENDATIONS**

This chapter discusses the conclusions and recommendations arising from the project regarding JVL implementation, evaluation and future evaluation of libraries.

4.1. **Conclusions**

The finished project fully meets the requirements of the client as described in the terms of reference. The establishment of a comparable list of a number of JVLs (see Appendix F) helps to improve the clients understanding of the varying capabilities of different libraries. Furthermore, the evaluation framework also enables the client to use a structured approach for future library evaluations (Section 3.3).

Despite the successful implementation of each JVL, there were some difficulties encountered in the project. For example, the dissimilar data format requirements for each individual JVL (Section 3.4). Each dataset had to be configured prior to implementation. This resulted in the creation of complex functions for data reconfiguration (see Appendix E.4). These front end functions for JSON manipulation are an extension of what was specified by the client. It is expected that the client can use these functions for future dashboard projects.

Surprisingly, the most difficult part of the implementation process was not the coding aspect, but the in depth analysis of each individual JavaScript API. For each implementation, it was clear what the requirements were, but it was not clear whether the JVL supported these requirements. Long periods were spent searching for functionality, only to find out that it’s simply not there in the first place. An example of this is with the amCharts JVL. The aim was to rotate labels on the horizontal axis of a bar chart. It was then discovered that amCharts does not support this capability.

The project required a significant amount of adaptability on behalf of the developer. With each new library came a new JavaScript API, which in turn meant a new set of classes and functions that had to be examined, understood, implemented and then evaluated.

At the time of writing there was nothing similar to this framework and with such a high level of detail available on the internet. The framework has been designed to allow for modification and extensibility. Features can and should be added depending on the dashboard design.

It was evident from the study that there was no superior JVL. Each individual library had some outstanding qualities, but no library can provide all of the functionality for every dashboard design. It can be concluded that JVL selection is dependent on dashboard design and context. There is no such thing as “one library fits all”.
4.2. **Recommendations**

**Update Regularly:**

The framework has been designed to be extensible and it is envisaged that the framework should be extended by the client when new versions of libraries are released and new JVL are evaluated. Furthermore, the framework was composed of some of the most active features and chart types among JVLs. The client should expand this list of features and chart types. This project did not involve maps which is something the client is interested in.

**Library Catalogue:**

A recommendation to the client would be to use the evaluation framework as documenting resource for future dashboard designs. This way, the client could build up a large repository of JVLs and their functionalities.

**Further Development:**

In order to provide a more comprehensive evaluation of JVLs, libraries should be implemented on an array of dashboards. It is recommended that for future evaluations, the client should create five to ten dashboards, and evaluate each library across these dashboards.
A. ORIGINAL PROJECT OUTLINE

Client: Deloitte Analytics
Project: Investigation and Comparison of JavaScript Visualisation Libraries
Location: Hatch Street Upper, Dublin 2
Client Contact: Richard Southern, 01 417 2408, risouthern@deloitte.ie
Ciarán Tobin, 01 417 2408, ctobin@deloitte.ie
David Horn, dahorn@deloitte.ie

Department Contact: Aideen Keaney

Client Background

Deloitte is the brand under which tens of thousands of dedicated professionals in independent firms throughout the world collaborate to provide audit, consulting, financial advisory, risk management, and tax services to selected clients.
Deloitte Analytics in Ireland deliver solutions to data, technology, and business challenges.
Our comprehensive approach to analytics is fuelled by our deep industry knowledge, broad functional experience, and mastery of technology.

Client Requirement

Dashboards are a valuable tool for analysing and understanding data. Deloitte Analytics have produced a variety of dashboards for specific clients and demonstration purposes. These dashboards are constructed using web technologies – HTML, CSS, RShiny and JavaScript.
Interactive JavaScript visualisations have become popular in recent years. There are a large number of new and interesting libraries, each with their own advantages and disadvantages.
The student is required to select a number of libraries and build a simple dashboard for each in order to compare them on ease of use, features, appearance, cost, interactivity, performance etc.
The student will be provided with a template dashboard and sample data, so that they may focus solely on implementing the JavaScript libraries. Example libraries include (among others): d3.js, dc.js, Google Charts, Highcharts, Polychart.js, amCharts, Chart.js, NVD3, gRaphaël, Vega, Rickshaw and xCharts.

What is involved for the student?

• The student will gain valuable experience working with and visualising data.
• Previous experience using HTML, CSS, and/or JavaScript will be beneficial but not essential. The student will be given one or more JavaScript tutorials if required.
• The student may decide which and how many libraries to test.
• They can also investigate new and interesting dashboard and visualisation methods.
B. **INTERIM REPORT**

**Project:** Investigation and Comparison of JavaScript Visualisation Libraries  
**Client:** Deloitte Analytics  
**Student:** Coleman Conmey  
**Supervisor:** Aideen Keaney

**Review of Background and Work to Date**

Deloitte Analytics have produced a variety of dashboards for specific clients for demonstration purposes. These dashboards are constructed using web technologies such as HTML, CSS, RShiny and JavaScript. JavaScript visualisations are used within dashboards to present data graphically using charts.

Interactive JavaScript visualisations allow the discovery and communication of meaningful patterns in data. As a result, they have become a popular resource for many organisations in recent years, leading to a considerable increase in the number of libraries being developed.

