Department of Mathematics
and Computer Science

University of Southern Denmark

Master Thesis in Data Science

Multilingual Textual Alignment:
Evolving and Modernizing an Open Source Analytics Tool

Author
Rikke Have Odgaard
Exam nr:
@student.sdu.dk

Supervisor
Stefan Jänicke

September 15, 2024

Abstract

English *TRAViz is an open-source analytics tool developed in JavaScript (JS) before ECMAScript6 was introduced [Int24]. It relies on jQuery, Qtip, and Rahpäel. The standards of ES6 have resulted in the rapid development of JS technologies, and the shift towards browser-based applications, has made a modernization of the original source code relevat, and has also shown that minimized dependency on external libraries might prepare for better sustainability of the software to remain compatible with contemporary web standards. The software has thus been modernized and the JS basis now more aligns with current standards. Dependence on jQuery and Qtip have been substituted by Vanilla JS. D3.js is leveraged for dynamic DOM manipulation, and for rendering, the interactive visualization via SVG, substituting Raphäel.js. A user-interactive multilingual analytic has been added to the source code as well. The process of development and the result are presented in the report, which constitutes a significant element of the modernized software documentation.*

Danish *TRAViz er et open-source JavaSript (JS) baseret software, til tekstanalyse, udviklet før ECMAScript6 [Int24]. Softwaren beror på anvendelsen af jQuery, Qtip og Raphael. Men ES6 har ført til en markant udvikling af JS-baserede teknologier, og samtidig er der sket et skift mod udbredelse af browserbaserede applikationer. Dette samlet, gør en modernisering af den originale kode relevant. Udviklingen viser også at en minimeret afhængighed af eksterne ibraries kan bidrage til øget bæredygtighed af en software, der ikke skal moderniseres i forhold så kompatibilitet med disse. Softwaren er blevet moderniseret, og JS-grundlaget i koden er nu mere overensstemmende med de nuværende standarder. Istedet for anvendelsen af jQuery og Qtip så benyttes der Vanilla Js. D3.js anvendes til dynamisk DOM-manipulation, og til at optegne den interaktive visalisering via SVG, hvilket erstatter Raphäel.js. Der er desuden integreret en udvidelse, således værktøjet nu også kan anvendes til flersproget analyse. Udviklingsprocessen og resultatet præsenteres i rapporten, og udgør et væsentligt element af den moderniserede software-dokumentation.*

Contents

Chapter 1

Introduction

TRAViz is an open source analytics tool developed for a humanities discipline ”close reading”, allowing inspection of similar passages from within a specific language, of different editions of text-excerpts, aligned in user-interactive directed Text Variant Graphs[Jän16]. TRAViz was licensed in 2014 [Jän14], and developed in an interdisciplinary setting during a PhD project at Leipzig University, in the context of, and with, other tools relevant within the field of Digital Humanities [Jän24]. An example alignment can be inspected in figure 1.1 below.

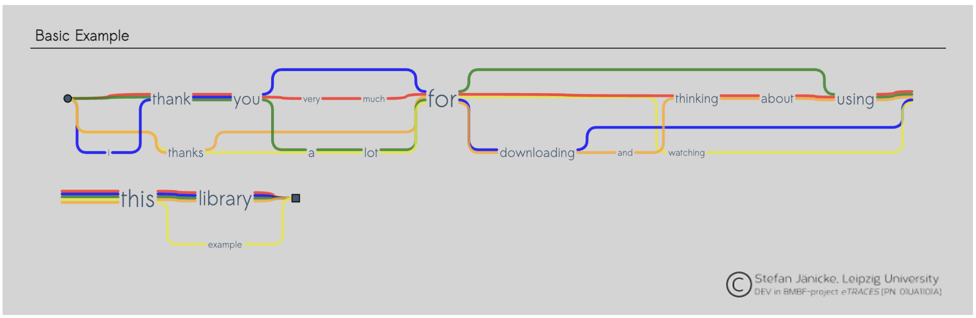


Figure 1.1: TRAViz: alignment of 5 similar sentences with similar intentions, but different syntax and wording. Source: www.traviz.vizcovery.org/

The TRAViz source code is a JavaScript (JS) code, and its interactions are based on a JS library jQuery [Fou24], which at that time gave a unique access to selection and manipulation of elements on a web page, working equally and similar well, across different browsers, and on Raphäel.js [Bar20] another library especially suited for rendering scalable graphics, in short form ”SVG”. JQuery was, and still is the most widespread library used in JS websites [W3T24]. However, two major developments have taken place since TRAViz was licensed 1) the browser has become the major application platform, and 2) concurrently browser-differences have diminished which has then lead to the plethora of software that is accessed in the browser. Synchronously with these develpments other libraries, and frameworks, have emerged and are competing with jQuery, and in many cases considered more currently relevant [Guo24], probably due to the setting of standards of scripting in JS language in 2015, which then re-awakened development in the JS langauge [Klø21] These newer libraries provide the same access to handle and manipulate the structure of a page, but due to different ways of managing the page-objects they can to some extent be incompatible with jQuery, at least for the non-experienced software user, or someone with only little coding-experience [Leg24].

Applications for presenting visual analytics are popular withing the field of Data Science, which at the same time is increasingly being populated by experts from other fields: finance, biology, medicine, and also from the humanities like political science, journalism, and literature who all wish to present analytic insights from within their fields, in browser-based visual environments. Often these specialists have little to no coding experience [IBM24]. Modernized simple tools allow access even to the non-expert for mediating analytics via simple visualization tools, often provided in Python-based applications for instance Dash [Plo24]. To accommodate these field experts, any tool for visualisation can benefit from adapting to this newer large, and growing, group of users [Sar21]. Thus relevance of continuous maintenance and modernization, evolving alongside other technology, and their advances both in terms of hardware and software environments, is well founded, and constitute the basic argument for the tasks described and reported in this master (see also appendix A for the full Project Description):

1. ***To further develop and modernize TRAViz, such that it accommodates current browser-based environments. And to add to TRAViz, a small extension allowing multilingual analysis.***
2. ***To create the basis for a collaboration platform, inviting the further maintenance and development of TRAViz.***
3. ***To document and comment the modernized code, and the extension, thereby making use, and further development of TRAViz continually accessible.***
4. ***In a final section to sum op the development of the product, and assess the learning outcome of the project.***

This will be reported in three following sections:

- In Chapter 2 I will give a very brief description of TRAViz, before presenting select examples of the modernization process, by comparing excerpts of the original source code with the modernized substitutions. In a similar detailed way I will describe the process of adding a multilingual feature for the source code.

  This chapter is the most technical and practical, and beyond documenting the work in the master project, it also makes up code documentation in itself, and it is aimed at answering point 1 above.

- In Chapter 3 I present a definition of documentation, and argue that the report, the collaborative effort to modernize it, as well as the documentation using JSDoc

[Col24], and commenting inside the source codce, also makes up documentation. I will also describe, and share details from the README presented on the established forum on GitHub, and this then answers point 2 and 3 from the tasks above.

- Each section contains a content-relevant conclusion, so in Chapter 4 I sum op with an overall assessment of the learning outcome and respectively the software development.

But before delving into the modernization, first two initial subsections. The first explaining the overall choice of style and other form decisions made, and the second discussing the use of generative AI in an academic setting, particularly in the context of this report.

Additionally: the code is handed in as an external appendix, as a compressed file. If the file structure is not changed, then the easiest way of quickly inspecting the result, is to open it in an IDE, then go to the ”3divs.html” file, and open it in a live server, on a local host. For alternative ways of inspecting it take a look at the README file, which introduces other methods to inspect it. In case the file becomes corrupted, there is also open access to the repository created for this assignment, which will remain open at least til after the examination: https://github.com/RHO-DK/MTRAViz

1.1 Style and Form

I have a background in Political Science, and by that a very different approach to choice and use of references, and other format decisions, than what I am adapting to, in the context of the Department of Mathematics and Computer Science at SDU. Writing a technical report is not something we have learned during the two year master of Data Science, nor have I been able to find precise demands as to how the department wants us to format our reports. As such this report then represents a polymorphic entity between institutions, not fully satisfactory in either silo, because it is a little too much of the other, in any of the contexts that I now belong to. This is the background for the choise of this section, and the elaborated information below.

I have left out sections about Philosophy of Science, though the backdrop of this field has guided my approach to this report, which in full accepts foundational principles for academic writing and I am attempting to create a fully transparent and well documented development process, in which both my results and relevant considerations will be shared and triangulated by means of comparison with other findings, or with technical know-how from practitioners in the fields related to this project.

Formatting of the references themselves, a reference with more than one writer includes a superscripted ”+”, and in the bibliography names of all writers are included, as no one to my knowledge at least, is named ”et al”.

Finally I write as ”I”, a choice taken to show a clear responsibility. It is not in any way to indicate that the code, or the extension added is my own personal invention, as the code, and its structure and logic(s) are not created by me, especially the main source code which has only been modified to adapt to more current standards. The choice to write as ”I”, ”me” and ”my”, has been taken because all errors in conception, in considerations, or in perceptions are fully my own.

1.2 ChapGPT

ChatGPT is developed by OpenAI [Cha24] and is a Large Language Model introduced in 2021 with a full global release in 2022 [CAS24]. Large Language Models rely on Transformer architecture, that serve as a generative core of the so-called ”Artificial Intelligence”(AI), and utilizes the high levels of computational power in modern technologies to process and predict pattern sequences in interaction with a user, for instance by receiving and responding to written prompts by users of the technology [Zha+23].

ChatGPT is thus a recent, and probably therefore still somewhat controversial tool, not least in the context of academia [SKM23]. Many arguments promote usage in learning contexts, and also in higher education, but they often seem fall along the line of ”if you can’t beat them, join them”, as it is difficult if not impossible to prevent use [SKM23]. Other arguments seem more prudent, yet are still promoting acceptance of usage, but focus more on possibilities and the potential for growth in motivation due to instantaneous and personalized feed-back matched to the individual student needs [Yu23].

Universities worldwide were initially hesitant to to embrace this leap in AI-assisted learning, and at SDU use was initially banned during exams [THS23]. Already by early 2024, and finalized as a decision in June 2024, with effect pr September 2024 this ban of usage at SDU was first minimized, and then lifted complete[Uni24].

In Political Science most research questions focus on decoding some social practice, or some other societal phenomenon. Much of the research deals with phenomenons which cannot easily be measured, and often they have some fleeting or polymorphic essence making them difficult to show or observe, and thus difficult to investigate, measure and report. This means the development of standardized methods for reporting and reflecting on use of means to capture data about the object of interest, in order to make research so transparent that it can be replicated. I view usage of ChatGPT in that perspective: it is a tool, which can divert me away from relevant sources, and as we all know it can even misrepresent sources, or it can fabricate them [Ath+23] as with any other media used, both students and professionals require training in the usage. At the moment we are all still learning where and for what it is relevant.

The guidelines are to reference direct use of ChatGPT, but this is easier described than it is executed. I have used it much like an ”informal Google”. Prompts like: ”Explain it to me as if I was a child of 10 years”, or ”could you describe to me what a ”node” is”, or ”is there a method in jQuery named ....”, ”What do I look for in code when I want to understand flow of data”, have been typical during the project period. These questions, and sometimes even longer conversations, have given me answers without which some tasks might have been beyond my expertise, as I have no background in, or real experience with, coding. I have received help in creating and inserting relevant logging demands in the code, especially in learning where to insert those, whether inspecting elements, or for instance investigating processes. And I have received help understanding the original code, as well as in the modernization or extension, for instance I had problems with an endless amount of dictionaries (see chapter 2, section 2.3.2) being loaded in many of the initial attempts using them, and I learned that I could choose to load those BEFORE initializing the code, which I had not even thought of.

For me access to ChatGPT has meant being able to ask those very basic of questions, and ask them repeatedly, insisting on repeated efforts of explanation, until something eventually made sense to me, and I could then move on to more relevant sources, and there investigate how to manipulate a newfound knowledge. I have used ChatGPT extensively for acquiring enough grasp on a subject, or even a vocabulary, to reach (or understand) relevant search results in other places, for instance in documentation, in blogs or other online forums.

Chapter 2

MTRAViz extends TRAViz

I have described above that TRAViz is a software used for aligning excerpts of different editions, for instance different translations, of a textual corpora [YS21]. Figure 1.1. in the section above was an image of such an alignment, and below in figure 2.1, you can see another text excerpt being aligned [1].

