D3 and Information Visualization Toolkits

Bechtold Oskar, Bobic Aleksandar, Jente Daniel, Schiffer Verena

Institute of Interactive Systems and Data Science (ISDS),
Graz University of Technology
A-8010 Graz, Austria

706.057 Information Visualisation SS 2018
Graz University of Technology

18 May 2018

Abstract

D3 is one of the most popular JavaScript libraries for information visualization and is mostly used in the creation of charts. Since D3 itself consists of several low level modules the implementation often requires vast amounts of code. Hence, several toolkits were developed on top of D3 to achieve better code readability and better ease of use while providing a variety of chart templates. In this survey, D3 and a selection of toolkits is presented that are later on compared against each other. In the end, different toolkits are recommend depending on the required task.

© Copyright 2018 by the author(s), except as otherwise noted.

This work is placed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

In recent years D3 has become one of the most popular libraries for information visualization with JavaScript in the Browser. While D3 is usually used to create charts, the library itself consists of several lower level modules with the goal of efficiently updating documents such as HTML files. The recent upgrade from version 4 to version 5 allows the usage of state-of-the-art JavaScript functionality such a Promises and async/await methods. Hence, D3 is a very powerful tool in today's web development. As the functionality of requested features increases, it gets harder to maintain readable code in D3, especially when creating basic chart types. To solve this problem different toolkits have been introduced which are based on D3. All of these toolkits highly vary in their functionality and complexity, as well as their online documentation. In this paper different aspects of D3 and information visualization toolkits are discussed:

- In Chapter 2 general information about D3 and its functionality is given. It is not only described how D3 works (Section 2.1) but also examples are presented (Section 2.2). Furthermore, new features of different versions of D3 are outlined and described (Section 2.3 and 2.4).
- Different toolkits using D3 are described in Chapter 3. A simple bar chart is implemented and the code as well as the graphs are compared to each other.
- Since all toolkits highly vary in their functionality, the main features are outlined in Chapter 4. In addition, toolkit recommendations for different applications are provided.

In the end, a broad overview of D3 and the different toolkits will be achieved. A developer is then able to estimate which toolkit is the right choice for a given task.

Chapter 2

D3

Data-Driven Documents, in short D3 or d3.js, is a JavaScript library utilizing HTML, SVG and CSS to manipulate the DOM (Document Object Model) in a data driven approach. D3 is currently on version 5 and is licensed under the BSD 3-Clause License. It was developed by the Stanford Visualization Group, specifically by Bostock et al. [2011], in 2011. Currently D3 follows a general update pattern which aims for more efficiency in comparison to native approaches such as directly using the W3C DOM API. This can already be seen for selections which is further depicted in the Listing 2.1.. While the approach with the W3C DOM API requires a for-loop to update all the elements, D3 can select elements with less code and chain several commands affecting that current selection.

The goal of D3 is to provide the necessary means in manipulating documents efficiently while still ensuring the required flexibility to solve arbitrary tasks. Rather then offering every feature out of the box, D3 focuses on supplying the demanded components to tailor and customize individual solutions. These components are further explained in Section 2.1. While it is possible to solve a broad range of challenges with D3, the amount of code to solve such tasks is still quite large. Especially for information visualization there are several toolkits that are build on top of D3 (see Chapter 3) to reduce the overall workload. Those toolkits take advantage of D3 or its low level modules and their ability to manipulate the DOM efficiently. The toolkits offer broad feature sets such as implementing specific kinds of charts or graphs while using less code than D3 for the implementation. However, the ability of the toolkits to adapt to unique requirements might often be fairly restricted in comparison to the native D3 approach.

```
1 // W3C DOM selection
2 var paragraphs = document.getElementsByTagName("p");
3 for (var i = 0; i < paragraphs.length; i++) {
4   var paragraph = paragraphs.item(i);
5   paragraph.style.setProperty("color", "white", null);
6 }
7
8 // D3 Selection
9 d3.selectAll("p").style("color", "white");
```

Listing 2.1: A for-loop is required to change the properties of all the individual elements of a W3C DOM selection. D3 only requires only one command. Changes to the D3-selection can be achieved by chaining functions after the select-command. Taken from Bostock [2018b].

2.1 Update Pattern

D3 aims for efficiently updating the DOM in a data driven approach. In D3, this is usually done by first selecting either an HTML- or SVG-element of the DOM and announcing to D3 the data set to use. Data for which the required DOM-elements still have to be created can then be accessed with the enter() method. In the case there are already existing DOM-elements and there are not enough data to populate those, those remaining DOM-elements can be accessed with the exit() method. With the help of the enter() and exit() methods it is now possible to generate or remove DOM elements as needed and accordingly style those to the user's wishes. Among the community this principle is usually understood as the `General Update Pattern` (see Bostock [2018c]) of D3 though it is not officially stated as such on the official website.

2.1.1 Selections

Changing a document, means making changes to the DOM. However, this requires finding certain elements and selecting them. Selections are one of the key features of D3. There are individual selections and also group selections. Independent from the selection type, changes to a selection can be applied by chaining functions to the results of the previous function as further depicted in Listing 2.2.

```
1 // individual selection
2 d3.select("body").style("background-color", "black");
3
4 // group selection
5 d3.selectAll("p").style("color", "white");
```

Listing 2.2: The actual selection happens in the select() or selectAll()-functions. The used style() methods change here a specified property of the selection to a static value. Taken from Bostock [2018b].

2.1.2 Applying Dynamic DOM Changes

While a lot of properties in a document are static and can be set directly, certain areas of the document such as charts should be populated based on some function or dynamic data. In case random colors should be assigned to the style of a selection, such a following function taken from Bostock [2018b] could be used:

```
1 d3.selectAll("p").style("color", function() {
2     return "hsl(" + Math.random() * 360 + ",100%,50%)";
3 });
```

The access to the index of an element in a selection can be quite useful for achieving certain effects such as oscillations or gradients. To generate such effects, an index can be processed with the modulo operation and the resulting value can then be mapped to a color value. The function that is executing such computations may have to contain the data element or index in the parameter list which is shown in the following example taken from Bostock [2018b]:

```
1 d3.selectAll("p").style("color", function(d, i) {
2   return i % 2 ? "#fff" : "#eee";
3 });
```

Figure 2.1: Enter and exit example for D3, using the code shown in Listing 2.3. Rectangles are already existing DOM-elements from the last iteration while circles are freshly created DOM-elements during the current iteration.

As already indicated such dynamic properties can also depend on input data. Assuming that there are as many DOM-elements in the selection as elements in the input data, such changes can directly depend on the data. For each DOM-element, the function accesses one data element and modifies the style according to the data. This can be seen in the following example from Bostock [2018b]:

```
1 d3.selectAll("p")
2   .data([4, 8, 15, 16, 23, 42])
3     .style("font-size", function(d) { return d + "px"; });
```

2.1.3 Enter and Exit

When a document is loaded for the first time, there is usually not the same amount of DOM-elements present as there is input data given. Most of the time this is also expected since the DOM-elements should be generated or destroyed dynamically rather than inserted manually. The approach is to provide the data to D3 after a selection was made. D3 detects if the generation or removal of DOM-elements is required based on the difference of DOM-elements and data elements. In the case more DOM-elements are needed, the data elements without a DOM-element can be accessed with the enter() method. The actual creation is triggered by the append() function. However, it is still necessary to inform those new DOM-elements how they should be styled according to their data. If there are instead too much DOM-elements, the DOM-elements without a data element can be addressed with the exit() function. The removal is executed with the remove() function. An example is given in Listing 2.3 and Figure 2.1 .

```
1  // Data
2  var gradient = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
3
4  // Create a row container in which the elements will be placed
5  var span = d3.select("body")
6   .append("span")
7   .style("display", "table");
8
9  // Update function
10 function update(data) {
11  var div = span.selectAll("div")
12    .data(data);
13
14  // Current DOM-elements are tagged as rectangles
15  div.attr("class", "rect");
16
17  // New DOM-elements are created for incoming data elements
18  div.enter().append("div")
19    .attr("class", "circle") // Those are tagged as circles
20    .merge(div)
21      .style("background-color", function(d, i) {
22        return "rgb(0, 100, " + d * 25 + ")";
23      });
24
25  // Leaving DOM-elements are removed
26  div.exit().remove();
27 }
28
29 // Call the update function in intervals
30 update(gradient);
31 d3.interval(function() {
32  var max = Math.floor(Math.random() * gradient.length);
33  update(gradient.slice(0, max));
34 }, 1500);
```

Listing 2.3: D3 `enter` and `exit` methods to automatically generate or destroy DOM-elments as needed. Here a sequence of rectangles and circles of random length is generated in time intervals. Rectangles represent the already existing DOM-elements and circles the freshly created ones. The `rect` and `circle` CSS classes are omitted for better readability. Inspired by the code from Bostock [2018c] though this example is almost completely rewritten.

```
1  // Data as Array
2  var data_ver_1 = [ 1, 3, 2, 4, 3, 5];
3
4  // Data as JSON
5  var data_ver_2 = [
6      {"name": "x1", "value": 1},
7      {"name": "x2", "value": 3},
8      {"name": "x3", "value": 2},
9      {"name": "x4", "value": 4},
10     {"name": "x5", "value": 3},
11     {"name": "x6", "value": 5}];
```

Listing 2.4: Data which are used in all bar charts during this survey.