There are now a large number of new JavaScript visualisation libraries; as a result, it is difficult to determine which libraries should be implemented. The aim of this project is to select a number of JavaScript visualisation libraries and build a sample dashboard for each in order to compare them on factors such as ease of use, features, appearance, cost, interactivity and performance. The evaluation of many different libraries will help the client to choose the most suitable libraries available, and the evaluation framework can be used in the future to evaluate other libraries.

Initially, time was spent analysing the various libraries available and the necessary computer languages required to implement such libraries. The following work has been completed to date:

- Tested template dashboard and sample data provided by the client.
- Started to gain an understanding and knowledge of relevant web development programming languages.
- Built two sample dashboards, using Google Charts and NVD3 libraries.
- Met with the client(s) to discuss the scope of project and choice of potential libraries.
Terms of Reference

The terms of reference are to:

- Select a number of JavaScript visualisation libraries based on research of various ways to display data – charts, canvas, html5 etc.
- For each library, build a dashboard and test using sample data.
- Design a framework for library evaluation, including factors such as ease of use, features, interactivity and performance.
- Using the framework, evaluate and compare each dashboard.
- Following a literature review make recommendations on best practices for dashboard design and data visualisation.

Further Work

Research will be carried out to determine the minimum functionality required for the client from a JavaScript visualisation library. The choice of libraries to review will then have to be finalised over the Christmas period to allow the process of dashboard design to begin. The evaluation framework will also need to be defined.

A critique on best practices for analysing and displaying data will be implemented after evaluation of the chosen libraries.

Conclusions

Developing many different dashboards will be time consuming and require extensive rewriting between libraries. During testing of the dashboards, the evaluation framework will be continually updated based on findings from each library evaluation.

After evaluation of each individual JavaScript visualisation library, it will be possible to compare libraries to one another, highlighting the advantages and disadvantages of each library.
C. DASHBOARD REQUIREMENTS

This appendix gives an overview of the styling format and feature requirements for each dashboard design. They are outlined below in table C.1. The types of requirements differed in every aspect except for titles. Although not a default feature in many JVLs, it is considered a mandatory requirement for chart design. The data for both dashboards was gathered from a list of internet sources which can be referred to in the References. Chart colours are presented through their hexadecimal format. A set of Deloitte logos and hexadecimal colours where provided by the client for the Custom Dashboard design.

**Table C.1: Default and Custom Dashboard Requirements.**

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Default dashboard</th>
<th>Custom dashboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart types</td>
<td>Render a scatter plot, column chart and combo chart (bar and line).</td>
<td>Render an area chart with data points (spline chart), combo chart (column and line) and bar chart.</td>
</tr>
<tr>
<td>Chart background colour</td>
<td>Default provided by the JVL.</td>
<td>The dashboard chart areas are set to #565349. The charts produced by the JVL must be formatted to #565349, or there background colour must be set to null/false if possible.</td>
</tr>
<tr>
<td>Animation</td>
<td>Default provided by the JVL.</td>
<td>Should be animated if possible.</td>
</tr>
<tr>
<td>Titles/axes titles</td>
<td>Default provided by the JVL.</td>
<td>Font family: Myriad Pro</td>
</tr>
<tr>
<td></td>
<td>Font size: 14px/12px</td>
<td>Title colour: #FFFFFF</td>
</tr>
<tr>
<td></td>
<td>Axes title colour: #92D400</td>
<td></td>
</tr>
<tr>
<td>Labels</td>
<td>Default provided by the JVL.</td>
<td>Font family: Myriad Pro</td>
</tr>
<tr>
<td></td>
<td>Font size: 12px</td>
<td>Axes title colour: #92D400</td>
</tr>
<tr>
<td></td>
<td>xAxis colour: #FFFFFF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>yAxis colour: #EDFD56</td>
<td></td>
</tr>
<tr>
<td>Tooltips</td>
<td>Default provided by the JVL.</td>
<td>Font family: Myriad Pro</td>
</tr>
<tr>
<td></td>
<td>Font size: 10px</td>
<td>Text colour: #565349</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Background: #FFFFFF</td>
</tr>
<tr>
<td>Legends</td>
<td>Default provided by the JVL.</td>
<td>Font family: Myriad Pro</td>
</tr>
<tr>
<td></td>
<td>Font size: 13px</td>
<td>Data series one: #00A1DE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data series two: #565349</td>
</tr>
<tr>
<td>Prefixes/ Suffixes</td>
<td>None.</td>
<td>Applicable to labels, legends and tooltips.</td>
</tr>
<tr>
<td></td>
<td>Minutes: min</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Millions: MN</td>
<td></td>
</tr>
</tbody>
</table>
D. SCREENSHOTS

This appendix contains sample screenshots from the implementation of each JVL using the default and custom dashboards. It also displays screenshots of important features discussed throughout the project.

FIGURE D.1: JavaScript Interactivity Example with Dropdown box. The data presented in the chart is manipulated by the interaction between the user and the dropdown box.
FIGURE D.2: JavaScript Interactivity Example with Tabs. The data presented in the chart is manipulated by the interaction between the user and the tabs.
FIGURE D.3: Default Dashboard Implemented using the amCharts JVL.
FIGURE D.4: Default Dashboard Implemented using the CanvasJS JVL.


Smartphone Market Share
Apple vs. Samsung in the United States

iPhone, iPad and iPod Sales

Market Share (%)
FIGURE D.5: Default Dashboard Implemented using the Google Charts JVL.
FIGURE D.6: Default Dashboard implemented using the Highcharts JVL.
FIGURE D.7: Default Dashboard Implemented using the NVD3.js JVL.
FIGURE D.8: Custom Dashboard Implemented using the amCharts JVL.