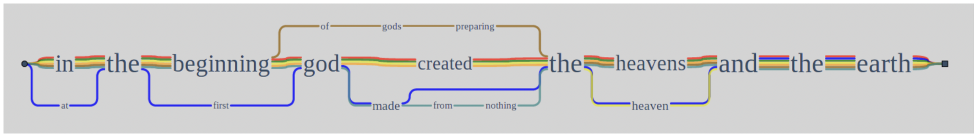


Figure 2.1: Alignment of Gen 1:1, six different bible translations.
Source: www.traviz.vizcovery.org/

A brief and overall description of TRAViz is based on both output from logging statements in the source code lending me insights into the flow of data, as well as on the dissertation describing the algorithm in detail [Jän16]. It is of course a software, written in JS, and utilizes libraries jQuery, Raphäel and Qtip. It visualizes graphs, by processing text data consisting of sentences from different editions of a text. Data is loaded into TRAViz and then sentences are immediately id'ed by their edition, and assigned a color. White space and unwanted characters are removed using Regular Expressions, they are normalized, and then single words (retaining their id's) are as such tokenized to become represented as nodes in the graph. These nodes, "vertices", have some different attributes, such as the text (token), and they of course have coordinates and they have editions, they have predecessor tokens, and successor tokens, as well as connections, ins and outs. These are then objects in the graph, whose arrays become populated during the processing. There are many and complex intertwined components in the graph, but the vertices are a main component.

The first processing after tokenization is the placement of the vertices, as they are first layered horizontally, according to how they can be merged. If tokens are alike, and they

[1]it should be noted that besides other references - all TRAViz code references are to Jänicke, Stefan (2014) TRAViz [Source code]. www.traviz.vizcovery.org/

can be merged without creating ensuing circling back, they will be merged. The larger the number of merges, the larger the size of the token (represented by fontsize). The first layer created is based on the edition which holds the most communal tokens, i.e. the most of the largest tokens, the main branch. The following layer is then the branch with most tokens in common with the initial layer (i.e. with the fewest tokens to place). And by iterations over all possibilities, all vertices are layered, and spread out as to both balance their placement in regards to each other, and to ensure there is place enough for all vertices, and by ensuring there are no vertices overlapping each other.

Edges in the graph represent paths between the vertices of each edition, and initially they may of course overlap vertices, and each other. To deal with this the edges are defined by type, depending on whether connection is straight and can be drawn without overlapping a vertex in the middle, or whether it must pass over one vertex, and connect to the next adjacent vertex, or if it must shift vertically to another branch. These three resulting possibilities, are determined for all connections, then they are ordered, both in form of preventing overlap as well as by attempting to simplify the connection-type, to that with the least amount of bends, giving a less complex graph structure. Eventually final adjustments can be made, to ensure connections do not overlap vertices, by shifting the vertex such that its position, which was decided by the initial layering and spreading, is only changed minimally.

TRAViz is configured with a larger number of optional user-configurations, and it allows for some user interaction with the graphs components. By mouseenter on an edge a tooltip will display the edition name represented by the given edge. By hovering over a vertex, it will display possible transpositions of that vertex, by encircling that and other transpositions by a black border, and by drawing a dotted line between these vertices. If clicking on the vertex and dragging, it can be merged with another vertex, which the user might perceive to have similar meaning. This merge, though, can only be performed, if that does not cause the directed graph to create a circle, in which case the merge will not be refused, and the user warned. Instead a split can also be created by clicking on a vertex, then a tooltip displays the number of editions attached, presenting their names in a table. Now the user can choose to separate the token of a particular edition from the graph, creating a new branch. This can then be exercised all the way to, or from the token, which the user could not merge, and thus can lead to the merge then becoming possible, if new branches make it possible, without creating a circle in the graph.

The source code is the practical application of the algorithm described above. None of this is changed in the modernization. The extension of the software by a multilingual analytic, is based entirely on the existing structure and internal logic of the the original code, and thus the modernized edition ”MTRAViz” is in fact an extension of TRAViz.

In the sections below I will first lay out an oversight of the modernization plan, which is then followed by subsections reporting more detailed code-examples comparing the original with new code, and a short section summing up the modernization process. Next in this chapter the addition of a multilingual analytic to the modernized code is described, using the same structure as the modernization section did: plan, process, and results. Finally in a last section I assess the overall result along the dimensions of 1) learning outcome, and 2) project outcome.

2.1 Modernizing: Plan, Process and Result

What I now present as a plan, is more a result of the afterthought of the processes of the modernization, as my background is not in coding, I was learning by doing.

The main tasks were: Modifying those elements of the source code which were using jQuery (and qtip). I furhter more decided that since Raphäel is no longer actively developed, I would also modernize those elements. Initially it was also part of the modernization plan to modularize the code further, but this was decided before I had enough insight into a develpment process of this kind, and any into JS coding, and as I will describe below, during an example, there has been very little modularization. Rather I learned that the original source-code was already semi-modularized, but I did not then have the competence, to determine that, when reading it.

After two failed attempts at modernizing the code, I did an online course on JS, which provided me with just enough of a foundation to decide a better approach. I was able to plan a more structured approach, and could gain better oversight of the processes I would have to go through. I prepared a dataset which was used at the same time on the old source code, and on a modernizing source code, which I was working on. Then I progressed by inserting logging statements in the old source code, and attempting to reproduce the result using slightly different methods, changing little things at first. Later I rewrote the entire code in a more complete effort to modernize, making it work reproduce the results of the original source code, by again comparing the results of comparable logging statements in both bodies of code. As time now also became of essence some of the modernization was rolled back and reverted more clearly older standards, in order to reproduce the exact same processing of data, as the old source code was undertaking.

The source code has now been carefully modernized, and is now more in line with modern JS standards. The code uses the class-method terminology, rather than the former prototype syntax. variable declarations are changed to ”let” and ”const” instead of ”var”. The modernized code now relies on Vanilla JS to access and/or affect DOM elements, where it previously relied on Raphäel.js (Raphäel), and Vanilla JS also manages eventhandlers, where the code previously utilized jQuery. The software no longer uses Qtip for styling of tooltips, that appear on user-interaction, but instead relies on d3 for accessing and binding nodes, and on CSS for styling, thus I have begun a decoupling of the styling, away from the visualization logics. All these changes have the purpose to not only adapt better to modern environments, but also to ensure that simply by enhancing readability of the code it is easier to access in future maintenance, and further development.

2.1.1 Plan

After some failed attempts, which then still made more familiar with the source code, the plan was to:

1. Generally modernize the code, such that it adhered with more current conventions of Javascript, thus enhancing readability
2. When encountering elements from jQuery, to substitute them with other solutions, mostly Vanilla JS, though in some places I might have used D3 as well, depending on the context. Eventhandlers for instance have been created with both, which is not optimal, as it should be the same across the code, but often I have simply used the method which would work. A sign that I might have lacked oversight in the process.
3. When encountering Raphäel, to find other ways to select and append items, and control visual renderings. Generally they were substituted for D3, in combination with style-definitions in CSS.
4. To substitute the Qtip elements, which in the original source code where handling both styling and interaction with the tooltip contents. These were substituted with also D3 in combination with CSS.

Generally I was expecting this to lead to a modernized code, with enhanced readability, and a higher level of modularity, written with current standards, and actively developed libraries.

2.1.2 Process: Class and Constructor Function

The concept of the Constructor in a Class in JS is a sort of blueprint, informing how to ”construct”: what is needed in this class, from other classes, and to assemble all you need in an initialized list, making all relevant objects, or arrays that will be populated or manipulated below, accessible inside the class [Cor24]. The logic is similar in both editions of the code, only syntactically they are somewhat different, and also the modernized code initializes in the constructor, what is in the original source code often initialized in the ”flow” when it becomes relevant. This means the modernized code lends you a more quick oversight, where as the old code lets you read ”along the flow” so to say. I now invite a concrete comparison, and refer you to the Listing 2.1 which is a such constructor function from the original source code.

```
function TRAViz(div,options){

  this.div = div;
  this.config = new TRAVizConfig(options);
  this.curveRadius = this.config.options.curveRadius;
  this.graph = new TRAVizGraph(this.config);
  this.startVertex = new TRAVizVertex(this.graph,'first','');
  this.graph.addVertex(this.startVertex);
  this.endVertex = new TRAVizVertex(this.graph,'last','');
  this.graph.addVertex(this.endVertex);
  this.graph.startVertex = this.startVertex;
  this.graph.endVertex = this.endVertex;
}
```

Listing 2.1: Initial Function in class in TRAViz

At the same time please also inspect Listing 2.2, which is an excerpt from the modernized code, of the comparable class/constructor function performing the similar task. Here the class is declared and a constructor, the blueprint, for the class is the initial function, and below this, then other methods written, can utilize what is initialized in the constructor directly. In this sense both codes perform this task in a similiar fashion.
A difference is also noticeable: the modernized constructor function initializes the arrays used across the class. These are "object-arrays" that are populated by all kinds of information about those objects, all refering back to the initial description of the algorithm, in the section above. The main difference in this modernization, is the instantiation of these objects within the confines of the instance, of this particular class. When a such array then is instantiated as below in Listing 2.2, later in methods inside the class, it is already defined which object is then used. In the original source code, if not previously defined like this - in a constructor context, an instantiation can then unknowingly be global. This particular modernization then, despite minor, also allows for more control over instantiations, as you simply ensure which instance is utilized, in the modern: it is the instance of this particular class.

```
class MTRAViz {
    constructor(div, options) {
        this.div = div;
        this.config = new MTRAVizConfig(options);
        this.curveRadius = this.config.options.curveRadius;
        this.graph = new MTRAVizGraph(this.config);
        this.originGraph = new MTRAVizGraph(this.config);
        this.startVertex = new MTRAVizVertex(this.graph, 'first', '');
        this.graph.addVertex(this.startVertex);
        this.endVertex = new MTRAVizVertex(this.graph, 'last', '');
        this.graph.addVertex(this.endVertex);
        this.graph.startVertex = this.startVertex;
        this.graph.endVertex = this.endVertex;


        this.edgeGroups = [];
        this.connections = [];
        this.vertexConnections = [];
        this.horizontalSlots = [];
        this.basicConnections = [];
        this.layers = [];
        this.layout = [];

    }
```

Listing 2.2: Class and Constructor Function in MTRAViz

2.1.3 Process: Methods

The decision to modernize using methods also was not a large deviation from the original code, especially since this is mainly a syntactic difference [Sim23]. This is then mainly done in an effort to adapt to current standards of JS [Int24], and also it becomes a consequence of using the class-constructor, as described in the section above.

```
1  function TRAVizConnection(v1,v2,type){
2    this.v1 = v1;
3    this.v2 = v2;
4    this.type = type;
5    this.links = [];
6  }
7
8  /**
9   * adds a link (vertical or horizontal) to the connection
10  */
11 TRAVizConnection.prototype.addLink = function(link){
12   this.links.push(link);
13 }
```

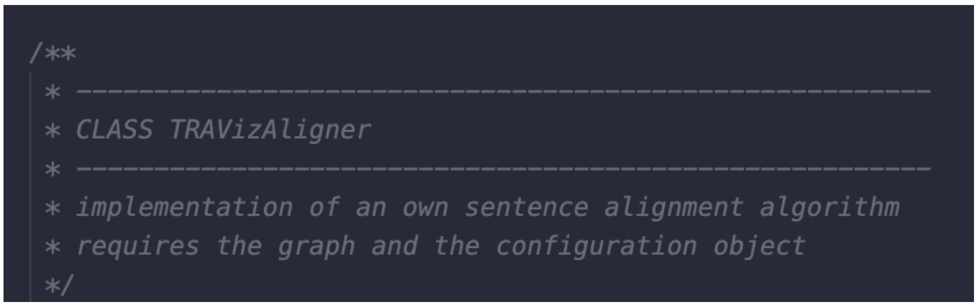
Listing 2.3: Prototype function in TRAViz

Listing 2.3 above contains an excerpt from the original source code using the pre-ES6 standards [Ama21]. In line 11 the prototype ”addLink” is declared (utilized when assigning the initial connection ”types” which were described above). In the modernized code, a function as this, was not defined as a prototype, instead it is integrated as a method in a class. This is exemplified by an excerpt from the modernized code in listing 2.4 below, see line 15. There is obvious syntactic difference, between the functions, but beyond that there is no difference. It was decided to implement this, as it followed logically from the class-constructor implications, and because it aligns better with current standards [Int24]. This way the decision also aligns with how the original developer abided by expectations during that period in JS development, where prototypes were described in the standards [Int11].

```
1  class MTRAVizConnection {
2      constructor(v1, v2, type) {


        this.v1 = v1;
        this.v2 = v2;
        this.type = type;
        this.links = [];
    }

    /**
     * Adds a link vertical or horisontal to the connection.
     * @param {Object} link - The link to be added.
     */
    addLink(link) {
        this.links.push(link);
    }
}

