Mika Alaoutinen

Developing interactive data visualizations for web UIs

Master's thesis of mathematical information technology

May 24, 2021

University of Jyväskylä

Faculty of Information Technology

Author: Mika Alaoutinen

Contact information: mika.a.alaoutinen@student.jyu.fi

Supervisors: Kaisa Miettinen, Giovanni Misitano and Johanna Silvennoinen

Title: Developing interactive data visualizations for web UIs

Työn nimi: Interaktiivisten datavisualisointien kehittäminen web-käyttöliittymille

Project: Master's thesis

Study line: Software Engineering

Page count: 76+10

Abstract: This thesis tackles the problem of developing interactive data visualization components that can be used in client-side web applications. The components are designed to be used in the context of multi-objective optimization, and especially with interactive methods. The thesis produced two new applications – desdeo-components and desdeo-frontend. Desdeo-components is a component library that includes several visualization techniques that were implemented with the Victory.js charting library. Desdeo-frontend is a single-page application that demonstrates how the components are used. Furthermore, desdeo-frontend may serve as a building block for creating a web user-interface for the DESDEO optimization software framework. The thesis explores different technologies that could be used for new applications and identifies design patters for handling user actions in a flexible manner.

Keywords: data visualization, interactive multi-objective optimization methods, DESDEO, web technologies, React, TypeScript

Suomenkielinen tiivistelmä: Tämän tutkielman parissa kehitettiin interaktiivisia datanvisualisointikomponentteja, jotka on tarkoitettu käytettäväksi asiakaspuolen web-sovelluksissa. Komponentit on suunniteltu monitavoiteoptimoinnin tarpeita silmällä pitäen, erityisesti interaktiivisten menetelmien tarpeet huomioiden. Tutkielma tuotti kaksi uutta sovellusta, joiden nimet ovat desdeo-components ja desdeo-frontend. Desdeo-components

on komponenttikirjasto, joka sisältää joukon visualisointitekniikoita. Visualisoinnit on rakennettu Victory.js-kirjaston pohjalta. Desdeo-frontend on single-page application -tyylillä toteutettu web-käyttöliittymäsovellus, joka demonstroi, kuinka komponentteja käytetään. Desdeo-frontend voi jatkossa toimia pohjana DESDEO-optimointisovelluksen web-käyttöliittymän kehitykselle. Tämä tutkielma kartoitti erilaisia web-teknologioita, jotka soveltuvat uusien sovelluksien tarpeisiin. Lisäksi tutkielmassa tunnistettiin suunnittelumalleja, joiden avulla käyttäjän antamia syötteitä voidaan käsitellä joustavasti.

Avainsanat: Datan visualisointi, vuorovaikutteiset monitavoiteoptimointimenetelmät, DESDEO, web-teknologiat, React, TypeScript

Glossary

AJAX	Asynchronous JavaScript and XML. A set of web development techniques used to make asynchronous queries from client to server.
API	Application Programming Interface. A method of abstraction that hides the underlying implementation details and only exposes a set of public actions that a software component supports.
ASP	Active Server Pages. Microsoft's server-side scripting language for creating dynamically generated web pages. Cf. JSP.
CI/CD	Continuous integration and continuous delivery or deployment. Processes that automate repetitive software development tasks.
CSS	Cascading Style Sheets. A style sheet language used to style HTML documents.
D3.js	JavaScript library for manipulating documents based on data. Used to create data visualizations that work in web browsers.
Decision maker	Person or group of people who can provide preference information and identify the best solution to a multi-objective optimization problem from a group of Pareto optimal solutions.
DESDEO	Open-source software framework for interactively solving multi-objective optimization problems. Developed by the Multiobjective Optimization Group at JYU.
Docker	Tool for creating and managing containers. A container is a standard package of software that includes source code and any dependencies needed to run the application. A container is a little bit like a lightweight virtual machine.

DOM	Document Object Model. DOM represents the structure and content of a web document. It is an API that allows programs to modify the content and style of a web page.
GUI	Graphical user interface.
HTML	HyperText Markup Language. A markup language used to construct web pages.
Interactive data visualization	Graphical data presentation technique that accepts user input to perform some meaningful action(s) with the data.
JavaScript	Programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. Enables dynamic behaviour on web pages.
JSP	Java Server Pages (old name) or Jakarta Server Pages (current name). A suite of technologies for creating dynamically generated web pages with Java.
JSX	JavaScript XML. An xml-like syntax introduced by React that allows HTML markup to be written directly into JavaScript code.
Multi-objective optimization	Optimization problem that has multiple conflicting criteria that need to be considered simultaneously.
Node.js	JavaScript runtime environment used to run JavaScript code outside a web browser.
npm	Node package manager. Can refer to: 1) A package management system for Node.js applications. Cf. Maven for Java or pip for Python. 2) The command line client for using the package manager. 3) The public package registry where Node packages are uploaded (npmsj.com). Cf. Maven Central for Maven libraries.

Pareto optimal	Optimization outcome where no criterion can be improved without making another criterion worse in multi-objective optimization problems.
PHP	General-purpose scripting language often used for web development.
Python	Interpreted general-purpose programming language.
React	JavaScript library made by Facebook for building web user interfaces. One of the most widely used technologies for frontend web development today.
REST	Representational state transfer. A software architectural style for stateless web services. A common way for applications to communicate over the Internet today.
Redux	JavaScript library for managing application state.
SDLC	Software development life cycle. A process for managing the life cycle of an information system, from initial planning to its end-of-life stage.
Storybook	JavaScript library for developing and displaying UI components in isolation. It could be described as a graphical UI component explorer. Works with many different web development frameworks, including React.
Transcompiler	Compiler that converts source code from one programming language to another. This process is called transcompilation. Also referred to as transpiler.
TypeScript	Programming language developed by Microsoft. It is a strict superset of JavaScript that adds static type definitions. TypeScript code transcompiles into JavaScript for execution.
UI	User interface.

Victory.js	React-based modular component library for creating data visualizations. Used as the basis for the custom data visualization components developed in this thesis.

List of Figures

List of Tables

Contents

1 Introduction

The subject for this thesis is developing interactive data visualization components that can be used in a web browser. Although the components should work with arbitrary data sets, they are primarily intended to be used in the context of multi-objective optimization, which has implications for the functionality of the components.

Multi-objective optimization tackles optimization problems where there are multiple conflicting criteria that need to be considered simultaneously. Due to the conflicting criteria, multi-objective optimization problems typically do not have a single optimal solution, but rather several solutions that are mathematically equivalent. Hence, solving a multi-objective optimization problem often requires input from a human decision maker, who can identify the best solution from the group of mathematically equivalent solutions candidates. Finding the most preferred solution often requires the decision maker to study the problem and gradually build his or her understanding of the problem and its solutions.

This thesis aims to develop tools to assist decision makers in their task. Some features that are useful for multi-objective optimization include support for exploring data sets to discover patterns and interdependencies between objectives, ability to filter uninteresting solution candidates, having a convenient way to provide preference information and providing a way to indicate that a solution is interesting. Hence, the components should look to support these types of user actions. Due to the emphasis on interactive visualizations and the medium of the web, this thesis aims to bring together the fields of multi-objective optimization, data visualization, and web development in a rather novel way.

The data visualization components developed in this thesis were released as a public open-source Node.js library on the Node package registry, which allows anyone to use them in their Node applications. In addition to the new component library, the thesis work includes creating a prototype web application that demonstrates the use of the new components. The prototype application could serve as the starting point for developing a web user interface (UI) for the DESDEO optimization framework. The source code for both the component library and the frontend application is available on GitHub (see Appendix B).

With this thesis, I hope to take a step towards lowering the barrier of entry into using the existing optimization tools with a modern web UI. Although providing any kind of graphical user interface (GUI) for existing multi-objective optimization tools would surely be valuable, I believe the approach of going with a web application should contribute to the novelty of this thesis. Taking a quick look at existing web-based visualization tools for multi-objective optimization, there does indeed seem to be a dearth of choice. To highlight some prior work, there is an interesting master's thesis by Kodžoman (2018), who developed a visualization framework for optimization algorithms using Jupyter Notebooks. Another interesting example is the Parallel Coordinates library (Chang n.d.), which is impressive work and could certainly be applied to the context of multi-objective optimization, however it is only a single visualization technique. Looking past web applications and into desktop applications for further inspiration, Hägele (2019) showcases an impressive looking Java application for visualizing optimization trajectories. With this thesis, I hope to test the waters regarding the viability of building complex, data intensive UIs with modern web technologies.

The overall structure of the thesis is as follows: Chapter 2 starts of by outlining the research problem. Chapters 3–7 present the theoretical background for the thesis. The topics discussed are multi-objective optimization, data visualization, the DESDEO software framework, a brief introduction to web development and an overview of data visualization with web technologies. Chapter 8 discusses the research approach, which was design science. Chapter 9 goes over the software requirements that were identified. Chapters 10 and 11 make up the more technical portion of the thesis, with Chapter 10 presenting the technologies that were used and Chapter 11 establishing key design principles for the new applications. Arriving at the home stretch, Chapter 12 summarizes the results of this thesis. Finally, Chapter 13 concludes the thesis with some suggestions for further research.

2 Research problem

This chapter presents the research problem of creating a protype web UI for visualizing data in the context of multi-objective optimization. First, I will briefly explain the motivation behind selecting this thesis topic. Next, I will list the research questions for this thesis, followed up by some discussion on the overall scope of the thesis. The chapter concludes with a note regarding risks associated with this thesis.

2.1 Motivation for the thesis

The topic for this thesis stems from the observation that there is a lack of good UI implementations for multi-objective optimization software (Tarkkanen, et al. 2013). The lack of suitable GUI software is unfortunate because there has been a fair amount of study into what types of visualization techniques could be used in the realm of multi-objective optimization and the relative strengths and weaknesses of the different techniques. Some examples of such work include Miettinen (2014), Tušar (2014), and Korhonen and Wallenius (2008). Unfortunately, these ideas have rarely been brought to life with tangible software. Indeed, Tušar (2014, 79) concludes her dissertation by raising the software implementation of the visualization methods as the "most important future work direction". As such, the groundwork – at least from the perspective of suitable visualization techniques – for a multi-objective optimization GUI software exists but is yet to be realized.

The lack of GUIs also raises the barrier for using the existing multi-objective optimization tools that are out there. One such tool is the DESDEO software framework, which is the context for this thesis. Considering the many applications of multi-objective optimization in, for example, different engineering disciplines, medicine and supply chain management (Stewart, et al. 2008), it seems reasonable to assume that having access to an optimization tool like DESDEO could be of interest to parties both in and outside of academia. Unfortunately, the current tools remain somewhat unapproachable due to the lack of a human-friendly UIs.

2.2 Research questions

Broadly speaking, the aim of this thesis is to first identify a set of technologies that can be used to build interactive data visualizations that work in web browsers, and then to build a prototype that demonstrates the use of those technologies. In addition to exploring different technical options, I expect to gain further insight into what types of limitations web applications impose for interactive visualizations and how much development effort is required to build functional data visualization components.

The research questions for this thesis are:

1. How feasible is developing interactive data visualizations for the web? Users should be able to interact with the visualizations in different ways; some examples could be making selections by mouse clicks or box selections, panning and zooming, and re-arranging elements by dragging them with a mouse.
2. What technologies (programming languages, frameworks, libraries, etc.) would make a good fit for this type of an application. Finding viable libraries for developing interactive charts is of particular interest.
3. How should the communication between the different components, or a component and the hosting frontend application, be handled?
4. How should the global application state be managed to facilitate chart reactivity? That is, how can it be ensured that the charts react fluidly to any changes in the data set that they display?

2.3 Scope and focus of work

Although the above section lists several tangible research questions, the initial scope and focus of this thesis were not very clearly defined at the start of the thesis project. This was partly explained by the close connection to on-going research and development and partly because of the need to find out what kind of approaches and tools would be feasible for the needs of the new software. There was a vision of producing a new client-side web application that connects to an existing server-side application, however there were many un-

knowns in the initial idea. As such, this thesis was rather experimental in nature, and a major focus point was exploring and prototyping technical solutions. In addition to the more programming-oriented work, the thesis involved mapping system requirements, exploring suitable technologies, doing technical and architectural planning, writing automated tests and, of course, reporting the results with this thesis. These activities – excluding reporting – are also found in the numerous software development life cycle (SDLC) models that have been suggested over the years, starting from Royce's (1970) paper that is often attributed as the foundation for the waterfall model. These different activities produced various artifacts for this thesis, including:

- software products,
- reference architecture for the software,
- knowledge of technical alternatives and their pros and cons,
- further insight into the challenges of developing this type of software.

2.4 Risks related to this thesis

Before moving on to the theory section of this thesis, I want to briefly highlight some risks that were identified with this thesis. Firstly, it was initially quite difficult to estimate the overall effort required to carry out the different SDLC activities discussed in the previous section, which resulted in a fair amount of uncertainty regarding the artifacts that this thesis could produce. In hindsight, it may have been useful to establish clearer priorities for the different artifacts. Secondly, while there are many data visualization libraries suitable for web applications, they often have little support for customizable event handling. As such, achieving a sufficient degree of interactivity to user actions with the visualization components was recognized as a potentially difficult technical problem to overcome. This initial lack of knowledge regarding suitable technology choices contributed to the unclear thesis scope, as was discussed above. Thirdly, DESDEO's backend application had no public application programming interfaces (APIs) at the time of writing this thesis. This meant that planning the integration between frontend and backend applications was based on (educated) guesswork on how the backend service might eventually work.

3 Multi-objective optimization

This chapter serves as an introduction into the world of multi-objective optimization. The first section defines some basic concepts of multi-objective optimization. The second and third sections introduce noninteractive and interactive optimization methods. The fourth and final section discusses the role of a human decision maker in more detail.

3.1 Defining concepts

To start from the basics, *optimization* is "the task of finding one or more solutions which correspond to minimizing (or maximizing) one or more specified objectives and which satisfy all constraints (if any)" (Branke, et al. 2008, V). In short, to optimize something is to select the best option from a group of options. Optimization problems can be divided into two categories: *single* and *multi-objective optimization* problems. Branke et al. (ibid.) explain that a single-objective optimization problem involves only one objective function to be optimized, which usually produces a single, optimal solution, whereas a multi-objective optimization problem is a problem with multiple objective functions that need to be optimized simultaneously. These objective functions usually conflict with each other, which means that improving the value of one objective function impairs the value of one or more other objective functions. The optimization problem may also contain constraints that must be met for a solution to be viable (Deb 2008, 67). Thus, the task of solving a multi-objective optimization problem is about finding the right balance between the different objectives to reach the best possible solution.

In contrast to single-objective optimization problems, multi-objective optimization problems typically do not have a single optimal solution, but rather a set of mathematically equivalent solutions (Miettinen 2014, 1–2). This is because in non-trivial multi-objective optimization problems, it is not possible to find a feasible solution with the optimal value for every objective function. The group of mathematically equivalent solutions is called the *Pareto set* or *Pareto frontier*, and finding it is referred to as reaching a Pareto optimal or Pareto efficient solution. A Pareto optimal solution is one where no objective function can be improved without making another objective function worse.

Multi-objective optimization falls under the category of *multiple criteria decision-making* problems. According to Miettinen (1999, xiii), multi-objective optimization problems can be divided into two distinct types, based on the properties of the feasible solutions: *multiattribute decision analysis* and *multiobjective optimization*. She clarifies that in multiattribute decision analysis, there is a finite and predetermined set of feasible solutions, whereas in multiobjective optimization, the set of feasible solutions is infinite, and the potential solutions are not known in advance. Miettinen (2014, 2) mentions choosing a car is an example of the former, whereas investing money into a stock portfolio is an example of the latter.

Since multi-objective optimization problems typically have multiple mathematically equivalent solution candidates (i.e. the Pareto set), reaching a satisfactory solution often requires evaluating and choosing one solution from the candidate pool as the most preferred one. Branke, et al. (2008, V) emphasize that the optimization task of finding a Pareto set and the decision-making task of choosing a most preferred solution are equally important. This selection task introduces ambiguity into the problem-solving process, because finding the best solution suddenly becomes a fundamentally subjective task that requires input from a human decision maker. This human element is one of the defining features of multi-objective optimization as a discipline, and it is, in my opinion as a software developer, the most interesting aspect of multi-objective optimization. The role of the decision maker will be discussed further in Section 3.4.

3.2 Noninteractive methods

The many multi-objective optimization methods can be classified into *noninteractive* and *interactive* methods. The focus of this thesis will be primarily on interactive methods, which will be discussed in the following section, however the basic ideas of noninteractive methods will be briefly presented here.

In noninteractive methods, there either is no decision maker present, or the decision maker is only able to specify preference information before or after the solution process (Miettinen 2008, 1–3). These three scenarios make up three sub-categories of noninteractive methods: *no-preference*, *a priori* and *a posteriori* methods. Miettinen (2008, 3) de-

scribes the three sub-categories as follows: In no-preference methods, there is no decision maker, and therefore a reasonable and neutral compromise solution should be found with no preference information. In a priori methods, the decision maker articulates any preference information before the solution process begins, and an *analyst* produces the best Pareto optimal solution that complies with the given preference information. An analyst is a person or a computer program who performs the mathematical modelling and computing involved in the solution process (Miettinen 2008, 2). A posteriori methods work in the reverse fashion – the representation of the Pareto set is discovered first, after which the decision maker performs the final selection (Miettinen 2008, 3).

3.3 Interactive methods

Interactive methods are the most extensive class of multi-objective optimization methods (Miettinen 2008, 3). They are iterative in nature, and they work by presenting interim solutions to the decision maker, who can provide preference information to help guide the optimization algorithm (also called solution process) towards more preferable solutions. The process of forming interim solutions and asking for feedback can be repeated multiple times, until a final, most preferred solution is reached (Miettinen, Ruiz and Wierzbicki 2008, 27–28).

The typical process for solving a multi-objective optimization problem using some interactive method has six steps, as described by Miettinen, Ruiz and Wierzbicki (2008, 28) and Miettinen (2014, 4). The steps are:

1. Initialization, which can involve calculating the lowest and highest objective function values in a Pareto set, for example.

2. Generating a Pareto optimal starting point. This can be done by starting with a neutral compromise solution or by asking the decision maker to provide the starting point. A neutral compromise solution is a point that is projected "'somewhere in the middle' of the ranges of objective values in the Pareto optimal set" (Miettinen 2008, 14). Such a point can be created by taking the average of the highest and lowest (*ideal* and *nadir*) values of each objective function (Miettinen 2008, 14–15).

3. Acquiring preference information from the decision maker to guide the solution process. The preference information can be, for example, desirable values of objective functions (aspiration levels) for some of the objective functions or the desired number of solutions for the next iteration.

4. Generating new Pareto optimal solution or solutions based on given preference information. The new solutions are presented to the decision maker, who is asked to select the best solution so far as the current solution.

5. Termination, if the decision maker is satisfied with the current solution.

6. Otherwise, continue the solution process from step #3 and perform the next iteration.

Tackling optimization problems in an interactive manner has the notable advantage of allowing the decision maker to adjust his or her preferences between each iteration, while also gaining a better understanding of the problem and the interdependencies that the different objective functions may have (Miettinen 2008, 3). In contrast to the noninteractive a priori and a posteriori methods, interactive methods also do not require the decision maker to have a global preference structure about the problem at hand (Miettinen, Ruiz and Wierzbicki 2008, 28). This means that it is not necessary to calculate all possible Pareto optimal solutions to find the most preferred one, but rather suitable solutions can be discovered gradually, based on the preference information provided by the decision maker. Not having to consider all potential solutions is beneficial both to reduce unnecessary computation work and to make the decision-making process less taxing for the decision maker, since they have fewer alternative solutions to compare (ibid.).

3.4 The role of a decision maker

A *decision maker* is a person or group of people who is qualified to make an informed decision on which of the options in a (Pareto optimal) solution set is the best solution for the given problem (Miettinen 2014, 4). A rational decision maker will always produce a Pareto optimal final solution (ibid.). The decision maker often relies on an analyst to help him or

her in the optimization process, for example by eliciting preference information and by interpreting the computation results and perhaps presenting them in a helpful manner (Miettinen 2008, 2).