Average Monthly Time Spent on Facebook (2013)

Unique U.S. visitors to Facebook and Google (2011-2014)

Top Users by Country (2012-2013)
FIGURE D.9: Custom Dashboard Implemented using the CanvasJS JVL.
FIGURE D.10: Custom Dashboard Implemented using the Google Charts JVL.

Average Monthly Time Spent on Facebook (2013):

- Jan 2013
- Oct 2012
- Jul 2012
- Apr 2012

Unique U.S. visitors to Facebook and Google (2011-2014):

- Jul 2011
- Jan 2012
- Jul 2012
- Jan 2013
- Jul 2013
- Jan 2014

Top Users by Country (2012-2013):

- United States
- Mexico
- United Kingdom
- Germany
- India
- France
- Italy
- Brazil
- Argentina
FIGURE D.11: Custom Dashboard Implemented using the Highcharts JVL.
FIGURE D.12: Custom Dashboard Implemented using the NVD3.js JVL.
FIGURE D.13: NVD3.js Implemented Without Cascading Style Sheets (CSS).

FIGURE D.14: Tooltip Example with Multiple Series on one Tooltip.

FIGURE D.15: Interactive Legend.
E. SAMPLE SOURCE CODE

This appendix contains the source code of the project. Since there are 10 HTML web pages, two JavaScript files and two CSS files within the project, amounting to several hundred lines of code, only sample code has been included.

E.1. File Directory

There is a file directory located in each dashboard HTML page. This references a number of relevant files (incl. jQuery) and the JVL source code. The user.js file represents the JavaScript file which contains the JSON formatted data, and the jQuery interactive elements. The style.css file contains the format code for the dashboard. A selection of library source codes is included here. N.B. only one JVL source code should be referenced normally.

```html
<!-- FILE DIRECTORY -->

<!-- jQuery 1.11 downloaded in March 2014 -->
<script type="text/javascript" src="common/jquery.js"></script>

<!-- Script with data and jQuery functions -->
<script type="text/javascript" src="common/user.js"></script>

<!-- Dashboard style -->
<link type="text/css" rel="stylesheet" href="common/style.css"></link>

<!-- JavaScript libraries -->
<!-- amCharts downloaded source code -->
<script language="javascript" type="text/javascript" src="amCharts/amcharts/amcharts.js"></script>
<script language="javascript" type="text/javascript" src="amCharts/amcharts/xy.js"></script>
<script language="javascript" type="text/javascript" src="amCharts/amcharts/serial.js"></script>

<!-- amCharts hyperlink source code -->
<script type="text/javascript" src="http://www.amcharts.com/lib/3/serial.js"></script>
<script type="text/javascript" src="http://www.amcharts.com/lib/3/xy.js"></script>

<!-- Custom style sheet -->
<link rel="stylesheet" type="text/css" href="amCharts/images/style.css">
```
E.2. Data Format

user.js is a JavaScript file that manages the JSON data and handles any events between the user and the dashboard through jQuery. The following file was referenced in every Default Dashboard HTML page. Similarly there was user.js JavaScript file for the Custom Dashboard with a different set of data.

```javascript
//Leading countries based on number of Facebook and Twitter users as of 2013 (in millions).
//JSON FORMAT
var getBarData = (function() {
    var data = {
        "Facebook": [
            {"x":"United States", "y":146.8},
            {"x":"Mexico", "y":35.7},
            {"x":"United Kingdom", "y":29.9},
            {"x":"Germany", "y":22.1},
            {"x":"India", "y":114.8},
            {"x":"France", "y":22},
            {"x":"Italy", "y":18},
            {"x":"Brazil", "y":69},
            {"x":"Argentina", "y":16.3}],
        "Twitter": [
            {"x":"United States", "y":22.9},
            {"x":"Mexico", "y":11.7},
            {"x":"United Kingdom", "y":6.6},
            {"x":"Germany", "y":2.4},
            {"x":"India", "y":33},
            {"x":"France", "y":2.2},
            {"x":"Italy", "y":3.1},
            {"x":"Brazil", "y":19.6},
            {"x":"Argentina", "y":6.3}],
    }
    return function(bar_data) {
        return data[bar_data];
    }
})();
```
E.3. Interactions Code

jQuery was used to manage interactions between the user and tabs or dropdown boxes. jQuery is indicated by the dollar sign.

```javascript
function switchTabs(tab, tabs, list) {
    var tab = $(tab);

    // Only do something if the user has selected an inactive tab.
    if (!tab.hasClass('tab-on')) {
        // Get currently active tab.
        var current = $(tabs + ' li.tab-on');

        // Remove currently active tab.
        current.removeClass('tab-on');

        // Highlight current tab.
        tab.addClass('tab-on');

        // Pull out the index of the clicked tab and return that element in the list.
        return list[tab.index()];
    } else {
        return list[tab.index()];
    }
}

$(document).ready(function()
{
    $('#line-choices li').click(function() {
        var line_choices = switchTabs(this, '#line-choices', ['Desktop', 'Mobile']);
        renderLine(line_choices);
    });
});
```
E.4. **JavaScript/jQuery Functions for Manipulation of Data**