```

Listing 2.4: Method in MTRAViz

It is apparent from reading the original code, that in in the context of TRAViz, the developer also conceptualized and utilized classes, so I believe this modernization is very much in line with how the source code was intended by the creator. Figure 2.2 below also describes a class, despite (of course) not using the standards later set for declaring class-constructor functions.



```
/**
 * ------------------------------------------------------------
 * CLASS TRAVizAligner
 * ------------------------------------------------------------
 * implementation of an own sentence alignment algorithm
 * requires the graph and the configuration object
 */
```

Figure 2.2: Screenshot of Original TRAViz Source Code Documentation

2.1.4 Process: Vanilla JavaScript

Vanilla JS means pure JS without frameworks or additional libraries [Nuñ23]. This makes it lightweight, and it also easier to maintain, and more powerful, more efficient, and potentially it is more sustainable, also from a climate perspective [Wad23a; Per20a]. During the modernization the selections, manipulations, or renderings that could relatively easily be substituted from libraries to Vanilla JS have been so. An example is presented below, in Listing 2.5, where jQuery used in the original source code, is utilized for the temporary manipulation of data. A temporary tag ”¡p¿”, paragraph, is created around the sentence (passed as a parameter in the method utilizing these lines of code), and by the jQuery ”.text()” method, is then parsed as pure text, and reassigned to ”sentence” [Fou24].

```
sentence = $("<p>"+sentence+"</p>").text();
```

Listing 2.5: Utilizing jQuery for interacting with data - excerpt from TRAVizAligner class, normalize.prototype

Below, Listing 2.6., displays how data is manipulated in the same way, and ensuring the same resut. but this time using Vanilla JS. It is more verbose, but easily readable. Initially creating a temporary DOM elememt ”div” assigned to ”tempEl” (convetion-naming for ”temporary element”). Then the innerHTML of this div contents, now accessed by via the const ”tempEl”, is set to ”sentence”, the parameter of the method, from which this excerpt is taken. Finally the ”.textContent, or .innerText, or undefined/empty” is then assigned to sentence [Cor24].

```
        const tempEl = document.createElement('div');
        tempEl.innerHTML = sentence;

        sentence = tempEl.textContent || tempEl.innerText || "";
```

Listing 2.6: Vanilla Javascript - excerpt from MTRAVizAligner class, normalize method

This method is admittedly more verbose, than the equivalent jQuery utilization. This of course was not the reason for using Vanilla JS, but rather when a code only in a limited amount utilizes the contents of a large library like jQuery, then it is still more efficient using Vanilla JS. We must write a few more lines, but beneath the hood, less information is still processed, making the code more efficient. And in my opinion more readable, and easier to understand.

Another example of substituting jQuery is presented in the comparison between the original code in Listing 2.7, and the modernized equivalent excerpt in Listing 2.8. In 2.7 jQeury directly interacts with the DOM, using the ”.width()” method, to fetch the size of the width of the container assigning it to the variable ”width”.

```
TRAViz.prototype.insertDummys = function(){
  var gap = 3*this.curveRadius;
  var width = $('#'+this.div).width();
```

Listing 2.7: Utilizing jQuery for concrete DOM manipulation - excerpt from TRAVIz class, insertDummys.prototype

In Listing 2.8 the corresponding way to access this information using Vanilla JS is shown. The element is selected by its id, "this.div", and the attribute of this div, its "offsetWidth" is then fetched and assigned to the variable "width".

```
    insertDummys() {
        const gap = 3 * this.curveRadius;
        const width = document.getElementById(this.div).offsetWidth;
```

Listing 2.8: Vanilla Javascript - excerpt from MTRAVIz class, insertDummys method, in modernized code

2.1.5 Process: D3 and CSS

In the original source code Qtip has been used to display a tooltip, containing some data-related information. The display is activated "mouseenter", as can be seen in the Listing 2.9 below. Qtip then hides the tooltip again, once the user moves the mouse away from the jQuery selected element [Tho09]. Substituting jQuery is discussed in the subsection above, and following in this context I will also describe substituting Qtip in the code. The excerpt in Listing 2.9 also contains information that Qtip handles content-insertion, dynamic positioning relative to the mouse, and even the style is managed by Qtip. It is a truly powerful, jQuery-based, library.

```
for( var j=0; j<g.edges.length; j++ ){
      $(g.edges[j].node).qtip({
        content: {
          text: tiptext
        },
        style: {
          tip: true,
          border: { width: 0, radius: 4 },
          width : { min: 100, max: 500 }
        },
        position: {
          target: 'mouse',
          corner: {
            tooltip: "bottomMiddle",
            target: "topMiddle"
          },
          adjust : { x: -6 }
        },
        show: {
          when: 'mouseenter',
          solo: true
        },
        hide: {
          when: { event: 'mouseleave' }
        }
      });
    }
```

Listing 2.9: Handling tooltips with Qtip, excerpt from TRAVIz class, prototype.computeEdgelabels

Substituting all jQuery elements to better allow utilization of the code in more current environments was an initial task, and therefor also Qtip elements are substituted. This though, allowed for some elements of modularization to be introduced. In Listing 2.10 below you will notice that now styles are handled by, or set in, CSS, and the software can in the future easier be be adapted to either current trends, or perhaps organization-internal preferred standards. The appearance, style, is now set in CSS in ".tooltip", ".tooltip-edge", and ".edge-tooltip", by attributing tooltip-classes [Doc24].

Listing 2.10 also exemplifies how elements interaction is handled by D3. Selecting nodes, eventhandlers on mouseenter, and mouseleave, and the dynamic positioning of the tooltip on the page [Obs24]. It is very apparent from the comparison of the two listings that the second method requires more manually implemented elements, i form of styles. This gives more controll to the user, and it makes the code easer to maintain and adapt to unique needs, in form of styles. As I had already decided to use D3, it was then consequently used interchangeably with Vanilla JS for selection, and for binding objects.

```
d3.select(edgeNode)
           .on("mouseenter", function(event) {
               // Append the tooltip div to the body and position it
               body.append("div")
                   .attr("class", "tooltip tooltip-edge edge-tooltip")
                   .html(tiptext)
                   .style("left", `${event.pageX + 10}px`)
                   .style("top", `${event.pageY + 10}px`)
                   .style("visibility", "visible");
           })
           .on("mousemove", function(event) {
               // Update tooltip position on mouse move
               d3.select(".edge-tooltip")
                   .style("left", `${event.pageX + 10}px`)
                   .style("top", `${event.pageY + 10}px`);
           })
           .on("mouseleave", function() {
               // Remove the tooltip when the mouse leaves the edge
               d3.selectAll(".edge-tooltip").remove();
           });
```

Listing 2.10: Handling tooltips with D3 and CSS, excerpt from MTRAViz class, method computeEdgelabels

Raphäel is used to handle scalable vector graphics, SVG, and in Listing 2.11 below it shows its power in a few simple steps, by an immediate creation of a paper, the drawing canvas, and then right after setting the dimensions of this canvas, assigning it to a variable ”r”, now ready for rendering a visualization.

```
var paperDiv = $("<div id='TRAVizPaperDiv"+this.div+"'/>")
    .appendTo('#'+this.div);
var r = Raphael('TRAVizPaperDiv'+this.div,10000,10000);
```

Listing 2.11: Manipulating the DOM with Raphäel, excerpt from TRAVIz class, prototype.visualize

In this case the modernized method is far from as elegant. But at the same time the preparing and setting of the SVG becomes easier for the reader to detect and maintain. I have inserted a method into the MTRAviz class, shown below in Listing 2.12, where the SVG is created and appended to the div container, which size is then also set to match the div perfectly. This method is then invoked in the same context, where the original code utilized Raphäel to create the SVG.

```
initSvg() {
    const container = document.getElementById(this.div);
    if (!container) {
        console.error(`Div with id ${this.div} does not exist`);
        return;
    }
    const width = container.clientWidth;
    const height = container.clientHeight;
    this.svg = d3.select(`#${this.div}`)
        .append('svg')
        .attr('width', width)
        .attr('height', height);
```

Listing 2.12: Manipulating the DOM wit Vanilla JS and D3, excerpt from MTRAViz class, method initSvg

2.1.6 Process: General Modernization

In this final section I select a few archetypal changes inherent to current standards of JS [Int24]. They can all be exemplified by comparing the two equivalent prototype/method in the codes ”addEdgeToGroup” (used if a user-configuration sets bundling of edges). This function is now used as point of departure for comparing and describing the changes. The original function can be inspected in Listing 2.13, and the modernized in Listing 2.14. Archetypes of changed standards chosen are variable scoping, modernized loops, and strict equality.

The function in both cases initially flags variable ”found” as ”false”, a boolean to ensure that if ”found = true” (line 8, respectively 9), then an edge is added to the group, and the loop breaks. And if not found (line 12, respectively 15), a variable newGroup is created, given attributes, and pushed to edgeGroups

Listing 2.13 uses a for loop, iterating over the indices of the edgeGroups, assigning one group to the variable ”g”, which then is a reference point for the ensuing iteration, which is undertaken item by item, until all groups have had the role of ”g”.
In the modernized code, this is handled by a ”for of” loop, which can iterate over items directly, without the need of manually assigning a ”reference group” (the task of the var ”g”, of the original source code, line 4, Listing 2.13). Several of the functions that have been modernized have not taken so ”kindly” to modernization of loops, and often I have had to revert the modernization, and use older type iterations. But this modernization worked and it has made the function more readable.

```
1  TRAViz.prototype.addEdgeToGroup = function(h,t,e,ids){
2    var found = false;
3    for( var i=0; i<this.edgeGroups.length; i++ ){
4      var g = this.edgeGroups[i];
5      if( g.h == h && g.t == t ){
6        g.edges.push(e);
7        g.ids = g.ids.concat(ids);
8        found = true;
9        break;
10     }
11   }
12   if( !found ){
13     this.edgeGroups.push({
14       h: h,
15       t: t,
16       edges: [e],
17       ids: ids
18     });
19   }
20 }
```

Listing 2.13: Protype addEdgeToGroup of class TRAViz

Another change I choose to describe, is the use of ”var” for variable assignment in the

original source code. The modernized standards are instead ”let” and ”const” [Int24]. This ensures a more stringent assignment, as well as declarations of variables themselves. The ”var” syntax can have both global- and function-scope, depending on where declaration happens. It can be re-declared, thus become overwritten, and the contents can also be re-assigned. In the modern standards, the ”let” and ”const” declarations are scoped within a block. ”let” variables can get reassigned content, but not re-declared, and ”const” does not allow re-assignment, but contents can be updated [Int24].
By comparingthe Listings 2.14, and 2.14, we can note that the ”var” usage, does not let the reader know if ”found” or ”g” are temporary or constant variables. Listing 2.14 instead initially declares ”found” with the ”let” keyword, making it apparent that this boolean will be used for a temporary, and as we see conditional check, and can then reassign its value (to ”true”). This especially makes the code more accessible to decode with regards to the rather complex role of the let variables iterated over, and the temporarily assigned ”g”. The use of ”const” in the ”!found” condition Listing 2.14 line 15, is used to assign a variable newGroup, clearly informing the reader that this variable will not be changed, it is in fact a ”constant”, although of course its contents can be modified. Also the constant is block-scoped, so once pushed to the edgeGroups, it can only be accessed through ”this.edgeGroups”.

```
 1 addEdgeToGroup(h, t, e, ids) {
 2
 3         let found = false;
 4
 5         for (let group of this.edgeGroups) {
 6             if (group.h === h && group.t === t) {
 7                 group.edges.push(e);
 8                 group.ids = group.ids.concat(ids);
 9                 found = true;
10
11                 break;
12             }
13         }
14
15         if (!found) {
16             const newGroup = {
17                 h: h,
18                 t: t,
19                 edges: [e],
20                 ids: ids
21             };
22             this.edgeGroups.push(newGroup);
23         }
24     }
```

Listing 2.14: Method addEdgeToGroup of class MTRAViz

A last general difference is use of strict equality in the modernized code. I have chosen the use of strict equality in this context because it is precise and does not allow a type conversion. It was a clear aim to enhance readability in the source code, and this particular choice plays a small part in that overall aim. Using loose equality as it is done in Listing 2.13 can lead to type-conversion "under the hood" [Cor24], and the results of the algorithm looses transparency. For instance if groups are compared, and two groups are found where one has attributes, h = '10', and t = 1, and the other has attributes, h = 10, and t = 1, then by loose equality these would be found true, the process would break, and the comparison would move on to iterate over the next group. By using the strict equality we ensure the values are indeed equal before moving on, ensuring enhanced control.

2.1.7 Results

The main take-away is that as shown different method-uses or libraries, modules, frameworks can be used creating the exact same result. However, they also do need adjustments when comparing across methods. It was very important that by every step, I could compare results with a running version of the old source code, processing the exact same data, letting me insprect every process in detail, thus diagnose the precise origin of divergence, where my modernization lead to such,
There is still work to be done on the modernization, and as it is it constitutes a work in progress. For instance some edges, edition paths, can hide behind others in the visualization, this is shown in figure 2.3. below. This can be due to "over-bundling" of the edges, not allowing each edge to be displayed.

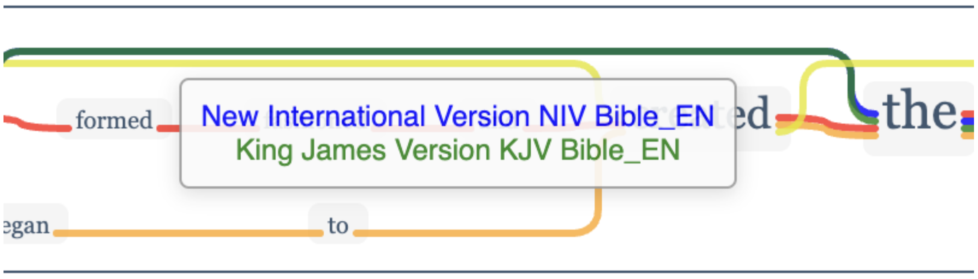

formed
New International Version NIV Bible_EN
King James Version KJV Bible_EN
ed
the
egan
to

Figure 2.3: Tooltip shown displaying edition-content of the green edge, exposing how the blue edge is almost hidden, MTRAViz.

By inspection of figure 2.4 it is also discernable, that edges are not completely smooth, certainly not as smooth as the original. This can be due to small differences in iterations for calculating the Bezier Curves between the original and the modernized source code, which are used for the D3 based generation of the visual paths.

Figure 2.4: Close-up of curvature on an edge, showing roughness, MTRAViz.

For a more full assessments of results I refer you to the hand-in of the modernized source code, or to inspect the code on the GitHub page, using the README to access a rendering with the example data.

2.2 MTRAViz: Plan, Process and Result

By modernizing the original source code I have gained a very good understanding of it, which made the implementation of a multilingual analytic component possible for me. I engaged on this with a plan, but there were several deviations, of which I will report some in the section below. Just as with the modernization, also this is a work in progress.

In the process section 2.2 , I will describe both the overall algorithm, as well as the development of two more detailed algorithms underlying it, and then briefly I will present the method utilizing these algorithms in the modernized source code.

I will describe the results and some drawbacks to decision made during the process, in the results section, which some images showing rending of the multilingual analytic.

2.2.1 MTRAViz: Plan

The work began structured with a ”tentative” plan:

1. I would create a simple interaction which could serve as example of analytics across language-based graph renderings. I decided to let a simple highlight enhance word-meaning similarities across the languages, and to ensure that no original user interactions would be affected by this. I also decided to let this interaction be integrated into an already existing user interaction when hovering over vertices, thus aligning with already created interactions in the source code.
2. Utilize information in the ”vertices” array, as it contained the tokenized words, that I would need to create the connections across languages. Both for translation-efforts, as well as of course for selecting them.
3. Create translation method for the data to be compared under the hood, on the basis of the token hovered over.
4. Create a user configuration that allowed the user to choose this added component.

5. Decide on precision measures: I decided to use synonyms not only direct translations.

6. Keep all helper functions in a Utilities file. If I had not been capable of modularizing much, during modernization, at least now I would, creating a sort of blueprint for further extensions of multilingual interest.

7. Create a method in the MTRAViz class, handling the process of decisions to hignlight tokens. This class handles visual display and user-interaction, and would be a fitting placement, and my implementation would thus only slightly impact on the existing source code.

2.2.2 MTRAViz Process: Creating an Algorithm

The process that ensued was of course not as linear, as I had hoped, but my previous work modernizing the code, was of great help when evaluating different attempts. Before being diverted by a lot of practical problems of implementation, I had an abstract algorithm in mind:

1. Translate tokens hovered over

2. Create a cluster of language-based synonyms

3. Include token hovered over in language-based array containing cluster

4. Create a method which would use the results from the translations, and the clusters of synonyms and translations, which would perform a check for token matches between clusters, and the tokens in the graph, eventually highlighting all matched tokens across all language-based graphs.

5. harmonize the method in the code structure, and letting it be invoked during processing of the already existing function, responsible for displaying tooltips by hovering over vertices, the setTooltip function, embedded in the visualize method, of the MTRAViz class.

Of course I had clearly under estimated the need for detailed and explicit steps in designing an algorithm!

I began by ensuring the basics for the language-based addition: a global access to the vertices and a way to sort them by language, without creating ripple effects in the rest of the code. I knew how complex the code was, but also how to avoid intervening with graph building. I could thus create only minor additions into existing methods, to capture the relevant information for selecting tokens (to translate), and for highlighting translations and synonyms (the actual analytic display of similarities across the graphs).

I added a language indicator to the data set, in all editions (a simple suffix indicating the language to each edition in the data). This way the vertices related to each language are specified, and can precisely be addressed. Next I created global access to the vertices object, and ensured the objects would be nested into this reference, based on language. Listing 2.15 below shows the the small addition to the the align method of MTRAViz class. This is point of entry for the data, and from there the data is then processed. Inserting it here, I had access to all information in the object.

```
if (languageSuffix && window.languageVertices[languageSuffix]) {
    window.languageVertices[languageSuffix].push(...this.vertices);
}
```

Listing 2.15: creating a global array, excerpt from align method class MTRAViz

Tokens across the graphs can now be accessed, for translation and selection and highlighting. This meant I was now ready to move on to the actual translations. I had envisioned, but not planned in detail, to use a python plugin for the translations. I had used this in a previous project, it utilized gooogletrans [Han20], a little plugin for creating state of the art translations, using Google Translate. But this then would mean, I now realized, that I would have to find means to execute the script from within the JS code, and it would also demand of a user, that they download other external modules, which would mean breaking with my aim at creating, in as far as possible, an independent software. I now deviated from the plan, deciding to look for other means of translation

This quickly evolved into a frustrating and time expensive divergence. It turned out that I had limited choices of translation, because good and fluent translations incur costs by access. And as I had obliged myself to present the code on GitHub, I did not want to include an open access to an API billing me, for every translation made by any random user. Eventually I went with a simple solution: I created an file containing token-relations, translations of the tokens in the data set. After working on implementing this, and integrating highlight effects, I realized this had some limitations: it would only work on the one dataset I had created, allowing future testing on different data, to be a tedious endeavour. I decided to find another method, returning to the initial plan of being able to not only translate but also to grow a ”synonym cluster”, highlighting not only translations, but also related concepts or words.

I now learned that also hard coded dictionaries cost money, and you are not allowed to share them widely. I needed something which I could legally share, which was free, and which could give the opportunity to create clusters of not only translations, but also of synonyms. Eventually I discovered Wiktionary [Ylo22; Ylo24], like its family-name indicates it is, as Wikipedia, a collaboratively gathered source, which contains dictionaries in several languages [Enc24].

Including synonyms concretely meant I was not only looking for word translations, but also for at means to create clusters of words with similar meanings. This I would then map across the language-based graphs, both in regards to direct translations, but preferably also to synonyms. For instance both from ”god” to ”gott”, but also from ”god” to "göttlich" (divine/celestial), or to "cielos" (divine/celestial). Due to this I was not just looking for combinations of dictionaries, but I was looking for all permutations. This means that order matters in this case, to grow as large clusters as possible. A simple enough math problem, I had learned during a Discrete Methods course, as the formula for permutations is given by[Ros19]:

$$P(n, r) = \frac{n!}{(n-r)!}$$

Where:

- n is the total number of items in our set (languages).
- r is the ordered item-set (dictionaries pr. language)

So in our case, where $n = 3$ and $r = 2$, the permutation is:

$$P(3, 2) = \frac{3!}{(3-2)!} = 6$$

A simple math problem to deduce, but practically I could find only four dictionaries:

- German-English
- Spanish-English and the item-set:

- Spanish-German
- German-Spanish

Even looking tokens up backwards (from for instance English to German, in the German-English dictionary), I assess this limits my cluster-sizes considerably, as from English to-dictionaries contain far more words than from Spanish, and some more than those from German [Wik24]. And I could not find a way to solve the problems of translating across conjugations in different languages, which is the cost of using dictionaries in this fashion.

But now at leas this pilot installation was prepared to be used with different data excerpts in the three chosen languages, and this was a good start. To compensate for the data insufficiency, both in form of actual translations, as well as in attaining synonyms, I decided to work on the cluster-creations, attempting to expand the amount of included (possible) tokens in the highlighter. This then meant I was still working intuitively from task to task without much thought of being guided or directed by my initial "abstract algorithm".

I wrote made a rather precise pseudo-code. A conceptual plan (also an example of an algorithm) to follow. The enumerated list below is an exact replica (except for correcting spelling errors) of the pseudo-code for the translateToken function, which can be found in the Utilities file of the modernized code:

```
if English
    1. save the token into the EN array –
    2. look it up backwards in the DE-EN
    3. save all German tokens found via backwards search in the DE array
    4. look the English toke up in the SP-EN
    5. save all Spanish tokens found via backwards search in the ES array
if German
    1. save the token into the DE array
    2. look it up in the DE-EN
    3. save all English tokens found via the look-up in the EN array
    4. look it up in the DE-EN
    5. save all Spanish tokens found via the look-up in the ES array
    6. look it up backwards in the ES-DE
    7. save all Spanish tokens found in the ES array
if Spanish
    1. save the token into the ES array
    2. look it up in the ES-EN
    3. save all English tokens found via the look-up in the EN array
    4. look it up in the ES-DE
    5. save all German tokens found via the lookup in the the DE array
    6. look it up backwards in the DE-ES
    7. save all German tokens found in the DE array
```

I was back following the steps of the initial "abstract algorithm", presented in the beginning of the section. It felt structured, and it worked well utilizing the posibilities I had to work with. The pseudo-code above is almost precisely how the function was also implemented. I did the same with a growSynonyms function for creating clusters, and this is also a replica from the Word file, where it was created:

```
take each entry in array EN
    1. look each entry in it up in DE-EN dict backwards
    2. add what comes up to the DE array
    3. look each entry up in the ES-EN dict backwards
    4. add what comes up to the ES array
take each entry in the DE array
    1. look each entry op in the DE-EN dict
    2. add what comes up to the EN array
    3. look each entry op in the DE-ES
    4. add what comes up to the ES array
take each entry in the ES array
    1. look each entry op in the ES-EN dict
    2. add what comes up to the EN array
    3. look each entry op in the ES-DE
    4. add what come up to the DE array
LAST iteration Take each entry in the EN array
```

1. look each entry op in the DE-EN dict backwards
2. add what comes up to the DE array
3. look each entry op in the ES-EN dict backwards
4. add what comes up to the ES array.

The "LAST iteration" is to compensate for the lack of dictionaries "from English to -". I also had to work a lot with this function, due to several problems integrating it and growing the clusters. In the actual code the process is more efficient, a small helper-function ensures the backward look-ups in the dictionary, when adding to the cluster (please see the attached code in the appendix, Utilities file, "growSynonyms function". This function also should be considered a work in progress, but the present edition, is the most efficient clustering attempt I have been able to write, giving only the slightest of pauses. For every token looked up, it will take the processing time of iteration over all entries in the dictionary, so that the larger the token-array, and the larger the amount of entries in the dictionary, the longer the time it will take to process. I have not been able to think of a way to make this process more efficient, without reducing the size of the clusters, and due to lack of dictionaries, and limitations related to conjugation, they are already very limited in size, thus also limiting the cross-language analytic considerably.

Finally in Listing 2.16 you can inspect the currently implemented method, which to a large degree matches the "abstract algorithm", in that it (by helper-functions, and invocation) highlights meaning-similarities of tokens in and across language-based graphs. This way it becomes a relevant extension, and internally in the graphs supplements the already implemented transposition functionality, which displays differently syntax internally in languages, and across graphs represents a multilingual analytic.