2.2 Examples

2.2.1 Charts in General

D3 focuses on manipulating documents in a data driven approach as explained in Section 2.1. The goal of D3 is not to provide a vast library of different chart types, but rather the possibility to compose them individually from basic components. Thus it is possible to create lots of unique chart types such as:

- Sunburst
- Calendar Views
- Streamgraph
- Treemap
- Dendrogram
- and many more.

Those are however, just some examples. A more detailed preview can be seen on the homepage of D3.

2.2.2 D3 - Bar Chart

Throughout this survey the chart type of comparison is a bar chart. The data set is either defined as an array or JSON as shown in Listing 2.4. The only difference between the two versions are the missing names for the values in the array. In those cases the indices in the array are used as the names. The bar chart created with D3 in Figure 2.2 is generated from the JavaScript code in Listing 2.5.

```
var svg = d3.select("svg"),
 margin = {top: 20, right: 20, bottom: 30, left: 40},
 width = +svg.attr("width") - margin.left - margin.right,
 height = +svg.attr("height") - margin.top - margin.bottom;

var x = d3.scaleBand().rangeRound([0, width]).padding(0.1),
 y = d3.scaleLinear().rangeRound([height, 0]);

var g = svg.append("g")
 .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

var data = [ 1, 3, 2, 4, 3, 5];
x.domain(data.map(function(d, i) { return i; }));
y.domain([0, d3.max(data, function(d) { return d; })]);

g.append("g")
 .attr("transform", "translate(0," + height + ")")
 .call(d3.axisBottom(x));
g.append("g")
 .call(d3.axisLeft(y));
g.selectAll(".bar")
 .data(data)
 .enter().append("rect")
   .attr("class", "bar")
   .attr("x", function(d, i) { return x(i); })
   .attr("y", function(d) { return y(d); })
   .attr("width", x.bandwidth())
   .attr("height", function(d) { return height - y(d); });
```

Listing 2.5: D3 code for a simple bar chart. Inspired by the code from Bostock [2018a].

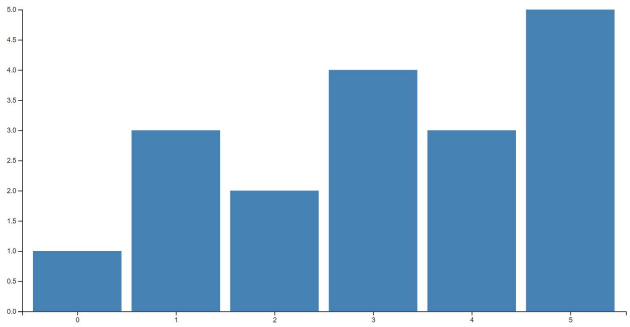

Figure 2.2: Simple bar chart created in D3, using the code shown in Listing 2.5.

2.3 What is New in D3 Version 4

D3v4 was a monumental update to the D3 library and brought many breaking changes to it, as it was a complete rewrite of version 3 [Bostock et al. 2018]. This also brought many back-compatibility issues, as the new version could not be used in projects which were using D3v3 without a substantial rewrite of the code. This chapter presents some of the larger changes and is based on the D3 change log.

2.3.1 Modules

One of the most notable changes introduced in this version of D3 was code modularization. The main idea behind modularization was the avoidance of cases where a developer would have to import the entire D3 library even if they do not use the majority of the features. Each module is a separate library and can be used independently of other modules. In case a user decides to import the default D3 library bundle they will receive approximately 30 default modules packed together. Each one of these small libraries is written as an ECMAScript (ES) 6 module. In addition, this influenced the symbol namespace in D3. Instead of using `d3.scale.linear` the user has now to use `d3.scaleLinear`. Furthermore, this affected the coding style of D3, as all ES6 modules are automatically written with strict mode enabled. Strict mode restrains the user from writing code that could cause future problems such as using a variable without declaring it. Finally, this also enables developers to easier understand a specific part of the D3 library and adopt that to their needs or even write their library or extension for D3.

2.3.2 ASCII Symbols

D3v4 uses only ASCII symbols for all of its variables in order to increase compatibility with more development environments. The previous version of D3 used special symbols for a more accurate mathematical representation (for example π). However, this caused some users to get JavaScript errors when using the not minified version of the library in wrong encoding.

2.3.3 Other Changes

There are many other changes not mentioned in the above sections which enable developers to do more with less code. Some of them will be described in the following section, however, in order to get the full grasp of these new features the reader is encouraged to read through the D3 changelog.

2.3.3.1 Axis Generation

First such change is the improvement in D3 axis generation which now requires less code and has a default styling. This modification means a developer can add an axis to their graph in no more than 1 line of code as seen in the following example: `d3.select('.axis-x').call(d3.axisTop(x));` The previous version of the library required the manual addition of styles and more lines of code for the actual generation of the axis (seen in Listing 2.6).

```
1 d3.select('.axis-x').call(
2     d3.svg.axis()
3         .scale(x)
4         .orientation('top'));
```

Listing 2.6: Simple axis example created in D3v3 without the needed styling. Inspired by a code sample from Bostock et al. [2018].

2.3.3.2 Color

D3 version 4 has an addition of a d3-color module which enables straightforward manipulation of color values. One of the notable new features is the possibility of adding opacity for colors in RBG and HSL space. Furthermore, it provides some simple methods for parsing color values into D3 color objects and to strings.

2.3.3.3 Easing

The easing has been overhauled by changing the way a developer defines what kind of easing they want. Before they would have to pass a string to an ease function and know the exact name of the easing they want which can be seen in the following example: `var e = d3.ease('elastic-out-in', 1.2);` Now they can call the symbol via the global d3 object which is displayed in the following code sample: `var e = d3.easeElastic.amplitude(1.2);` A lot of the easing symbols have an alias which defaults to -in-out in most cases. For example a user can write `d3.easeQuad` instead of `d3.easeQuadInOut`. In addition, there were some bug fixes and optimization's added to the easing methods.

2.4 What is New in D3 Version 5

Unlike version 4 of the library, D3v5 was only an incremental update to the already established product [Bostock et al. 2018]. It introduced one significant change which is the move from callbacks to promises and many smaller improvements and feature additions. This section will attempt to explain why the transition to promises is a significant one and briefly touch other changes to the library. Just like the previous Section 2.3, this one is also based on the D3 changelog.

2.4.1 Promises in D3

The addition of promises simplified the way developers handle the reading of data in D3. Until D3v5 they had to use callback functions for this action which could result in a situation called `callback hell`. This is the name for hard to read code which consists of multiple nested callback functions. Such a code sample can be seen in Listing 2.7.

```
randomFunction(param, (error, data) => {
    if (error) {
        throw error;
    }
    someOtherRandomFunc(param, data, (error, result) => {
        if (error) {
            throw error;
        }
        thirdRandomFunc(result.data, (error, data) => {
            if (error) {
                throw error;
            }
            // We must go deeper!
            lastFunc(data, (error, data) => {
                if (error) {
                    throw error;
                }
                // Do some additional stuff with the data
                console.log(data);
            });
        });
    });
});
```

Listing 2.7: Example of callback hell in JavaScript. The functions are made up as they do not matter for the illustration of this situation.

Promises help the developers avoid callback hell by enabling them to write code which does not need to be deeper than one level. Even though the syntax is different, the code still works in an asynchronous way just like the one which uses promises. An example of reading data from a file using promises can be seen in Listing 2.8.

```
d3.csv('data.csv')
    .then(data => {
        console.log(data);
    })
    .catch(error => {throw error;});
```

Listing 2.8: Example of reading a data file using promises in D3v5. Inspired by a code sample from [Bostock et al. 2018].