A number of JavaScript and jQuery functions were created and used to transform the JSON data into the right format depending on the requirements of the JVL.

```javascript
$(‘#bar-choices li’).click(function() {
    var bar_choices = switchTabs(this, ’#bar-choices’, ['Facebook', ’Twitter’]);
    renderBar(bar_choices);
});

// First time the page loads the charts need to be all rendered with defaults.
renderLine(”Desktop”);
renderBar(”Facebook”);
renderCombo();
});
```

/* Canvas JS */

/* Function to replace ”x” field name with ”label” string, bar, column, pie charts etc only recognise categories with the field name of ”label” */

function changeFieldName(data){
    for(var i = 0; i < data.length; i++){
        if(data[i].hasOwnProperty(”x”)){
            data[i][”label”] = data[i][”x”];
            delete data[i][”x”];
        }
    }
}

/* Google Charts */

/* JSON to DataTable */
function JSONtoDataTable(data) {
  var x = new google.visualization.DataTable();

  x.addColumn('date', 'Year');
  x.addColumn('number', 'Sales');

  for (var i=0; i<data.length;i++){
    x.addRows([[new Date(data[i].x.substring(0,4),data[i].x.substring(5,7)-1,data[i].x.substring(8,10)) , data[i].y]]);
  }

  return x;
}

/* Highcharts */

//Convert data into 2D array */

function convertData(Data) {
  var data  = Data;
  var x = new Array(data.length);

  for (var i=0;i<data.length;i++){
    x[i] = new Array(2);
  }

  for (var i=0;i<data.length;i++){
    x[i][0] = Date.UTC(data[i].x.substring(0,4), data[i].x.substring(5,7)-1, data[i].x.substring(8,10));
  }

  for (var i=0;i<data.length;i++){
    x[i][1] = data[i].y;
  }

  return x;
}
E.5. **JavaScript Implementation**

This Appendix gives an example of a JVLs API format for implementing its JavaScript library. The provided example is xCharts implemented on the Custom Dashboard.

```javascript
function renderBar() {

    /* JSON TO DataTable*/

    function makeArray(data) {
        var x = new google.visualization.DataTable();
        x.addColumn('string', 'Year');
        x.addColumn('number', 'Sales');
        for (var i=0; i<data.length;i++) {
            x.addRows([[data[i].x, data[i].y]]);
        }
        return x;
    }

    var rowData1 = getBarData('iPod');
    var rowData2 = getBarData('iPhone');
    var rowData3 = getBarData('iPad');

    var x = new google.visualization.DataTable();
    var y = new google.visualization.DataTable();
    var z = new google.visualization.DataTable();

    x = makeArray(rowData1);
    y = makeArray(rowData2);
    z = makeArray(rowData3);

    var data = [];
    data[0] = x;
    data[1] = y;
    data[2] = z;

    var options = {
        title: 'iPhone, iPad and iPod Sales',
        vAxis: {
            title: 'Sales (in million units')
        },
        animation: {
            duration: 1000,
            easing: 'out',
        }
    };
```
var chart = new google.visualization.ColumnChart(document.getElementById('bar'));

function drawChart(value) {
    chart.draw(data[value], options);
}

// Initialise chart with first values
    drawChart(0);

$('#dropDown').change(function() {
    drawChart($('#dropDown option:selected').val());
});

// Code to render bar chart to 'bar' div.

} // Initialise chart with first values
    drawChart(0);

$('#dropDown').change(function() {
    drawChart($('#dropDown option:selected').val());
});

// Code to render bar chart to 'bar' div.
}
F. EVALUATION FINDINGS

This appendix outlines the findings of the evaluation process for each JVL. The results are categorized under five main sections.

- Appendix F.1 to F.2 documents the findings after evaluating the amCharts JVL.
- Appendix F.2 to F.3 documents the findings after evaluating the CanvasJS JVL.
- Appendix F.4 to F.5 documents the findings after evaluating the Google Charts JVL.
- Appendix F.5 to F.6 documents the findings after evaluating the Highcharts JVL.
- Appendix F.7 to F.8 documents the findings after evaluating the NVD3.js JVL.

The proceeding tables outline the advantages and disadvantages of the various JVLs evaluated as desired by the client. The charts evaluated are; amCharts, CanvasJS, Google Charts, Highcharts and NVD3.js.

F.1. amCharts Evaluation Table

<table>
<thead>
<tr>
<th>Library Source Code</th>
<th>JavaScript API Format</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Hyperlink</td>
<td>✓ Easy</td>
<td>✓ JSON</td>
</tr>
<tr>
<td>✓ Downloadable</td>
<td>✓ Moderate</td>
<td>❑ Two-Dimensional Array</td>
</tr>
<tr>
<td>❑ CSS</td>
<td>❑ Hard</td>
<td>❑ DataTable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Axes</th>
<th>Titles/Labels</th>
<th>Tooltips</th>
<th>Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Range</td>
<td>✓ Format titles</td>
<td>✓ Format tooltip</td>
<td>❑ Static</td>
</tr>
<tr>
<td>✓ Time series</td>
<td>✓ Format labels</td>
<td>✓ Format text</td>
<td>✓ Interactive</td>
</tr>
<tr>
<td>✓ Gridlines</td>
<td>✓ Add prefix/suffix to labels</td>
<td>❑ Manipulate data</td>
<td>✓ Position</td>
</tr>
<tr>
<td></td>
<td>✓ Format time series labels</td>
<td>❑ HTML</td>
<td>✓ Format</td>
</tr>
<tr>
<td></td>
<td>✓ Rotate labels¹</td>
<td>❑ Multiple Series</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓ Limit labels²</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default Appearance</th>
<th>Custom Appearance</th>
<th>Animation</th>
<th>Chart Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Effective axes scaling</td>
<td>❑ Excellent</td>
<td>❑ Flow</td>
<td>✓ SVG</td>
</tr>
<tr>
<td>❑ Legend</td>
<td>✓ Good³</td>
<td>❑ Bounce</td>
<td>✓ VML</td>
</tr>
<tr>
<td>✓ Tooltips</td>
<td>❑ Poor</td>
<td>❑ Transition</td>
<td>✓ Resizable</td>
</tr>
<tr>
<td>❑ Animation</td>
<td></td>
<td></td>
<td>❑ Rendering time</td>
</tr>
<tr>
<td>✓ Issues³</td>
<td></td>
<td></td>
<td>420 milliseconds</td>
</tr>
</tbody>
</table>
### Comments