From the perspective of the decision maker, the process of solving a multi-objective optimization problem with an interactive method often involves two distinct phases: a *learning phase* and a *decision phase* (Miettinen, Ruiz and Wierzbicki 2008, 29). In the learning phase, the decision maker explores the interim solutions provided by the analyst and builds up understanding of the problem as well as its feasible solutions to identify a region of interest. In the decision phase, the decision maker arrives at the most preferred solution. The decision-making process is, therefore, fundamentally a process of learning. Indeed, Miettinen, Ruiz and Wierzbicki (ibid.) describe it as a constructive process where the decision maker builds conviction of what kinds of solutions are possible, while mirroring these discoveries against his or her preferences that may also change over the decision-making procecess.

4 Data visualization in multi-objective optimization

This chapter discusses data visualization and its role in multi-objective optimization. The chapter starts by outlining general data visualization concepts and gradually moves towards covering topics that are more closely related to the domain of multi-objective optimization. The first two sections serve as an introduction to data visualization and some of its basic concepts. The third section moves on to discuss the purposes of presenting data in a visual form. The next two sections discuss data visualization in the context of multi-objective optimization, particularly from the perspective of interactive methods. The last section presents the visualization techniques that are implemented in this thesis.

4.1 Data, information knowledge, and wisdom

The data, information, knowledge, and wisdom (DIKW) pyramid (Ackoff 1989) is a classic way to present the idea that raw data is meaningless, however data can be refined to extract meaning out of it. The thesis of the DIKW pyramid is simple: information may be acquired from data, knowledge may be acquired from information, and wisdom may be acquired from knowledge. One way to look at the DIKW pyramid is as a metaphor for how humans learn things by acquiring and processing information.

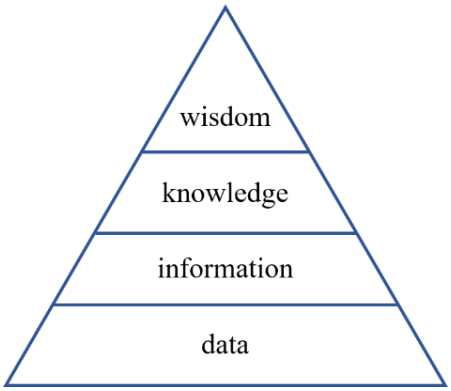


Figure 1. DIKW pyramid, as it is commonly depicted

According to Wallace (2007, 13–14), the basic idea of the DIKW pyramid is commonly attributed to T. S. Eliot's 1934 play The Rock, however he continues to argue that the ori-

gins are unclear. Wallace (2007, 14) does, however, mention Russell Ackoff as one of the popular candidates for receiving the credit. Regardless of the true origins, the idea of a hierarchical data pyramid has stuck, and the concept is a mainstay among many fields of study, particularly in information science.

Moving on from the history lesson, Ackoff (1989, 3–5) describes the four components in the DIKW pyramid. Data is made up of "symbols that represent properties of objects, events and their environment" and it is something that can be observed (Ackoff 1989, 3). Ackoff also notes that data is inherently worthless and must be refined into information to be useful. Information is contained in descriptions and found in answers to questions (ibid.). Knowledge, Ackoff (1989, 4–5) continues, is the know-how of how systems or other things work, which can be codified into instructions and allows systems to be controlled efficiently. Finally, wisdom is the "ability to increase effectiveness", which then leads to development, or an increase in value in a process (Ackoff 1989, 5). Curiously, Ackoff (1989) mentions *understanding* as an additional step in the pyramid between wisdom and knowledge, however understanding is not very commonly included in the acronym nowadays.

While the DIKW pyramid is a widely used concept, its use is not without criticism. Chen et al. (2009, 12) comment that the terms data, information and knowledge are often ambiguously used to imply "different levels of abstraction, understanding, or truthfulness". Furthermore, they point out that the meanings for the terms can vary depending on the field of study. For the purposes of this thesis, it should suffice to draw some parallels between the concept of refining data into more useful forms and the task of solving a multi-objective optimization problem. Multi-objective optimization problems are defined by the data that shapes the problem. To see viable solution candidates, the data must often first be processed somehow, which is where different optimization methods come into play. From this perspective, the act of optimization could be seen as a process that refines data into information; that is, an optimization method takes in meaningless data and transforms it into a more useful form, for example as an interim solution set. In the DIKW terminology, these interim solutions could perhaps be considered information. The final step in both multi-objective optimization and this analogy is finding the most preferred solution to the prob-

lem, hence refining the available information even further. I suppose reaching the most preferred solution indicates that the decision maker has gained some new insight into the problem, which in the DIKW terminology might translate to acquiring knowledge.

4.2 What is data visualization?

To start from the ground up, visualization – in its modern connotation – can be understood as a "graphical representation of data or concepts" (Ware 2012, 2). Data visualization, then, is about presenting a (numerical) data set in a graphical form. The need for data visualization is clear; studies have shown that humans strongly rely on vision to acquire information (Ware 2012, 2), which naturally makes visualizations an appealing option for communicating information. Before moving on to discuss data visualization in more detail, I will briefly mention a couple of related terms. First, *infographics* is closely related but not equivalent to data visualization. An infographic is an explanatory graphic presentation that aims to tell a story to the viewer, whereas data visualization is a tool for analysing data (Koponen, Hildén and Vapaasalo 2016, 20–22). Secondly, *information design*, is "the art and science of preparing information so that it can be used by human beings with efficiency and effectiveness" (Horn 2000, 15). While data visualization and infographics produce concrete outcomes in the form of graphics, information design is a much more abstract endeavour that ponders how phenomena could be expressed visually to make it understandable.

Data visualizations generally have two purposes: they can assist in sense-making (i.e. data analysis) and serve as communication tools (Few n.d.). Although both purposes are important and powerful, the focus of this text will be more on the data analysis aspect. Senay and Ignatius (1994, 40) state that the purpose of data visualization is to "gain insight into an information space by mapping data onto graphical primitives". Regarding the uses for data visualization, Grinstein and Ward (2001, 21) point out that visualizations can summarize data, provide qualitative overviews of complex data sets, and help identify interesting local patterns in data. This is because, according to them, humans naturally look for "structure, features, patterns, trends, anomalies, and relationships" in data, and visualization can help expose those features in a data set. This emphasis on pattern recognition is also ech-

oed by Tou (2011, 7), who discusses data visualization in the context of science and engineering. He notes that data mining often involves searching for patterns and looking for useful data structures, typically from complex and multivariate data. In such context, extracting meaningful information from the data is very important, but also challenging.

It should also be noted that a data set can be visualized in different ways and the choice can affect how people decipher the data. Going back to the idea of mapping data into graphical primitives, Senay and Ignatius (1994, 40) remark that several mappings are possible, and the choice leads to different visualization techniques. Grinstein and Ward (2001, 21) note that presenting data in different forms and highlighting the different interactions in the data is useful to help with data analysis. In an engineering context, Tou (2011) goes into great depth about how the selection and arrangement of elements can affect the outcome of a visualization. Understandably, he emphasizes the importance of simplicity and consistency of graphical representations as well as ensuring that the quantitative data remains easily readable. This is an interesting contrast to other fields where visualizations are also commonly used, like digital marketing, where the design emphasis may be quite different.

There is a vast number of visualization techniques[1], ranging from basic and familiar (f. ex. bar, line or pie charts) to increasingly complex and specialized presentations (f. ex. heatmaps, flow diagrams or exploded view drawings). Although the techniques may differ greatly from one another, producing them shares some common principles. Senay and Ignatius (1994, 36) identify three steps in the data visualization process: *data manipulation*, *visualization mappings*, and *rendering*. The data manipulation step converts a data set into a workable form; in the language of modern data engineering, this step might also be called data wrangling. The visualization mapping step binds the data to chosen visualization primitives, such as positional parameters, colour and texture. The goal of mapping is to identify the set of primitives that can effectively convey the information in the data set. The rendering step produces the final image, based on the design created in the mapping step.

[1] See, for example, https://datavizproject.com/ for examples of different visualization techniques.

To tie data visualization back to the DIKW pyramid, Chen et al. (2009, 13) argue – albeit with rather obfuscated notation – that it is difficult for humans to process information from a raw data set, which makes it difficult to acquire meaningful information from the data. Hence, visualizing data facilitates the process of transforming data into information, which may then be further refined into knowledge and wisdom. Looking at data visualization from that perspective, one could argue that it is a vehicle for information discovery.

4.3 Purposes of data visualization

The previous section argued that data visualization is fundamentally a tool that helps humans analyse raw data. In terms of the DIWK pyramid, data visualization helps us climb up the pyramid, towards more valuable insights. While that is not a wrong conclusion to make, there is more nuance to the topic.

It was already discussed above that a data set can be visualized in different ways, and the choice can affect how the visualization is read. This leads into the notion that visualizations can serve different purposes. Grinstein and Ward (2001, 22) outline three use cases for visualizations, which are exploring data, confirming a hypothesis, and manipulation. In exploratory visualizations, the user may not know what he or she is looking for, but rather the user is exploring the data set to discover structure and patterns that could be meaningful. Exploratory visualizations place great emphasis on interacting with the data to facilitate learning. Confirmatory visualizations are done so that the user can test a hypothesis. In production visualizations, the goal is to present the data in an optimized and purposeful way and with the intention of manipulating the viewer in some way. Grinstein and Ward (ibid.) mention a marketing brochure as an example of this type of visualization.

Continuing the theme of visualization as a means of influencing people, Pandey et al. (2014, 2211), argue that visualizations are increasingly harnessed as tools to convey powerful messages. Their study was focused on the question of *does presenting data graphically (as opposed to textual or tabular presentation) make the message more pervasive*. They discovered that using charts and graphs did indeed affect the overall persuasiveness of the presentation, although the effectiveness depended on people's initial attitudes to-

wards the topic being discussed. This chapter has, so far, inspected data visualization as a tool for making data understandable for the purpose of analysing it. From this perspective, it is intuitive to strive to present the data in an objective and truthful manner. Although delving deep into the subject of data visualization as a tool for manipulation and persuasion is outside the scope of this thesis, it is an interesting topic and something to be aware of even in the context of this thesis.

4.4 Data visualization in multi-objective optimization

The previous sections have discussed data visualization in a general sense, without a specific context in mind. This section, meanwhile, looks at data visualization through the lens of multi-objective optimization, where visualization plays an important role in aiding decision makers. In multi-objective optimization, the primary purpose of visualizations is to help decision makers explore the data and gain understanding of the problem they are solving. A good visualization is, therefore, easy to comprehend and intuitive to use; it should not lose too much of the original information, but it should also not introduce unintentional information into the data (Miettinen 2014, 5–6).

There are a couple of important issues to consider when designing visualizations for multi-objective optimization. Firstly, the complexity of the visualization is primarily dependent on two factors: the number of objectives and the number of alternatives (Korhonen and Wallenius 2008, 195). Multi-objective optimization problems often have more than two objectives to consider, which poses a problem for visualization: how does one represent multivariate data in two dimensions? Korhonen and Wallenius (2008, 198–199) state that in statistics, this problem is generally solved with one of two options: either the dimensionality of the problem is reduced, or the multivariate observation is plotted as an object. There are several standard techniques that demonstrate the latter approach, such as spider-web charts (also called radar charts). Secondly, the number of alternatives may be uncountable; as an example, consider the case of selecting a sub-region of an objective space (Korhonen and Wallenius 2008, 195). This of course limits what types of elements can be used to meaningfully express that data – two-dimensional coordinates, for example, probably do not make much sense in this case.

Finally, it is good to remember that although good visualizations can make complex data easier to digest, one should still apply restraint when presenting data. This is because humans have rather limited working memory, which means that we cannot efficiently process large amounts of information (Huang, Eades and Hong 2009, 140). Indeed, a well-known theory of *magical number seven plus or minus two* suggests that an average human can only store roughly seven items in his or her working memory (Miller 1956). A similar concern is also raised by Miettinen (2014, 6), who discusses the limitations of humans' information processing capabilities in the context of visualizing alternatives in multi-objective optimization. She reminds us that humans have a limited capacity for processing and remembering information, and as such, one should consider the amount of information that is displayed at a time.

4.5 Interacting with visualizations in multi-objective optimization

Working with interactive multi-objective optimization methods presents some additional challenges for visualization. As was discussed in Sections 3.3 and 3.4, the defining features of interactive methods are their iterative solution process and the active interplay between the decision maker and the analyst. Enabling this interplay in the problem-solving process poses challenges for both the individual visualizations and especially for UI design of multi-objective optimization software.

Taking a closer look into the interactive problem-solving process reveals some common interactions. These include evaluating a solution, comparing several solutions, indicating that a solution is interesting (Tarkkanen, et al. 2013, 3222), setting parameters for a desired solution, and filtering out uninteresting solutions (Miettinen 2014, 2–3). Tarkkanen, et al. (2013, 3221) presents the following as the main challenges for UI design for interactive methods: supporting the practical decision-making process, making it intuitive to specify preference information, and analysing the data produced by the analyst. Preference information could be given, for example, by specifying a reference point for desirable objective function values (Tarkkanen, et al. 2013, 3223) or by indicating "a desired direction of simultaneous improvements" (Ojalehto and Miettinen 2019, 78).

Finding ways to support the above-mentioned common actions is a priority for multi-objective optimization software, and something to consider in this thesis. More specifically, the thesis is interested in exploring useful ways to display data and demonstrating options for providing preference information in a convenient way. These two concerns play an important role in supporting analysis and guiding the interactive optimization method.

4.6 Selected visualization techniques

The visualization techniques implemented in this thesis are (grouped) bar charts, two-dimensional scatter plots, value paths and a simple data table. Additionally, some work was done to develop a spider-web chart, however that was left in an unfinished state due to implementation difficulties and a lack of time. The charts are roughly similar in their design to the ones demonstrated in Miettinen's (2014) paper.

The above-mentioned visualization techniques were chosen primarily due to their simplicity, familiarity and versatility. From a multi-objective optimization perspective, they have been mentioned as viable visualization techniques by Miettinen (2014) as well as Korhonen and Wallenius (2008). They are also commonly used in other contexts as well, which should help decision makers understand them in the context of multi-objective optimization. From a technical perspective, the familiarity again works in their favour because these types of visualizations (apart from value paths) are often found in charting libraries. The option of leveraging existing libraries makes the overall development effort more manageable. On the downside, because the charts are simple, they offer little unique utility that would be of special value in the context of multi-objective optimization.

For ideas on additional visualizations to implement, see Korhonen and Wallenius (2008), Miettinen (2014), Filipič and Tušar (2018) and Wilke (2019). The first three examine visualization specifically in the context of multi-objective optimization, whereas Wilke offers a more general look into different types of visualization techniques with good advice on how to apply them.

5 DESDEO

DESDEO is an open-source software framework that can be used for interactively solving multi-objective optimization problems. It is being developed by the Multiobjective Optimization Group at the University of Jyväskylä, and it consists of multiple modules, which are implemented in Python (DESDEO n.d.). The goal of DESDEO is to make interactive optimization methods readily available to both researchers and practitioners, hence allowing people to solve optimization problems that involve multiple conflicting objectives (Ojalehto and Miettinen 2019). The work done in DESDEO is noteworthy because there is a clear lack of open-source software in the domain of multi-objective optimization, and software support for interactive methods is particularly poor (ibid.). In fact, Ojalehto and Miettinen (ibid.) state that they are not aware of any open-source frameworks that are suitable for the needs of interactive multi-objective optimization methods, and even proprietary options are rare.

5.1 DESDEO's architecture

Figure 2 presents the high-level architecture of the DESDEO framework, as it was perceived in 2019. The architecture is divided into four layers, which are called *problem*, *optimization*, *method*, and *elicitation*. The different layers communicate via predefined channels, which is intended to promote reusability of the components (Ojalehto and Miettinen 2019, 74–75). The framework has no built-in UI, however it can be connected to an external UI implementation, such as IND-NIMBUS (ibid.). Hence, a UI could be considered a fifth layer. The basic structure of the application follows a common multitier software architecture design.

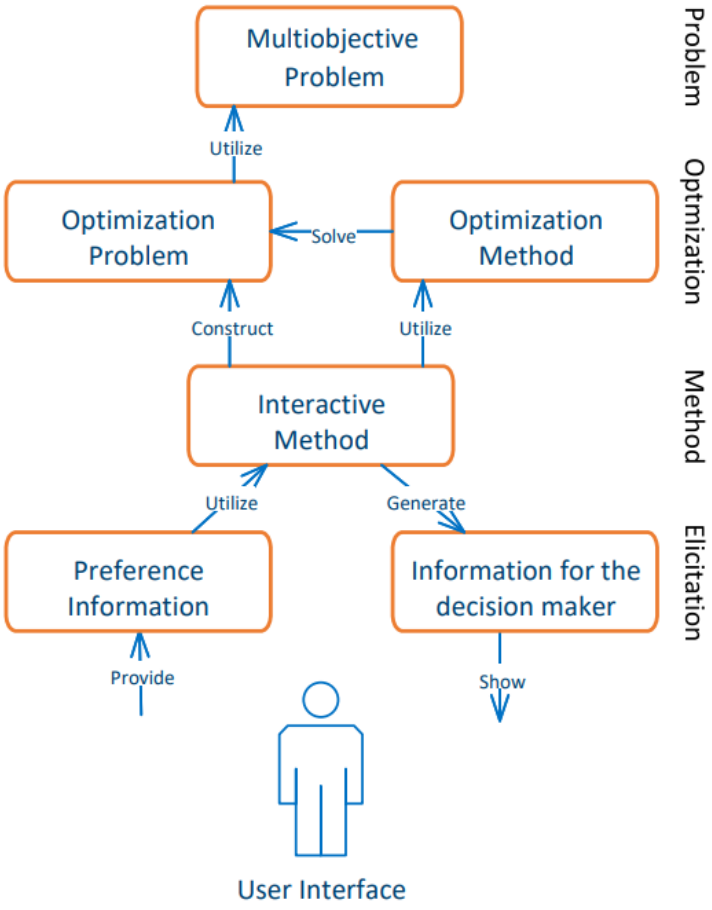


Figure 2. DESDEO framework overview (Ojalehto and Miettinen 2019, 75)

This thesis is mostly interested in the desired functionality of the UI and the mechanisms that are used to connect the UI to the backend services provided by DESDEO. As the above figure nicely illustrates, the UI serves two primary functions: it accepts user input and displays the output back to the user. More specifically, the decision maker passes preference information to guide the analyst (i.e. the DESDEO framework) towards more preferable solutions. Ojalehto and Miettinen (2019, 78) mention as examples that preference information could be provided as "a desired direction of simultaneous improvements", "classification of objective functions", or by "specifying a reference point of aspiration levels". The optimization framework then takes the preference information, processes it and returns an output that the UI displays to the user. The type of preference information

given depends on the optimization method used. The output might be a set of interim solutions, for example.

5.2 DESDEO's modular structure

The work on DESDEO has continued steadily since its initial release. The current design direction for DESDEO emphasizes modularity, which is demonstrated by the fact that the current version of the software is divided into several modules with well-defined purposes (DESDEO n.d.). At the time of writing, DESDEO's documentation lists seven modules: *desdeo*, *desdeo-problem*, *desdeo-tools*, *desdeo-emo*, *desdeo-mcdm*, *desdeo-vis*, and *desdeo-mix* (Multiobjective Optimization Group 2020). The last two are labelled as being under construction. The purposes of the different modules are summarized in Figure 3.

The modular structure is intended to make it easier to expand DESDEO's suite of optimization methods, while also ensuring their cross-compatibility (DESDEO n.d.). Dividing software into several modules is also a natural way of creating logical boundaries – a module typically hides its own implementation details and only exposes certain details about itself via a public API (Hall, et al. 2011, 25). The data visualization component library developed in this thesis can, in the future, be used by the visualization module.