A significant advantage of supporting promises is that a developer can use the newer syntax for promises introduced in ES7 which uses the async/await pattern. Unlike in classic promise calls, the async/await pattern enables the developers to write the code in a way which mimics a synchronous syntax. The only difference is in the keyword async in front of the function which contains the asynchronous code, and the keyword await in front of every function which returns a promise. This means that there can be multiple sequential calls to certain functions which return a promise without the deep nesting of the then keywords. An example of reading the data of two files using the normal promise syntax can be seen in Listing 2.9. For comparison the same function can be seen in Listing 2.10 using the async/await syntax.

```
readData() {
    return d3.csv('data_block_1.csv')
        .then(firstBlock => {
            return Promise.all(firstBlock, d3.csv('data_data_2.csv'))
        })
        .then(blockArray => {
            return mergeBlocks(...blockArray);
        })
        .catch(error => {throw error;});
}
```

Listing 2.9: Example of reading two data files using the normal promise syntax in D3v5.

```
async readData() {
    try {
        const firstBlock = await d3.csv('data_block_1.csv');
        const secondBlock = await d3.csv('data_data_2.csv');
    } catch (error) {throw error;}

    return mergeBlocks(firstBlock, secondBlock);
}
```

Listing 2.10: Example of reading two data files using the async/await syntax in D3v5. It can be seen that this way of writing is easier to understand and write.

2.4.2 Other Changes

In addition to moving from callbacks to promises, D3 version 5 offers further incremental improvements, bug fixes and feature extensions which do not introduce breaking changes. One of the most noticeable improvements is the change of color schemes. D3 no longer offers the `d3.schemeCategory20` as it was flawed and implied false relations between data due to a flaw in the color selection. Instead, version 5 of the library includes a new `d3-scale-chromatic` module which offers many predefined color schemas without the mentioned flaw. In addition, this version introduces new functions for d3-contour, d3-selection, and d3-geo.

Chapter 3

Toolkits

Creating charts with D3 can be quite tedious since the library consists mostly of low level components rather than specific chart types. Toolkits are usually building on top of D3 or using the underlying modules while abstracting the way the different charts are created. However, the approach of their implementation varies between them. While some of them such as D3FC (see Section 3.1) are still tightly intertwined with the D3 library other libraries such as Plottable (see Section 3.2) or Britecharts (see Section 3.3) wrap the D3 functionality mostly behind another API layer. There are also toolkits such as Vega (see Section 3.8) which try to reduce the amount of code that has to be written and instead formulate the charts in JSON format. In this Chapter an overview of the toolkits is given while highlighting the variety among them. An extensive list of toolkits using D3 can be found on the following website: https://github.com/wbkd/awesome-d3.

```
var data = [ 1, 3, 2, 4, 3, 5];

// Specify how to interpret the data input
var series = fc.autoBandwidth(fc.seriesSvgBar())
 .align("left")
 .crossValue(function(d, i) { return i; })
 .mainValue(function(d, i) { return d; })

var yExtent = fc.extentLinear()
 .include([0])
 .accessors([function(d, i) { return d; }]);

// Prepare Chart
var chart = fc.chartSvgCartesian(d3.scaleBand(), d3.scaleLinear())
 .chartLabel('D3FC Bar Chart')
 .xLabel("Data")
 .xDomain(data.map(function(d, i) { return i; }))
 .xPadding(0.2)
 .yDomain(yExtent(data));

chart.plotArea(series);

// Place and draw chart given the data
d3.select("#cartesian")
 .datum(data)
 .call(chart);
```

Listing 3.1: D3FC code for a bar chart. Inspired by the code from D3FC [2018].

3.1 D3FC

D3FC is a toolkit that is currently built on top of D3v4. It is licensed under the MIT License and was developed by Scott Logic [2018]. The official website however does not specify the exact version of D3 during the installation with npm, so most likely most of the code works already with D3v5. The general idea of D3FC is to expand the D3 context with additional charting components instead of completely wrapping it up behind yet another API layer. D3FC is still tightly intertwined with D3 since the components of D3FC support selections and data joins. With the help of the decorator pattern it is possible to adjust the style of the chart depending from the current data input.

In Listing 3.1 an example is given on how to create a bar chart with D3FC. The series variable specifies how to interpret that data input which can be quite important if the data are in a JSON format to access the correct fields. After the layout of the chart was defined, it is linked with the series to inform the chart on how to interpret the incoming data. The actual placement and rendering off the chart is still executed with d3 commands at the end of the code snippet. The resulting chart can be seen in Figure 3.1. The decorator pattern allows even more customization by changing the style of the chart depending on the used input. In Listing 3.2 a threshold was introduced to color all bars green which exceed a certain threshold. This is done in the decorate function and the results are depicted in Figure 3.2.

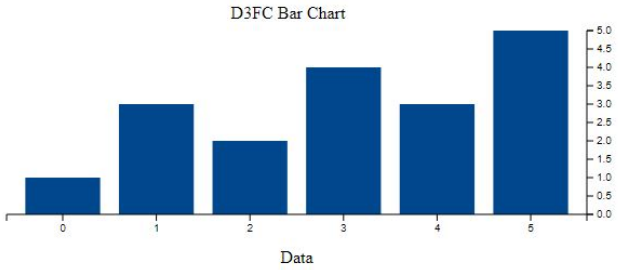

Figure 3.1: Simple bar Chart created in D3FC, using the code shown in Listing 3.1.

```
var data = [ 1, 3, 2, 4, 3, 5];
var threshold = 3;

var series = fc.autoBandwidth(fc.seriesSvgBar())
 .align("left")
 .crossValue(function(d, i) { return i; })
 .mainValue(function(d, i) { return d; })
 // further adapt the style depending on the input data
 .decorate(function(selection) {
   selection.select('.bar > path')
     .style('fill', function(d) {
       return d <= threshold ? 'inherit' : '#0c0';
     });
 });

...
```

Listing 3.2: D3FC code for a simple bar chart using the decorator pattern. Inspired by the code from D3FC [2018].

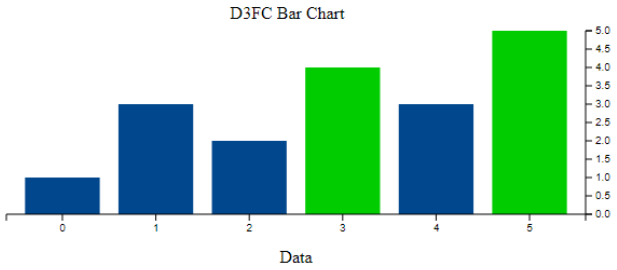

Figure 3.2: Simple bar chart created in D3FC utilizing the decorator pattern and using the code shown in Listing 3.2.

3.2 Plottable

Plottable was invented by Palantir Technologies [2018], is currently on version 3.0.0 and uses the MIT License. As of version 3.0.0 Plottable upgraded to D3v4 while previous versions used D3v3. Plottable is split into different components. Every different type of plot, as well as different axes, legends, labels, and gridlines are separate components. Since the types of the available charts are limited to a small amount, it is possible to add new components via native D3. To simplify the readability it is feasible to add different interactions, that can be bound to each component [Palantir Technologies 2018].

As seen in Listing 3.3, data points can be added as a list of dictionaries, where every x and y value is specified (line 1 to 8). To create the preferred plot, the x and y-axis are added with the corresponding scale (line 9 to 12) and the type of chart is stated (line 13). Afterwards, all components are combined to one plot (line 14 to 21) and bound to an HTML-div (line 22).

Although Plottable has a very good documentation it has some problems since upgrading to the new version of D3. Since this upgrade, the documentation is not up-to-date and the examples shown on the website are not working. Furthermore downgrading to Plottable version 2.0.0 is not possible since the Git Repository does not exist anymore. Due to that, no working bar plot could be created.

```
1  {
2  var data = [
3    { "x": 0, "y": 1 },
4    { "x": 1, "y": 3 },
5    { "x": 2, "y": 2 },
6    { "x": 3, "y": 4 },
7    { "x": 4, "y": 3 },
8    { "x": 5, "y": 5 }
9  ]
10 var xScale = new Plottable.Scales.Linear()
11 var yScale = new Plottable.Scales.Linear()
12 var xAxis = new Plottable.Axes.Numeric(xScale, "bottom")
13 var yAxis = new Plottable.Axes.Numeric(yScale, "left")
14 var plot = new Plottable.Plots.Bar()
15   .addDataset(new Plottable.Dataset(data))
16   .x(function(d) { return d.x; }, xScale)
17   .y(function(d) { return d.y; }, yScale)
18
19 var chart = new Plottable.Components.Table([
20   [yAxis, plot],
21   [null, xAxis]
22 ])
23 chart.renderTo("#chart")
```

Listing 3.3: Plottable code for a bar chart.

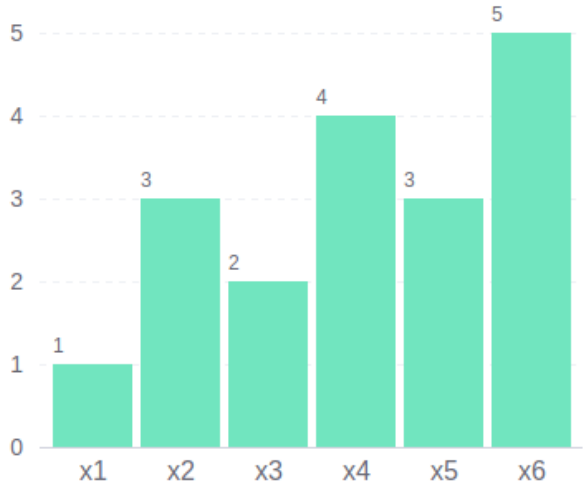

Figure 3.3: Simple bar chart created in Britecharts, using the code shown in Listing 3.4.