1. Only the horizontal axis can be rotated. If you set this for the vertical axis, the setting will be ignored.
2. It is possible to limit the number of labels shown. Although, could not expand the category axis for the custom dashboard to show all labels (see Appendix D, p D.8).
3. By default, even barcharts aren't filled in. The simplest and most primitive charting library encountered so far, but has allot of potential to create impressive charts.
   The titles overlap the chart area if they are too large.
4. The animation sequence is impressive.
5. amCharts use JSFiddle as part of their demos provided on their website.
7. Use ‘My Tickets’ section to put in direct support requests to amCharts staff. This section can be used by any amCharts user. Of course, paid customers with active Priority Support contracts will be served first.
8. Single website application: $140.
   For multiple websites: $700.
   Distribution purposes: $1400.
   Web link to amCharts.com is placed on charts if you don’t have a license.

### F.2 CanvasJS

<table>
<thead>
<tr>
<th>Library Source Code</th>
<th>JavaScript API Format</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperlink</td>
<td>Easy</td>
<td>JSON</td>
</tr>
<tr>
<td>Downloadable</td>
<td>Moderate</td>
<td>Two-Dimensional Array</td>
</tr>
<tr>
<td>CSS</td>
<td>Hard</td>
<td>DataTable</td>
</tr>
</tbody>
</table>
Axes

- Range
- Time series
- Gridlines

Titles/Labels

- Format titles
- Format labels
- Add prefix/suffix to labels
- Format time series labels
- Rotate labels
- Limit labels

Tooltips

- Format tooltip
- Format text
- Manipulate data
- HTML
- Multiple series

Legend

- Static
- Interactive
- Position
- Format

Default Appearance

- Effective axes scaling
- Legend
- Tooltips
- Animation
- Issues

Custom Appearance

- Excellent
- Good
- Poor

Animation

- Flow
- Bounce
- Transition

Chart Rendering

- SVG
- VML
- Resizable
- Rendering time
  - 20 milliseconds

Documentation

- JavaScript API documentation
- Tutorials
- Demos

Community Support

- Forums
- GitHub
- JSFiddle

Personalised Support

- Email
- FAQ
- Features request
- Bug fixes
- Other

Licensing

- Free for all usage
- Free for non commercial usage

Comments

1. Has built in dynamic functions to deal with this. As a result it is easy to implement.
2. Chart titles scale to the size of the chart area, making for some unnecessarily large titles. JVL struggles to render a combo bar and line chart (see Appendix D, p. D.4).
3. Very impressive. Although scale for combo chart is poor. Animation is only available for pie and donut charts.
4. Features and bugs are resolved through a peronlaised form on the CanvasJS website.
5. Single website license: $79.
   - Team license: $700. With priority support: $1299.
   - Unlimited developer license: $1999.
   - Web link to CanvasJS.com is placed on charts if you don’t have a license.
6. Priority support includes: unlimited email based for one year; 24-hour turnaround time; hot fixes; and early access to to upcoming features (upon request).

Bugs:

1. Scatter plot in default dashboard does not load on action. When an interaction is made post, the scatter plot appears.
# F.3. Google Charts