The hovered token is basis for all processing, and is initially assigned to a const "hoveredToken". A "Guard Clause" is inserted [ama23], to ensure that if the process is not initialised by a proper string then the processing aborts. In line 9, the language suffix is then extracted and passed to the const, to be used as a parameter, together with the hoveredToken, in the invocation of the translationCluster function, in line 11. In line 14 growSynonyms is invoked, growing the clusters. Lines 16-18 pushes hoveredToken to the cluster based on language, if it is not already present. Lines 19-30 constitute the process of matching, based on the "language-keys". Language by language the global array with the tokens is accessed, and a test performed; if the tokens IN the graph are included also in the tokens in the cluster, then they are highlighted.

```
1 highlightCrossLanguageMatches(vertex) {
2     const hoveredToken = vertex.token;
3 
4     if (typeof hoveredToken !== 'string') {
5         console.error('Invalid hovered token (not a string):',
6             hoveredToken);
7         return;
8     }
```

```
    const languageSuffix = this.getLanguageSuffix(this.editions[0]);

    let translationCluster = translateToken(hoveredToken,
        languageSuffix);

    translationCluster = growSynonyms(translationCluster);

    if (!translationCluster[languageSuffix].includes(hoveredToken)) {
        translationCluster[languageSuffix].push(hoveredToken);
    }
    Object.keys(translationCluster).forEach(language => {
        const matchingTokens = translationCluster[language];

        if (matchingTokens.length > 0) {
            window.languageVertices[language].forEach(vertex => {
                if (matchingTokens.includes(vertex.token)) {
                    d3.select(vertex.rect.node())
                    .style("fill", "lightgreen");
                    d3.select(vertex.textNode.node())
                        .style("font-weight", "400")
                        .style("fill", "darkgreen");
                }
            });
        } else {
            console.log(`No matching tokens found for language:
                ${language}`);
        }
    });
}
```

Listing 2.16: Method hightlightCrossLanguageMatches of class MTRAViz

2.2.3 MTRAViz: Results

The final result is presented in figure 2.5 below. It shows the result of a user hovering over the a vertex, by which the vertex itself and synonyms, or translations across the langauge-based graphs are highlighted. The test dataset contains bible excerpts from three different languages, English, Spanish, and German, each in five bible editions, and all edition excerpts are from Genesis 1:1 [sti23]. The addition of the interactive component did not aim at changing or modifying any other interactive components, it is based on a user configuration, where the user can select to use this analytic or not (true or false in the configurations).

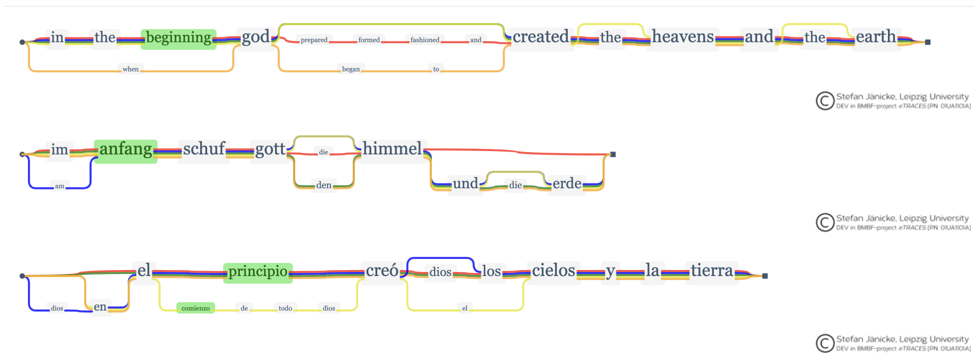


Figure 2.5: User-interactive component across language-based graphs, MTRAViz.

A drawback is the reliance of dictionaries. The lack of capacity to handle conjugations, can be seen in figure 2.6 where the implemented solution cannot reach across the conjugated German term ”schuff”, find the infinitive ”zu schaffen” (created/to create), and translate to ”created”, and ”creò”, which are then the matchin tokens in the Germand, respectively the Spanish graph.

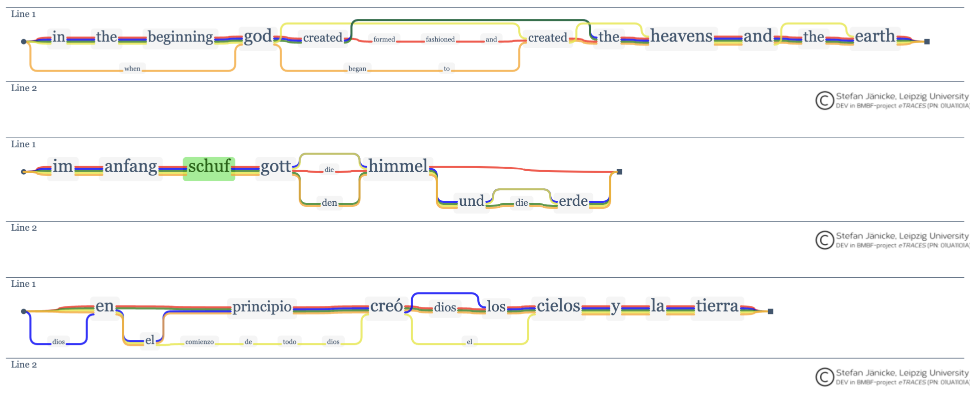


Figure 2.6: Hovering over the vertex ”schuff”, but the highlighter not showing relations to ”create” and ”creò”, MTRAViz

A more expanded algorithm in the clustering creation might be able to add certain often used conjugations, for instance by attempting to match to some stable and typical conjugations formats (suffixes for instance ”-ed” in English past tense is an example of this), these are often present in dictionaries, and could serve as a guide for designing an algorithm for this purpose. It would be tedious to implement, though, and it would still not catch all conjugations, as many languages have more exceptions than they have rules. A better solution would be to opt for a professional solution implementing an API based translation which in real-time could translate, like for instance Google Translate. This would incur costs upon the user, but it would add very competent translations, and of course synonyms could still be grown to heighten full capture of similar tokens in, and across languages. Perhaps a combination of the online translations, and the dictionary look-ups could then also be combined, heightening the final capture of highlighted tokens.

The choice to use a highlighter is based on the decision not to interfere with any originally developed and implemented configurations, and not to override any of these. TRAViz already possesses a similar component, the transposition-component. This allows the user to hover over a token, and if the graph then has that same token placed syntactically somewhere else in the sentence (the graph), this relation is visualized by encircling the tokens, and drawing a dotted line between these, please see figure 2.7. Allowing that feature, combined with the new extension presence at the same time, now combines the transposition of precisely matched tokens, with the highlights of possible synonyms both inside the singular graph, as well as across languages.

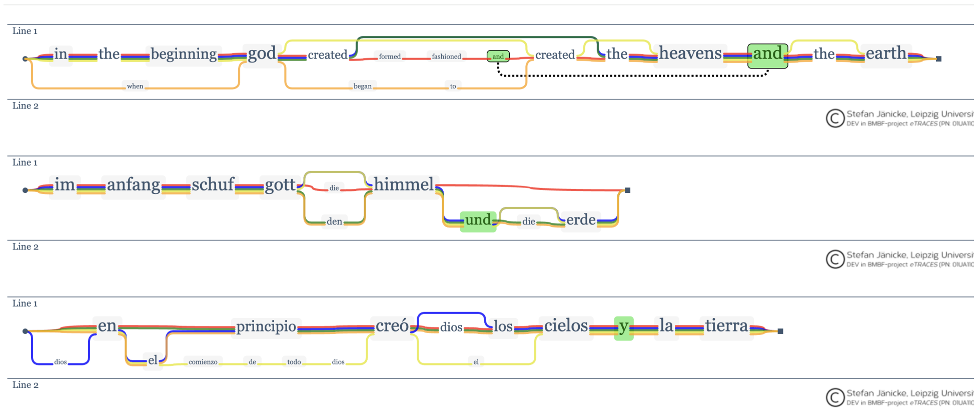

Line 1
in the beginning god created formed fashioned and created the heavens and the earth
when began to
Line 2
Stefan Jänicke, Leipzig Universi
Line 1
im anfang schuf gott die den himmel und die erde
Line 2
Line 1
en principio creó dios los cielos y la tierra
dios el comienzo de todo dios el
Line 2

Figure 2.7: Hovering over "and" with both highlights and transpositions shown, MTRAViz.

2.3 Modernization and MTRAViz: Overall Assessment

An overall assessment of both elements - the modernization and the implementation of a new feature in the modernized software can be taken along different dimension, I have chosen two: 1) learning outcome, and 2) product outcome. I will list the major outcomes by this order below

1. The learning outcome has been immense. I have had to both acquire understanding of a programming language, that I in the very beginning knew so little of, that I was surprised to learn that "Java" and JS are in fact NOT the same language. In addition I have learned about some major JS libraries, about its history, and about many components of web development, and about the differences of scripting

standards over the years, making me realize that of course there had to be standards ensuring compatibility, not only across interacting systems, but also across time. I thus also understood, that these very clearly described standards, were a basis for maintenance and development.

By learning by trial and error, I have also developed an understanding for the intricacies of project planning, of pseudo-coding, of algorithm-building, and I understand how sometimes even small development processes need these to keep them aimed at a purpose, and to restrict deviations.

2. Product-wise I consider it a roughly sketched preliminary product. The hard work has in many ways been done. The code logic(s), and the flow of data, is for a major part well understood - which is an important part of modernizing source code [Ros24].

   There is still much work to be done: finalizing the visual components, for instance as described "smoothing" the edges. Also deciding how to ensure space enough for several renderings on the screen. Other translator technologies should also be considered. There is considerable testing to be done, and collaborating with users. These are only some examples of what still needs doing. Another relevant consideration, but more on the outskirts of this project, and I mention it only because it relates back to the original idea of further delveloping TRAViz as a language learning tool, in that case implementing sound by configuration, could be a further development, such that the user can hear and audio, of a token spoken out.

   The code no longer relies on libraries that are either not in active current development, or which might gate-keep from utilizing it in local environments that conflict with it.

Summing up it has been a process which has taught me much, and it has resulted in a product which I consider a preliminary modernized version, but which certainly also has been updated and now relatively easy can be accessed and implemented in a modern environment, for further development.

Chapter 3

Documentation as Collaboration

Documentation of modern software consists of a multitude of sources which all serve to support ”design, implementation, comprehension, maintenance, and evolution”, and also contains sources such as: ”blogs, Twitter, stackOverflow, instant messaging apps, forums, Github README” [Rag^{+}23]. This provides a very broad definition, and obviously also with the potential to completely alter what might more traditionally be perceived as documentation, which would be based on thorough testing, transparency, and include coding examples, based on similar effects across different implementations, with use of different data, or at least with exceptions, and conditions there off clearly described. But even large commercial software and companies, who mainly adhere to this understanding of documentation, have to at least to some extend, had to adapt (implement and react) to the many open source platforms, that share and develop documentation as small examples, none tested unique occurrences, and so on [Nya21].
During the process of this project I have also used all these sources of documentation. I have coded along with bloggers, on traditional blogs, or on YouTube. I have accessed multiple GitHub repositories looking for coding examples, and explanations of software. I have used stackOverFlow, or commercial sites teaching code, and I have even relied on code-snippets shared by fellow students on a day to day basis. If course I have also extensively used more traditional documentation for the code, or the library itself, but to me those manifold sources of information and exemplifying usage of code, have been very important. I believe that without these, the field of programming, would have been inaccessible to me. Having a background in the humanities, I have needed to use any and every entry-point to coding, I could find, and most have been of tremendous help. Especially blogs, or those of larger commercial companies tutoring in programming, as they are part of what has made the traditional documentation more accessible to me, by delivering much in a low-level practical language, often only expecting limited technical know-how of the user.

Documentation also impacts on collaboration on development as they are closely intertwined [Sit23], and the popularity of collaborative repositories even depend on documentation [VC22]. Documentation is then not only what invites collaboration, but it is, according to the definition above, created in a collaborative effort and is thus itself a collaboration.

The following is based these two broad and interconnected definitions of both documentation an of collaboration in the field of programming. They define almost any interaction which heightens understanding of code, as both code collaboration and as code documentation. I will thus describe some examples of documentation, both in the report, in the modernized code itself, and on the platform, created to present the modernization, document the process, and the code, and invite to further collaboration.

3.1 Documentation in Report

During the development process I have used all accessible resources on the original source code intensely this has included the code itself, a website created for presentation, quite a few articles and the dissertation describing the algorithm in detail [Jän14; Jän$^+$15a; Jän$^+$15b; Jän16; Jän$^+$20]. This all constitutes very thorough documentation of the code, which I did not realize when beginning this work, because it seemed very inaccessible to me, as I had no coding experience. What I needed was a way to understand the overwhelming amount of documentation actually existing on the software.

While I wrote this report, which was done in steps during the whole project, I realized that writing it often made be gain a far better understanding of what I had done, or decisions made. Choosing between what to document and what not to, and deciding on good examples, and analyzing and describing them, and also trying to argue them with triangulation to official documentation sources, meant I had to, to at least to some extent, abstract my understanding of not only my own code, but of ”coding” generally. I also read some interesting articles regarding automation of code documentation [BS17; Hu$^+$22], and appreciating the extreme amount of time and effort, which documentation demands, I understand that good automation is somewhat of a holy grail. Not only to spare efforts of development, but also to ensure good documentation for a future maintenance, or modernization. I also appreciate that computer scientists, and software engineers are trained at generalizing tasks and at avoiding redundancy, thus are most effeciency experts in their own right. I understand that automating code documentation is a tempting exercise.

But my personal experience of documentation also means I believe we have to also consider this from another perspective as well: creating documentation, and considering fitting examples of tests might lead to better code, and less redundancy in the code-base, because I believe even experienced programmers will get to know their code better by working so intensively with it. When a software is large, it is hard to hold it in your head, and keep the necessary oversight, if you do not constantly return and abstract your understanding of it, and how it can be best mediated. I have also attempted to use Chat-GPT for creating in-code documentation, and it HAS taught me a lot of syntax, and has guided my understanding in what to ”write where”, but mostly when I have attempted to use it, I have found it so abstract, as to have almost no essential meaning to me. There is a level between abstract, and practical, which is maybe only humanly achievable. And in any case, the learning from creating documentation of code is immense.

Summing this section up, all of chapter 2, the report of the coding process, is an in depth example of documentation. The report as a whole lends insights to considerations, evaluations, and assessments, and the eventual decisions I have made during the process. The report is clearly sectioned with meaningful chapter- and section names - especially as regards the concrete coding examples, and constitutes a major part of the documentation.

3.2 Documentation in Code

The code itself has been been commented, and documented on the basis of JSDoc [Col24]. In Listing 3.1, is an example of a such formatted code-internal documentation. Some of this content was also in the original code, and some are added such as description of constructor parameters, and comments in line where I have found it relevant.

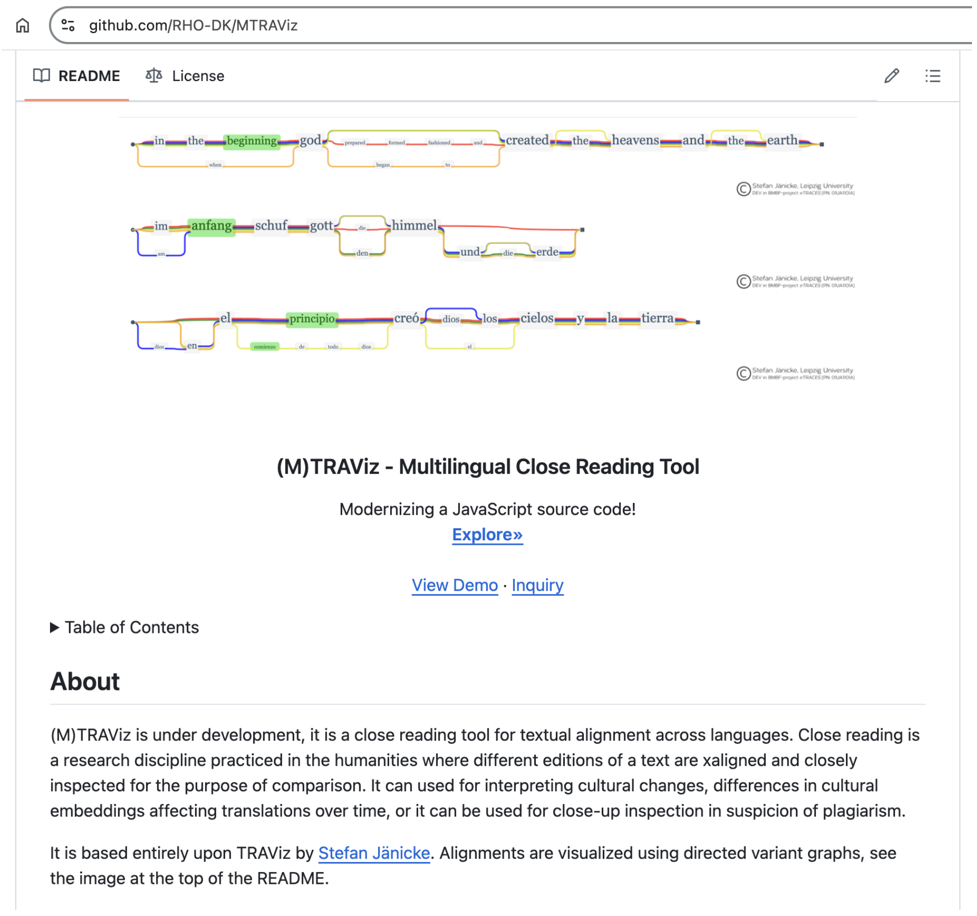
```
/**
 * --------------------------------------------------------
 * CLASS MTRAVizVertex
 * --------------------------------------------------------
 * Represents a vertex object in the graph.
 * Each vertex corresponds to a token in the text variant graph
 * holds connections to its predecessors and its successors.
 * Has been reverted mostly back to the original!
 */
class MTRAVizVertex {
    /**
     * Constructor for the Vertex
     * @param {Object} graph - The graph of the vertex
     * @param {number} index - The index of the vertex, in the graph
     * @param {string} token - The token of the particular vertex
     */
    constructor(graph, index, token) {
        this.graph = graph;
        this.token = token;
        this.successors = [];
        this.predecessors = [];
        this.count = 1;                 //frequency of the vertex
        this.traced = false;
        this.linked = true;
        this.sources = [];
        this.index = index;             //vertex index in the graph

    }
```

Listing 3.1: Documentation in the code, of class and constructor, MTRAVizVertex

Commenting and documenting during coding has been important for the modernization efforts, and for even understanding the code. The copy of the old code that I have used thougout the process is in some places commented line by line, as I understood the code. There is some evidence that working on documentation also enhances code quality [DR10a], and it certainly is a relevant element when modernizing or maintaining software [SDZ07a]

3.3 Documentation on GitHUb

I have created a GitHub repository which by a README presents the project, the code and other relevant information, and below I will go into a few details of this, and of course you are also invited to check it out at: https://github.com/RHO-DK/MTRAViz. This is of course all according to the definition both collaboration, and concrete code documentation.


github.com/RHO-DK/MTRAViz
README
License
in the beginning god created the heavens and the earth
im anfang schuf gott himmel und die erde
el principio creó dios los cielos y la tierra

(M)TRAViz - Multilingual Close Reading Tool

Modernizing a JavaScript source code!
Explore»

View Demo · Inquiry

▶ Table of Contents

About

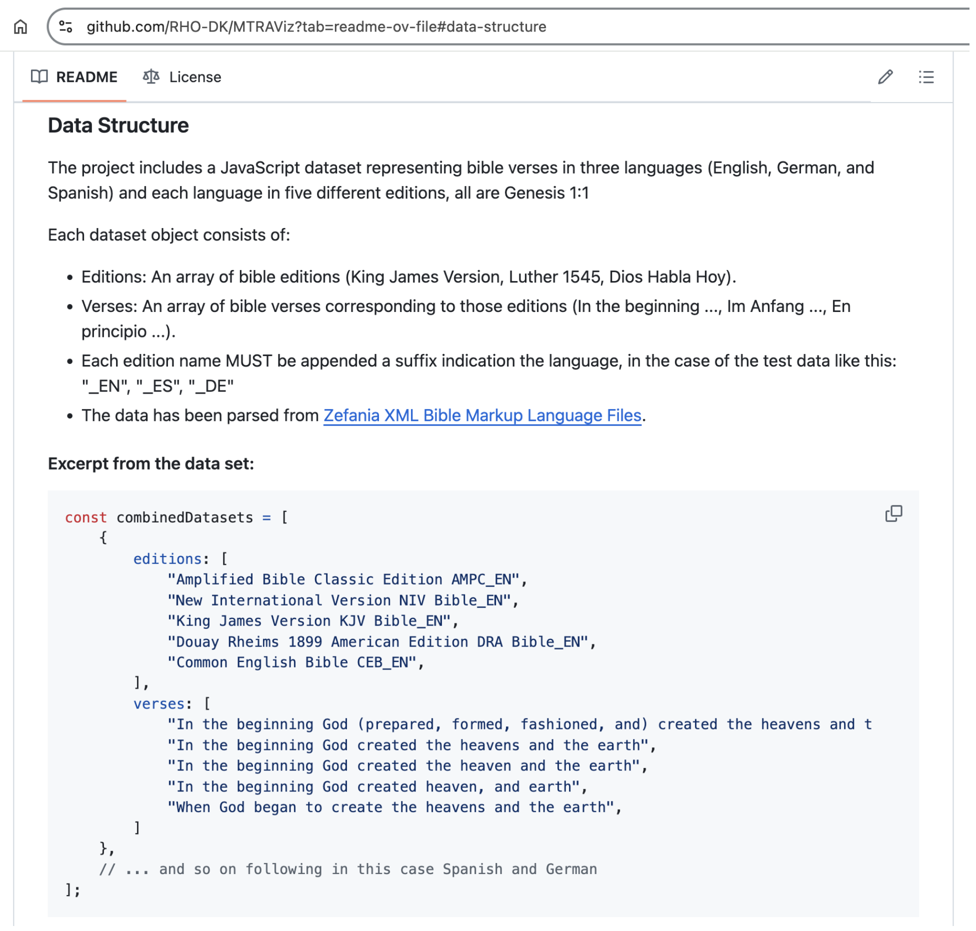
(M)TRAViz is under development, it is a close reading tool for textual alignment across languages. Close reading is a research discipline practiced in the humanities where different editions of a text are xaligned and closely inspected for the purpose of comparison. It can used for interpreting cultural changes, differences in cultural embeddings affecting translations over time, or it can be used for close-up inspection in suspicion of plagiarism.

It is based entirely upon TRAViz by Stefan Jänicke. Alignments are visualized using directed variant graphs, see the image at the top of the README.

Figure 3.1: README file on GitHub, "About".

Above in figure 3.1, I present the introductory ”About” in the README, which is very shortly sums up the intent of the code. There is also link to a gif, which presents a small demo of the visualization.

In figure 3.2 below you can see an excerpt of how the README describes the data structure. It also contains information about the translations, what they are, where they are from, and which structure they have. It presents the original work, it contains this report, and contact information, as well as information on the template i based the README file on [Dre24]



Data Structure

The project includes a JavaScript dataset representing bible verses in three languages (English, German, and Spanish) and each language in five different editions, all are Genesis 1:1

Each dataset object consists of:

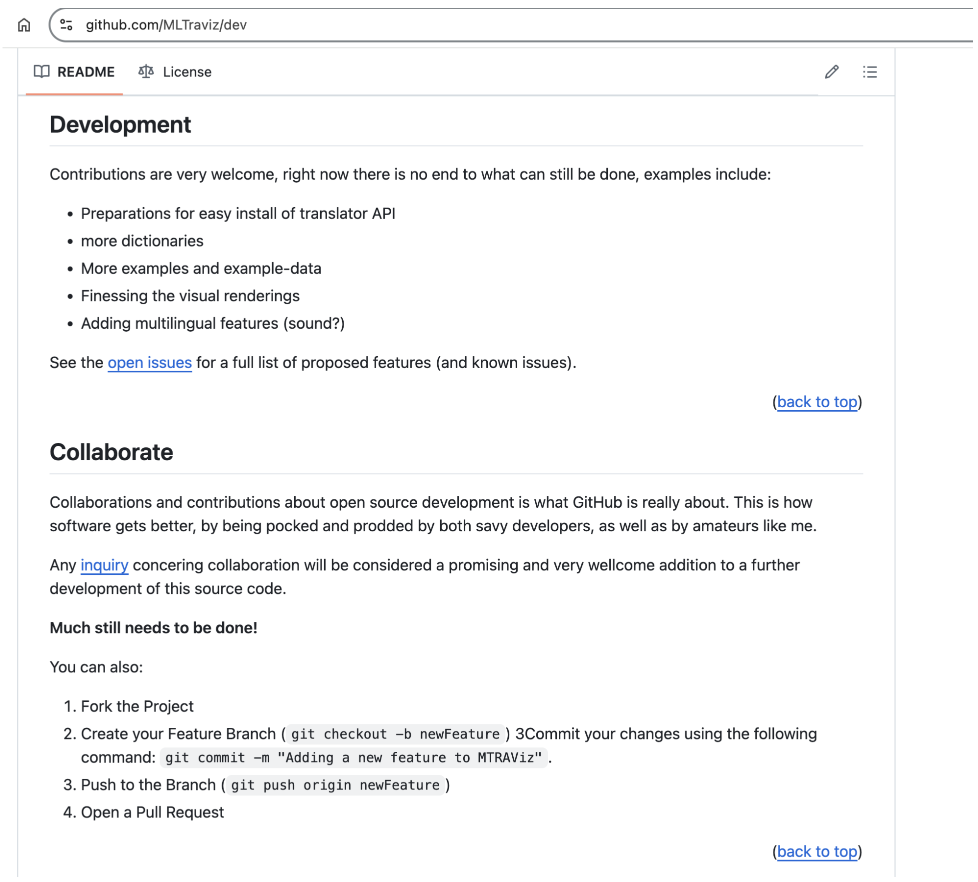
- Editions: An array of bible editions (King James Version, Luther 1545, Dios Habla Hoy).
- Verses: An array of bible verses corresponding to those editions (In the beginning ..., Im Anfang ..., En principio ...).
- Each edition name MUST be appended a suffix indication the language, in the case of the test data like this: "_EN", "_ES", "_DE"
- The data has been parsed from Zefania XML Bible Markup Language Files.

Excerpt from the data set:

```
const combinedDatasets = [
    {
        editions: [
            "Amplified Bible Classic Edition AMPC_EN",
            "New International Version NIV Bible_EN",
            "King James Version KJV Bible_EN",
            "Douay Rheims 1899 American Edition DRA Bible_EN",
            "Common English Bible CEB_EN",
        ],
        verses: [
            "In the beginning God (prepared, formed, fashioned, and) created the heavens and t
            "In the beginning God created the heavens and the earth",
            "In the beginning God created the heaven and the earth",
            "In the beginning God created heaven, and earth",
            "When God began to create the heavens and the earth",
        ]
    },
    // ... and so on following in this case Spanish and German
];
```

Figure 3.2: README file on GitHub, ”Data structure”.

These efforts all have the intent making it easy for potential collaborators to begin accessing the code and become involved in further developments. They are also directly invited to so, and instructed how, as you can see in figure 3.3. below. Important issues, and suggestions for new features are listed under the heading ”Development”, and under ”Collaborate” I share concrete information as to how collaboration can proceed.



Development

Contributions are very welcome, right now there is no end to what can still be done, examples include:

- Preparations for easy install of translator API
- more dictionaries
- More examples and example-data
- Finessing the visual renderings
- Adding multilingual features (sound?)

See the open issues for a full list of proposed features (and known issues).

(back to top)

Collaborate

Collaborations and contributions about open source development is what GitHub is really about. This is how software gets better, by being pocked and prodded by both savy developers, as well as by amateurs like me.

Any inquiry concering collaboration will be considered a promising and very wellcome addition to a further development of this source code.

Much still needs to be done!

You can also:

1. Fork the Project
2. Create your Feature Branch (`git checkout -b newFeature`) 3Commit your changes using the following command: `git commit -m "Adding a new feature to MTRAViz"` .
3. Push to the Branch (`git push origin newFeature`)
4. Open a Pull Request

(back to top)

Figure 3.3: README file on GitHub, "Development" and "Collaborate".

3.4 Collaboration as Documentation

An overall assessment of both elements - development and establishing collaboration will, as outlined in the introduction, and as already seen in chapter 2, also be taken along the same two dimensions 1) learning outcome, and 2) product. In recognizable manner I will list the major outcomes by this order below

1. Documenting my process, and writing the documentations in the code, and the README for GitHub, meant I had to iterate in my mean forth an back between the general level of an abstract understanding, and the unique practical implementation, has provided a considerable amount of learning.

   I have also moved from mainly using documentation from logs, or learning platforms or books, to mainly consulting the traditional code documentation. Now this is far more efficient, where at the beginning of my project most of it was incomprehensible. This has especially been a consequence of having to understand how to look for, and describe malfunctions in ones code, because it is necessary in order to find which solutions might prove helpful.

2. Product-wise I also consider this a relevant contribution to the software. The

explanations I have created might not help someone very savvy in programming, but they can provide a platform for an amateur, to access and utilize the software. By this they may also reach an interest in further developing it.

Summing up: in both dimensions the work has been relevant. It has created a good opportunity for a relevant learning outcome, especially because it was necessary to abstract from the actual implementation, to the level of creating a more generalized documentation. The outcome is a level of documentation suited for inviting collaboration by students of programming, as well as for those fields where this is a relevant tool.

Chapter 4

Conclusion

As outlined in the introduction, and utilized both chapters 2, and 3, I will draw these final conclusions along the two different dimensions 1) learning outcome, and 2) product outcome. For more detailed considerations and suggestions of further development, or for perspectives on other relevant elements of the report I refer to the relevant chapters and sections.

1. An overall learning outcome has been the tendency to none-linearity despite all intentions, in a development process. In 1976 two researchers in the evolving field of software engineering, set out three laws of Program Evolution Dynamics of large systems: 1) change [occurs] in adaptation to technological developments in hardware, as well as in its surrounding also adapting meta-system of other software, 2) increasing lack of stringent structure, as a result of these adaptations, and 3) continuing growth [BL76]. I initially considered this a guideline on what to avoid, and ended up realizing that once you decide on adapting a software and further evolving it, you inevitable also create a less structured, larger, even if technologically adapted system.

   This particular learning outcome has made me aware of another relevant area that I need to explore and learn to utilize if I want to continue learning to develop software: creating helpful flowcharts of "software processing" to create easily gathered oversight of how a system works. Also to plan for processes in development, would be helpful, especially a tool which easily lets you get from the practical implementation and back to the general aspect, and the oversight of a process. A visualization tool which could aid an easy switch between those levels of a development process, would be nice to know and utilize.

2. Product-wise I consider the development as "passing". The software has been modernized, and a method of multilingual analytics has been added. Both are well documented, and they both work well enough to be presented, and discussed for further development. It will be easy to create more data in the realm of the dictionary-languages, and to document more examples. To use this to find bugs,

or improve the described details, that have been determined to need more work. Documentation for the none-field-experts has also been developed, allowing easier initial access for collaborative efforts from those potential future and current users of TRAViz.

Along both dimensions it is my assessment that the plans have lead to relevant results. For further information I refer to the full source code, which has been handed in as external appendix, and to the GitHub platform, which also serves as documentation.

Bibliography

[Ama21] Amarachi Amaechi. *The bleeding edge of JavaScript classes.* https://blog.logrocket.com/the-bleeding-edge-of-javascript-classes/ [Accessed: (9.9.2024)]. 2021.

[ama23] amanv09. *Guard Clause in JavaScript.* https://www.geeksforgeeks.org/guard-clause-in-javascript/[Acessed: (september 2024)]. 2023.

[Ath+23] Sai Anirudh Athaluri, Sandeep Varma Manthena, VSR Krishna Manoj Kesapragada, Vineel Yarlagadda, Tirth Dave, and Rama Tulasi Siri Duddumpudi. "Exploring the boundaries of reality: investigating the phenomenon of artificial intelligence hallucination in scientific writing through ChatGPT references". In: *Cureus* 15.4 (2023).

[Bar20] Dmitry Baranovskiy. *raphael.* https://github.com/DmitryBaranovskiy/raphael [Accessed: (2024)]. 2020.

[BS17] Antonio Valerio Miceli Barone and Rico Sennrich. "A parallel corpus of python functions and documentation strings for automated code documentation and code generation". In: *arXiv preprint arXiv:1707.02275* (2017).

[BL76] Laszlo A. Belady and Meir M Lehman. "A model of large program development". In: *IBM Systems journal* 15.3 (1976), pp. 225–252.

[Cha24] ChatGPT. *OpenAI.* https://chat.openai.com/chat [Accessed: (juni-sep 2024)]. 2024.

[Col24] Collaborative. *@use JSDoc.* https://jsdoc.app/ [Accessed: (spring 2024)]. 2024.

[Cor24] Mozilla Corporation. *MDN Web DocsᴊavaScript.* https://developer.mozilla.org/en-US/docs/Web/JavaScript [Accessed: (jan - sep 2024)]. 2024.

[CAS24] Debby R. E. Cotton, Cotton Peter A., and Reuben J. Shipway. "Chatting and cheating: Ensuring academic integrity in the era of ChatGPT". In: *Innovations in Education and Teaching International* 61.2 (2024), pp. 228–239.

[DR10a] Barthélémy Dagenais and Martin P Robillard. "Creating and evolving developer documentation: understanding the decisions of open source contributors". In: *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering.* 2010, pp. 127–136.

[DR10b] Barthélémy Dagenais and Martin P Robillard. "Creating and evolving developer documentation: understanding the decisions of open source contributors". In: *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. 2010, pp. 127–136.

[Dar18] Frank P. Darveau. *You SHOULD Learn Vanilla JavaScript Before JS Frameworks*. https://medium.com/hackernoon/you-should-learn-vanilla-javascript-before-js-frameworks-88fb727ab362 [Accessed: (5.6.2024)]. 2018.

[Doc24] MDN Web Docs$_H$*TML*. *HTML: HyperText Markup Language*. https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/class [Accessed: (jan-sep 2024)]. 2024.

[Dre24] Othneil Drew. *Best-README-Template*. https://github.com/othneildrew/Best-README-Template [Accessed: (sep 2024)]. 2024.

[Enc24] HandWiki Encyclopia. *Wiktionary*. https://encyclopedia.pub/entry/32178[Acessed: (11.9.2024]. 2024.

[FSZ16] Joseph Feliciano, Margaret-Anne Storey, and Alexey Zagalsky. "Student experiences using GitHub in software engineering courses: a case study". In: *Proceedings of the 38th International Conference on Software Engineering Companion*. 2016, pp. 422–431.

[Fou24] OpenJS Foundation. *.text()*. https://api.jquery.com/text/text [Accessed: (jan-sep 2024)]. 2024.

[Guo24] Danny Guo. *The History and Legacy of jQuery*. https://blog.logrocket.com/the-history-and-legacy-of-jquery/ [Accessed: (5.6.2024)]. 2024.

[Han20] S. Han. *Googletrans PyPI)*. https://pypi.org/project/googletrans/description[Acessed: 2023]. 2020.

[Hu+22] Xing Hu, Qiuyuan Chen, Haoye Wang, Xin Xia, David Lo, and Thomas Zimmermann. "Correlating automated and human evaluation of code documentation generation quality". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31.4 (2022), pp. 1–28.

[IBM24] IBM. *What is Data Science*. https://www.ibm.com/topics/data-science [Accessed: (7.9.2024)]. 2024.

[Int11] ECMA International. *Standard ECMA-262 - 5.1 Edition - June 2011 - ECMAScript® Language Specification*. https://262.ecma-international.org/5.1/ [Accessed: (September 2024)]. 2011.

[Int24] ECMA International. *ECMAScript® 2015 Language Specifications*. https://262.ecma-international.org/6.0/sec-object.is [Accessed: (August 2024)]. 2024.

[Jän14] Stefan Jänicke. *TRAViz*. https://www.traviz.vizcovery.org/source.html [Accessed: (summer 2024)]. 2014.

[Jän16] Stefan Jänicke. *Close and Distant Reading Visualizations for the Comparative Analysis of Digital Humanities Data*. PhD Thesis, Leipzig University. 2016.

[Jän24] Stefan Jänicke. *Vizcovery.* https://www.vizcovery.org/ [Accessed: (spring 2024)]. 2014-2024.

[Jän+18] Stefan Jänicke, Judith Blumenstein, Michaela Rücker, Dirk Zeckzer, and Gerik Scheuermann. "TagPies: Comparative Visualization of Textual Data." In: *Visigrapp (3: Ivapp).* 2018, pp. 40–51.

[Jän+15a] Stefan Jänicke, Greta Franzini, Muhammad Faisal Cheema, and Gerik Scheuermann. "On Close and Distant Reading in Digital Humanities: A Survey and Future Challenges". In: *EuroVis (STARs)* (2015), pp. 83–103.

[Jän+15b] Stefan Jänicke, Annette Geßner, Greta Franzini, Melissa Terras, Simon Mahony, and Gerik Scheuermann. "TRAViz: A visualization for variant graphs". In: *Digital Scholarship in the Humanities* 30.suppl_1 (2015), pp. i83–i99.

[Jän+20] Stefan Jänicke, Pawandeep Kaur, Pawel Kuzmicki, and Johanna Schmidt. "Participatory Visualization Design as an Approach to Minimize the Gap between Research and Application." In: *VisGap@ Eurographics/EuroVis.* 2020, pp. 35–42.

[Kak21] Nikita Kakuev. *Understanding front-end data visualization tools ecosystem in 2021.* https://cube.dev/blog/dataviz-ecosystem-2021 [Accessed: (23.5.2024)]. 2021.

[KP23] Blanka Klimova and Marcel Pikhart. "Cognitive Gain in Digital Foreign Language Learning". In: *Brain Sciences* 13.7 (2023), p. 1074.

[Klø21] Åsmund Aqissiaq Arild Kløvstad. "Essay on History of JavaScript". In: *History of Programming Languages: Collection of Student Essays Based on HOPL IV Papers* (2021), pp. 38–42.

[Leg24] React Legacy. *Integrating With Other Libraries.* https://legacy.reactjs.org/docs/integrating-with-other-libraries.html/ [Accessed: (5.6.2024)]. 2024.

[Nuñ23] Leandro Nuñez. *The Evolution of JavaScript: From Vanilla to Modern ES2023 Features.* https://dev.to/digitalpollution/the-evolution-of-javascript-from-vanilla-to-modern-es2023-features-5bj0 [Accessed: (9.9.2024)]. 2023.

[Nya21] Hillary Nyakundi. *@use JSDoc.* https://www.freecodecamp.org/news/how-to-contribute-to-open-source-projects-beginners-guide/ [Accessed: (spring 2024)]. 2021.

[Obs24] Multiple Observable. *D3 API Reference.* https://devdocs.io/d3~6/ [Accessed: (summer 2024)]. 2024.

[PFY21] Chiara Palladino, Maryam Foradi, and Tariq Yousef. "Translation Alignment for Historical Language Learning: a Case Study". In: *Digital humanities quarterly* 15.3 (2021).

[Per20a] Morgan Persson. *JavaScript DOM Manipulation Performance: Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks.* https://www.diva-portal.org/smash/record.jsf?dswid=-3964&pid=diva23A1436661 [Accessed: (7.9.2024)]. 2020.

[Per20b] Morgan Persson. *JavaScript DOM Manipulation Performance: Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks*. 2020.

[Plo24] Plotly. *React for Python Developers: a primer*. https://dash.plotly.com/react-for-python-developers [Accessed: (september 2024)]. 2024.

[Rag+23] Marco Raglianti, Csaba Nagy, Roberto Minelli, Bin Lin, and Michele Lanza. "On the rise of modern software documentation (pearl/brave new idea)". In: *37th European Conference on Object-Oriented Programming (ECOOP 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik. 2023.

[00] *React Documentation*. httpAddr//legacy.reactjs.org/docs/integrating-with-other-libraries.html[Accessed: (2023)]. 2000–2099.

[Ros19] Kenneth Rosen. *Discrete Mathematics and Its Applications. Eight edition*. Eighth edition. New Your, NY: McGraw-Hill, 2019.

[Ros24] Omer Rosenbaum. *Modernizing Legacy Code: It Starts with Understanding*. https://overcast.blog/modernizing-legacy-code-it-starts-with-understanding-6271b04beae9[Accessed: (september 2024)]. 2024.

[Sar21] Iqbal H Sarker. "Data science and analytics: an overview from data-driven smart computing, decision-making and applications perspective". In: *SN Computer Science* 2.5 (2021), p. 377.

[SDZ07a] Daniel Schreck, Valentin Dallmeier, and Thomas Zimmermann. "How documentation evolves over time". In: *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting*. 2007, pp. 4–10.

[SDZ07b] Daniel Schreck, Valentin Dallmeier, and Thomas Zimmermann. "How documentation evolves over time". In: *Ninth International Workshop on Principles of Software Evolution: In Conjunction with the 6th ESEC/FSE Joint Meeting*. IWPSE '07. Dubrovnik, Croatia: Association for Computing Machinery, 2007, pp. 4–10. ISBN: 9781595937223. DOI: 10.1145/1294948.1294952. URL: https://doi.org/10.1145/1294948.1294952.

[Sch20] Maximilian Schwarzmüller. *Clean Code*. https://github.com/academind/clean-code-course-code [Accessed: (spring 2024)]. 2020.

[Sim23] Jonathon Simpson. *How JavaScript Works: Master the Basics of JavaScript and Modern Web App Development*. 1. 2023;1; Berkeley, CA: Apress, 2023.

[Sit23] Supavas Sitthithanasakul. "Understanding How Developers Present Code Snippets in README Files". PhD thesis. Nara institute of Science and Technology, 2023.

[Soh17] SM Sohan. *Automated example oriented REST API documentation*. 2017.

[Sta24] StatCounter. *StatCounter GlobalStats*. https://gs.statcounter.com/browser-market-share [Accessed: (5.6.2024)]. 2024.

[sti23] stingerone. *Zefania XML Bible Markup Language Files*. https://sourceforge.net/projects/zefania-sharp/files/Bibles/ [Accessed: (summer 2024)]. 2023.

[SKM23] Miriam Sullivan, Andrew Kelly, and Paul McLaughlan. “ChatGPT in higher education: Considerations for academic integrity and student learning”. In: *Journal of Applied Learning Teaching* 6.1 (2023), pp. 1–10.

[Tho09] Craig Thompson. *Qtip1.* https://github.com/Craga89/qTip1/blob/master/1.0.0-rc3/jquery.qtip-1.0.0-rc3.js~ [Accessed: (jan-sep 2024)]. 2009.

[THS23] Rie Troelsen, Hansen, and Pernille Stenkil. *Nyhedsbrev Marts 2023 - Undervisning og læring med AI.* https://www.sdu.dk/da/om-sdu/institutter-centre/c_unipaedagogik/liste_nyheder/nyhedsbrev_marts_2023 [Accessed: (june 2024)]. 2023.

[Uni24] Syddansk Universitet. *Vejledning - Brug Af Generativ AI i Undervisning og Eksamen.* https://mitsdu.dk/da/mit_studie/kandidat/civilingenioer_i_robotteknologi/vejledning-og-support/aipaasdu/vejledning-gai [Accessed: (june 2024)]. 2024.

[VC22] Akhila Sri Manasa Venigalla and Sridhar Chimalakonda. “An Empirical Study On Correlation between Readme Content and Project Popularity”. In: *arXiv preprint arXiv:2206.10772* (2022).

[W3T24] W3Tech. *Usage statistics and market share of jQuery for websites.* https://w3techs.com/technologies/details/js-jquery [Accessed: (7.9.2024)]. 2024.

[Wad23a] Malin Wadholm. *Green and Sustainable JavaScript: a study into the impact of framework usage.* https://www.diva-portal.org/smash/record.jsf?pid=diva23A1768632&dswid=1598 [Accessed: (7.9.2024)]. 2023.

[Wad23b] Malin Wadholm. *Green and Sustainable JavaScript: a study into the impact of framework usage.* 2023.

[War13] Colin Ware. *Information visualization: perception for design.* Waltham, MA Morgan Kaufmann, 2013.

[Wik24] Wikipedia. *Raw Data Wiktionary.* https://en.wikipedia.org/wiki/List_of_dictionaries_by_number_of_words[Acessed: (september 2024)]. 2024.

[Ylo22] Tatu Ylonen. “Wiktextract: Wiktionary as machine-readable structured data”. In: *International Conference on Language Resources and Evaluation.* European Language Resources Association (ELRA). 2022, pp. 1317–1325.

[Ylo24] Tatu Ylonene. *Raw Data Wiktionary.* https://kaikki.org/dictionary/rawdata.html[Acessed: (september 2024)]. 2024.

[YS21] Tariq Yousef and Jänicke Stefan. “A Survey of Text Alignment Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (Sept. 2021). //ieeexplore.ieee.org/document/9220137, pp. 1149–1159. DOI: 10.1109/TVCG.2020.3028975.

[Yu23] Hao Yu. “Reflection on whether Chat GPT should be banned by academia from the perspective of education and teaching”. In: *Frontiers in Psychology* 14 (2023), p. 1181712.

[Zha+23] Emma Yann Zhang, Adrian David Cheok, Zhigeng Pan, Jun Cai, and Ying Yan. “From Turing to Transformers: A Comprehensive Review and Tutorial on the Evolution and Applications of Generative Transformer Models”. In: *Sci* 5.4 (2023).

Appendix A

Project Description

**UPDATED - Project Aim

Further development of an open source analytics tool, as well as modernizing of code, such that the tool accommodates current browser based demands.

Tasks

To partake in developing methods for the alignment of multilingual texts which can be integrated with the Variant Graph based currently monolingual text alignment tool TRAViz [Jän$^+$15b].

- TRAViz needs to be modified to accommodate visual presence of several, or at least three languages.
- Variant Graphs tokenizes input text [Jän$^+$15b] such that the words become the nodes of the Variant Graphs. A possible solution for displaying several tokens by user interaction is using Tag Pies visualizing the contents of any node [Jän$^+$18]
- Multilingual textual alignment contain several challenges: should there be a main language? How can transpositions be displayed? How to tackle syntactic (structural) differences between languages [PFY21]. Any solution must also accommodate limitations in perception, and avoid cognitive overload [War13], and the interface should also ensure shifts in methods and contemplative tools to optimize positive cognitive development from language learning [KP23].

To partake in the integration of these methods such that they can work seamlessly with TRAViz.

- Integrating an extension into an already existent tool, evaluated during development, and which was build in an academic setting and in close cooperation with end users [Jän16], is an efficient way to use resources in development.
- There are concrete challenges to this decision that must tackle the issue of TRAViz being jQuery based, and therefore libraries that create virtual DOMs should possibly be avoided, excluding the otherwise immediate choice of Dash for an interface. [00]. A solution must balance the limited time period and the need for creating a seamlessly functioning prototype.

****TO BE OMITTED - To develop a method to evaluate efficacy both during and after development.**

****UPDATED - To create and maintain a project architecture allowing for working with collaborators on development and implementation.**

- Collaboration can help ensure a digital tool will be utilized in a targeted domain [Jän+20]
- Utilizing a collaboration platform such as GitHub is a relevant skill i Data Science [FSZ16].
- Version control will document progress [FSZ16], and will also create basis for better documentation and examples, that stem from working through development.
- Modularization of source code will enhance readability, and ultimately make a later update and modernization a more manageable task by contributing to both better documented, as well as divisible shares.

****NEW - To create documentation of code and implementation.**

- Documentation is relevant for using and maintaining code [SDZ07b]
- Examples are important in documentation as they save time and error solving efforts [Soh17].
- Preparing code documentation enhances final code quality [DR10b].

****NEW - To move away from frameworks: Vanilla Javascript**

- Using Vanilla JS minimizes carbon emissions compared to JS in/with bundles, libraries, or frameworks [Wad23b], and it enhances efficiency and rendering time, especially in Chrome [Per20b], which has a market share of more than 65% [Sta24]
- JQuery is simply not necessary and keeping the code pure means that in any local context, and possible different frameworks, it can be implemented [Dar18].

****NEW - To move from Raphaël: D3 or Vanilla**

- When already modernizing and adapting the code it is apparent to also move from Raphaël which is not maintained actively compared to D3 [Kak21]
- It is possible the process will reveal that it is preferable to only use Vanilla JS.

****NEW - Modularization of source code**

- When already modernizing and adapting the code it is apparent to also move from Raphaël which is not maintained actively compared to D3 [Kak21]
- It is possible the process will reveal that it is preferable to only use Vanilla JS.

**UPDATED - The Project

Taking on the task of updating and modernizing the source code of an analytics tool which was released in 2012, is an industry-relevant task, as it relates both directly to the profession of analytics, as well as to the reality of modern technology developing with an extremely high speed, necessitating constantly evolving source code, based on constantly evolving or even new languages.

As the method of the analytics tool and its application were all developed in extension of a larger Thesis [Jän16], and as such is well described, as is its relevance, this will not be a large focus of the assignment, and will entail only a short description of the tool itself. Instead focus is on the further development, documentation, organization, and modernization/updating of the existing source code, as described and referenced above.

Appendix B

Source Code

The source code has been handed in via SDU digital exam as a separate appendix, as it has not been converted to pdf format. In the file you will find a folder "no comments", which contains the uncommented code, in case you prefer to read that, for ease of access, for those that do understand and easily read code [Sch20]