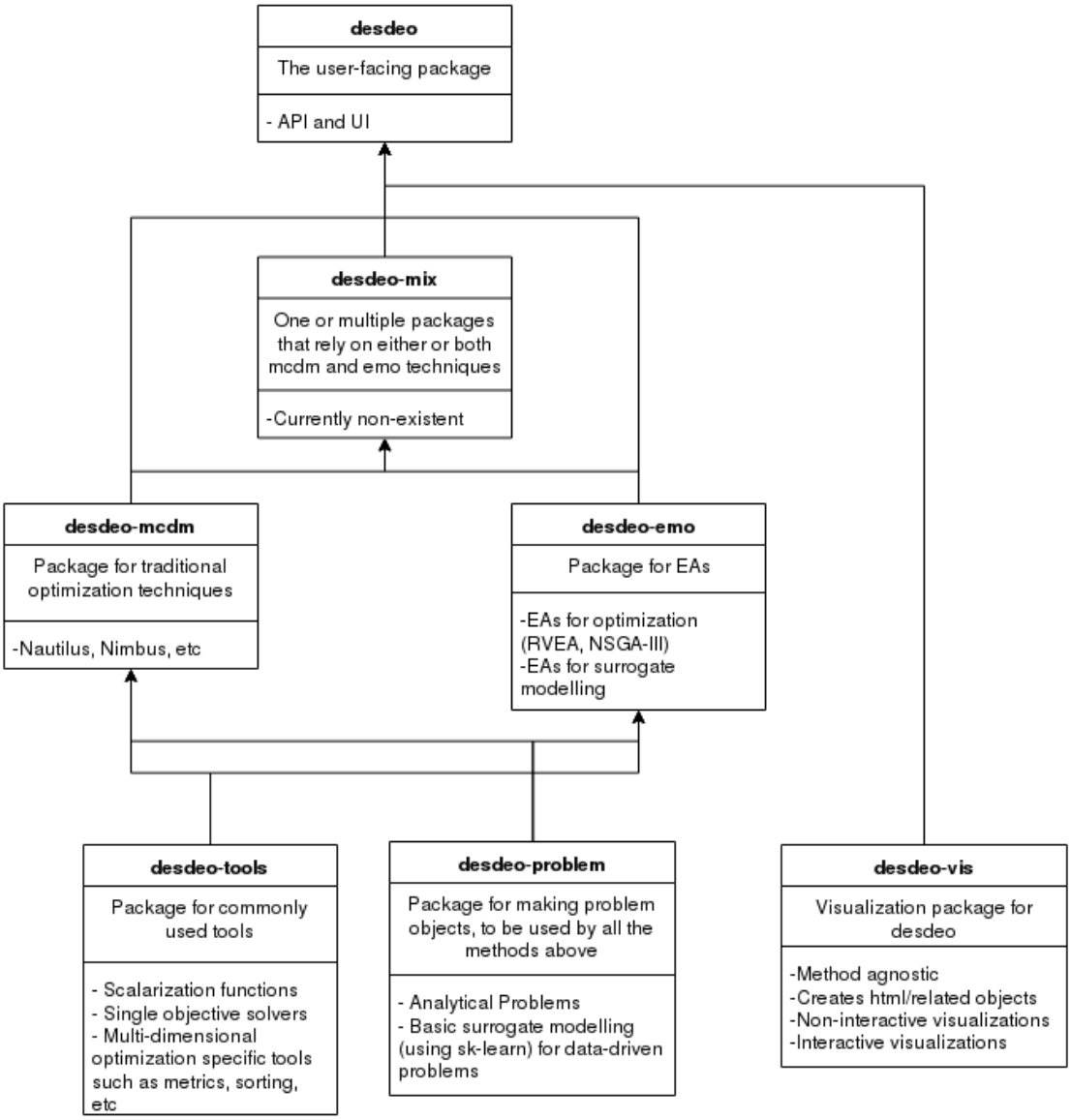


Figure 3. DESDEO's modular structure (DESDEO n.d.)

5.3 User interface features

After introducing the DESDEO framework, let us summarize the main features that DESDEO's UI should have. As was outlined in the previous section, the primary responsibilities of the UI are accepting preference information input from the decision maker and displaying interim solutions and other relevant information back to the decision maker.

Based on the discussion in this section and in Section 4.5, the UI should be capable of handling the following types of inputs from the decision maker.

Input related to providing preference information:

- Giving preference information to guide the solution process.
- Setting parameters to guide the solution process.

Input related to the solution candidates:

- Filtering out uninteresting solutions and otherwise exploring the data set (f. ex. to discover relations between the objective functions).
- Indicating that a solution is interesting, and the same solution may be returned to later.
- Selecting a solution as the most preferred solution, ending the optimization task.

In addition to handling input, the UI should, of course, be able to present the problem data and any interim solutions in a useful and understandable way. See Section 4.4 for further discussion on visualization in the context of multi-objective optimization.

6 Introduction to web development

This chapter goes over some of the most essential aspects of developing web applications, especially from the perspective of frontend (or client-side) development. The purpose of this and the following chapter is to provide context for the technology choices that were made for the new software. The technology choices will be outlined in Chapter 10. This chapter discusses what web applications are, how they work, and what kinds of approaches there are for building web applications.

6.1 In search of web applications

As was mentioned already in the introduction, the topic of the thesis is developing a *web application*. In 2021, web application is a term one can expect to hear even in everyday conversation, however defining what is and is not a web application is not entirely straightforward. Furthermore, web applications have evolved significantly over the last two to three decades, as can be surmised from the following discussion.

To start things off, Fraternali (1999, 228) describes web applications as a "hybrid between hypermedia and an information system". According to him, web applications deal with concerns such as handling both structured and nonstructured data, provide support for exploratory access via navigational interfaces and support proactive behaviour, such as recommendations and filtering. Conallen (1999, 63–64) provides a simpler but relevant perspective on the topic, stating that a web application is a "Web system (Web server, network, HTTP, browser) in which user input (navigation and data input) effects the state of the business". He continues to say that web applications generally follow a client–server model and emphasizes that the frontend of a web application is delivered via a web system. Scouring the Internet, it is also possible to find definitions that associate web applications closely with Java and its servlet specification; see, for example, Chaffee (2012). This perspective, while not incorrect, may come as a surprise to the modern reader. For the purposes of this thesis, it is perhaps sufficient to re-iterate that web application are used via a web browser and they contain logic and/or state that users can manipulate (i.e. a plain HTML document is not a web application, even though it can be viewed with a browser).

6.2 Servers and clients

As was mentioned above, web applications are typically built with a client–server model, where the server stores documents or other information and the client requests to see specific documents or information (Jazayeri 2007, 200). The mechanism that the server and client use to communicate is the Hypertext Transfer Protocol (HTTP), which is a request–reply protocol (Jazayeri 2007, 201). Using HTTP, a client may, for example, ask the server for a specific resource (a GET request) or submit information to the server (the HTTP method used depends on the situation; POST is probably the most common).

In the early days of the World Wide Web, web pages were made up of static documents that were linked with one another. In this system, the server's role was to serve the clients (i.e. web browsers) web pages that they can then display to the users (Jazayeri 2007, 201). Jazayeri (ibid.) notes that software engineers quickly discovered that it was possible to generate web pages programmatically, for example based on information queried from a database. This discovery was the first step towards producing dynamic web pages. The idea of servers returning "ready to consume" web pages to clients is known as *server-side rendering*. Some traditional technologies for server-side rendered web applications include PHP, Java Server Pages (JSP) and Active Server Pages (ASP). Note, however, that although a web page may be constructed dynamically with the above technologies, the created page still has no dynamic behaviour from the client's perspective; the client is still served a static HTML page. Producing web pages with client-side dynamic behaviour requires further effort and some new tools.

Working with client-side applications involves four fundamental components: Hypertext markup language (HTML), Cascading Style Sheets (CSS), JavaScript, and the Document Object Model (DOM). HTML is a markup language that forms the structure and contents of web pages, CSS is used to style HTML documents, and JavaScript is a scripting language that allows programmers to manipulate the contents of web pages via DOM (Mikkonen and Taivalsaari 2008, 320). DOM is an API for HTML and XML documents that presents the contents of the document in a tree-like hierarchy and allows programs to edit the content and style of web documents (MDN Web Docs 2021). In summary, HTML

is the content of a page, CSS defines its looks and JavaScript (via DOM) defines the page's behaviour. Out of these four web components, JavaScript deserves a closer look.

6.3 Dynamic web applications with JavaScript

JavaScript was first released in 1995 as a scripting language for the Netscape Navigator web browser, with the aim of enabling dynamic behaviour for web pages (Haverbeke 2018). Since its inception, JavaScript – or more precisely ECMAScript – has grown to be the dominant programming language for client-side web applications. ECMAScript is both a specification and a programming language that implements the specification (Ecma International 2020). ECMAScript (the standard) aims to ensure openness and interoperability of the web, based on the notion that "Anybody should be able to create a Web page that can be hosted by a variety of Web servers from different vendors and accessed by a variety of browsers" (Wirfs-Brock and Eich 2020, 3). Although JavaScript began as a scripting language for the web, is has since greatly expanded its reach, propelled by the release of Node.js. Node is a JavaScript runtime environment built on top of Chrome's V8 JavaScript Engine (OpenJS Foundation n.d.). Node allows JavaScript code to be run outside a web browser, thus making JavaScript a viable language for server-side development, among other things.

As was discussed above, JavaScript can be used to modify contents of a web page via the DOM API, which is powerful in its own right. More impressively, JavaScript can be used to query the server without triggering a page load on the browser. This was originally achieved by using AJAX (Asynchronous JavaScript and XML), which is an asynchronous communication protocol for client–server communication (Jazayeri 2007, 207). The discovery of AJAX was an outright revolution in how web applications were developed, as noted by Fink and Flatow (2014, 4). This is because enabling direct asynchronous communication between the client and server circumvents the classic dynamic of retrieving rendered documents in favour of retrieving snippets of data that the client then uses as it sees fit. Indeed, Crane, Pascarello, and James (2006) discuss the advantages of AJAX in length, noting that it enables, among other things, the development of "rich clients". A rich client

supports a variety of input methods and responds to them in an intuitive and timely fashion (Crane, Pascarello and James 2006, 5).

Moving on from the exciting discovery of AJAX and towards the present day, *single-page applications* (SPA) are, in my opinion, the next major shift in client-side application development. SPA is a software architectural style that is, in many ways, a logical continuum for the concepts pioneered by AJAX. An SPA is a web application that only has one page that functions as a shell for web pages and other content that the application may have (Fink and Flatow 2014, 11). From a technical perspective, the core idea behind SPAs is that clients receive data in JSON format from the server and independently decide how to display the data (Fink and Flatow 2014, 12). Furthermore, the data is retrieved without incurring a page load, which results in a seamless browsing experience (Fink and Flatow 2014, 13). SPA has nowadays become a popular approach for building frontend applications.

Although JavaScript is a crucial part of client-side web development, it is certainly possible to use other programming languages as well. If a different language is used, the source code can be transpiled into JavaScript to allow it to run in a browser (Poudel 2018). Some alternatives to JavaScript for frontend development include TypeScript, Elm, and ClojureScript.

6.4 Software frameworks

Modern software development is often done with the help of software frameworks. A software framework is a “library that offers opinions about how software gets built” (MDN Web Docs 2021). Frameworks may be used, on one hand, as guidelines for overall software architecture (Okanović 2011, 1–2), or, on the other hand, as tools for solving common tasks or problems in building software. Examples of common issues that may be handled by a framework (or libraries) are managing database connections, session handling and user authentication. While the framework sets the rough guidelines for application development, further application-specific customization is required to produce working software for a specific context (Okanović 2011, 2). In addition to generally making devel-

opment easier, frameworks can increase code quality and maintainability. This is because frameworks often impose opinions on how certain things should be done in the code, thus resulting in a more homogenous and predictable code base (MDN Web Docs 2021).

It is worth noting that there are different types of frameworks, with varying scope and focus. Fernández-Villamor, Díaz-Casillas and Iglesias (2008) make a distinction between "Web application frameworks" (WAF) and "Agile web frameworks" (AWF). They argue that WAFs usually focus only on one layer of an application, such as persistence or web flow. AWFs, meanwhile, address the entire application, which is why they are also known as "full stack web frameworks". The canonical example of such framework is probably Ruby on Rails. While I am not sure that the term "agile web framework" has lived on since the release of their paper, the principle of having frameworks with varying scopes certainly still applies today.

This thesis focuses on frontend development, and thus modern client-side JavaScript frameworks are of particular interest. Probably the most popular general purpose frontend frameworks at the time of writing are Angular, React[2], Svelte, and Vue. The question of which framework to use for the new applications is addressed in Section 10.2.

[2] React is technically a library, however it is commonly supplemented with a suite of other libraries, which together make it function like a framework.

7 Data visualization for the web

This chapter presents the problem of creating data visualizations for the web and outlines some potential solutions. The first section discusses different techniques for displaying graphics on web pages, the second section introduces a well-known and influential data visualization library, and the third section explores some of its modern competitors.

7.1 Web graphics

One of the main goals for this thesis was to explore data visualization options for the web. The World Wide Web Consortium (W3C) lists a few different ways of presenting graphics on a web page: raster images (in PNG file format) Scalable Vector Graphics (SVG), the Canvas API, and Web Computer Graphics Metafiles (WebCGM). They explain that these techniques are used for different purposes; for instance, interactive line art and data visualization are a good fit for SVG and the Canvas API (W3C 2016). While the Canvas API is an interesting alternative for producing web graphics, the following text will focus on SVG because it is the technique that is used by the father of modern web data visualization solutions. More on this in the next section.

SVG is a text-based standard for producing images that work well with other web standards, like CSS, DOM and JavaScript (MDN Web Docs 2021). The SVG standard has been developed by W3C since 1999 (ibid.). Quint (2003, 99–100) describes SVG as a drawing solution that can be used to draw primitive shapes like lines, polygons, rectangles, and ellipses, as well as paths. These shapes, or objects, can be transformed and styled, for example in terms of their scale, rotation and opacity. In addition to creating static objects, SVG also provides more advanced dynamic features, like adding animations and scripts to objects (Quint 2003, 100).

One of the key benefits of SVG is that SVG images, unlike raster images, scale cleanly at any size (MDN Web Docs 2021). Furthermore, many programming languages, including JavaScript, can generate SVG images (W3C 2016). In summary, SVG is a flexible and powerful tool for producing graphics for the web, making it an essential tool for data visu-

alization in web context. One noteworthy tool that heavily features SVG is the D3.js charting library (MDN Web Docs 2021), which will be introduced next.

7.2 D3.js

There is no way to discuss data visualization in the web without bringing up D3, so here we go. Above, I titled D3 the father of modern web data visualization, and I do not think that is an exaggeration; there are plenty of books and countless tutorials written about using D3 – Murray (2013), Dale (2016), and Zhu (2013) all seem like good resources to start learning D3. Furthermore, as we will soon discover, D3 is the foundation for many (possibly even most) other JavaScript-based data visualization libraries.

D3 stands for Data-Driven Documents, and it is a JavaScript charting library first released in 2011 by Michael Bostock, Vadim Ogievetsky and Jeffrey Heer of the Stanford Visualization Group (Bostock, Ogievetsky and Heer 2011). Bostock, Ogievetsky and Heer (2011, 2302) describe D3 as a "visualization kernel" that solves the problem of "efficient manipulation of documents based on data". They mention jQuery as a suitable analogue to D3, since both are *document transformers*.

At its core, D3 offers a handful of basic operations for manipulating and displaying data. Bostock, Ogievetsky and Heer (2011, 2302–2303) highlight the selection of specific nodes in a DOM tree as a requirement for enabling *data-driven transformations* and present it as a core feature of the library. In D3, selection is an atomic operand that finds a set of elements from a document, based on given filters. These selections can then be manipulated by *operators* that modify the selection in some way. In addition, *transitions* can be used to construct animations and user input can be captured with *event handlers* that enable documents to contain interactive behaviour (Bostock, Ogievetsky and Heer 2011, 2303).

The main selling point of D3 is its raw power – there are few limitations for what can be done with the library, provided that the programmer is skilled enough. Unfortunately, the powerful feature set comes with a cost. Firstly, D3 is commonly considered to be a difficult tool to learn (Murray 2013, 4). Perhaps the biggest reason for this is the fact that D3 is a relatively low-level library, as noted by King (2014), particularly when compared with

many of the more modern data visualization libraries that will be discussed in the next section. This low-level approach means, among other things, that D3 does not offer any predefined visualizations that can be used as-is, but rather D3 offers a toolbox for creating your own visualizations (Murray 2013, 8). Secondly, D3 may not always play nicely with other libraries that want to have control over the DOM, such as React. The problem is that both D3 and React want to control the DOM tree and dictate what is rendered on the page at any time, which may lead to conflicts and surprising bugs. The issues with combining D3 and React (and other frontend frameworks) are well-known, and there are some approaches to working around them. See Iglesias (2018) for potential solutions.

7.3 Competitors to D3

For this thesis, I examined the overall landscape of JavaScript-based data visualization libraries to identify potential tools for the task at hand. I had identified D3 as a viable option for this project early in the planning stage, however I also wanted to see if there were other, easier-to-use alternatives available. The libraries I looked at were ApexCharts, Chart.js, Highcharts, Nivo, Plottable.js, React-vis, Recharts, and Victory, which were suggested in online articles[3] related to JavaScript data visualization. An important observation to make about these libraries is that several of them are, in fact, built on top of D3. To summarize my findings, the major differences between the above-mentioned libraries and D3 are the types of components that they provide out-of-the-box and their level of abstraction. Furthermore, many of the libraries listed above lacked sufficient support for adding interactive behaviour for the charts.

Regarding the types of components that the libraries offer, they take the approach of providing several types of visualization techniques as ready-to-use components (e.g. they may have a bar chart component or a scatter chart component). As was mentioned above, D3 does not offer such convenience. D3's approach of building all charts from the ground up has its pros and cons. On one hand, building completely custom visualization components provides full freedom regarding their functionality and looks. On the other hand,

[3] See, for example, Majorek (2020) and Saring (2018).

building everything from scratch is a significant amount of work, and of course requires a fair amount of expertise with the library.

The above-mentioned libraries also had a higher level of abstraction when it comes to rendering data. As an example, drawing a simple bar chart in D3[4] involves drawing rectangles with the correct dimensions, styling them, and rendering them as a SVG image with manually defined axes. Creating a similar chart with ApexCharts[5] is more straightforward; the chart is created with an object and its data and style are defined with properties.

One area where the libraries differed was their integration with popular JavaScript frameworks. Some libraries provided the charts as plain JavaScript, others had different versions or additional wrapper libraries to make them compatible with different frameworks, and couple were designed specifically to integrate with React. From an ease-of-use perspective, having an easy integration with popular frameworks is, of course, desirable for this project.

Finally, many of the above libraries suffered from a surprisingly poor support for implementing custom functionality on chart interactions, like when a piece of data is clicked with a mouse. It is possible that this type of more advanced functionality does, in fact, exist in these libraries, but it is buried deep enough into their documentation that a cursory look does not reveal it. Be as it may, I certainly got the impression that many of these libraries are intended to be used in a largely static manner, to display data, but not to interact with the data in a very meaningful way. Interactivity was an important feature for the new visualization components that were developed in this thesis, as will be discussed in Chapter 9, and thus the apparent lack of interactive features ruled out some of the above libraries from consideration.

[4] Bar chart using D3: https://observablehq.com/@d3/bar-chart
[5] Bar chart using ApexCharts: https://apexcharts.com/javascript-chart-demos/bar-charts/basic/

8 Research approach

The research approach for this thesis is *design science*. This chapter starts off by introducing design science, followed up by discussion on the artifacts that design science research can produce. Next, the design science research process is briefly outlined. The chapter closes with an overview of how this thesis project was carried out.