<table>
<thead>
<tr>
<th>Library Source Code</th>
<th>JavaScript API Format</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Hyperlink</td>
<td>☐ Easy</td>
<td>☑ JSON</td>
</tr>
<tr>
<td>☐ Downloadable</td>
<td>✔ Moderate¹</td>
<td>☑ Two-Dimensional Array</td>
</tr>
<tr>
<td>☐ CSS</td>
<td>☐ Hard</td>
<td>✔ DataTable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Axes</th>
<th>Titles/Labels</th>
<th>Tooltips</th>
<th>Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Range</td>
<td>✔ Format titles²</td>
<td>✔ Format tooltip⁵</td>
<td>✔ Static</td>
</tr>
<tr>
<td>✔ Time series</td>
<td>✔ Format labels</td>
<td>✔ Format text</td>
<td>☐ Interactive</td>
</tr>
<tr>
<td>✔ Gridlines</td>
<td>✔ Add prefix/suffix to labels</td>
<td>☐ Manipulate data</td>
<td>✔ Position</td>
</tr>
<tr>
<td></td>
<td>✔ Format time series labels</td>
<td>✔ HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✔ Rotate labels³</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✔ Limit labels⁴</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default Appearance</th>
<th>Custom Appearance</th>
<th>Animation</th>
<th>Chart Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Effective axes scaling</td>
<td>✔ Excellent⁸</td>
<td>☐ Flow</td>
<td>✔ SVG</td>
</tr>
<tr>
<td>✔ Legend⁶</td>
<td>❑ Good</td>
<td>☐ Bounce</td>
<td>✔ VML</td>
</tr>
<tr>
<td>✔ Tooltips</td>
<td>❑ Poor</td>
<td>✔ Transition⁹</td>
<td>✔ Resizable</td>
</tr>
<tr>
<td>☐ Animation</td>
<td></td>
<td></td>
<td>☐ Rendering time¹⁰ 160 milliseconds</td>
</tr>
<tr>
<td>❑ Issues⁷</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Documentation</th>
<th>Community Support</th>
<th>Personalised Support</th>
<th>Licensing</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ JavaScript API documentation</td>
<td>✔ Forums¹²</td>
<td>✔ Email</td>
<td>✔ Free for all usage¹⁵</td>
</tr>
<tr>
<td>✔ Tutorials¹¹</td>
<td>❑ GitHub</td>
<td>✔ FAQ</td>
<td>❑ Free for non commercial usage</td>
</tr>
<tr>
<td>❑ Demos</td>
<td>❑ JSFiddle</td>
<td>✔ Features request¹³</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>✔ Bug fixes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>✔ Other¹⁴</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. The API format is easy to understand, but having to use DataTables made the process allot more difficult when compared with other charting libraries.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Titles cannot be centred.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Only works for the horizontal axis.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Google Charts cannot limit the number dates shown on a time series. It defaults to five or less. In order to work with time series labels effectively, the data must be presented in ‘string’ format in the DataTable, instead of ‘date’.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Only through the use of HTML or the addition of another row to the DataTable. It is not possible to manipulate the data shown in the tooltip by adding suffixes or prefixes.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Legends run out of chart area if they are too long (see Appendix D.5).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Labels run out of the chart area if they are too long.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Legends take their names from corresponding column id’s in the DataTable. Because of this, jQuery cannot be used to manipulate the text that appears in the legend when a user switches between tabs or dropdown boxes. Can change the size of the chart area to maximise the use of available space. This ability can be used to solve the issues with legends and labels. Can also change the direction of a series.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Transition Animation takes away from the initial visual experience that occurs with other JVLs. Trying to integrate animation into Google Charts is extremely tedious (see code in Appendix E.6). A DataTable made the manipulation of datasets on one chart awkward as all data must be combined into one DataTable to allow for the transition sequence to occur.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Tutorials are simple but leave allot of the customisation process out. Developer is expected to refer to the API reference manual for these aspects.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. NB: Rendering was performed without an animation sequence.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Forums are presented in the form of Google Groups.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Provides a catalogue for bug reports and feature requests.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. Google’s Hall of Fame where users can submit their developed visualisations posted on the web page.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. Although it is free for all usage, it is not open source. It is important to make the distinction between the two. For an explanation of open source see Appendix H.1.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bugs:

1. Cannot use “google” as a variable name when implementing Google Charts.
# F.4. Highcharts

<table>
<thead>
<tr>
<th>Library Source Code</th>
<th>JavaScript API Format</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>☑ Hyperlink</td>
<td>☑ Easy</td>
<td>☑ JSON</td>
</tr>
<tr>
<td>☑ Downloadable</td>
<td>☑ Moderate</td>
<td>☑ Two-Dimensional Array</td>
</tr>
<tr>
<td>☑ CSS</td>
<td>☑ Hard</td>
<td>☑ DataTable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Axes</th>
<th>Titles/Labels</th>
<th>Tooltips</th>
<th>Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>☑ Range</td>
<td>☑ Format titles</td>
<td>☑ Format tooltip</td>
<td>☑ Static</td>
</tr>
<tr>
<td>☑ Time series</td>
<td>☑ Format labels</td>
<td>☑ Format text</td>
<td>☑ Interactive</td>
</tr>
<tr>
<td>☑ Gridlines</td>
<td>☑ Add prefix/suffix to labels</td>
<td>☑ Manipulate data</td>
<td>☑ Position</td>
</tr>
<tr>
<td></td>
<td>☑ Format time series labels</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>☑ Rotate labels</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>☑ Limit labels</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default Appearance</th>
<th>Custom Appearance</th>
<th>Animation</th>
<th>Chart Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>☑ Effective axes scaling</td>
<td>☑ Excellent</td>
<td>☑ Flow</td>
<td>☑ SVG</td>
</tr>
<tr>
<td>☑ Legend</td>
<td>☑ Good</td>
<td>☑ Bounce</td>
<td>☑ VML</td>
</tr>
<tr>
<td>☑ Tooltips</td>
<td>☑ Poor</td>
<td>☑ Transition</td>
<td>☑ Resizable</td>
</tr>
<tr>
<td>☑ Animation</td>
<td></td>
<td></td>
<td>☑ Rendering time</td>
</tr>
<tr>
<td>☑ Issues</td>
<td></td>
<td></td>
<td>118 milliseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Documentation</th>
<th>Community Support</th>
<th>Personalised Support</th>
<th>Licensing</th>
</tr>
</thead>
<tbody>
<tr>
<td>☑ JavaScript API documentation</td>
<td>☑ Forums</td>
<td>☑ Email</td>
<td>☑ Free for all usage</td>
</tr>
<tr>
<td>☑ Tutorials</td>
<td>☑ GitHub</td>
<td>☑ FAQ</td>
<td>☑ Free for non commercial usage</td>
</tr>
<tr>
<td>☑ Demos</td>
<td>☑ JSFiddle</td>
<td>☑ Features request</td>
<td></td>
</tr>
</tbody>
</table>
## Comments

1. Highcharts was by far the best and easiest JVL to implement to the Default Dashboard. The number of features provided by default matched that of the Custom Dashboard.
2. As well as the JavaScript API, a general documentation is provided.
3. Demos are performed through JSFiddle.
4. ‘GitHub Issues’ is used to report bugs.
5. Features are requested through ‘user voice’, which is similar to stack overflow. People can post a feature, and depending on its popularity it will get votes to be implemented.
   Five developer license: $1500. With premium support: $2250.
   Ten developer license: $2500. With premium support: $3600
   Web link to Highcharts.com is placed on charts if you don’t have a license.