3.3 Britecharts

Eventbrite [2018] has invented Britecharts, which is an Information Visualization toolkit that is based on D3v5 and uses the Apache license version 2.0. The syntax used in Britecharts is very similar to native D3, as can be seen in Listing 3.4. This library is very limited to only a few plot types. However, new plots can be easily added via native D3. One major advantage of Britecharts is that it is very intuitive to use and has an exceptional documentation [Eventbrite 2018].

As seen in Listing 3.4, a container for the chart needs to be selected (line 1). Data points can be easily added via a list of a dictionary containing the labels for the *x*-axis as well as the values for the *y*-axis (line 2 to 8). In line 9 the type of the chart, in this example, a bar chart, is defined. Afterwards, the chart needs to be further specified, defining the width and height as well as the labels (line 11 to 14). In the end, the chart that has been defined needs to be called (line 15), which results in a chart as seen in Figure 3.3.

```
1  var container = d3.select("#chart")
2  data = [
3      {"name": "x1", "value": 1},
4      {"name": "x2", "value": 3},
5      {"name": "x3", "value": 2},
6      {"name": "x4", "value": 4},
7      {"name": "x5", "value": 3},
8      {"name": "x6", "value": 5}]
9  barChart = new britecharts.bar()
10 barChart
11     .width(400)
12     .height(300)
13     .enableLabels(true)
14     .labelsNumberFormat('1')
15 container.datum(data).call(barChart)
```

Listing 3.4: Britecharts code for a bar chart.

3.4 NVD3

This section describes the NVD3 library and is based on Novus Partners [2018]. This is a reusable chart component library which is built on top of D3 and intends to simplify the way a developer can create graphs while keeping the customization power of D3. The example in Listing 3.5 demonstrates how easy it is to create a bar chart with NVD3. The developers first have to create a chart object and specify all the desired options. They can use a minimalistic configuration as can be seen in the Listing, or add a lot of configuration options for specific parts of the graph such as tooltips. Next, they have to combine the graph data with the graph object itself and render them on the SVG element of their choice. The library can create graphs with the default styling with only a simple configuration from the developer. However, the library is still using D3 version 3 and is still in the process of migrating to version 4. This could represent an issue for someone who would like to use it in a new project as it would mean that they would probably need to do a complete rewrite of that part in the future.

```
1  // Data creation
2  chartData = [{
3       key: "Cumulative Return",
4       values: [
5           { label: "x1", value: 1 },
6           { label: "x2", value: 3 },
7           { label: "x3", value: 2 },
8           { label: "x4", value: 4 },
9           { label: "x5", value: 5 },
10      ]
11   }];
12
13   nv.addGraph(() => {
14      // Chart object creation
15      const chart = nv.models.discreteBarChart()
16          .x(data => data.label)
17          .y(data => data.value);
18
19      // Addition of chart object and data to the svg element
20      d3.select('#chart')
21          .datum(chartData)
22          .call(chart);
23
24      // Function for window resize handling
25      nv.utils.windowResize(chart.update);
26
27      return chart;
28   });
```

Listing 3.5: Example of NVD3 bar chart code.

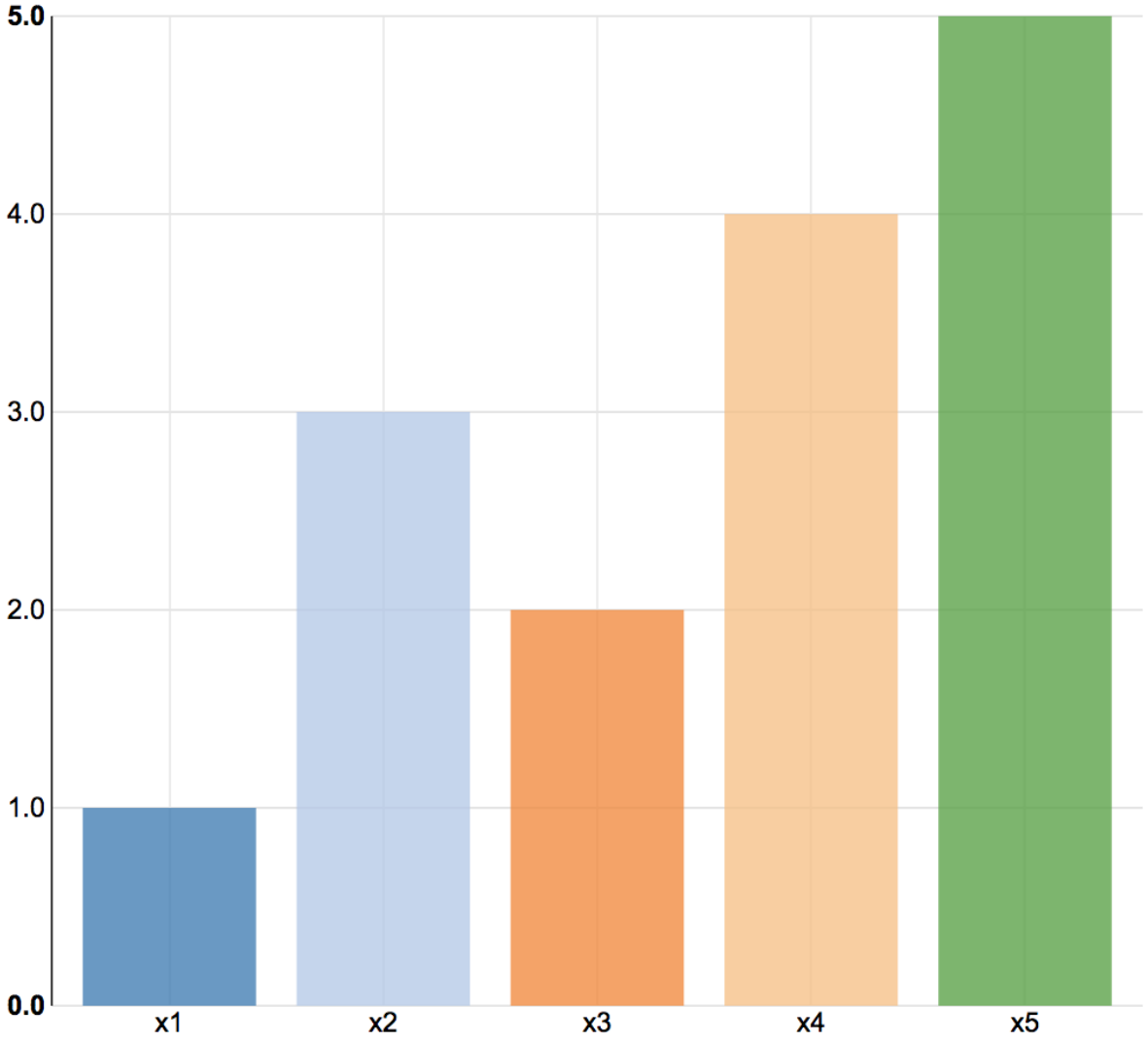

Figure 3.4: Simple bar chart created with NVD3, using the code shown in Listing 3.5

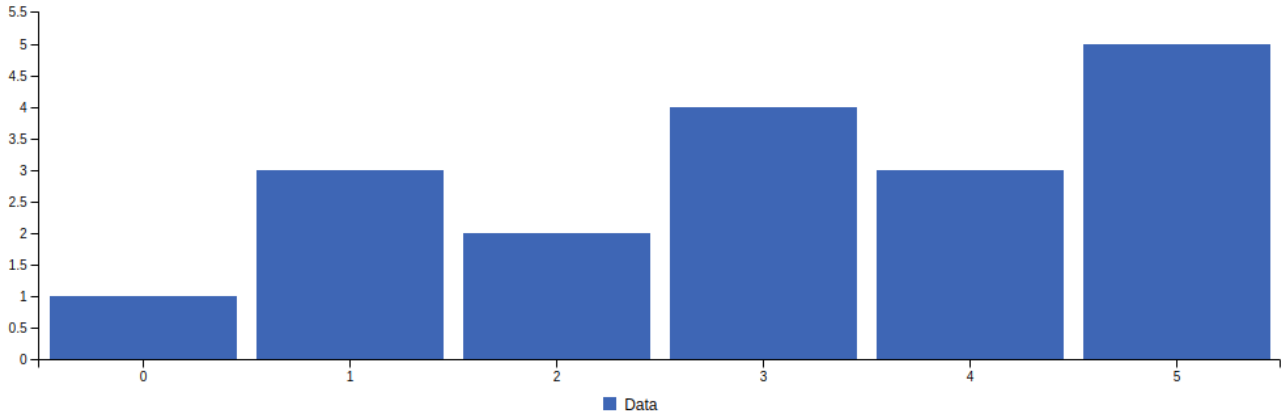

Figure 3.5: Simple bar chart created in Billboard, using the code shown in Listing 3.6.

3.5 Billboard

Billboard was invented by NAVER Corp. [2018] and is an information visualization toolkit that has been forked from C3.js, as C3.js still uses D3v3 and Billboard has migrated to D3v4. Since the D3 versions are the only differences in these two toolkits it is possible to migrate C3 code to Billboard code and vice versa with minor code changes. Billboard currently uses the MIT Licence and ES6+ [NAVER Corp. 2018].

As seen in Listing 3.6 the code is entirely written in JSON. Data points can be easily added via lists (line 3). To define the type of chart that should be used it is possible to specify the type of chart via the *type* codeword (line 4). To further customize the appearance of these bars, attributed like width and color can be changed (line 6 to 10). Afterward, the created chart needs to be bound to an HTML-div (line 11). The final chart can be seen in Figure 3.5

One unique feature of Billboard is the online playground, that makes it possible to try code online without installing the whole toolkit. Similar to many other tools, Billboard has a very good online documentation.

```
1  var chart = bb.generate({
2    data: {
3      columns:[["Data", 1, 3, 2, 4, 3, 5]],
4      type: "bar"
5    },
6    bar: {
7      width: {
8        ratio: 0.9
9      }
10   },
11   bindto: "#chart"
12 })
```

Listing 3.6: Billboard code for a bar chart.

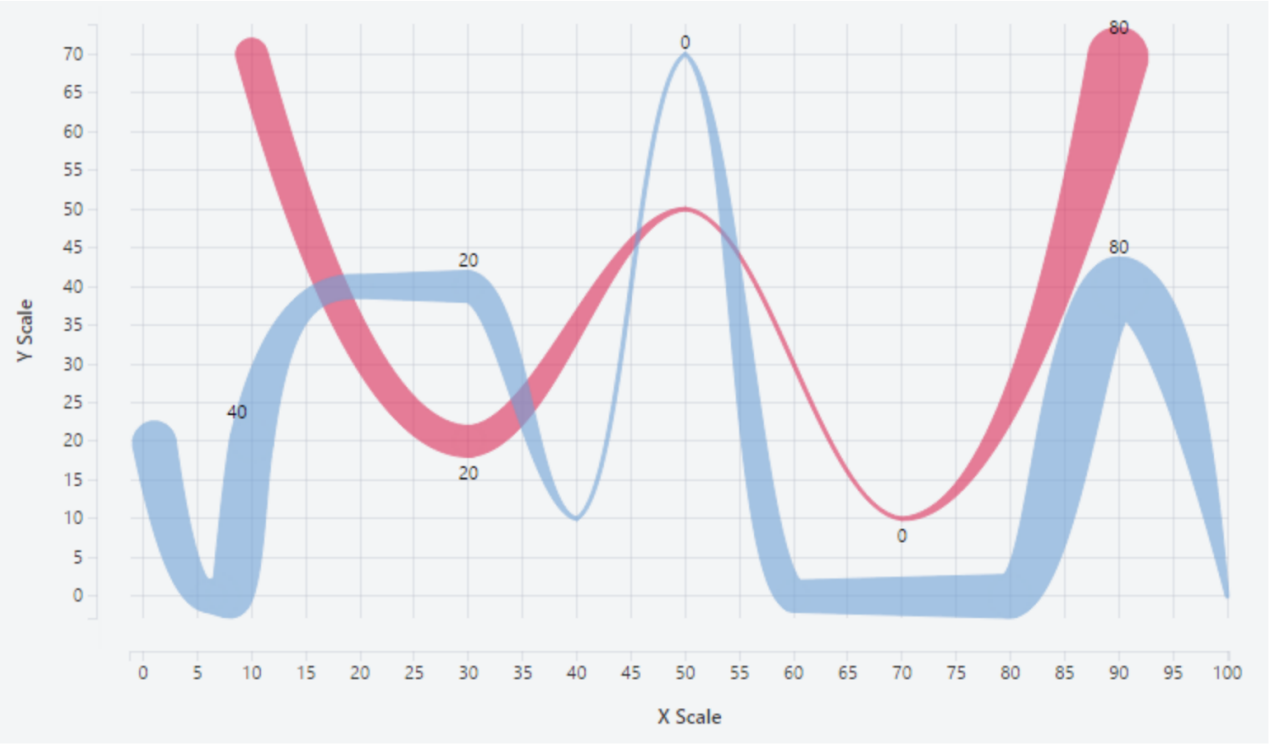

Figure 3.6: Example line chart created with Taucharts.
[Screenshot of Taucharts, captured by the authors from targetprocess [2018b].]

3.6 Taucharts

Taucharts attaches importance to perfect visual design. It is very flexible and thus powerful. Taucharts is extensible by the use of plugins. Plugins can also be custom written. As stated on their website, Taucharts is powered by Chart Intermediate Language (CIL) which is based on the 'Grammar of Graphics' concept as introduced by Leland Wilkinson [2005].

Many commonly used chart types are supported and can be implemented by selecting the chart type when creating a new chart. To name just a few of the supported types: line chart, scatter plot, bar chart, facets. The usage of Taucharts is very easy, basic charts can be created in only a few lines of code as seen in Listing 3.7. The documentation provides quite extensive instructions and lots of examples. Also a 1 min tutorial and a 5 min tutorial are provided. That makes the learning curve quite flat. Taucharts offers responsiveness out of the box. The width is automatically adapted and to preserve readability of the chart and its labels, the labels are moved out of the way on smaller devices. Charts Intermediate Language (CIL) is a declarative language to describe the visual appearance of the charts. CIL splits the chart into three aspects: data, data mapping and layout which are described separately. These aspects are specified in a JSON description of the graph. Rendering is then happening in three steps: Building the logical structure, calculating the layout and finally drawing the graph. Providing interactivity out of the box lets Taucharts stand out from other toolkits. Tooltips, fading away of data sets that are not highlighted by hovering, completely hiding of data sets by selecting them in the legend etc. make Taucharts really powerful and interactive.

Taucharts is published under the apache license version 2.0 and is maintained by targetprocess [2018a]. In the examples and in the documentation D3 is referenced in version 3.5.17 and 4.13.0. The basic examples created for this survey have been developed using D3 version 5.1.0.

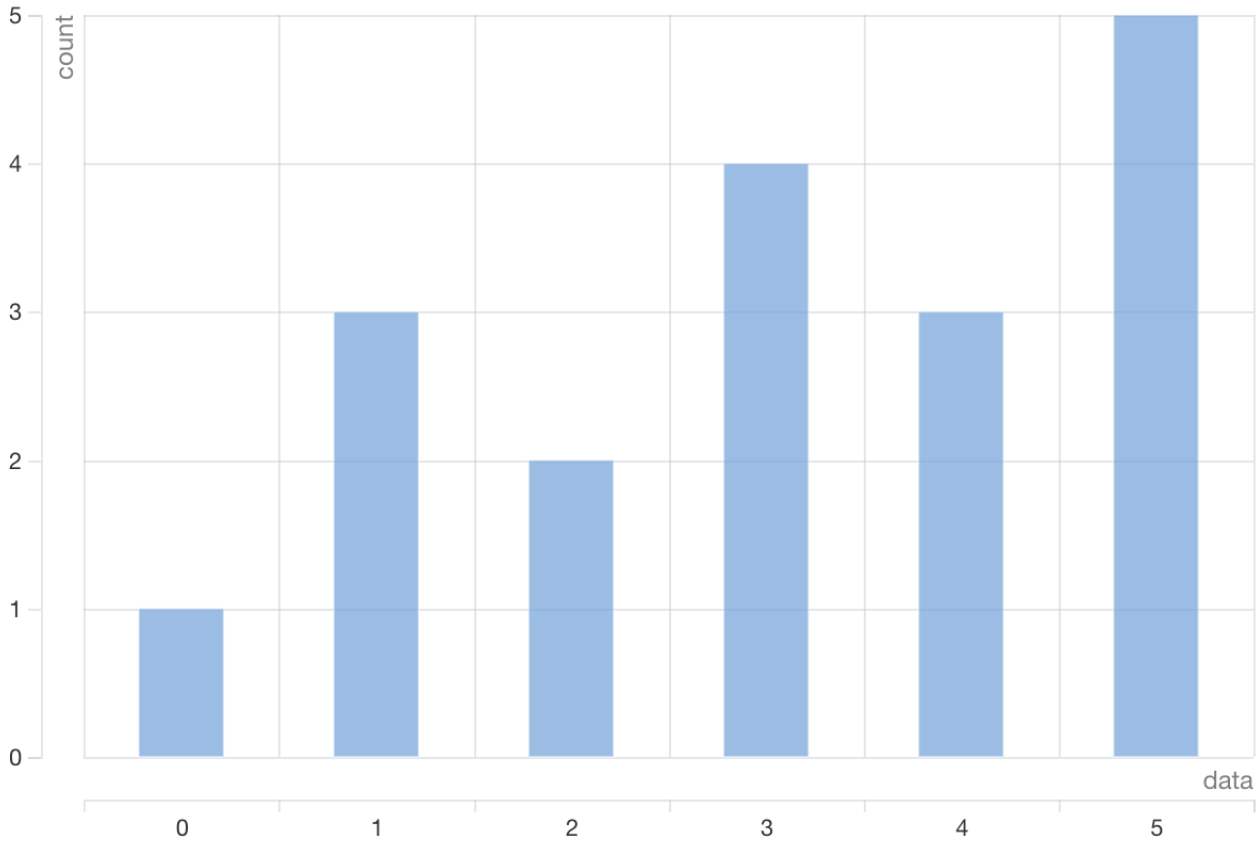

Figure 3.7: Simple bar chart created with Taucharts, using the code shown in Listing 3.7.

```
var defData = [
  {"data": "0","count": 1,},
  {"data": "1","count": 3,},
  {"data": "2","count": 2,},
  {"data": "3","count": 4,},
  {"data": "4","count": 3,},
  {"data": "5","count": 5,}
];
var chart = new Taucharts.Chart({
  data: defData,
  type: 'bar',
  x: 'data',
  y: 'count'
});
chart.renderTo('#bar');
```

Listing 3.7: Example of Basic Taucharts Line Chart.

3.7 dagre-d3

Dagre is a JavaScript library for creating directed graphs. These graphs are rendered on the client. Since dagre only focuses on the graph layout it needs an extra library providing a rendering engine. This is where dagre-d3 comes into play, it is a D3-based renderer for dagre.

Dagre has multiple ways of describing the graphs. Nodes and edges can be added via JavaScript as seen in Listing 3.9. But an extension for dagre to read the DOT language, a graph desription language, can be used for a simplified desription. An example can be found in Listing 3.8. Dagre-d3 is very easy to use. As shown in Listing 3.9 the renderer is created with dagre-d3, the SVG-element to render the graph has to be selected with D3. In the end they are combined to run the renderer. Dagre-d3 does not provide many interactivity features on it's own. But, as seen in the example in Listing 3.9, these can easily be implemented with D3 or plain JavaScript to center and zoom the graph. Responsiveness has to be added manually. Documentation is available and consists of basic examples, it could be more extensive though. Dagre-d3 is licensed under the MIT license. Officially D3v4 is listed as dependency but it has shown to be working with D3 in version 5.1.0.

```
digraph {
    /* Note: HTML labels do not work in IE, which lacks support for <
        foreignObject> tags. */
    node [rx=5 ry=5 labelStyle="font: 300 14px 'Helvetica Neue', Helvetica"]
    edge [labelStyle="font: 300 14px 'Helvetica Neue', Helvetica"]
    A [labelType="html"
       label="A <span style='font-size:32px'>Big</span> <span style='color:red
           ;'>HTML</span> Source!"];
    C;
    E [label="Bold Red Sink" style="fill: #f77; font-weight: bold"];
    A -> B -> C;
    B -> D [label="A blue label" labelStyle="fill: #55f; font-weight: bold;"];
    D -> E [label="A thick red edge" style="stroke: #f77; stroke-width: 2px;"
        arrowheadStyle="fill: #f77"];
    C -> E;
    A -> D [labelType="html" label="A multi-rank <span style='color:blue;'>HTML
        </span> edge!"];
}
```

Listing 3.8: Dot language of a graph for dagre-d3. Taken from dagre-d3 [2018].

```
// Create the input graph
var g = new dagreD3.graphlib.Graph()
  .setGraph({})
  .setDefaultEdgeLabel(function() { return {}; });

// Set up nodes.
g.setNode(0,  { label: "A", class: "type-start" });
g.setNode(1,  { label: "B" });
g.setNode(2,  { label: "C" });
g.setNode(3,  { label: "D", class: "type-end" });
g.setNode(4,  { label: "E" });
g.setNode(5,  { label: "F", class: "type-end" });

// Set up edges.
g.setEdge(0, 1, {
  label: "A to B",
```

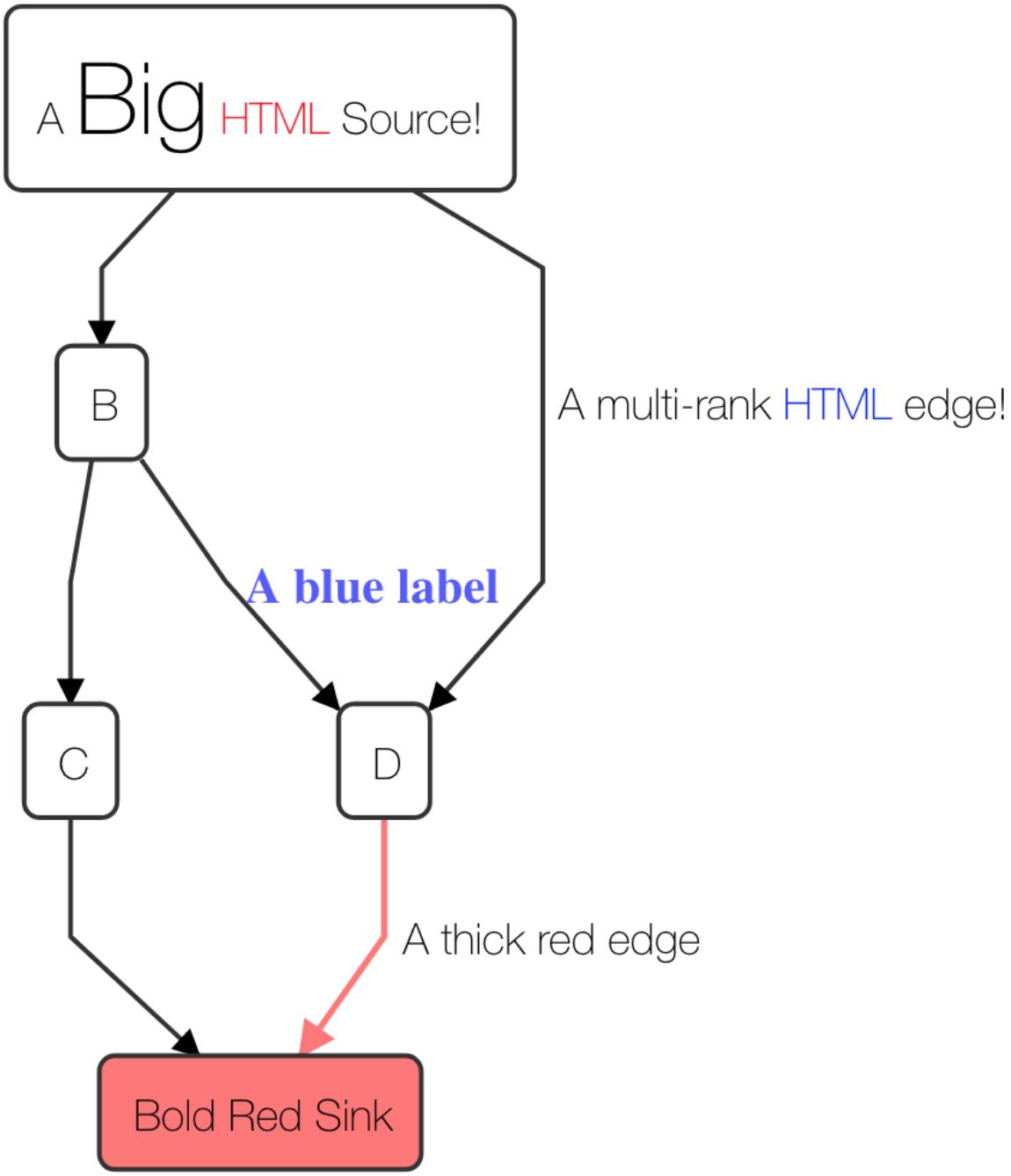

Figure 3.8: A simple directed graph showing node styles with html content and different label and edge style. Created with the DOT language shown in listing 3.8.
[Screenshot of dagre-d3 [2018] captured by the authors.]

```
  style: "stroke: #f66; stroke-width: 3px; stroke-dasharray: 5, 5;",
  arrowheadStyle: "fill: #f66"
});
g.setEdge(1, 2);
g.setEdge(1, 4);
g.setEdge(2, 3);
g.setEdge(2, 3);
g.setEdge(4, 3);
g.setEdge(4, 5);

g.nodes().forEach(function(v) {
  var node = g.node(v);
  // Round the corners of the nodes
  node.rx = node.ry = 100;
});

// Create the renderer
var render = new dagreD3.render();

// Set up an SVG group so that we can translate the final graph.
var svg = d3.select("svg"),
    svgGroup = svg.append("g");

// Set up zoom support
var zoom = d3.zoom().on("zoom", function() {
      svgGroup.attr("transform", d3.event.transform);
    });
svg.call(zoom);

// Run the renderer. This is what draws the final graph.
render(svgGroup, g);

// Center the graph
var initialScale = 1.5;
svg.call(zoom.transform, d3.zoomIdentity.translate((svg.attr("width") - g.graph
    ().width * initialScale) / 2, 20).scale(initialScale));

svg.attr('height', g.graph().height * initialScale + 40);
```

Listing 3.9: Example of Basic dagre-d3 Directed Graph.

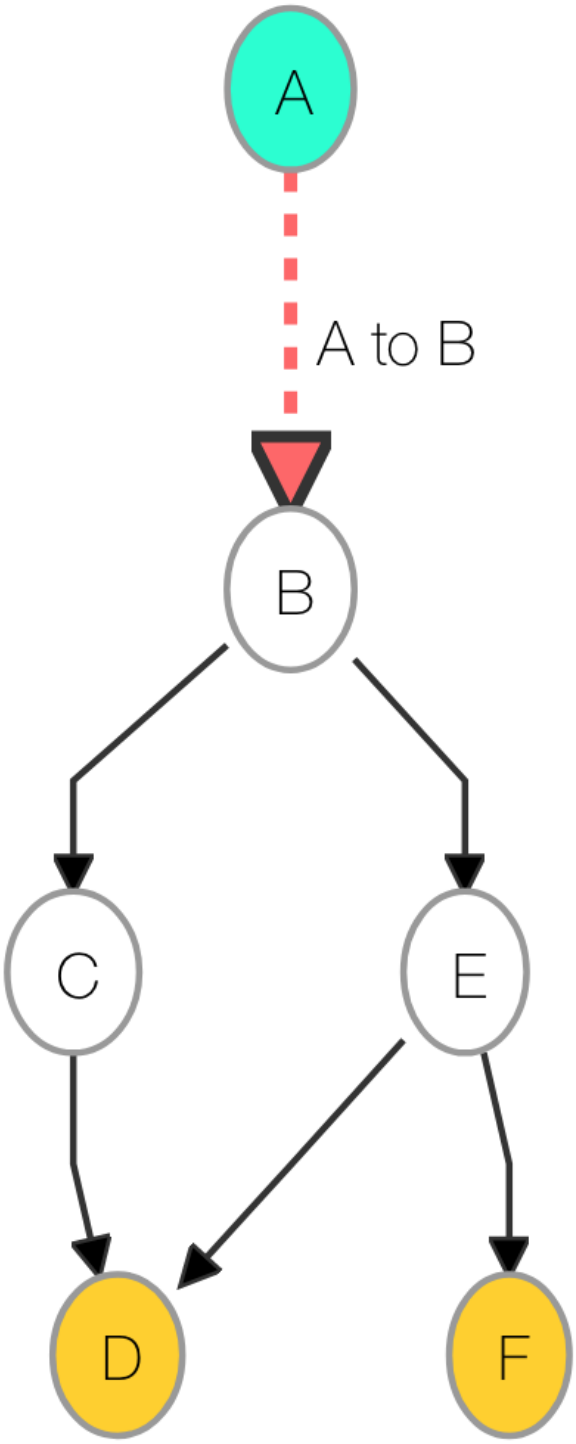

Figure 3.9: Simple graph created with dagre-d3, using the code shown in listing 3.9.

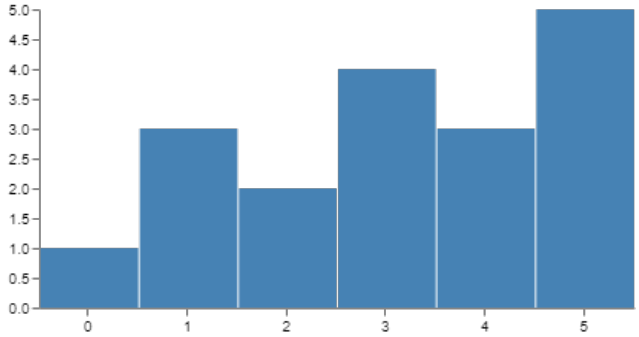

Figure 3.10: Simple bar chart created in Vega, using the code shown in Listing 3.10.

3.8 Vega

Vega was developed by Satyanarayan et al. [2014] and is currently available under version 3.3.1. However, the first release candidate of version 4 can already be downloaded. There is also a light version called Vega-Lite developed by Satyanarayan et al. [2017]. Vega itself does not directly include a D3 version but rather requires the low level modules of D3 such as d3-array, d3-format and d3-interpolate. The package.json of the Vega-Lite version includes TypeScript typings of D3v4 and thus hints to the fact that those included components are D3v4 compatible. Vega extensively uses a JSON to define the chart and the displayed data. An online editor on the offficial Vega website allows to create fast prototypes with practically no setup overhead. Installations with npm can be troublesome since Vega does not only depend from JavaScript but also from Python.

The bar chart example with Vega can be seen in Figure 3.10 using the code from Listing 3.10. The JSON requires certain fields to be present such as data, scales, axes and marks. Scales are further defining the style of the axes while the marks define how to insert the data. The decision to keep most of the chart design in a JSON has advantages since the coding requirements are a little bit reduced. On the drawback, a chart description in JSON format requires usually more space then their code equivalents.

```
{
  "$schema": "https://vega.github.io/schema/vega/v3.0.json",
  "width": 400,
  "height": 200,
  "padding": 5,
  "data": [
    {
      "name": "table",
      "values": [
        {"category": "0", "amount": 1},
        {"category": "1", "amount": 3},
        {"category": "2", "amount": 2},
        {"category": "3", "amount": 4},
        {"category": "4", "amount": 3},
        {"category": "5", "amount": 5}
      ]
    }
  ],
  "scales": [
    {
      "name": "xscale",
      "type": "band",
      "domain": {"data": "table", "field": "category"},
      "range": "width"
    },
    {
      "name": "yscale",
      "domain": {"data": "table", "field": "amount"},
      "nice": true,
      "range": "height"
    }
  ],
  "axes": [
    {"orient": "bottom", "scale": "xscale"},
    {"orient": "left", "scale": "yscale"}
  ],
  "marks": [
    {
      "type": "rect",
      "from": {"data": "table"},
      "encode": {
        "enter": {
          "x": {"scale": "xscale", "field": "category", "offset": 1},
          "width": {"scale": "xscale", "band": 1, "offset": -1},
          "y": {"scale": "yscale", "field": "amount"},
          "y2": {"scale": "yscale", "value": 0}
        },
        "update": {"fill": {"value": "steelblue"}}
      }
    }
  ],
  "config": {}
}
```

Listing 3.10: Vega bar chart code example. Inspired by the code from Vega [2018].

Chapter 4

Toolkit Comparison

There is no one toolkit to rule them all. All toolkits vary on a broad range of features, chart types, interactivity and ease of us. Thus declaring a best performing toolkit without specifying the requirements is not possible. It is rather suggested to determine the toolkit to use based on the goals and desired features of the project and the skills of the developer or development team. In the following, the key criteria are given to decide when a specific toolkit should be used.

Chart / plot types is the first criterion to think about. What kind of charts have to be implemented? Not all toolkits support all chart types, so this is the biggest limitation on the toolkits. While there are quite a lot of toolkits supporting basic chart types such as bar charts and line charts, already fewer support stacked charts or scatter / bubble charts. Especially some rarer chart types like directed graphs or dendrograms are supported by only a few toolkits. The chart / plot types supported by the investigated toolkits are listed in Table 4.1

Interactions / features is the second dimesion of criteria to consider. While all interactivity can be created from scratch, some toolkits provide interactivity out of the box. More commonly supported features of interactivity are scaling, tooltips, en- / disabling of datasets, selecting timelines, etc. If one of those basic types of interactivity is needed without lots of customization some toolkits provide more convenience out of the box. The interactions supported out of the box by the investigated toolkits are listed in Table 4.2

Ease of use in regards to creating charts by coding vs. configuring them via JSON files is something to be considered as well. A professional JavaScipt programmer might want to select a toolkit that requires to code the chart and thus give you more freedom. An unskilled programmer such as a designer or project manager with the goal of fastly showing some data might prefer a toolkit that configures charts via a JSON file.

Metadata is also something which the toolkits handle in a broad variety. It is recommended to check the documentation for completeness and how easy it is to understand the toolkit. Also choosing frequently updated tools can be of advantage since they assure ongoing support and the introduction of new features. Since the toolkits are usually hosted on github, the last commit date can be evaluated and also the commit history offers further details about the update frequency. While some of the presented toolkits are rarely updated and have intervals of up to two years between updates, others are more frequently updated. The version of D3 used should also be considered. The objective is to look for toolkits using D3 version 4 or 5 since there are almost no breaking changes between those two versions and D3 version 5 is still quite new. However, it is not suggested to use tools using D3 version 3 anymore when starting a new project. Metadate of the investigated tookits can be seen in Table 4.3

	NVD3.js	D3FC	Britecharts	billboard.js	Plottable	dagre-d3	Taucharts
Area Plot	x	x		x	x		
Area Range Chart				x	x		
Bullet Chart	x		x				
Bar Plot	x	x	x	x	x		x
Clustered Bar Plot	x	x	x	x	x		x
Line Plot		x	x	x	x		x
Multiple XY Line Chart				x			
Line Chart with Regions				x			
Cumulative Line Chart	x						
Pie Plot	x			x	x		
Donut Chart	x		x	x			
Gauge Chart				x			
Rectangle Plot					x		
Scatter Plot	x		x	x	x		x
Bubble Chart		x		x			
Segment Plot					x		
Stacked Area Plot	x		x	x	x		x
Stream Area Plot	x						
Expanded Area Plot	x						
Stacked Bar Plot	x	x	x	x	x		x
Mondrian					x		
Gantt Chart					x		
Calendar Heatmap					x		
Time Series				x	x		
Spline Chart (curved line)	x		x	x			
Step Chart			x	x			
Combination Chart	x			x			
Indented Tree							
Small multiples		x					
Directed graphs						x	
Facet chart							x

Table 4.1: Plot / chart / graph types supported by the investigated toolkits.

For basic chart types like bar charts or line charts, the toolkits Taucharts and Billboard.js are recommended to use. These provide the biggest set of features and interactivity out of the box. They are very flexible to custom feature requirements while still being easy to use.

	NVD3.js	D3FC	Britecharts	billboard.js	Plottable	dagre-d3	Taucharts
Hover	x	x	x	x	x	x	x
Click					x		
Double Click					x		
Key					x		
Pan		x		x	x		
Zoom		x		x	x		
En-/Disable Datasets	x			x	x		x
Subchart							
Select / Zoom	x		x	x	x		
Highlight Datasets				x			x
Highlight Columns	x		x	x			x
Tooltips	x		x	x			x
Crosshair		x					x

Table 4.2: Interaction supported out of the box by the investigated toolkits.

	NVD3.js	D3FC	Britecharts	billboard.js	Plottable	dagre-d3	Taucharts
Github	Github	Github	Github	Github	Github	Github	Github
Website	Web	Web	Web	Web	Web		Web
D3 Version	3.x	4.x	4.x	4.x	4.x	4.x / 5.x	4.x / 5.x
Documentation	Docs	Docs	Examples	Examples	Tutorials	Examples	Docs
Ease of use	easy					easy	easy
License	Apache v2	MIT	Apache v2	MIT	MIT	MIT	Apache v2
Last commit	6.4.2018	16.10.2017	6.5.2018	11.5.2018	4.5.2018	18.3.2018	26.4.2018

Table 4.3: Metadata of the investigated toolkits. Github, website and documentation are hyperlinks to the corresponding webpages.

Chapter 5

Concluding Remarks

In this survey, the popular JavaScript library D3 and a variety of toolkits building on top of D3 were investigated. The fundamentals of D3 were explained and the varieties between the individual versions have been highlighted. While the transition from D3v3 to D3v4 included many breaking changes, the upgrade from D3v4 to D3v5 was rather small. Many examples written in D3v4 can still be executed in D3v5 since only data reading modules were affected from the D3v5 change. While D3 focuses on efficient updates of documents and is building charts from low level components, the toolkits usually directly provide a variety of different chart templates.

To highlight the differences between D3 and the toolkits, a bar chart was used as the object of comparison. Further investigation showed that D3 requires in general more code than the given toolkits and is harder to grasp because of its low level nature. D3FC is still tightly intertwined with the D3 code base and requires similar amounts of code while providing chart templates and the ability to create custom ones. There are no working examples for Plottable and thus requires patching from the developers side. Britecharts offers a small variety of chart types but enables the possibility to add additional ones using native D3. NVD3 is still using D3v3. Thus despite its provided functionality it is not recommended to use it because of the implicated maintenance overhead in the future. Billboard is a designer friendly toolkit specifying the complete chart in a JSON. Taucharts is a state-of-the-art toolkit using D3v4 and offers a broad variety of chart types. dagre-d3 focuses on rendering graphs and thus satisfies another visualization branch. Vega is also a designer friendly toolkit since the whole chart is defined in a JSON while the binding to the document happens in a JavaScript function.

In the toolkit comparison different criteria were presented to find the optimal toolkit depending on a given task. dagre-d3 should be used if the project meets the specific niche requirements. For regular uses cases, Billboard and Taucharts are probably the best toolkits since they require decent amounts of code while offering broad functionality.

Bibliography

Bostock, Michael, Vadim Ogievetsky and Jeffrey Heer [2011]. "D3: Data-Driven Documents". *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* [2011]. http://vis.stanford.edu/papers/d3 (cited on page 3).

Bostock, Mike [2018a]. *Bar Chart - bl.ocks.org*. Blog, GNUv3 License. 13th May 2018. https://bl.ocks.org/mbostock/3885304 (cited on page 8).

Bostock, Mike [2018b]. *D3 Website*. D3 Website. 10th May 2018. https://d3js.org/ (cited on pages 3–5).

Bostock, Mike [2018c]. *General Update Pattern - bl.ocks.org*. Blog, GNUv3 License. 13th May 2018. https://bl.ocks.org/mbostock/3808218 (cited on pages 4, 6).

Bostock, Mike, Curran Kelleher, Micah Stubbs, Jeferson Koslowski, Gordon Woodhull and Barrie Treloar [2018]. *D3 Changelog*. D3 Github Repository. 3rd May 2018. https://github.com/d3/d3/blob/master/CHANGES.md (cited on pages 9, 11).

D3FC [2018]. *D3FC Bar Chart Example*. D3FC Website. 13th May 2018. https://d3fc.io/introduction/building-a-chart.html (cited on pages 14–15).

dagre-d3 [2018]. *dagre-d3 interactive demo*. Github Page. 10th May 2018. https://dagrejs.github.io/project/dagre-d3/latest/demo/interactive-demo.html (cited on pages 23–24).

Eventbrite [2018]. *Britecharts Information Visualization Toolkit*. Britecharts Website. 10th May 2018. eventbrite.github.io/britecharts/index.html (cited on page 17).

NAVER Corp. [2018]. *Billboard Information Visualization Toolkit*. Billboard Website. 10th May 2018. naver.github.io/billboard.js/ (cited on page 20).

Novus Partners [2018]. *NVD3 Re-usable charts for d3.js*. NVD3 Website. 9th May 2018. http://nvd3.org/ (cited on page 18).

Palantir Technologies [2018]. *Plottable Information Visualization Toolkit*. Plottable Website. 3rd May 2018. plottablejs.org/ (cited on page 16).

Satyanarayan, Arvind, Dominik Moritz, Kanit Wongsuphasawat and Jeffrey Heer [2017]. "Vega-Lite: A Grammar of Interactive Graphics". *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* [2017]. http://idl.cs.washington.edu/papers/vega-lite (cited on page 27).

Satyanarayan, Arvind, Kanit Wongsuphasawat and Jeffrey Heer [2014]. "Declarative Interaction Design for Data Visualization". In: *ACM User Interface Software & Technology (UIST)*. 2014. http://idl.cs.washington.edu/papers/reactive-vega (cited on page 27).

Scott Logic [2018]. *D3FC Information Visualization Toolkit*. D3FC Website. 10th May 2018. https://d3fc.io/ (cited on page 14).

targetprocess [2018a]. *targetprocess*. targetprocess Website. 10th May 2018. https://www.targetprocess.com (cited on page 21).

targetprocess [2018b]. *Taucharts Github*. Taucharts Github Repository. 3rd May 2018. https://github.com/TargetProcess/tauCharts (cited on page 21).

Vega [2018]. *Vega Bar Chart Example*. Vega Website. 1st May 2018. https://vega.github.io/vega/tutorials/bar-chart/ (cited on page 28).

Wilkinson, Leland [2005]. *The Grammar of Graphics*. Springer, 2005. ISBN 978-0-387-28695-2 (cited on page 21).