8.1 Introducing design science

Design science is a commonly used research approach in information systems, computer science and software engineering (Iivari 2007, 39). The premise of design science is to explore a problem by creating an item or an idea that solves said problem. These tangible items and intangible ideas are collectively referred to as *artifacts*. The process of creating the item is thought to produce novel ideas that can be generalized into new knowledge of the problem, thus adding to the existing body of knowledge regarding the problem space. More precisely, Hevner and Chatterjee (2010, 5) define design science research as a research paradigm where "a designer answers questions relevant to human problems via the creation of innovative artifacts, thereby contributing new knowledge to the body of scientific evidence. The designed artifacts are both useful and fundamental in understanding that problem." Hevner and Chatterjee (ibid.) continue to state that the first principle of the method is that "knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact". In much the same vein, Vaishnavi, Kuechler and Petter (2004, 4) define design science as knowledge of how to create artifacts that satisfy pre-determined functional requirements. Design science research, according to them, is research that "creates this type of missing knowledge using design, analysis, reflection, and abstraction".

8.2 Design science artifacts and theories

So, design science looks to gain insight into a problem space by applying existing design theory to create artifacts. Although artifact is perhaps the most essential concept in all of

design science research, there is no universally accepted notion on what it is or should be. To start the discussion, March and Smith (1995, 254) established the basic vocabulary for design science with *constructs*, *models*, *methods*, and *instantiations*, which are the products of design science study. Baskerville et al. (2018, 362) summarize the concepts as follows: constructs are the basic concepts and language that are used to describe problems and solutions, models represent the real-world context of a problem, and methods define processes. Instantiations take constructs, models and methods and implement them as a working information system that solves a problem. Alternatively, Vaishnavi, Kuechler and Petter (2004, 17) say that instantiation is "the realization of the artifact in an environment". Vaishnavi, Kuechler and Petter (2004, 16) also make a distinction between *abstract* and *material* artifacts. Abstract artifacts include constructs, models, frameworks, architectures, design principles, and methods, whereas material artifacts are context-bound instantiations of the abstract artifacts.

Together with IT artifacts, *design theories* are the second type of research outcome that is commonly recognized for design science studies (Baskerville, et al. 2018, Vaishnavi and Kuechler 2004). While both outcomes are considered important, the lower-level design artifacts are typically seen as the groundwork that builds towards high-level design theories, which contribute "new knowledge to the body of scientific evidence", as Hevner and Chatterjee (2010, 5) put it. An example of this line of thinking is found in Gregor and Hevner (2013, 341–342), who discuss design artifacts from the perspective of their abstraction level. In short, they divide research efforts into three levels, where the first is "situated implementation of artifact", the second is "nascent design theory", and the third is "well-developed design theory" (see Table 1). They argue that the higher the abstraction level of the produced artifact, the more universally applicable – and therefore more valuable and with more academic merit – the study findings are.

This dichotomy between design artifacts and design theories is recognized by Baskerville et al. (2018, 358–359), who note that research in information systems could arguably be divided into an "artifact camp" and a "design theory camp". Furthermore, they make a point of how particularly some of the "gatekeepers of IS journals" emphasize theoretical contribution as a sign of quality in information systems studies. That said, Baskerville et al.

(ibid.) advocate for the value of design artifacts as important contributions to existing design knowledge.

	Contribution Types	**Example Artifacts**
More abstract, complete, and mature knowledge	Level 3. Well-developed design theory about embedded phenomena	Design theories (mid-range and grand theories)
↕	Level 2. Nascent design theory — knowledge as operational principles/architecture	Constructs, methods, models, design principles, technological rules.
More specific, limited, and less mature knowledge	Level 1. Situated implementation of artifact	Instantiations (software products or implemented processes)

Table 1. Design science research contribution types (Gregor and Hevner 2013, 342)

There has also been criticism of the often rather abstract nature of artifacts proposed in studies that apply design science as a method, particularly in the field of information systems. In their influencial paper that examined how IT artifacts are conceptualized in information systems research, Orlikowski and Iacono (2001) argue that research efforts have been overly focused on conceptual issues surrounding the IT artifact, at the expense of studying the artifacts themselves. They criticize prior research for adopting an overly simplified and utilitarian view of the IT artifact, stating that the artifacts have commonly been considered unproblematic, taken for granted and viewed through the researchers' "disciplinary lenses". The result, according to Orlikowski and Iacono (2001, 130), is that IT artifacts are "absent, black-boxed, abstracted from social life, or reduced to surrogate measures" in information systems research.

Overall, it is not easy to give a satisfactory summary of what the outcome of a design science study should be. There clearly are different types of artifacts, ranging from tangible to intangible, from a piece of software to abstract theories about solving problems. At the risk of overgeneralizing, it seems reasonable to suggest that these ideas fit quite comfortably into the different disciplines where design science is most often applied: computer scientists and software engineers may first and foremost be interested in the IT artifact itself,

whereas more business-oriented information systems researchers are perhaps more interested in the context of the artifact and its utilitarian value in an organization.

8.3 Design science process

As was discussed in the previous sections, design science research aims to produce new artifacts, which are then evaluated and studied to extract generalizable theoretical knowledge from them. This implies that design science research process must include at least an artifact creation phase and an analysis phase. Indeed, one common model, presented by both Vaishnavi, Kuechler and Petter (2004, 11) and Baskerville et al. (2018, 365), depicts a five-step research process. The steps in this model are *awareness of problem*, *suggestion*, *development*, *evaluation*, and *conclusion*.

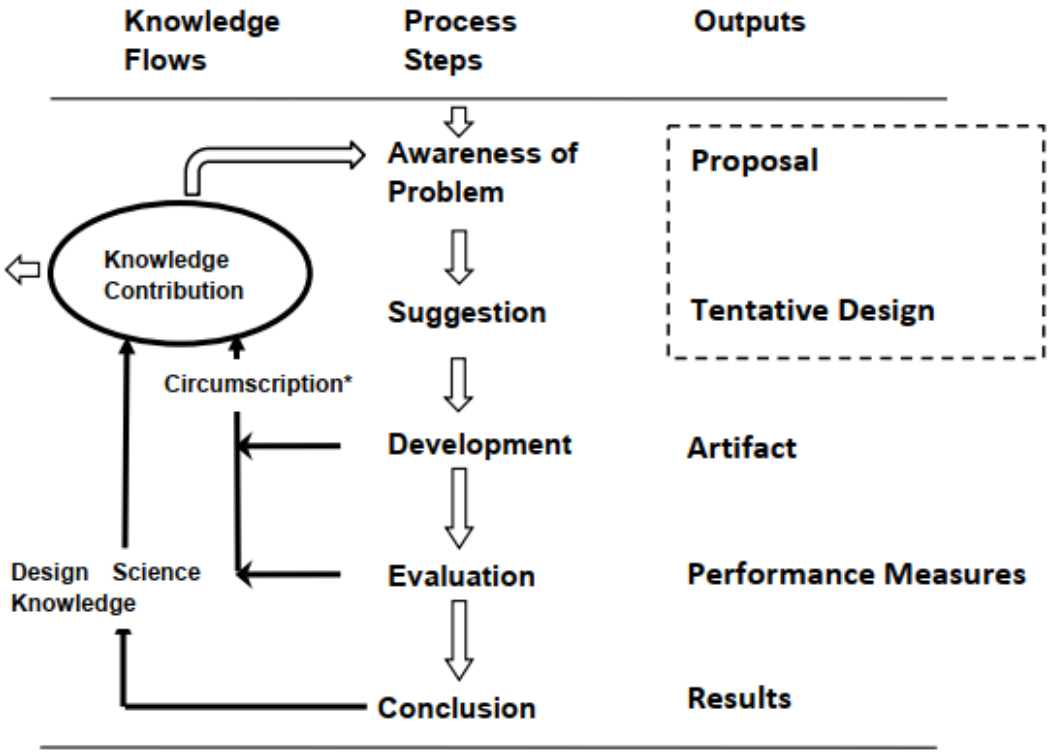


Figure 4. Design science research process (Vaishnavi and Kuechler 2004, 11)

The first two are the preliminary steps where a problem is identified, and possible solutions are explored. The development phase is where a new IT artifact is created to test out a pos-

sible solution to the identified problem. Vaishnavi, Kuechler and Petter (2004, 12) emphasize that the artifact does not have to be refined, but rather the interest is on establishing a functional design for solving the problem. Conversely, the inherent value of the artifact is highlighted by other scholars, such as Baskerville et al (2018) and Orlikowski and Iacono (2001). The evaluation phase, of course, analyses the new artifact and its impact in solving the problem. Vaishnavi, Kuechler and Petter (2004, 12) point out that any deviations from the initial expectations for the artifact should be analysed and tentatively explained in this phase. They continue to say that the lessons learned from the evaluation may then be carried over to another iteration of the design process, starting from the suggestion phase. Finally, after one or more iterations, the research process is concluded.

8.4 Conducting design science

The research process model presented in Figure 4 fits reasonably well into this thesis. The initial problem, namely a lack of UI implementation for multi-objective optimization, served as the initial motivation for this thesis. Once the research problem was defined with some accuracy, the main thesis process began. Most of the thesis work was spent in cycles of Suggestion, Development and Evaluation phases. The suggestion step involved activities like exploring viable technologies for development work, establishing guidelines for the overall design of the software, and considering new visualization techniques to prototype. The main concerns were, on one hand, planning viable technical solutions for desired features, and, on the other hand, looking for technical problems and limitations that could prevent the development of viable IT artifacts. The most time and effort went into the development phase, where the IT artifacts – a web application and a component library – were developed. The development work was monitored throughout the thesis process to identify problems and to guide future development work. Evaluating the produced artifacts with well-defined metrics is an important part of conducting design science research (Hevner, et al. 2004, 85–87), however finding suitable evaluation methods and metrics proved to be a challenge. As was discussed in Chapter 2, there was a fair amount of uncertainty regarding the direction that this thesis would take in its early stage, which meant that it was difficult to establish goals for how the artifacts should be evaluated. Hevner, et al.

(2004, 86) present five categories of evaluation methods, out of which Testing fits this thesis the best. The data visualization components have automated tests that apply both black and white box methods to ensure the functionality of each component. In addition, this thesis looks into some notable technical design choices regarding how the visualization components integrate into the frontend application, which could be considered Architecture Analysis. Finally, this thesis is the obvious conclusion of the research process. The thesis presents the new IT artifacts and discusses the lessons learned from their development. The most important technical and design concerns are addressed in Chapters 10 and 11. Chapters 12 and 13 summarize the results of the development work and address the question of how feasible the initial vision for the web UI was from a technical perspective. Together, these chapters hope to provide some insight into how this type of software could be built.

9 Software requirements

This chapter outlines the requirements for the software that was developed in this thesis. The first section briefly introduces the discipline of requirements engineering. The second section outlines the initial ideas for the new software. These initial ideas and constraints are then discussed in more detail in the following three sections, using Laplante's (2018, 4-12) system of dividing requirements into functional, non-functional and domain requirements.

It should be noted that the division to functional, non-functional and domain requirements is not the only way to categorize requirements. A different approach, suggested by Sommerville (2011, 83–84), would be to divide requirements into user and system requirements. I find functional, non-functional and domain to be a simple and understandable system for this case, however more thought should be put into analysing requirements if a proper requirements document were to be written. Regarding requirements documents, there are many ways to present requirements, ranging from informal natural language to visual models or even highly structured formal or mathematical models (Laplante 2018, Sommerville 2011). In this thesis, requirements are discussed rather informally using natural language.

9.1 What is requirements engineering?

To start with basic terminology, *requirements* define the stakeholders' (f. ex. users, customers, and developers) needs for a system under development, and thus they are the basis for every project (Hull, Jackson and Dick 2005, 2). More simply, requirements are a description of what the system should do (Sommerville 2011, 83). The act of eliciting, evaluating, and documenting a project's requirements is referred to as *requirements engineering*. More specifically, Laplante (2018, 2–3) defines requirements engineering as follows: "Requirements engineering is the branch of engineering concerned with the real-world goals for, functions of, and constraints on systems. It is also concerned with the relationship of these factors to precise specifications of system behaviour and to their evolution over time and across families of related systems." It should be noted that requirements en-

gineering is not specific to software development, but rather it can be applied to development of any system. As an example, Laplante (2018, 2) introduces requirements engineering with an anecdote about building a bridge.

9.2 Initial requirements

The list below summarizes the initial set of requirements that were identified in the preliminary discussions between the thesis supervisors and me. Some of the requirements are also mentioned in the thesis topic proposition paper, which is included as Appendix A. These initial ideas for the software are discussed here to provide context of the starting point of the endeavour and to capture some of its original vision.

The initial requirements for the project, in a rough order of importance:

1. The software must be open source.
2. The software must be a web application. In other words, the software must be usable via a web browser. The frontend application must be able to communicate with DESDEO's backend services.
3. The visualization components must be implemented in a modular way, so that each component is re-usable.
4. The new visualization components must be interactive. The exact nature of the interactivity was initially vague, although there was a notion of supporting various mouse gestures. The purpose of gestures could be, for example, to provide preference information to a multi-objective optimization method.
5. The visualization components must be able to display the same data set and react to changes in the data set. In other words, the components should apply the *linked data view* paradigm (Wills 2008, 218), where the same data is displayed with multiple simple views that are linked with one another. Being linked means that interactions with one view are reflected in all linked views. For example, if a piece of data is removed from the data set, the change must be reflected across all views. Satisfying

this requirement allows the same data to be displayed with multiple visualization techniques.

6. The visualization components should be dynamic and accessible.

7. The software should be written in a maintainable and extendable manner.

As can be seen from the above list, some of the requirements were concrete and well-defined from the start, whereas others were more vague. Out of the above requirements, I suspected that achieving a sufficient level of interactivity with the UI would be the most difficult problem to solve.

9.3 Functional requirements

Functional requirements describe what services the system should provide, how it should react to inputs, and how the system should behave in specified situations (Sommerville 2011, 84–85). Functional requirements should also explicitly define behaviours that are not allowed (Laplante 2018, 6). In short, functional requirements define what a system should and should not do. Sommerville (2011, 86–87) states that the functional requirements of a system under development should be defined in a complete and consistent way. He adds, though, that in large and complex systems, achieving this is practically impossible. It should also be noted that the distinction between a functional and non-functional requirement is not always clear, as Sommerville (2011, 85) points out.

There are some functional requirements that can be extracted from the initial requirements. The first and most obvious one is requirement #2, which dictates that the application should be a web application, as opposed to a desktop or a mobile application. Requirement #2 also explicitly specifies the need to establish a communication mechanism with DESDEO's backend services. Secondly, requirement #3 emphasizes the importance of developing the application in a modular manner. The desire for modularity guided the development efforts towards delivering the visualization components as an independent application, rather than implement them directly into the frontend application. Hence, the components were eventually delivered as an independent component library. Thirdly, re-

quirement #4 regarding interactivity of the visualization components was a theme that was heavily emphasized throughout the development process. It was perhaps the most novel aspect of the project, but also the one that presented the biggest challenges. Fourthly, requirement #5 suggested that there was a need for an application-level state management solution, as indicated by the need to pass the same data set to multiple components and keep the data set synchronized.

9.4 Non-functional requirements

If functional requirements describe what a system should do, then non-functional requirements describe how the system should do those things. Laplante (2018, 7–11) explains that non-functional requirements deal with some observable attributes of a system, such as reliability, reusability or maintainability. He divides non-functional requirements into five categories, which are *quality*, *design*, *economic*, *operating,* and *political/cultural*. Sommerville (2011, 87) explains that non-functional requirements are issues that are not directly related to the services provided by the system. Instead, they are more likely to describe emergent system properties (f. ex. response times) or system constraints (f. ex. capabilities of input and output devices). Sommerville (ibid.) continues to emphasize the importance of non-functional requirements, stating that they are often more critical than individual functional requirements. This is because, according to him, users can typically work around inadequacies in system functionality, whereas failing to meet non-functional requirements may render the entire system unusable. Sommerville (ibid.) provides an example of an aircraft control system that fails to meet reliability requirements – in such a case, the system is most likely deemed unsafe, and therefore never accepted into use.

There are a handful of non-functional requirements that can be identified from the list of initial requirements. The most obvious one is requirement #1 that mandates for the software to be open source. Secondly, requirement #7 states that the software should be developed “in a maintainable and extensible manner”. The implications of this statement are multi-faceted and difficult to meet and evaluate. For some actionable guidelines, favouring a statically typed programming language, dedicating some effort into writing automated tests, and producing documentation for the software should help with maintainability. En-

suring extensibility is a difficult topic to address briefly, however dividing the software into modules with well-defined public APIs and applying the Open-Closed principle (Martin 2017) should provide a good starting point. Thirdly, requirement #6 states that the UI should be accessible. Again, this is a complex topic to tackle, and it mostly falls outside the scope of this thesis. Ultimately, the look and feel of the finished UI components was largely dictated by the libraries that were used in the implementation, and as such, I had limited control over how accessible or inaccessible the finished software is.

9.5 Domain requirements

Domain requirements are derived from the specific needs of the application domain (in this case multi-objective optimization) rather than from the needs of system users (Sommerville 2011, 86). Laplante (2018, 11) points out that domain requirements can be domain-specific functional requirements, additional constraints placed for certain functional requirements, or specifications for how certain computations must be performed. Sommerville (2011, 86) comments that domain requirements are problematic, because software developers often do not have the required domain knowledge to tell if domain requirements have been missed out or if they conflict with other requirements. As an example, a functional requirement for a banking system might be that it needs to log users' actions in adequate detail. A domain requirement, on the other hand, might be that the system must not log certain sensitive customer information.

There were few clearly domain-based requirements in the initial requirements. The one that was mentioned already in the thesis proposal was that a decision maker should be able to pass preference information via "interactive graphical elements". This concern closely relates to the requirement #4 regarding interactivity, which was discussed in the functional requirements section above. Another feature that was recognized as valuable for multi-objective optimization was the ability to visualize a data set using different techniques in a linked manner, with the aim of helping decision makers analyse the data. Using multiple visualization techniques for the same data ties back into requirement #5, which maintains that changes to the data should be reflected across all visualizations.

10 Technology choices

This chapter presents the technology choices for the project with some discussion on why the selected tools were chosen. More specifically, the first three sections go over the choices of programming languages, frameworks, and data visualization libraries. The entire technology stack is summarized in the fourth section.

10.1 Programming language

The goal of this thesis was to produce a proof-of-concept interactive UI that runs in a web browser. Hence, the use case is a rather obvious fit for a SPA[6], which can provide a rich and responsive user experience. Taking the SPA route firmly guides the choice of technologies towards JavaScript and its numerous frameworks and libraries. Furthermore, JavaScript was suggested in the thesis proposal (see Appendix A) as a suitable programming language for the task. Ultimately, I decided to forego JavaScript in favour of its younger sibling TypeScript as the programming language for both the component library and the frontend application.

TypeScript is a superset of JavaScript that adds optional static type definitions to the language; it is essentially JavaScript with static types. TypeScript is a rather popular language, having over 19 million weekly downloads on npm at the time of writing. Furthermore, in 2020, TypeScript was cited as the fourth most used language on GitHub (GitHub 2020). TypeScript is fully compatible with JavaScript, which means that TypeScript applications are free to use any JavaScript frameworks and libraries without an issue, which was a crucial consideration for this project.

TypeScript was picked as the language for this project because it is, in my opinion, a better JavaScript due to the virtue of static typing. Static vs. dynamic typing is a classic topic of debate among software developers, with valid arguments on both sides of the fence[7]. Personally, I lean more towards favouring static typing, and I believe that having static types

[6] See Chapter 6 for further discussion on client-side web applications and SPAs.
[7] See, for example, Meijer and Drayton (2004) and Ousterhout (1998) for perspectives on the topic.

provides benefits for development and especially maintenance of the software. A potential downside of TypeScript is the fact that it requires a build step, where the code is transpiled into JavaScript so that browsers can understand it. With React applications, though, this additional build step does not particularly matter because React applications typically have a build step in any case to package the code for deployment. See Evans (2019) for more information on building React applications.

10.2 Frontend framework

There were two main candidates for a general-purpose web development framework for this project: React and Vue. The reasons for considering those two are simple – they are both mainstream frameworks at the time of writing, and I had prior experience with both. I did not see a compelling reason to explore more exotic frameworks for the purposes of this project, because the requirements of the new software – from the perspective of these general-purpose frameworks – are fairly basic. The data visualization aspects are a different story, however the choice between React, Vue, or some other framework should have relatively little impact on that.

Between React and Vue, I went with the former for a few reasons. Firstly, React is an older and arguably more mature technology than Vue, which should reduce the chances of running into unexpected problems with the framework. Secondly, and perhaps more importantly, React has a larger ecosystem of libraries to choose from. This is especially noticeable when looking into more niche use cases, such as data visualization. While many of the more popular data visualization libraries may, to varying extents, be agnostic about the frameworks they are paired with, coupling them with React is probably the safest bet to avoid obscure bugs and other problems. Thirdly, React is most likely the more popular framework of the two[8], which should help with introducing new people into the project in the future.

[8] At the time of writing, React has roughly five times as many weekly downloads on npm as Vue

10.3 Data visualization library

While the programming language and framework choices were quite easy to make, tackling the challenge of data visualization with JavaScript was significantly more challenging. In fact, I consider selecting a suitable data visualization solution to be the most important tooling-related decision that was made in this thesis. The choice boiled down to deciding if it would be better to go with the perhaps safer but also more difficult to use D3, or one of the higher-level charting libraries introduced in Section 7.3.

Although D3 was a strong contender, my choice was ultimately Victory.js. Victory is an open-source, modular charting library for React applications. It is developed by a company called Formidable, and the first version of the library was released in 2015 (Formidable n.d.). Like many of the other charting libraries mentioned in this thesis, Victory is built on top of D3, and it leverages the low-level rendering functionality of D3. In my evaluation, Victory provides a reasonable amount of baseline components that work well out-of-the-box, while also allowing the programmer to write low-level code to customize the components, as necessary. Furthermore, I believe that Victory has sufficient support for adding and customizing chart element interactions, which was a key consideration for this project. That said, chart interactivity is a topic that should be continuously evaluated as development of the software proceeds. It is also nice that Victory's custom components are exported as React components, which makes integrating Victory into a React application effortless. Lastly, compared to some of the other charting libraries mentioned in Section 7.3, Victory has reasonably good documentation.

Overall, the main reason for choosing a higher-level charting library instead of D3 was to reduce the amount of effort that was required to develop a single visualization component. This way, it was possible to prototype multiple types of visualizations and test their integration with a frontend application. I doubt a similar scope for the thesis would have been possible to achieve, had I implemented the components using D3 instead. That said, exploring the possibilities of D3 (or other charting libraries) would certainly be an interesting endeavour for the future.

10.4 Summary of technologies

Below is a summary of the entire technology stack, including the most noteworthy libraries, used to build the frontend application and the data visualization component library.

Component library and frontend application:

- Written with TypeScript, using React as the framework.
- Unit tests are done with Jest, which is a JavaScript testing framework (Facebook 2021).
- Linters are used to maintain code quality and uniform style. Linting is done with Eslint and Prettier.
- Applications run inside Docker containers. Containers are used to give future developers more options on running their own development environments.
- Continuous integration (CI) workflows are done with GitHub Actions. These workflows involve automatically running tests and linters on code changes.
- Continuous delivery (CD) consists of automatically creating and pushing latest container images to Docker Hub.

Component library only:

- Uses the Victory charting library as the foundation for the custom charts.
- Uses the Storybook.js (Figure 5) library as a local development environment. Storybook is a component explorer tool that can be used to develop UI components in isolation (Chroma Software n.d.). Storybook can also double as a documentation tool, which is convenient.
- Application is packaged as a library using Rollup. Rollup is a module bundler that compiles JavaScript code into a production-ready package (Rollup n.d.).

Frontend application only:

- State management is done with Redux. State management is discussed in Section 11.2.

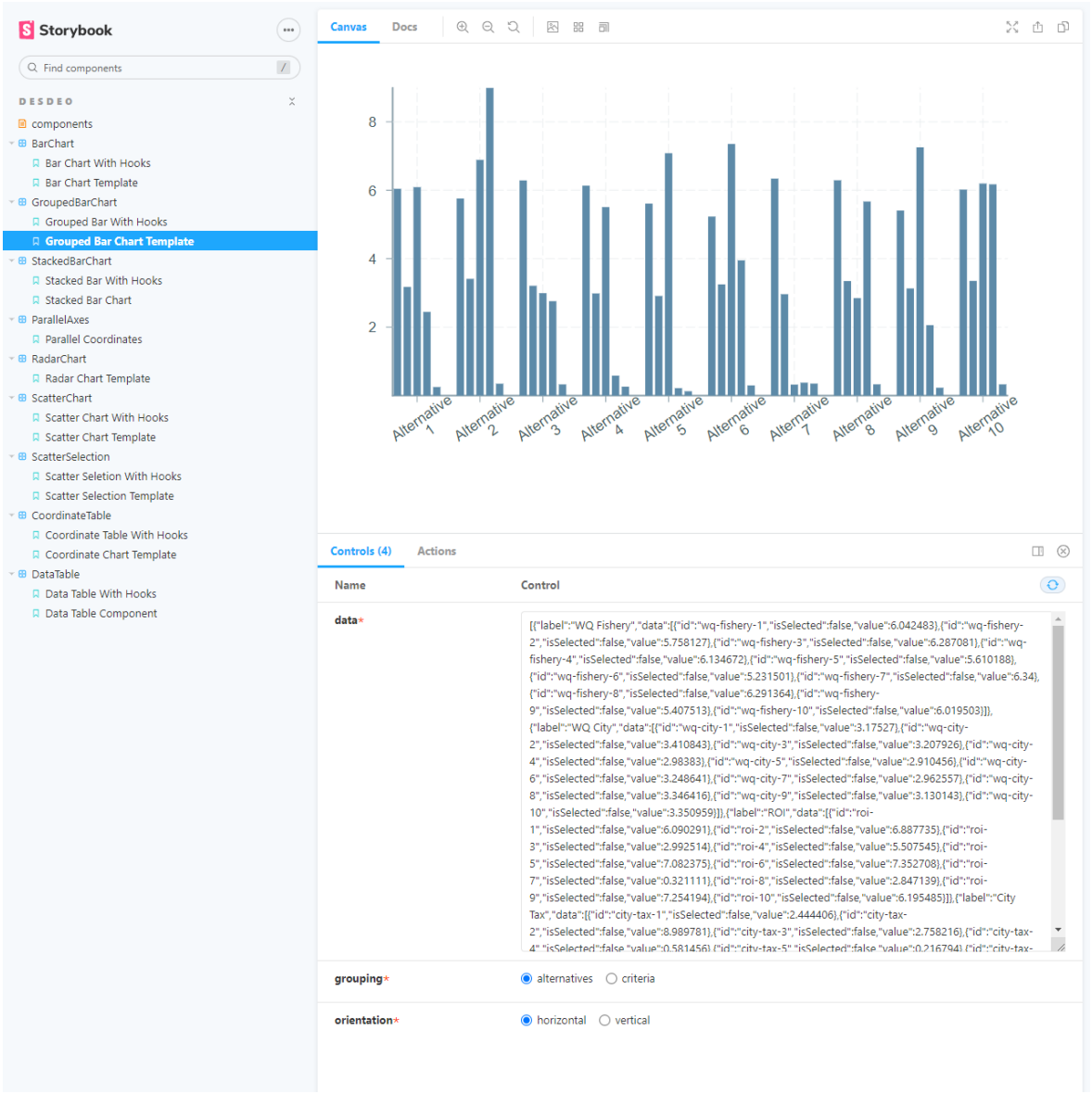


Figure 5. Storybook's UI

11 Software design choices

This chapter discusses the most noteworthy design decisions that were made when building the frontend application and the visualizations components. Some of the choices were established in the planning stage of the project, whereas others were informed by discoveries made during application development.

11.1 Delivering visualizations as a component library

One of main requirements – requirement #3 in Section 9.2 – was to develop the visualization components in a modular manner, so that each component could function independently. The focus on modularity culminated in developing the visualization components as a stand-alone library. Hence, there were separate code bases for the frontend UI application and the data visualization components. The component library was labelled *desdeo-components* and the frontend UI application was called *desdeo-frontend*. Desdeo-components was released into the Node package registry, where it is publicly available under the same name. Before discussing the implications of the component library approach, I want to briefly go over the history behind software components, and their more modern connotations.

McIlroy (1969) is commonly credited for popularizing the idea of dividing software into components. In his paper, McIlroy argued for adopting “mass production techniques” in software development and drew parallels between traditional (at the time) industrial processes and software development practices. McIlroy’s thesis was that software industry was weakly founded, because there was no “software components subindustry”, which could produce “interchangeable standard parts” for software. To illustrate his point, McIlroy said that writing a compiler begins by asking “What table mechanism shall we build?”, when he believed that the correct question to ask was “What mechanism shall we use?”. In other words, he thought that the problem was common enough that it should have an “off the shelf” solution.

Moving on to the 21st century, *component* has become a mainstay in the software development lingo; indeed, Crnkovic and Larsson (2001) note *component-based development* as a new trend in software development. A decade later, Crnkovic et al. (2011, 593) consider *component-based software engineering* to be an established area of software engineering, with its roots in, among other things, object-oriented design. While it is generally agreed that software can be built using components, there is no universally agreed-upon definition for what a component is. Crnkovic and Larsson (2001) offer four definitions found from literature. To quote the most relevant parts for this thesis, a component is "a non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function..." and "A software component can be deployed independently and is subject to composition by third party.". These definitions fit desdeo-components well.

Regarding the way in which component-based systems are built, Crnkovic et al. (2011, 596) maintain that component development occurs in multiple stages, which are "requirements, design, implementation, deployment, and execution". Further, they present these stages in the form of a component lifecycle model (Figure 6). The development work of desdeo-components closely matched this component lifecycle model.

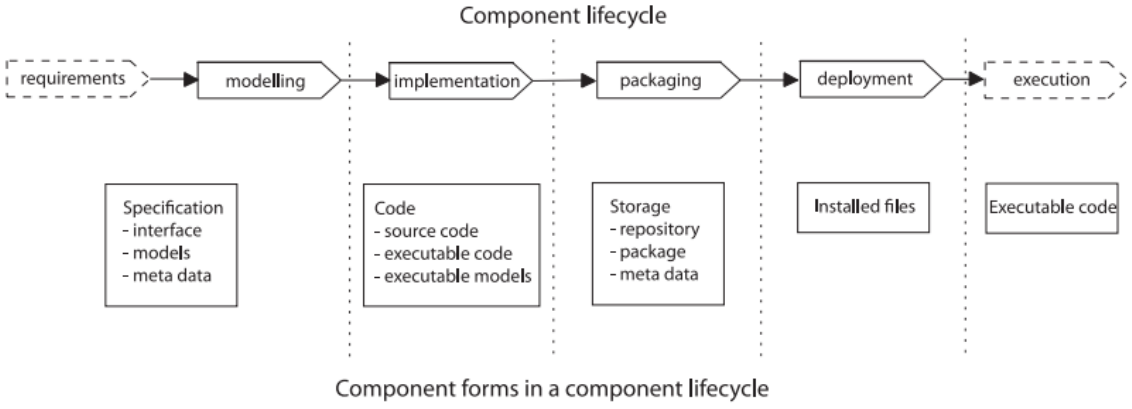


Figure 6. The component lifecycle (Crnkovic, Sentilles, et al. 2011, 596)

From a development perspective, the primary benefit of the component library approach was that it allowed components to be developed and tested in isolation, without having to worry about the context in which they are used. Additionally, I found that while it was initially more difficult to fully grasp the interplay between the visualization components and

the frontend application, creating a hard separation between the two forced me to pay close attention to the components' public APIs, thus hopefully improving their design. Finally, having separate code bases meant that it was possible to freely work on the components without affecting the frontend application in any way. The most notable downsides were the increased complexity in managing component interaction with the frontend application and the need for additional tooling to develop the components (Storybook being the most notable development tool used).

11.2 Managing state in the frontend application

Requirement #5 in Section 9.2 identified developing an application-level state management solution as an important requirement for the project. The purpose of the requirement was to ensure that any modification to the data set would be reflected in all visualization components. Fulfilling this requirement quite naturally leads towards adopting a "single source of truth" approach to managing application state. Single source of truth is a principle that states that there should be a single authoritative source for a piece of data. The principle has been applied in many contexts, from network automation (McGillicuddy 2020) to service-oriented architecture (Pang and Szafron 2014) and state management in web applications (Reduxjs n.d.) – the last of which is of interest to this thesis.

State management is a rather common problem in frontend web applications, and various solutions have sprung up to address it over the years. Some well-known examples include the Flux architecture and libraries like MobX and Redux. Without going into specifics, they use a centralized data store that is somewhat akin to a database in a server-side application. The store is accessed with various mechanisms to read the application state and to make modifications to the state. In addition to standalone state management libraries, popular frontend frameworks also provide tools for managing application state. React, for example, addresses state management via class components[9] and state hooks[10]. Desdeo-frontend uses Redux for state management because its event-driven state management system fits rather naturally into the event-based communication style of the visualization

[9] See https://reactjs.org/docs/state-and-lifecycle.html for code examples
[10] See https://reactjs.org/docs/hooks-state.html for code examples

components (see the following section for discussion on event handling). Furthermore, Redux is a commonly used state management library, particularly with React applications, which should make it familiar to future developers as well. Ultimately, though, the specific state management technique is probably not very important, and a working solution could be achieved with other tools as well. For example, React's built-in state hooks would probably be sufficient for desdeo-frontend's current needs.

What is important, however, is to note that the visualization components are stateless, and all application state is maintained solely in the frontend application. Alternatively, it would have been possible to have each component manage its own state or perhaps devise some sort of a state management solution that is integrated into the component library and shared across all components. Overall, I am happy with the choice of going with stateless components and lifting the application state up to the frontend application. This way, application state is managed in one place and data only flows one way[11], from frontend application to a component. Furthermore, not managing state in the components increases their flexibility, because they can be used in a static way to present arbitrary data. If one wants to take advantage of the interactive features, which are important in the context of interactive multiobjective optimization methods, the frontend application should implement an event-driven state management system. This will be discussed next.

11.3 Event handling in visualization components

Requirement #4 in Section 9.2 identified interactivity as an important feature for the visualization components. Coming up with a way to provide sufficient interactivity was perhaps the most difficult challenge tackled in this thesis. As was hinted above, the final design was to leave the event handling functionality as the responsibility of the frontend application, not the visualization components.

As an example of an event, clicking on a datum is recognized in most visualization components. When a visualization component registers a click event, it passes it on to the

[11] For an explanation of one-way data flow, see, for example, https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow.

frontend application via an event handler function. These event handler functions are given to the component as a prop[12] by the frontend application. Due to this, the frontend application has control over what specific user actions (like clicking on a datum) do in the application; clicking on a datum might, for instance, remove it from the data set, duplicate it, or indicate that it has been selected. It is, of course, also possible for the frontend application to not implement event handlers for some or all user actions, in which case an action simply does nothing.

```
9  const GroupedBarChart: React.FC = () => {
10   const data = useDataset()
11   const dispatch = useDispatch()
12
13   const onClick = (clicked: Value): void => {
14     dispatch(selectDatum(clicked))
15   }
16
17   return (
18     <GroupedBar data={data} grouping={'alternatives'} orientation={'vertical'} onClick={onClick} />
19   )
20 }
21
22 export default GroupedBarChart
23
```

Figure 7. The “onClick” callback function is passed to a component as a prop

The event handler functions discussed above are an example of a *callback function* (sometimes called a *call-after function*). Callback functions are commonly used in JavaScript to sidestep issues that stem from the fact that JavaScript is a single-threaded[13] programming language. A callback function is “a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action” (MDN Web Docs 2020). Alternatively, Dabek et al. (2002, 186) note that callback functions may be used in situations where “a program cannot complete an operation immediately because it has to wait for an event”. The above is true, however my case was somewhat different; the reason to use callback functions was to expose the event handling functionality out of the components. To phrase that in another way, using a callback function in

[12] Props are a way to pass values to React components. Cf. function parameters.
[13] Being single-threaded means that only one thing can ever happen at a time in code. This is bad in situations where certain operations take a long time to complete, because the program is forced to wait until the action is completed. A common example of a potentially slow operation is calling a web endpoint, which may be slow to respond.

a component allows the component to perform an operation that it cannot do by itself, like manipulate an external data store.

Relegating the implementation of event handling – including the responsibility of defining what the events do – as the frontend application's responsibility has the clear benefit of making the visualization components flexible, and thus re-usable. This is because the components' behaviour is not predetermined, but rather defined by the context in which they are used. There are, however, also downsides. Firstly, I discovered that the Victory charting library has an internal event handling system that is designed to manage user actions. My solution essentially bypasses the internal system, which introduces some limitations for how the charts can react to events. For example, it would be possible to achieve fairly granular styling for chart elements by using the internal event handling, however styling elements via "external" events is comparatively limited. Secondly, my solution adds some complexity into the interaction between visualization components and frontend application.

11.4 Wrapper and rendering components

Most of the visualization components were implemented with two layers; they have an inner private rendering component and an outer public wrapper component. The inner rendering component implements all the functionality in a particular visualization component, such as drawing the chart and handling any user actions that the component supports. The inner rendering component is wrapped inside an outer component, which serves two purposes: Firstly, it exposes the public API of a chart and hides its implementation details. The implementation details remain hidden from the consumers of the library because the library only exports the wrappers. Secondly, a wrapper component processes the inputs that are given to it by the frontend application so that the inner rendering component can easily use them. These inputs can be things like controlling the layout of the chart (horizontal vs. vertical orientation) or, most notably, the data set that should be rendered. Regarding data sets, it is important to understand that data must be given in a correct shape to the rendering components.

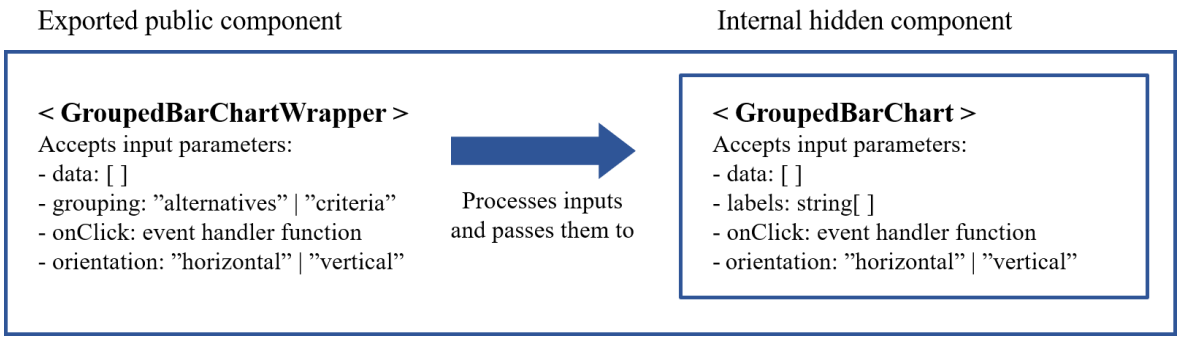


Figure 8. Example of a wrapper component and a rendering component

As was mentioned above, the rendering components are given a data set as a parameter, and the shape of the data set matters. This is because the rendering components are built on top of the Victory charting library. The charts in Victory render their elements (bars, coordinates, lines, etc.) based on the data they are given, which means that the shape of the input data often directly determines how the charts are drawn. As an example, the data set for grouped bar charts (Figure 9) is made of nested arrays that create the bar groups (“alternatives”) seen in the visualization. One might expect that the same nested array data structure can be given to other types of visualization components as well, however that is not always the case. For instance, if one wants to try out a visualization technique where the groups (alternatives) are stacked on top of one another, the items in the nested arrays must be organized differently. Hence, the same data set must be transformed into specific shapes that the different chart types can understand and display correctly. Doing these data transformations is one of the main responsibilities of the wrapper components.

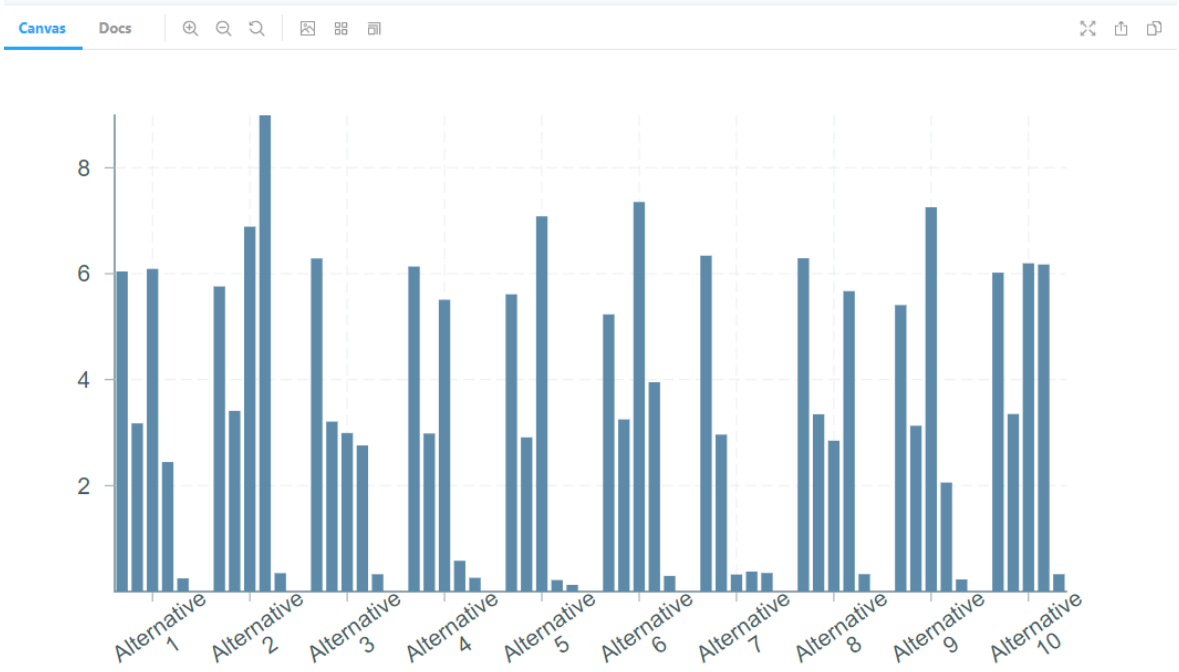


Figure 9. Grouped bar chart shown in Storybook

It would have been possible to not make a distinction between wrapper and rendering components, and rather combine the two into a single entity. I maintain, however, that having wrappers and renderers improves the overall design of the components, because they help enforce a clear separation of concerns. The wrappers handle external inputs (from a frontend application) and perform much of the conditional logic involved in using a particular visualization component (such as the question of horizontal or vertical orientation). Importantly, using wrappers allows the frontend application to pass the same data set, in the same shape, to the different visualization components. The wrapper then transforms the data set into a shape that the rendering component understands. Handling these concerns in wrappers leaves the rendering components relatively free of logic. The result is that the rendering components can be written in a mostly declarative style, where they simply render elements based on the inputs they receive from their wrappers.

12 Results

This chapter summarizes the results of the thesis. The chapter starts off by outlining the work that was done in this thesis. The following two sections describe the tangible IT artifacts and other intangible artifacts that were produced. The last two sections highlight the successes and shortcomings of the thesis project.

12.1 Summary of work

The most notable types of work done in this thesis were:

1. Requirements gathering to identify what I should be building. The requirements were evaluated and refined throughout the thesis process.
2. Identifying technologies – including programming languages, frameworks, and libraries – that are suitable for the new applications.
3. Defining architectural styles and design patterns to serve as the foundations of the development work.
4. Development work, including writing automated test suites.
5. Building CI/CD pipelines to assist development work and improve code maintainability.
6. Writing this thesis to report the results.

It should be noted that although the above summary of work is presented as a numbered list, the work was not done in a strictly ordered fashion, and some activities continued throughout the project.

12.2 IT artifacts

This thesis produced two new applications – desdeo-components and desdeo-frontend. The first is a component library that implements multiple data visualization techniques and the second is a SPA that serves as a web user-interface for the new components. Links to the

source code for both applications are found in Appendix B. Screenshots of the visualization components are found in Appendix C.

Desdeo-frontend is a rather conventional client-side web application that was built in a SPA style with React. It uses Redux for state management and a handful of common libraries for things like routing, unit testing and linting. The primary responsibilities of the application are implementing basic page layout and navigation features, integrating the visualization components with one another via a global state management system, and defining the behaviour of charts on user actions (f. ex. mouse clicks or selections) via event handlers. Additionally, the frontend needs to communicate with DESDEO's backend services via some mechanism. There were some initial plans for how the integration with the backend could be done, however that fell outside of the scope of this thesis. I think the most interesting features of the frontend application are the way it communicates with the data visualization components via events and the way those events are used to manipulate the application's state. These topics were discussed in Sections 11.2 and 10.3.

Desdeo-components contains several types of interactive visualization techniques. The visualizations were designed with the domain of multi-objective optimization in mind, however they should work in other contexts as well. The visualization techniques that were implemented include bar chart, scatter chart, value paths, data table and an incomplete spider-web chart. The spider-web chart lacks any interactive functionality, but it can still be used in a static way to display data sets. The spider web chart was left in an unfinished state due to lack of time and development difficulties with the chart's event handling. The component library was built with React and it heavily leverages the Victory charting library. The library was designed with the expectation that it would be expanded in the future, and that is reflected in its modular design. The component library is publicly available in the Node package registry, which means that it can freely be used in any frontend Node application.

WQ Fishery	WQ City	ROI	City Tax	Plant Resources
6.042483	3.17527	6.090291	2.444406	0.248895
5.758127	3.410843	6.887735	8.989781	0.346752
6.287081	3.207926	2.992514	2.758216	0.326688
6.134672	2.98383	5.507545	0.581456	0.259547
5.610188	2.910456	7.082375	0.216794	0.126336
5.231501	3.248641	7.352708	3.951754	0.295807
6.34	2.962557	0.321111	0.377181	0.35
6.291364	3.346416	2.847139	5.67065	0.328574
5.407513	3.130143	7.254194	2.057297	0.228541
6.019503	3.350959	6.195485	6.173211	0.327455

Figure 10. The same data set with three linked visualizations

Some of the most important issues that were considered during development were chart interactivity to user input and reacting to changes in their data. I think this thesis was reasonably successful in addressing the latter problem of linking the charts together so that they all react to any changes in the data set. Figure 10 is a screenshot of desdeo-frontend, where two grouped bar charts and a data table are shown on the same page. All three charts display the same data set. As can be seen from the screenshot, several items in the data set have been selected, which is indicated by the red colour. Selecting or unselecting an item in one chart immediately updates all other linked charts as well. Handling user input was a more difficult problem to tackle, as was foreseen in Section 2.4. The visualization with best support for user interactions is probably value paths (Figures 11 and 12), which allows user to brush on the vertical bars that represent different criteria to filter out undesirable alternatives (alternatives are shown as orange horizontal lines). Furthermore, hovering on a vertical brush bars displays a label with the value of the objective function next to the mouse cursor. Most other visualization components support data selection with mouse clicks, as well as show relevant labels on mouse hover.

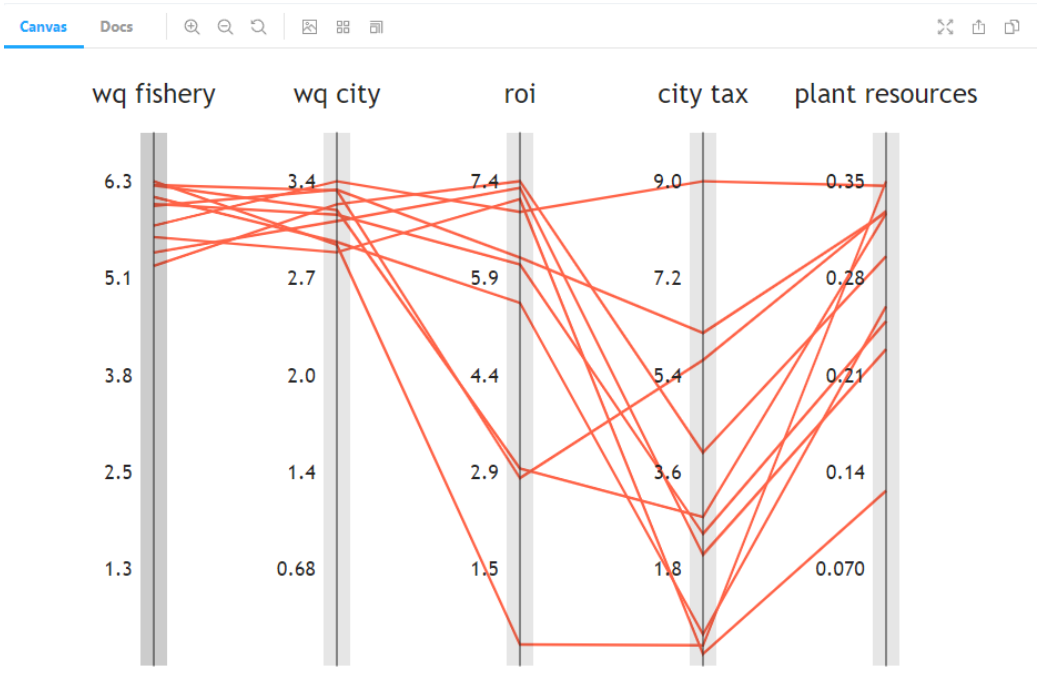


Figure 11. Value paths – initial state

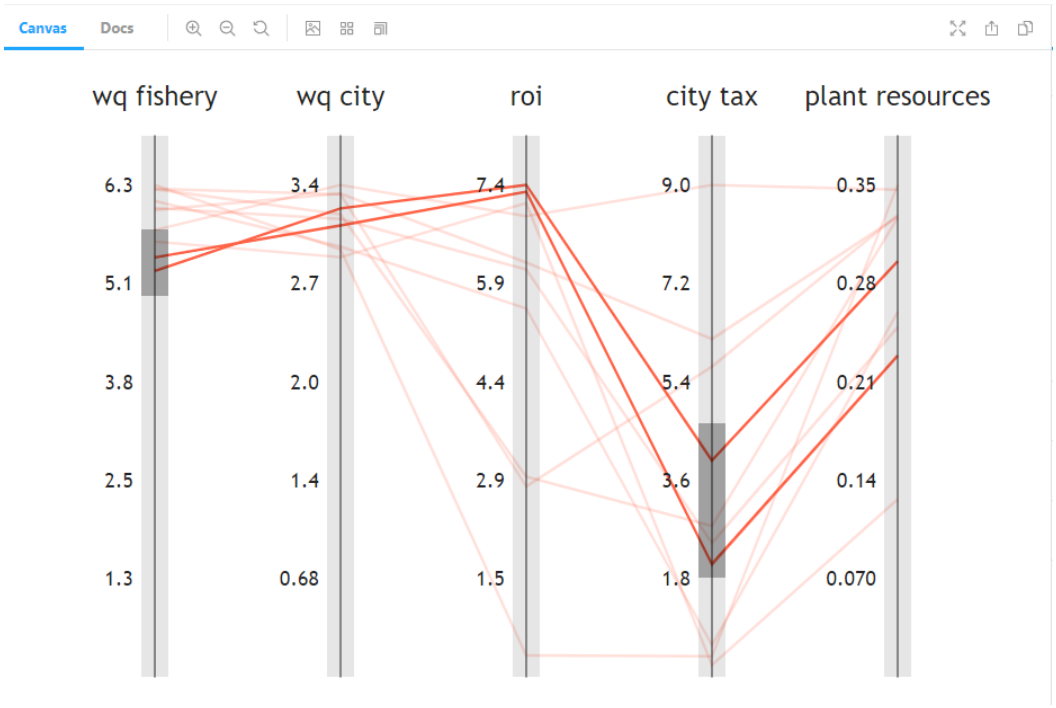


Figure 12. Value paths – some alternatives have been filtered out

Desdeo-components and desdeo-frontend demonstrate one possible approach for building interactive web UIs using customizable data visualization components. Hence, I present the two applications as the tangible IT artifacts produced in this thesis. Additionally, I present the design and architectural features that were discussed in Chapter 11 as complementary intangible artifacts. These design directions explain the functionality of the applications and establish guidelines that can be applied for further development.

12.3 Other intangible artifacts

In addition to the IT artifacts discussed above, the thesis produced other, less tangible artifacts. These artifacts emerged, on one hand, from the information that enabled the development of the new applications and, on the other hand, from lessons learned while developing the new software.

Before development work could begin in earnest, there was a need to establish basic requirements for the new applications to define what I was building. The results of the ad-hoc requirements gathering are presented in Chapter 9. While the requirements gathering

process was unrefined, it still managed to uncover a few key details to guide development efforts. Most notably, interactivity and reactivity (to changes in data) were seen as important features to prototype in the new software, and they will most likely continue to be essential requirements for future development as well. Hence, I present the current software requirements as an intangible artifact that could serve as the starting point for more strategic requirements gathering and analysis.

It was initially not clear what technologies would be suitable for the project, and therefore exploring available options required a fair amount of effort early in the project. Chapters 6 and 7 lay the groundwork for the choice of technologies and Chapter 10 presents and explains the final choices. I present the list of technologies – as summarized in Section 10.4 – as another intangible artifact that outlines a viable suite of tools for developing interactive data visualization components for the web. The tools identified here could be applied to develop rich and interactive UIs in other contexts as well.

On a more general note, this thesis takes a glimpse into the challenges and possible approaches of visualizing solutions in multi-objective optimization. Although that is an established area of research, re-contextualizing the problem into the domain of the web is rather novel. As such, this thesis attempts to draw together the problem spaces of multi-objective optimization, data visualization, and web development and tackles them together. While this thesis does not produce new design theories, I hope that it has done enough to serve as an interesting case study in this rather unique and challenging problem space.

12.4 What went well

First, I am overall happy with the choice of technologies for this project. The technologies used are a modern, mainstream, and sufficiently mature to facilitate further development. Sticking with widely used technologies should make it easier for others to contribute to the projects in the future. I am also confident that going with TypeScript over JavaScript will contribute to the long-term maintainability of the code base. See Chapter 10 for discussion on technology choices.

Secondly, implementing the data visualization components as a separate component library was almost certainly a good choice. Delivering the components separately from the frontend application makes them re-usable, for example in the case that the frontend application is scrapped and a new one is developed from scratch. The component library approach also allows the charts to be developed independently from the frontend application, which has been a bit of a trend in recent years[14]. See Section 11.1 for further discussion.

Thirdly, going with Victory as the charting library was perhaps a controversial choice. In retrospect, I still think Victory was the correct choice for this thesis, however I suspect that switching to D3 might serve the project better in the long run. The advantages of Victory over D3 are its relatively gentle learning curve (at least in my opinion) and its more high-level approach to building visualizations, which allows one to create basic visualizations fairly easily. See Sections 7.2, 7.3, and 10.3 for further discussion on these topics. These benefits were essential to achieving the volume of work that was done with this thesis; I believe that implementing a single visualization component with D3 to the current standard would have been enough work for the thesis. As such, going with an easier-to-use – but less flexible – library allowed for more exploration of the overall problem space. All that being said, D3 is probably still the premiere choice for complex data visualizations for the web, and thus it may eventually be the way to go for DESDEO.

Finally, the sprawling and perhaps even unfocused nature of this thesis was both a benefit and a hindrance. The benefit was that the thesis was quite successful in identifying several areas of further study and development. These suggestions for further work are discussed in the conclusion of the thesis. The drawback was that this thesis did not delve very deeply into any one topic.

12.5 What could have been improved

The first note regarding technical implementation is that exploring chart interactivity was perhaps left underdeveloped. The charts implemented can handle some user actions, but the feature set is rather limited. As such, adding some more advanced visualization tech-

[14] See component-driven development: https://www.chromatic.com/blog/component-driven-development/

niques with more complex user interactions may still be required to gain confidence regarding the feasibility of the visualization components.

The second note regarding technical implementation is that Victory is rather particular about the shape of the data that it receives, as was discussed in Section 11.4. To get the charts to work correctly, a significant amount of data wrangling (i.e. transforming data from one shape to another) had to be done with the different visualization techniques. Repeatedly transforming the same data set into slightly different shapes is certainly not ideal for application performance, nor for code readability. With small data sets, like the one in Appendix D, the data transformations are not a problem, as indicated by the functionality of the current software. With larger data sets, however, application performance may become a concern. The current solution is to hide the data transformations inside the component library to make the visualization components simpler to use, however I am not sure if that was the best solution to the problem. Further thought should be put into how data sets are passed to each visualization component to reach a good balance between performance and ease-of-use of individual components.

Thirdly, the use of the research method should have been planned more carefully. Although I think design science was a fine choice for this thesis, it was applied in haphazard way. Firstly, it would have been useful to consider what types of artifacts the thesis is most interested in producing. Secondly, the overall research process should have been planned more carefully, particularly in terms of how the artifacts are evaluated. Thirdly, it may have been a good idea to frame the software development and thesis writing process more firmly into predefined iterations. That way, each iteration could have had a specific goal to meet, and the overall process may have been easier to analyse.

13 Conclusions

This thesis explored the problem space of building a web UI for a data-intensive application in a component-driven style. The web UI was developed with the assumption that it may eventually be used as the UI for DESDEO, which implies that the UI should be capable of handling a wide range of user actions in a reactive manner. The thesis mapped out some possibilities and limitations of building such a UI and delivered a proof-of-concept implementation of a web UI with multiple independent visualization techniques. Section 2.2 outlined four research questions for this thesis; those questions are re-visited below.

The first question was if it is feasible to build interactive data visualization components for the web. I would present a tentative yes as the response to this question; although this is a complex problem to tackle, my experiences with this thesis suggest that delivering a functional software product that is fit for purpose is feasible. That said, I expect the development effort required to produce a polished and high-quality piece of software to be substantial. Furthermore, I believe that more requirements gathering and analysis should be done before investing a great deal of resources into further development work.

The second question was about identifying suitable technologies for the application. As was mentioned in the previous chapter, I am happy with the technology choices for the project. I would, however, still put further effort into prototyping other alternatives as well. In particular, I would encourage trying out other data visualization libraries and evaluating how well they meet the requirements set for the application. Out of the libraries mentioned in this thesis (see Section 7.3), D3 is an obvious contender.

The third question was to establish a method for allowing the data visualization components to communicate with one another, as well as the frontend application. This was one of the more difficult problems to solve. The solution that is demonstrated with the frontend application relies on callback event handlers that the frontend application passes to components as props. In this solution, all functionality that the components have is defined by the frontend application, and all communication happens via the frontend application. This means that an individual component is not aware of any other components, nor does it

know anything about the context in which it is used. Although this solution introduces some complexity into the overall architecture, it also serves to reduce the coupling between the components and the application that uses them, which should make the components more flexible and re-usable. I believe that the callback event handler pattern demonstrated in this thesis is a suitable solution to the problem of handling user actions in components.

The fourth and final question asked how application state should be managed. This question turned out to be deeply interwoven with the third question, at least in my implementation. To this, I offer a somewhat open-ended answer that the exact state management solution used is probably not very important. Alternatively, it might be more accurate to say that the way state management should be done depends on how question three is addressed. My solution for handling component communication was event-based, which is a natural fit for an event-driven state management solution. Hence, the frontend application employs Redux as the state management system, and it works as expected. Using a simpler state management solution, like React's state hooks, would most likely also work fine.

Looking at the finished product, I think it is a solid first attempt at establishing the groundwork for future development. It is clear, though, that much work remains to be done. Due to the sprawling nature of this thesis, there are several avenues for further work, suitable for people with varying skillsets. Some ideas are presented below.

Technical topics:

- Try creating similar visualization components with a different charting library and compare the results. My recommendation would be to try out D3 to see if its more flexible toolkit is worth the extra development effort required.

- Try replacing the React frontend application with a server-side rendered application. Writing the application with Python would probably allow it to be included in the current DESDEO framework as the UI module.

- Design and implement a mechanism that allows the frontend application to communicate with DESDEO's backend services. One option that was considered in this thesis was a RESTful API, however other alternatives could also be explored.

- Deploy the frontend application into a production environment and improve the current CI/CD pipelines. For example, build a pipeline for automating the releases for the component library.
- Expand the current selection of data visualization components and/or improve the ones that were implemented in this thesis.
- Benchmark the application's performance with larger data sets.

Topics related to software requirements:

- Review and expand the requirements gathering portion of this thesis. Gather the requirements for the data visualization components and the frontend application in a strategic manner (compared to the ad-hoc requirements gathering done in this thesis) and write the requirements into a proper document.
- Evaluate whether the chart components are versatile enough to be used in different contexts as well.

Topics related to UI design:

- Start planning the UI for DESDEO. Create UI mock-ups.
- Study what types of data visualization techniques would be best suited for the context of interactive multi-objective optimization.

In conclusion, I think this thesis was moderately successful in delivering proof-of-concept software that demonstrates how to build interactive data visualization elements that work in a web browser and can be used in the context of multi-objective optimization. Although the software lacks polish, I think it has value as laying the groundwork for further development. In my opinion, the most notable merits of the thesis are identifying and demonstrating some design patterns and technologies that can be used for further development, as well as uncovering several directions for future work.

Bibliography

Ackoff, Russell L. 1989. “From Data to Wisdom.” *Journal of Applied Systems Analysis* 16: 3-9.

Baskerville, Richard, Abayomi Baiyere, Shirley Gregor, Alan Hevner, and Matti Rossi. 2018. “Design Science Research Contributions: Finding a Balance between Artifact and Theory.” *Journal of the Association for Information Systems* 19 (5): 358–376.

Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer. 2011. “D3: Data-Driven Documents.” *IEEE Transactions on Visualization and Computer Graphics* 17 (12): 2301–2309.

Branke, Jürgen, Deb Kalyanmoy, Kaisa Miettinen, and Roman Słowiński. 2008. “Preface.” In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, edited by Jürgen Branke, Deb Kalyanmoy, Kaisa Miettinen and Roman Słowiński, V–XII. Berlin: Springer.

Chaffee, Alex. 2012. *What is a web application (or "webapp")?* 4 May. Accessed March 29, 2021. https://www.jguru.com/faq/view.jsp?EID=129328.

Chang, Kai. n.d. *Parallel Coordinates. https://syntagmatic.github.io/.* Accessed March 5, 2021. https://syntagmatic.github.io/parallel-coordinates/.

Chen, Min, David Ebert, Hans Hagen, Robert S. Laramee, Robert van Liere, Kwan-Liu Ma, William Ribarsky, Gerik Scheuermann, and Deborah Silver. 2009. “Data, Information and Knowledge in Visualization.” *IEEE Computer Graphics and Applications* 29 (1): 12–19.

Chroma Software. n.d. *Storybook.* Accessed April 21, 2021. https://storybook.js.org/.

Conallen, Jim. 1999. “Modeling Web application architectures with UML.” *Communications of the ACM* 42 (10): 63–70.

Crane, Dave, Eric Pascarello, and Darren James. 2006. *Ajax in Action.* Greenwich: Manning Publications.

Crnkovic, Ivica, and Magnus Larsson. 2001. “Component-based Software Engineering – New Paradigm of Software Development.” *Inviteted talk & Invited report, MIPRO 2001 proceedings.*

Crnkovic, Ivica, Séverine Sentilles, Aneta Vulgarakis, and Michel R.V. Chaudron. 2011. “A Classification Framework for Software Component Models.” *IEEE Transactions on Software Engineering* 37 (5): 593–615.

Dabek, Frank, Nickolai Zeldovich, Frans Kaashoek, David Mazières, and Robert Morris. 2002. “Event-driven programming for robust software.” *Proceedings of the 10th workshop on ACM SIGOPS European workshop (EW 10).* New York: Association for Computing Machinery. 186–189.

Dale, Kyran. 2016. *Data Visualization with Python and JavaScript.* Sebastopol: O'Reilly Media.

Deb, Kalyanmoy. 2008. “Introduction to Evolutionary Multiobjective Optimization.” In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, edited by Jürgen Branke, Deb Kalyanmoy, Kaisa Miettinen and Roman Słowiński, 59–96. Berlin: Springer.

DESDEO. n.d. *About.* Accessed October 17, 2020. desdeo.it.jyu.fi/about/.

—. n.d. *Software.* Accessed April 4, 2021. desdeo.it.jyu.fi/software.

Ecma International. 2020. *ECMAScript® 2020 Language Specification.* Vers. 11. June. Accessed March 31, 2021. https://262.ecma-international.org/11.0/.

Evans, Jacob. 2019. *Creating a Production Build.* 3 October. Accessed April 10, 2021. https://create-react-app.dev/docs/production-build/.

Facebook. 2021. *Jest.* Accessed April 21, 2021. https://jestjs.io/.

Fernández-Villamor, José, Laura Díaz-Casillas, and Carlos Iglesias. 2008. “A comparison model for agile web frameworks.” *Proceedings of the 2008 Euro American*

Conference on Telematics and Information Systems (EATIS '08). New York: Association for Computing Machinery. 1–8.

Few, Stephen. n.d. *35. Data Visualization for Human Perception.* Accessed March 25, 2021. https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/data-visualization-for-human-perception.

Filipič, Bogdan, and Tea Tušar. 2018. "A taxonomy of methods for visualizing Pareto front approximations." *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18).* New York: Association for Computing Machinery. 649–656.

Fink, Gil, and Ido Flatow. 2014. *Pro Single Page Application Development: Using Backbone.js and ASP.NET.* 1st. New York: Apress.

Formidable. n.d. *Victory.* Accessed April 2, 2021. https://formidable.com/open-source/victory/.

Fraternali, Piero. 1999. "Tools and Approaches for Developing Data-Intensive Web Applications: A Survey." *ACM Computing Surveys* 31 (3): 227–263.

GitHub. 2020. *The 2020 State of the Octoverse.* Accessed March 9, 2021. https://octoverse.github.com/.

Gregor, Shirley, and Alan Hevner. 2013. "Positioning and Presenting Design Science Research for Maximum Impact." *MIS Quarterly* 37 (2): 337–356.

Grinstein, Georges G, and Matthew O. Ward. 2001. "Introduction to Data Visualization." In *Information Visualization in Data Mining and Knowledge Discovery*, edited by Usama Fayyad, Georges G. Grinstein and Andreas Wierse, 21–46. San Francisco: Morgan Kaufmann.

Hägele, David. 2019. "Visualizing Optimization Trajectories." *Master's thesis.* University of Stuttgart, 27 November. Accessed March 5, 2021. https://elib.uni-stuttgart.de/bitstream/11682/10841/1/main-english-digital.pdf.

Haverbeke, Marijn. 2018. *Eloquent JavaScript.* 3rd. San Francisco: No Starch Press. Accessed March 31, 2021. https://eloquentjavascript.net/.

Hevner, Alan R., Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. "Design Science in Information Systems Research." *MIS Quarterly* 28 (1): 75–105.

Hevner, Alan, and Samir Chatterjee. 2010. *Design Research in Information Systems: Theory and Practice.* 1st. New York: Springer.

Horn, Robert E. 2000. "Information Design: Emergence of a New Profession." In *Information Design*, edited by Robert Jacobson, 15–33. Cambridge: MIT Press.

Huang, Weidong, Peter Eades, and Seok-Hee Hong. 2009. "Measuring effectiveness of graph visualizations: A cognitive load perspective." *Information Visualization* 8 (3): 139–152.

Hull, Elizabeth, Ken Jackson, and Jeremy Dick. 2005. *Requirements Engineering.* 2nd. London: Springer.

Iglesias, Marcos. 2018. "Bringing Together React, D3, And Their Ecosystem." *Smashing Magazine.* 21 February. Accessed April 2021, 2. https://www.smashingmagazine.com/2018/02/react-d3-ecosystem/.

Iivari, Juhani. 2007. "A Paradigmatic Analysis of Information Systems As a Design Science." *Scandinavian Journal of Information Systems* 19 (2): 39–64.

Jazayeri, Mehdi. 2007. "Some Trends in Web Application Development." *Future of Software Engineering (FOSE '07).* Minneapolis: IEEE. 199–213.

King, Ritchie S. 2014. *Visual Storytelling with D3: An Introduction to Data Visualization in JavaScript.* 1st. Boston: Addison-Wesley Professional.

Kodžoman, Marija. 2018. "A software framework for interactive visualization of optimization algorithms." *Master's thesis.* University of Zagreb, June 15. Accessed march 5, 2021. https://bib.irb.hr/datoteka/1009209.Final_0036472084_56.pdf.

Koponen, Juuso, Jonatan Hildén, and Tapio Vapaasalo. 2016. *Tieto näkyväksi: informaatiomuotoilun perusteet.* Helsinki: Aalto-yliopisto.

Korhonen, Pekka, and Jyrki Wallenius. 2008. "Visualization in the Multiple Objective Decision-Making Framework." In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, edited by Jürgen Branke, Deb Kalyanmoy, Kaisa Miettinen and Roman Słowiński, 195–212. Berlin: Springer.

Laplante, Philip A. 2018. *Requirements Engineering for Software and Systems.* 3rd. Boca Raton: CRC Press.

Majorek, Jakub. 2020. *14 JavaScript Data Visualization Libraries in 2021.* 23 August. Accessed April 21, 2021. https://www.monterail.com/blog/javascript-libraries-data-visualization.

March, Salvatore T., and Gerald F. Smith. 1995. "Design and natural science research on information technology." *Decision Support Systems* 15 (4): 251–266.

Martin, Robert C. 2017. *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* 1st. Boston: Pearson.

McGillicuddy, Shamus. 2020. "A Network Source of Truth Promotes Trust in Network Automation." *Cisco.* May. Accessed March 27, 2021. https://www.cisco.com/c/dam/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/white-paper-sp-ema-trust-in-network-automation.pdf.

McIlroy, Malcolm D. 1969. "Mass produced software components." Edited by Peter Naur and Brian Randell. *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968.* Brussels: Scientific Affairs Division, NATO. Accessed March 27, 2021. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF.

MDN Web Docs. 2020. *Callback function.* 18 December. Accessed March 27, 2021. https://developer.mozilla.org/en-US/docs/Glossary/Callback_function.

—. 2021. *Introduction to client-side frameworks.* 19 February. Accessed April 1, 2021. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction.

—. 2021. *Introduction to the DOM.* 16 March. Accessed March 29, 2021. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

—. 2021. *SVG: Scalable Vector Graphics.* 15 March. Accessed April 2, 2021. https://developer.mozilla.org/en-US/docs/Web/SVG.

Meijer, Erik, and Peter Drayton. 2004. "Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages." Accessed April 4, 2021. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.394.3818&rep=rep1&type=pdf.

Miettinen, Kaisa. 2008. "Introduction to Multiobjective Optimization: Noninteractive Approaches." In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, edited by Jürgen Branke, Deb Kalyanmoy, Kaisa Miettinen and Roman Słowiński, 1–25. Berlin: Springer.

—. 1999. *Nonlinear Multiobjective Optimization.* Boston: Kluwer Academic Publishers.

Miettinen, Kaisa. 2014. "Survey of Methods to Visualize Alternatives in Multiple Criteria Decision Making Problems." *OR Spectrum* 36 (1): 3–37.

Miettinen, Kaisa, Francisco Ruiz, and Andrzej P. Wierzbicki. 2008. "Introduction to Multiobjective Optimization: Interactive Approaches." In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, edited by Jürgen Branke, Deb Kalyanmoy, Kaisa Miettinen and Roman Słowiński, 27–57. Berlin: Springer.

Mikkonen, Tommi, and Antero Taivalsaari. 2008. "Web Applications - Spaghetti Code for the 21st Century." *Proceedings of the 2008 Sixth International Conference on*

Software Engineering Research, Management and Applications (SERA '08). IEEE Computer Society. 319–328.

Miller, George. 1956. “The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.” *Psychological Review* 63 (2): 81–97.

Multiobjective Optimization Group. 2020. *Modular Structure.* Accessed April 29, 2021. https://desdeo.readthedocs.io/en/latest/software.html#modular-structure.

Murray, Scott. 2013. *Interactive Data Visualization for the Web: An Introduction to Designing with D3.* 1st. Sebastopol: O'Reilly Media.

Ojalehto, Vesa, and Kaisa Miettinen. 2019. “DESDEO: An Open Framework for Interactive Multiobjective Optimization.” In *Multiple Criteria Decision Making and Aiding: Cases on Models and Methods with Computer Implementations*, edited by Sandra Huber, Martin Geiger and Adiel de Almeida, 67–94. Cham: Springer International Publishing.

Okanović, Vensada. 2011. “Designing a Web Application Framework.” *18th International Conference on Systems, Signals and Image Processing.* Sarajevo: IEEE. 1–4.

OpenJS Foundation. n.d. *Node.js.* Accessed March 31, 2021. https://nodejs.org/en/.

Orlikowski, Wanda J., and Suzanne C. Iacono. 2001. “Research commentary: Desperately seeking the ”IT” in IT research – A call theorizing the IT artifact.” *Information Systems Research* 12 (2): 121–134.

Ousterhout, John K. 1998. “Scripting: Higher-Level Programming for the 21st Century.” *Computer* 31 (3): 23–30.

Pandey, Anshul Vikram, Anjali Manivannan, Oded Nov, Margaret Satterthwaite, and Enrico Bertini. 2014. “The persuasive power of data visualization.” *IEEE Transactions on Visualization and Computer Graphics* 20 (12): 2211–2220.

Pang, Candy, and Duane Szafron. 2014. “Single Source of Truth (SSOT) for Service Oriented Architecture (SOA).” In *Service-Oriented Computing*, edited by Xavier

Franch, Aditya K. Ghose, Grace A. Lewis and Sami Bhiri, 575–589. Berlin: Springer.

Poudel, Pawan. 2018. *Elm Compiler.* Accessed March 5, 2021. https://elmprogramming.com/elm-compiler.html.

Quint, Antoine. 2003. “Scalable vector graphics.” *IEEE MultiMedia* 10 (3): 99–102.

Reduxjs. n.d. *Three Principles.* Accessed March 27, 2021. https://redux.js.org/understanding/thinking-in-redux/three-principles.

Rollup. n.d. *rollup.js.* Accessed April 21, 2021. https://rollupjs.org/guide/en/.

Royce, Winston. 1970. “Managing the Development of Large Software Systems.” *Technical Papers of Western Electronic Show and Convention (WESCON).* Los Angeles: IEEE. 328–338.

Saring, Jonathan. 2018. *11 Javascript Data Visualization Libraries for 2019.* 11 September. Accessed April 21, 2021. https://blog.bitsrc.io/11-javascript-charts-and-data-visualization-libraries-for-2018-f01a283a5727.

Senay, Hikmet, and Eve Ignatius. 1994. “A Knowledge-Based System for Visualization Design.” *IEEE Computer Graphics and Applications* 14 (6): 36–47.

Sommerville, Ian. 2011. *Software Engineering.* Boston: Pearson.

Stewart, Theodor, Oliver Bandte, Heinrich Braun, Nirupam Chakraborti, Matthias Ehrgott, Mathias Göbelt, Yaochu Jin, Hirotaka Nakayama, Silvia Poles, and Danilo Di Stefano. 2008. “Real-World Applications of Multiobjective Optimization.” In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, edited by Jürgen Branke, Deb Kalyanmoy, Kaisa Miettinen and Roman Słowiński, 285–327. Berlin: Springer.

Tarkkanen, Suvi, Kaisa Miettinen, Jussi Hakanen, and Hannakaisa Isomäki. 2013. “Incremental User-Interface Development for Interactive Multiobjective Optimization.” *Expert Systems with Applications* 40 (8): 3220–3232.

Tou, Stephen. 2011. *Visualization of Fields and Applications in Engineering.* 1st. Chichester: Wiley.

Tušar, Tea. 2014. "Visualizing Solution Sets in Multiobjective Optimization." *PhD diss.* Jožef Stefan International Postgraduate School. Accessed March 11, 2021. https://dis.ijs.si/tea/Publications/Tusar14phd.pdf.

Vaishnavi, Vijay, and William Kuechler. 2004. *Design Science Research in Information Systems.* Edited by Vijay Vaishnavi, William Kuechler and Stacie Petter. Accessed April 16, 2021. http://www.desrist.org/design-research-in-information-systems/.

W3C. 2016. *Graphics.* Accessed April 2, 2021. https://www.w3.org/standards/webdesign/graphics.

Wallace, Danny P. 2007. *Knowledge Management: Historical and Cross-Disciplinary Themes.* Westport: Libraries Unlimited.

Ware, Colin. 2012. *Information Visualization: Perception for Design.* 3rd. Amsterdam: Morgan Kaufmann.

Wilke, Claus O. 2019. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures.* 1st. Sebastopol: O'Reilly Media. https://clauswilke.com/dataviz/index.html.

Wills, Graham. 2008. "Linked Data Views." In *Handbook of Data Visualization*, by Chun-houh Chen, Wolfgang Härdle and Antony Unwin, 217–241. Berlin: Springer.

Wirfs-Brock, Allen, and Brendan Eich. 2020. "JavaScript: the first 20 years." *Proceedings of the ACM on Programming Languages.* 1–189.

Zhu, Nick. 2013. *Data Visualization with D3.js Cookbook.* Birmingham: Packt Publishing.

Appendices

A Thesis topic proposition, topic 2

Topic 2 - A graphical frontend for DESDEO

DESDEO is an open-source Python based software framework for developing and experimenting with interactive multiobjective optimization methods. While a good amount of interactive multiobjective optimization methods have already been implemented in DESDEO, these methods lack a usable graphical interface. Our goal is to have an accessible web-based interface, which would work in any modern web browser. Currently, we are still prototyping with different interfaces, so we do have usable applications, but development of the final version of the interface is yet to be started.

By choosing this topic, you will learn about many different interactive multiobjective optimization methods. You will also learn about designing and developing graphical web-based interactive interfaces, and you will learn about the archetypical structuring of software where a separate backend and frontend application work in tandem. Lastly, you will be expected to develop interactive graphical elements, which can be used by a user to express various preferences related to the different interactive multiobjective optimization methods being supported by the frontend.

Many of the interactive multiobjective methods implemented in DESDEO do lack an openly available graphical user interface. Thus, you will be answering a clear software need currently existing in our field of research. We expect the graphical frontend to be developed in a way that it can be run in any modern web browser. This most likely means developing the frontend using javascript. You should also be able to combine the frontend with the backend using existing web frameworks, for example flask (https://flask.palletsprojects.com/en/1.1.x/). Moreover, you should also be ready to work on the DESDEO backend to contribute to the API used to communicate with DESDEO. You can expect to write a thesis with a technical emphasis on the frontends and its communication with the backend.

However, we do not expect you to write a comprehensive interface. It does not have to support every currently existing method. However, the frontend should be developed in a maintainable and extensible manner so that it can be later developed to support more methods.

This thesis has a heavy emphasis on developing software. We will still expect you to show familiarity with the basic literature on multiobjective optimization and the development of graphical interfaces. After completing this thesis, you will have a solid basis to start working on other similar projects and your GitHub profile will be ornated with a nice-looking mosaic, which will surely catch the eye of interested parties...

You will learn:

• A hands-on experience on full-stack development where the backend has been developed in Python and the frontend in javascript (or possibly some other web-friendly language!).

• A solid understanding on the basics of visual design. What makes a user-friendly interface user friendly? Come and find out!

• How it feels like to develop pioneering software solutions.

B Source code

Frontend application: https://github.com/mika-alaoutinen/desdeo-frontend

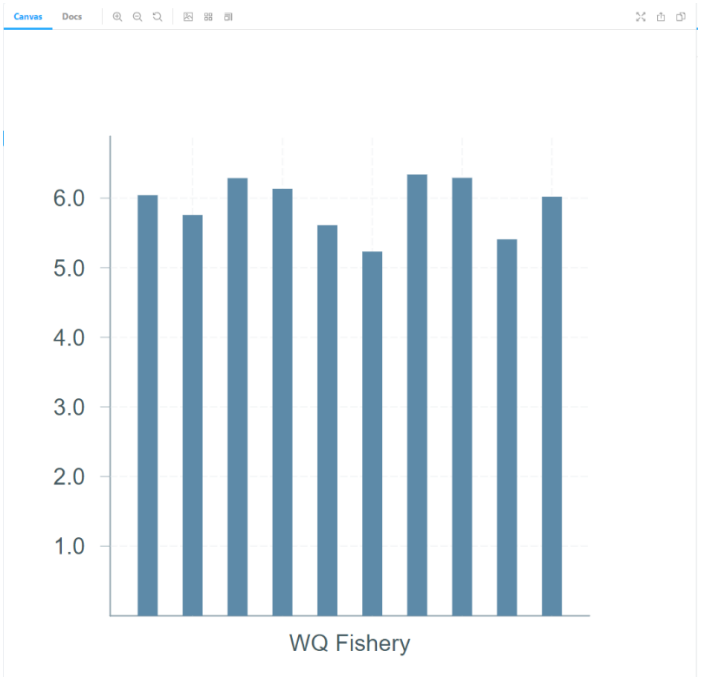
Data visualization components: https://github.com/mika-alaoutinen/desdeo-components

Released npm package: https://www.npmjs.com/package/desdeo-components

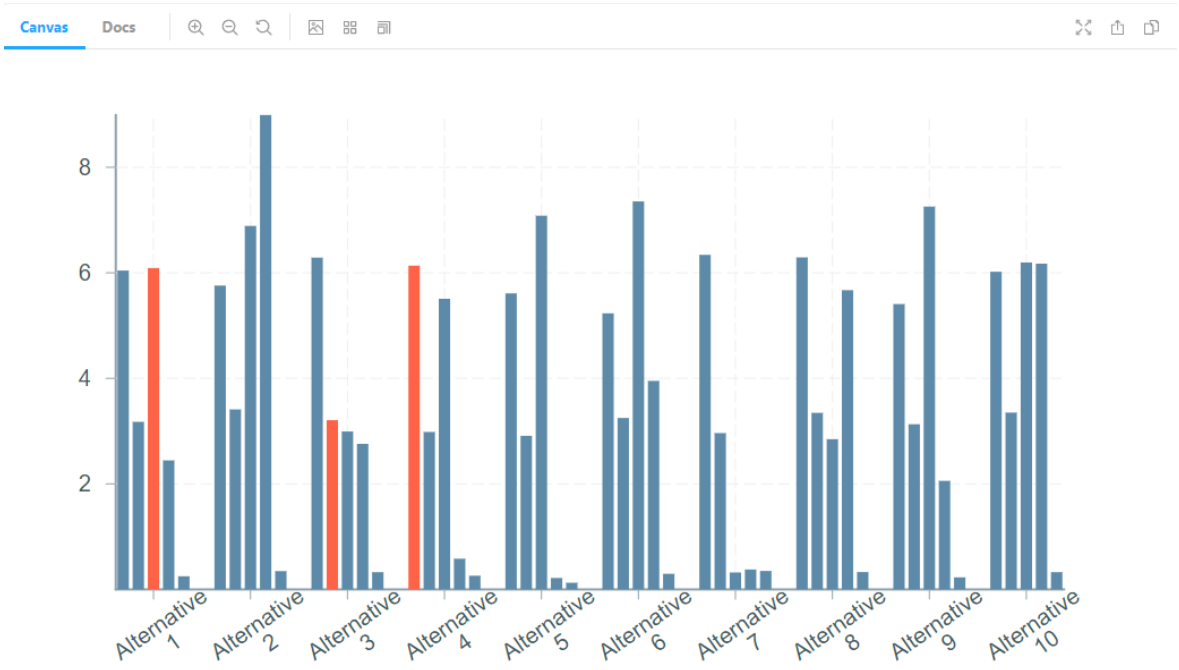
C Visualization component screenshots

Screenshots are taken from a Storybook development environment. The red colour in some visualizations indicates that a piece of data has been selected.

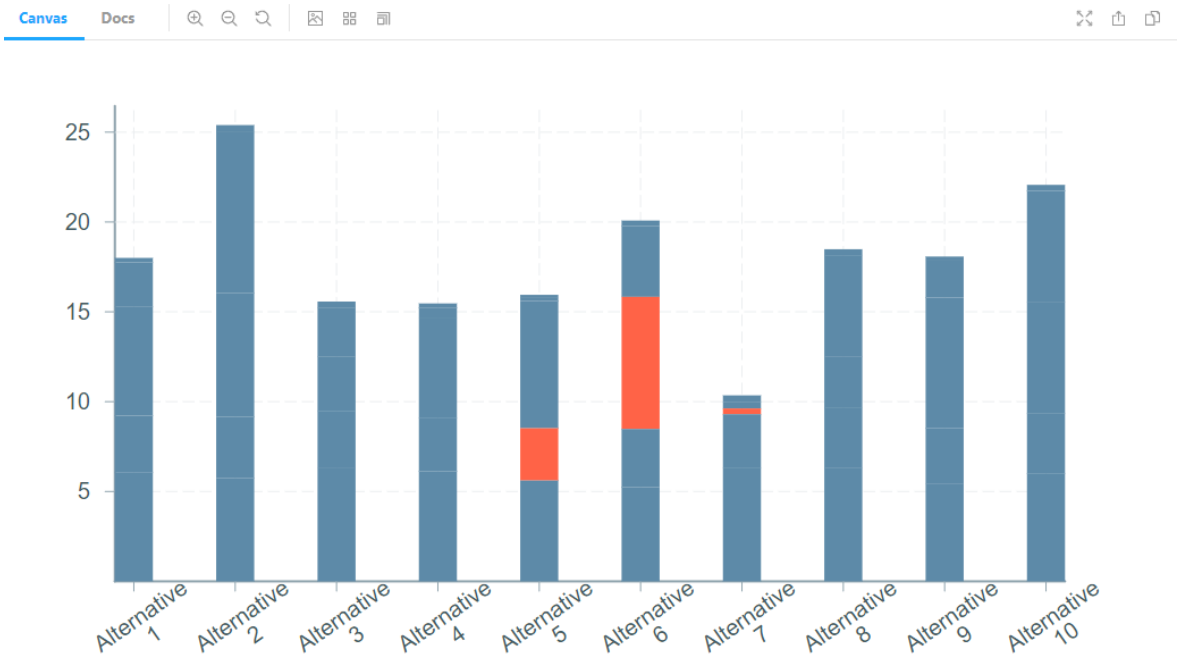
Component #1: simple bar chart



Component #2: grouped bar chart

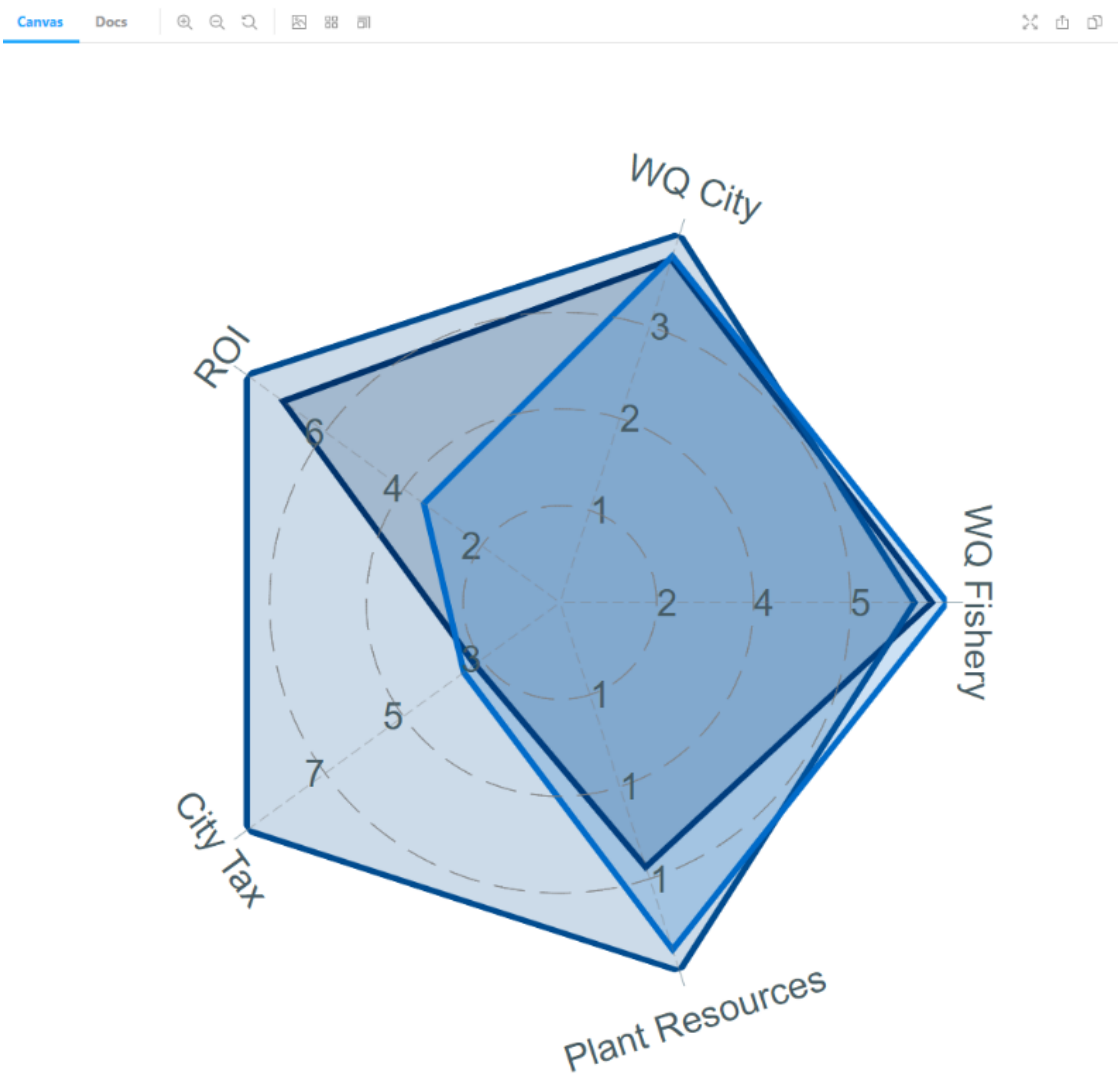


Component #3: stacked bar chart

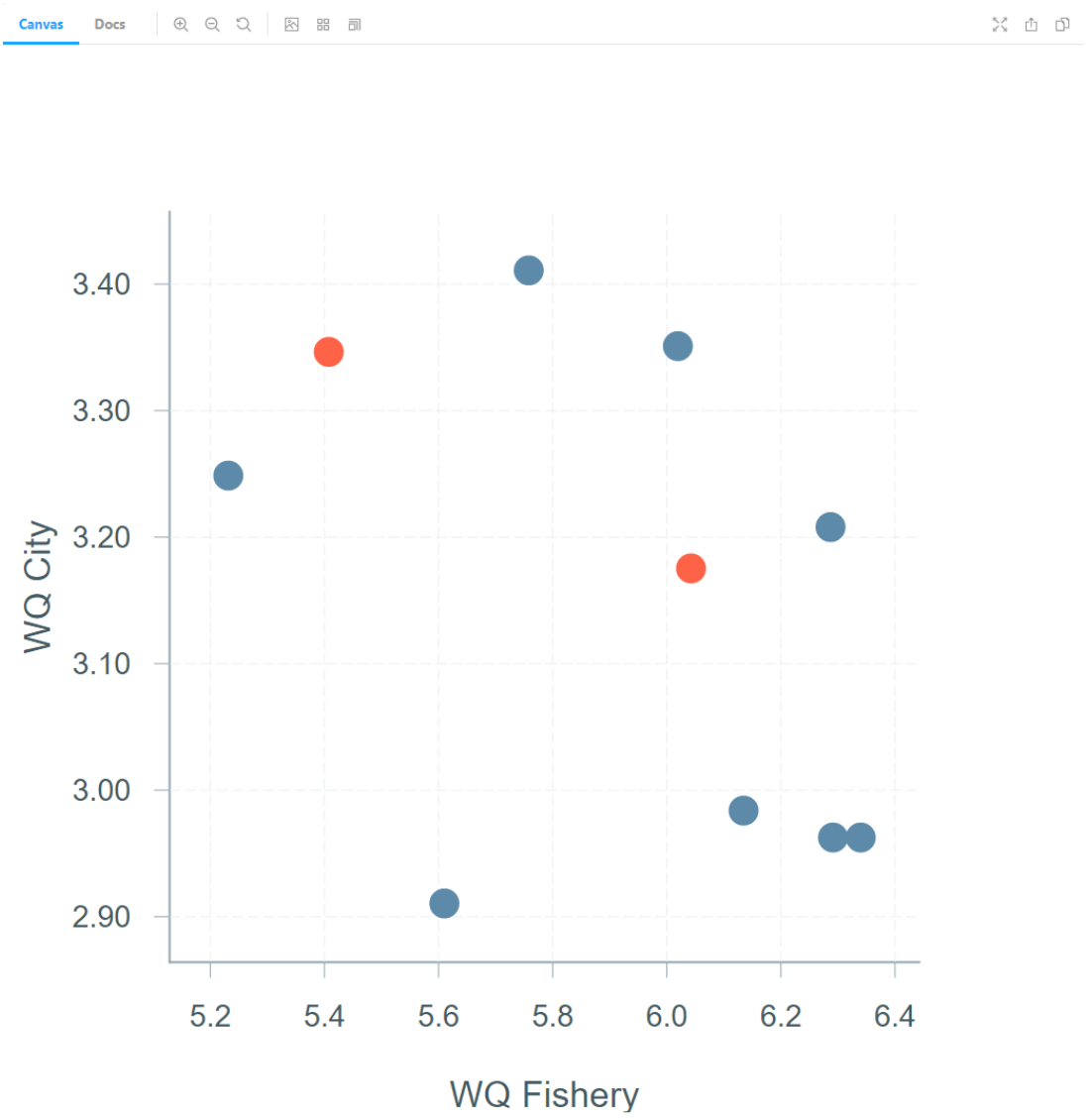


Component #4: value paths

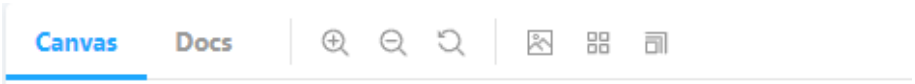
Component #5: spider-web chart



Component #6: scatter plot

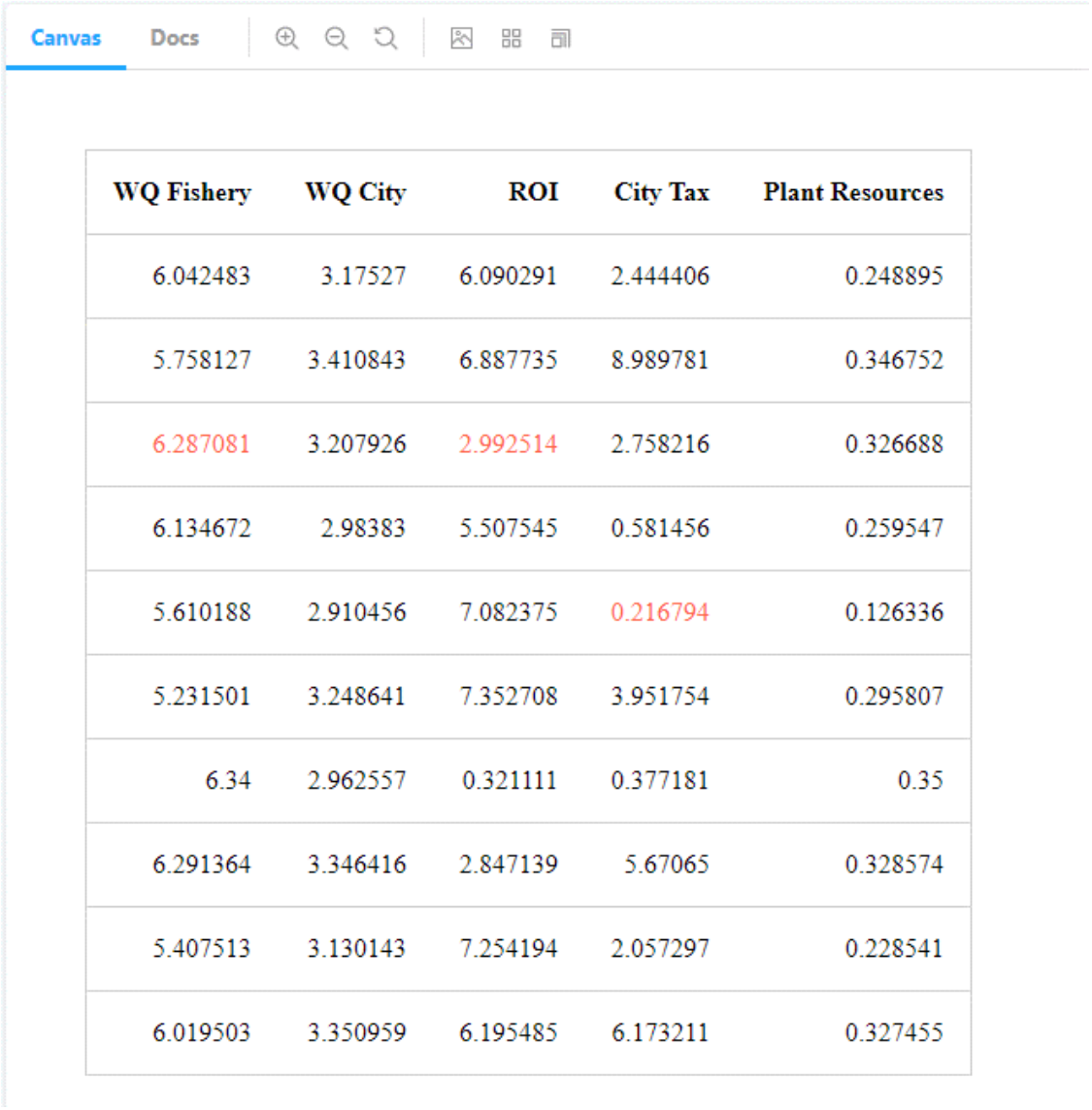


Component #7: basic table



Label	X	Y	Selected
no label	6.042483	3.17527	false
no label	5.758127	3.410843	false
no label	6.287081	3.207926	true
no label	6.134672	2.98383	false
no label	5.610188	2.910456	false
no label	5.231501	3.248641	false
no label	6.34	2.962557	false
no label	6.291364	2.962557	false
no label	5.407513	3.346416	true
no label	6.019503	3.350959	false

Component #8: data table



WQ Fishery	WQ City	ROI	City Tax	Plant Resources
6.042483	3.17527	6.090291	2.444406	0.248895
5.758127	3.410843	6.887735	8.989781	0.346752
6.287081	3.207926	2.992514	2.758216	0.326688
6.134672	2.98383	5.507545	0.581456	0.259547
5.610188	2.910456	7.082375	0.216794	0.126336
5.231501	3.248641	7.352708	3.951754	0.295807
6.34	2.962557	0.321111	0.377181	0.35
6.291364	3.346416	2.847139	5.67065	0.328574
5.407513	3.130143	7.254194	2.057297	0.228541
6.019503	3.350959	6.195485	6.173211	0.327455

D Multi-objective optimization example data

An example data set for a multi-objective optimization problem. This data set is used in the screenshots in Appendix C.

WQ Fishery	WQ City	ROI	City Tax	Plant Resources
6.042483	3.17527	6.090291	2.444406	0.248895
5.758127	3.410843	6.887735	8.989781	0.346752
6.287081	3.207926	2.992514	2.758216	0.326688
6.134672	2.98383	5.507545	0.581456	0.259547
5.610188	2.910456	7.082375	0.216794	0.126336
5.231501	3.248641	7.352708	3.951754	0.295807
6.34	2.962557	0.321111	0.377181	0.35
6.291364	3.346416	2.847139	5.67065	0.328574
5.407513	3.130143	7.254194	2.057297	0.228541
6.019503	3.350959	6.195485	6.173211	0.327455