### F.5. NVD3.js

<table>
<thead>
<tr>
<th>Library Code</th>
<th>Source Code</th>
<th>JavaScript API Format</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hyperlink</td>
<td>Easy</td>
<td>JSON</td>
</tr>
<tr>
<td></td>
<td>Downloadable</td>
<td>Moderate</td>
<td>Two-Dimensional Array</td>
</tr>
<tr>
<td></td>
<td>CSS</td>
<td>Hard</td>
<td>DataTable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Axes</th>
<th>Titles/Labels</th>
<th>Tooltips</th>
<th>Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Format titles</td>
<td>Format tooltip</td>
<td>Static</td>
</tr>
<tr>
<td>Time series</td>
<td>Format labels</td>
<td>Format text</td>
<td>Interactive</td>
</tr>
<tr>
<td>Gridlines</td>
<td>Add prefix/suffix to labels</td>
<td>Manipulate data</td>
<td>Position</td>
</tr>
<tr>
<td></td>
<td>Format time series labels</td>
<td>HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rotate labels</td>
<td>Multiple series</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Limit labels</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default Appearance</th>
<th>Custom Appearance</th>
<th>Animation</th>
<th>Chart Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effective axes scaling</td>
<td>Excellent</td>
<td>Flow</td>
<td>SVG</td>
</tr>
<tr>
<td>Legend</td>
<td>Good</td>
<td>Bounce</td>
<td>VML</td>
</tr>
<tr>
<td>Tooltips</td>
<td>Poor</td>
<td>Transition</td>
<td>Resizable</td>
</tr>
<tr>
<td>Animation</td>
<td></td>
<td></td>
<td>Rendering time</td>
</tr>
<tr>
<td>Issues</td>
<td></td>
<td></td>
<td>300 milliseconds</td>
</tr>
</tbody>
</table>
### Documentation
- JavaScript API documentation
- Tutorials
- Demos
- Forums
- GitHub
- JSFiddle

### Community Support
- Email
- FAQ
- Features request
- Bug fixes
- Other

### Personalised Support
- Email
- FAQ
- Features request
- Bug fixes
- Other

### Licensing
- Free for all usage
- Free for non-commercial usage

### Comments
1. The JavaScript API functions for NVD.js is based off D3, which is very difficult to work with because many of the functions for customising the chart are simply not apart of the libraries arsenal. It takes a considerable amount of effort to make any changes through D3.
2. Titles overlap legends (see Appendix D). Unable to do a combo bar and line chart, and for the line and bar chart it places two y-axes, which in the end effects the information being received by the user.
3. This library failed with respects to the Custom Dashboard. It lacks allot of customising capabilities, which are available through D3 but would take allot of effort to implement (see Appendix D).
4. Animation between the charts is not synchronised, as a result the visualisation produced is disrupted and disjointed.
5. Charts are primarily rendered in SVG so NVD3.js cannot be used for older browsers that do not support SVG. Every chart requires svg tags or else the chart will not render.
6. The website is poor and rarely updated. The architect of the library states this in the repository on Github where the most up to date examples are kept. Although the examples here are just examples of text with no visualisations, therefore it is a very difficult library to work with.

### Bugs:
1. The legend is only somewhat interactive, as sometimes it doesn’t allow the user to remove the dataset from the chart unless an interaction has been made between the user and the data. Furthermore, the Custom Dashboard ran into difficulties when trying to implement jQuery through tabs with NVD3.js. It would work initially and then crash.
G. TECHNICAL REPORT

This appendix details the technical environment of the project.

G.1. Client-Side Scripting Languages

Client-side scripting refers to the class of computer programs on the web that are executed client-side, by the user’s web browser, instead of server-side (on the web server). JavaScript represents the client-side scripting language used in all JavaScript Visualisation Libraries. JavaScript code is written into a HTML page. The language is supported by all major web browsers. JavaScript allows interactions between the web page and users creating a dynamic web environment. For example a web page can have different and changing content depending on user input, such as tab and dropdown box selections.

The syntax of JavaScript can be quite difficult to work with. As such, research revealed several JavaScript libraries created with the intention of simplifying this syntax. This eventually led to the usage of JQuery. JQuery is a JavaScript library. Once the library is referenced in the header of any HTML page, JQuery syntax is then recognized. The purpose of jQuery is to make it much easier to use JavaScript on your website. jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code. JQuery also simplifies a lot of the complicated things from JavaScript, like DOM manipulation.

G.2. Dynamic HTML

Dynamic HTML, or DHTML, is an umbrella term for a collection of technologies used together to create interactive and animated web sites by using a combination of a static markup language (such as HTML), a client-side scripting language (such as JavaScript), a presentation definition language (such as CSS), and the Document Object Model (DOM).

G.3. Document Object Model

The Document Object Model (DOM) is the model that describes how all elements in an HTML page, like input fields, images, paragraphs etc., are related to the topmost structure: the document itself. By calling the element by its proper DOM name, we can influence it.

G.4. CSS

Cascading Style Sheets (CSS) is used to format the appearance and layout of content displayed by XHTML. There is no viable alternative to CSS since it is widely supported by all major web browsers. CSS is most commonly stored in an external .css file. It is then referenced at the top of each page as the page loads.

The main difficulty with CSS is web browser support. While it is supported by all major browsers in some fashion, the manner in which it is interpreted is often quite different from one browser to another. All modern browsers (Internet Explorer10, Mozilla Firefox 27, Chrome 33 etc.) follow CSS standards as defined by the World Wide Web Consortium (W3C). CSS in such browsers performs consistently.
G.5. **SVG**

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics that has support for interactivity and animation. The SVG specification is an open standard developed by the World Wide Web Consortium (W3C) since 1999.

SVG is used to define vector-based graphics for the Web. SVG defines the graphics in XML format. SVG graphics do not lose any quality if they are zoomed or resized. Every element and every attribute in SVG files can be animated. SVG is a W3C recommendation. SVG integrates with other W3C standards such as the DOM and XSL.

G.6. **Minification**

Minification in computer programming languages and especially JavaScript, is the process of removing all unnecessary characters from source code without changing its functionality. These unnecessary characters usually include white space characters, new line characters, comments, and sometimes block delimiters, which are used to add readability to the code but are not required for it to execute.

Minified source code is especially useful for JavaScript because it reduces the amount of data that needs to be transferred onto the internet. Most JVLs provide both minified and non minified source code. It is important to have non minified source code so users can develop the libraries. Otherwise, they are unreadable in minified form.

G.7. **D3**

D3.js is a JVL that uses digital data to drive the creation and control of dynamic and interactive graphical forms which run in web browsers. It is a tool for data visualization in W3C-compliant computing, making use of the widely implemented Scalable Vector Graphics (SVG), JavaScript, HTML5, and Cascading Style Sheets (CSS3) standards.

In contrast to the other libraries studied in this project, D3 allows for far superior control over the final visualisation. The implementation and development of D3 is at a far lower level of granularity than any of the other libraries. Such that, many open source libraries are built off D3.js, using it powerful visualisations for re-usable charts (e.g. NVD3.js).
H. **GLOSSARY AND RELEVANT TERMS**

This appendix contains a glossary and descriptions of various terms referred to during the report.

**H.1. Glossary**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Performance Indicators</td>
<td>A Key Performance Indicator (KPI) is a measureable value that is used to evaluate an organisation's success, or to evaluate the success of a particular activity within an organisation.</td>
</tr>
<tr>
<td>Open source</td>
<td>Denoting software that for which the original source code is made freely available and may be redistributed and modified.</td>
</tr>
<tr>
<td>Proprietary owned</td>
<td>Proprietary owned software is software that is licensed under exclusive legal right of the copyright holder with the intent that the licensee is given the right to use the software only under certain conditions, and restricted from other uses, such as modification, sharing, studying, redistribution, or reverse engineering.</td>
</tr>
</tbody>
</table>

**H.2. Features**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legend</td>
<td>A legend (or Key) is an explanation section located on a chart to illustrate information in the chart. It may be made up of symbols and colours that are meant to represent different things in the chart.</td>
</tr>
<tr>
<td>Tooltip</td>
<td>A tooltip is a common graphical user interface element. It is used in conjunction with a cursor. A message appears when a cursor is positioned over a data point in a chart, containing relevant information about that data point.</td>
</tr>
<tr>
<td>Drilldown</td>
<td>Drilldowns within charts involves the analysis of a specific dataset more closely. To drill down on a column chart means to select a column to inspect it more closely. This column would then be broken up into another series of more specific columns.</td>
</tr>
</tbody>
</table>
I. USER GUIDE

This appendix gives a brief guide to using the dashboards which have been created in the project. Accompanying this document is a CD with all ten dashboards stored on it. Also stored on the CD is an index.html which can be used to access each dashboard.

```
index.html

Default Dashboards
  amCharts Default Dashboard
  CanvasJS Default Dashboard
  Google Charts Default Dashboard
  Highcharts Default Dashboard
  NVD3.js Default Dashboard

Custom Dashboards
  amCharts Custom Dashboard
  CanvasJS Custom Dashboard
  Google Charts Custom Dashboard
  Highcharts Custom Dashboard
  NVD3.js Custom Dashboard
```

From here, each dashboard can be accessed, and rendering from the necessary source files that are also stored on the CD. To access each HTML page individually follow these steps:

1. Click on Dashboards folder.
2. Choose either DefaultDashboard folder or CustomDashboard folder.
3. Select a HTML page to run.

Within either folder is a repository of JVLs libraries aptly named. Each folder is necessary to run the dashboard. Unless the platform you are working off has the internet, then Google Charts, amCharts and Highcharts will work.
REFERENCES

JavaScript Visualisation Information Sources

http://kraskniga.blogspot.ie/2012/06/comparison-of-javascript-data.html
http://www.creativeblog.com/design-tools/data-visualization-712402
http://www.fusioncharts.com/javascript-charting-comparison/

Open Data Sources: