FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen

Fachbereich 9

Medizintechnik und Technomathematik

Evaluation von JavaScript Bibliotheken zur Darstellung statistischer Ergebnisdaten von Gießsimulationen

Seminararbeit

im Studiengang Angewandte Mathematik und Informatik

Christine Ophoven

Matrikelnummer: 3201716

15.12.2020

1. Prüfer: Prof. Dr. rer. nat. Stephan Bialonski
2. Prüfer: Dipl. Mat. Andreas Rombey

Kurzfassung

Mithilfe der Simulationssoftware MAGMASOFT® für Unternehmen in der Gießereibranche, erhalten Benutzer unter anderem die Möglichkeit, verschiedene Versionen ihres Bauteils und Gießprozesses zu simulieren. Um diese Versionen zu vergleichen und zu bewerten, werden Möglichkeiten angeboten, parametrisierte Werte und Zielfunktionen in verschiedenen Graphen und Darstellungen anzuzeigen.

Auf der technischen Ebene soll im Rahmen einer Bachelorarbeit von der Verwendung von C++ und OpenGL zum Darstellen der Graphen auf eine Implementierung in HTML und JavaScript umgestellt werden, um den Wartungsaufwand zu verringern und die Benutzerfreundlichkeit zu erhöhen.

Als Vorbereitung darauf soll in dieser Ausarbeitung eine JavaScript Bibliothek ausgewählt werden, mit der die Umsetzung erfolgen kann. Zur Anforderungsanalyse wurden verschiedene Anwender von MAGMASOFT® zu ihren Erfahrungen und Wünschen in Bezug auf das Auswerten der Simulationsergebnisse befragt. Nach der Evaluation verschiedener Optionen stellte sich die Bibliothek plotly.js als diejenige heraus, die die gestellten Anforderungen erfüllen kann. Um die Tauglichkeit zu überprüfen wurde eine beispielhafte Implementierung in den Bestandscode von MAGMASOFT® eingebaut.

Inhalt

1. Einleitung

Benutzerfreundliche Auswertung großer Datenmengen mithilfe von JavaScript Graphen - ist das überhaupt möglich? Um diese zentrale Fragestellung zu beantworten, sollen verschiedene graphische JavaScript Bibliotheken miteinander verglichen werden. Die Kriterien zur Bewertung orientieren sich hierbei an den Anforderungen und Bedürfnissen von MAGMASOFT®-Anwendern und Kunden, hinsichtlich der Bereitstellung von benötigten Diagrammen und Bedienbarkeit.

Die Motivation, diese Methode zu evaluieren, resultiert aus der aktuellen Darstellung statistischer Simulationsergebnisse in der Software MAGMASOFT®, mit deren Hilfe Unternehmen in der Gießerei-Industrie die Möglichkeit erhalten, durch Simulation ihre Prozesse zu optimieren. Die Funktionsweite reicht von der Definition eines Bauteils, also einer dreidimensionalen Geometrie, über das Einstellen von Materialeigenschaften bis hin zur Simulation des Gießvorgangs. Nach der Simulation werden die Ergebnisse in Form von dreidimensionalen Bildern und Kurven zur Auswertung bereitgestellt. Es besteht zusätzlich die Möglichkeit, eine statistische Versuchsplanung zu definieren, mit der verschiedene Versionen des Prozesses, im Folgenden Designs genannt, anhand ausgewählter Zielfunktionen direkt miteinander verglichen werden können. Die Software ist dabei in verschiedene Perspektiven gegliedert, die jeweils einen Schritt zur Definition eines Simulationsprojektes abbilden. Angefangen in der Geometrie Perspektive, in welcher der Benutzer sein Bauteil definiert, wird er über die Vernetzung, das Einstellen der Prozess- und Materialeigenschaften und die Eingabe von Freiheitsgeraden und Zielfunktionen zur Berechnung von Optimierungen und DOEs geführt. Um die Simulationsergebnisse auszuwerten, gibt es zwei weitere Perspektiven, in welchen die 3D-Bilder als Graphiken und die statistischen Werte in Graphen dargestellt werden. Ziel der Arbeit soll es sein, herauszufinden, ob die Verwendung von JavaScript Graphen in der Perspektive zur Auswertung der statistischen Daten, der sogenannten Assessment Perspektive, die Benutzerfreundlichkeit erhöht, die Implementierung vereinfacht und welche neuen Möglichkeiten zur Auswertung und Gestaltung diese bietet.Am Beispiel des Vergleiches der Werte der Zielfunktionen in Abhängigkeit der verschiedenen Prozessparameter soll evaluiert werden, ob sich eine JavaScript Bibliothek anstelle der aktuell verwendeten C++ Bibliothek eignet, um in die bestehende Java Implementierung zur Darstellung der Graphen eingebunden zu werden. Ein großer Vorteil einer JavaScript Implementierung liegt darin, dass sie wesentlich flexibler ist. So kann im nächsten Schritt sogar dazu übergegangen werden, die Auswertung als Dashboard im Webbrowser anzubieten. Zudem existiert in der Software bereits eine Schnittstelle zur Integration von HTML und JavaScript, sodass die neuen Elemente mit verhältnismäßig geringem Aufwand integriert werden könnten. Die aktuelle C++ Variante ist eine Eigenimplementierung, die einen hohen Wartungsaufwand bedeutet. In JavaScript gibt es dagegen viele Bibliotheken, die genau für den oben beschriebenen Anwendungsfall spezialisiert sind und besonders bezüglich der Skalierbarkeit deutlich besser abschneiden als die aktuelle Implementierung. Obwohl durch die Umstellung einige Abstriche im Hinblick auf die Individualisierbarkeit gemacht werden müssten, da hier die Möglichkeiten durch die vorgegebenen Schnittstellen und zur Verfügung gestellten Daten durchaus eingeschränkt sind, gäbe es hier gute Möglichkeiten die Graphen den eigenen Bedürfnissen anzupassen. Außerdem sind viele der Bibliotheken Open Source, sodass im Zweifelsfall eine Anpassung oder Erweiterung möglich wäre.

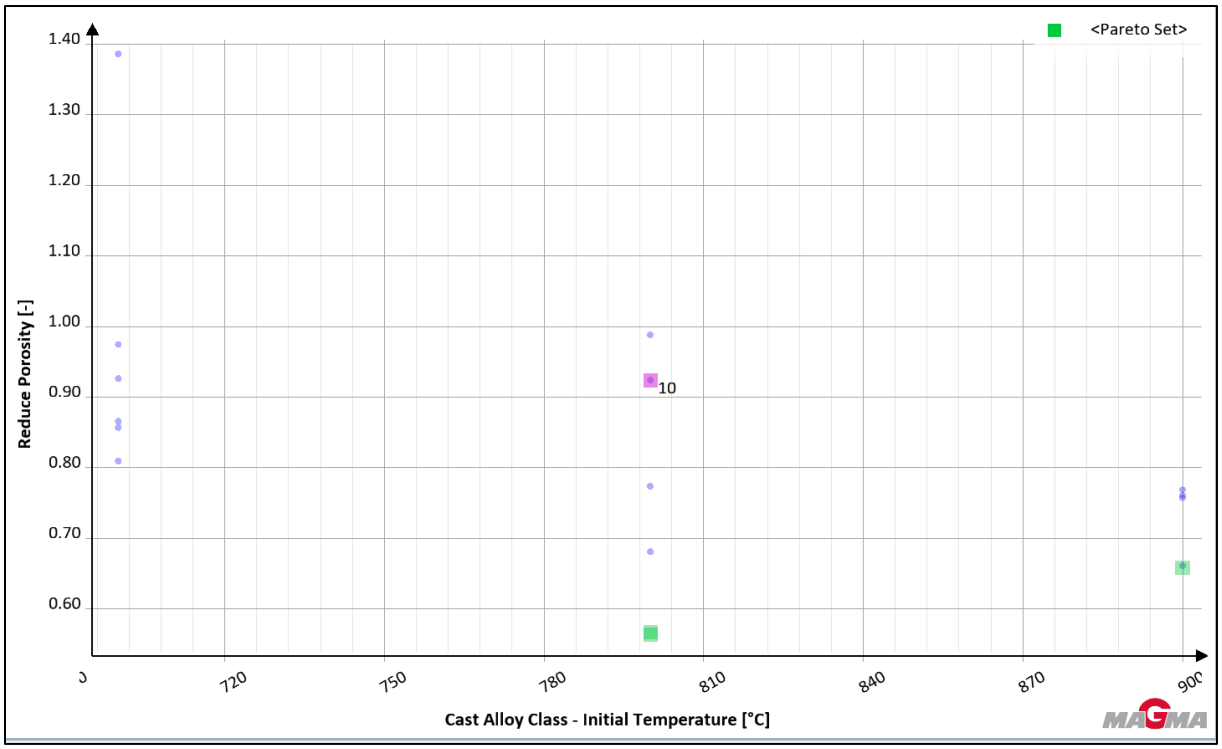


Abb. 1 Aktueller Scatter Chart in MAGMASOFT®

Am Beispiel des Scatter Charts, in welchem hier auf der x-Achse die Anfangstemperatur des Gussmaterials als Designvariable und auf der y-Achse die Zielfunktion zur Reduktion von Porosität dargestellt ist, werden bereits einige Probleme der Skalierung deutlich. So ist hier die überlappende Legende zu sehen, wodurch Punkte, die an dieser Stelle liegen nicht mehr zu erkennen sind. Zudem sind einige Teile der Werte der x-Achse abgeschnitten, obwohl ausreichend Platz für die vollständige Anzeige zur Verfügung steht.

Die Ausarbeitung wird sich vorrangig mit gängigen Bibliotheken und deren verschiedenen Techniken beschäftigen. Die Auswahl der vorgestellten Bibliotheken erfolgte mithilfe von Blogbeiträgen, Nutzungsstatistiken und durch eigene Erprobung von Implementierung und Performance. Hilfreich für einen Überblick über die jeweiligen Funktionen und Möglichkeiten sind besonders die Webseiten der jeweiligen Bibliotheken, da sie viele Beispiele mit verschiedenen Einstellungen vorstellen.

Am Ende der Evaluation wird eine prototypische Darstellung der statistischen Daten eines mit MAGMASOFT® simulierten Gießprojektes entstehen, um die Auswertungsmöglichkeiten mithilfe der am besten abschneidenden Bibliothek im Anwendungsfall zu prüfen.

2. Bereitstellung von Kriterien

Um eine sinnvolle und zielführende Evaluation durchzuführen, müssen im Vorfeld Anforderungen formuliert werden, anhand derer über die Eignung einer Bibliothek entschieden werden kann. Hierbei gibt es verschiedene Kategorien von Anforderungen. Dazu gehören zum Beispiel Funktionsumfang, Benutzerfreundlichkeit oder Lizenzbedingungen. Auf welche Aspekte genauer geachtet werden soll, wird im Folgenden spezifiziert.

2.1 Befragung der Anwender

Um herauszufinden, worauf bei der Auswahl eines passenden Frameworks geachtet werden muss, wurden verschiedene internationale Anwender von MAGMASOFT® gefragt, welche Funktionen sie am meisten benutzen. Sie sollten auch beantworten, welche sie gar nicht benutzen und welche Ideen oder Wünsche an neuen Funktionen sie haben, die mit dem neuen Framework realisiert werden können, damit ihre Möglichkeiten, Simulationsergebnisse auszuwerten noch besser werden. Zurzeit existieren verschiedene Tabellen und Graphen, die eine Auswertung der statistischen Ergebnisse ermöglichen. Aktuell zeigt die Assessment Perspektive beim ersten Aufruf lediglich die Werte der Zielfunktionen als Tabelle und stellt die Möglichkeit bereit, die Geometrie der jeweiligen Designs anzuzeigen. An dieser Stelle besteht die Überlegung, ein Dashboard zu integrieren, um auf den ersten Blick zu sehen, welche Designs in die engere Auswahl genommen werden können. Auch wenn hier noch keine konkrete Designvorstellung existiert, ist es von den Usern gewünscht, dass auf dem Dashboard benutzerdefinierte Graphen und Darstellungen angezeigt werden. An dieser Stelle sollen außerdem lediglich Tendenzen aufgezeigt werden, um das Simulationsergebnis zu bewerten. Aktuell ist die Einstiegsseite eine Tabelle, die die einzelnen Werte der Zielfunktionen enthält.In der britischen Vertriebsstelle besteht außerdem der Wunsch, die Gewichtung der Zielfunktionen besser zu visualisieren. Aktuell wird durch das gewichtete Summieren ein Ranking der Designs bestimmt, wobei die Gewichte benutzerdefiniert sind. Hier ist der Vorschlag, die Bestimmung eines Kompromisses für das beste Design zu bestimmen, indem zwei Zielfunktionen gewichtet gegeneinander aufgetragen werden, um diesen im Kreuzungspunkt der beiden Kurven zu finden. Danach hat der Anwender die Möglichkeit, mithilfe von einem nach Größe sortierten Säulendiagramm gegenläufige Tendenzen zwischen zwei Zielfunktionen zu erkennen. Hier wünschen sich besonders die Anwender aus den USA eine andere Visualisierung der gleichen Daten, um sie leichter auszuwerten. Zur besseren Visualisierung könnte ein Liniendiagramm eingeführt werden. Mithilfe eines Scatter Charts können Anwender die Relation zwischen Zielfunktionen und Design Variablen genauer untersuchen. Es besteht die Auswahl zwischen einem 2D- und 3D Scatter Plot sowie einem Bubble Chart, wobei die 3D Variante bei kaum einem Anwender Anklang findet. Damit Redundanzen verringert werden, wurde der Vorschlag eingebracht, dem Scatter Chart eine zweite y-Achse hinzuzufügen, sodass hiermit auch die einzige genutzte Funktionalität des Säulendiagramms abgebildet wäre.[1]

[1] Vergleiche Abb. 2

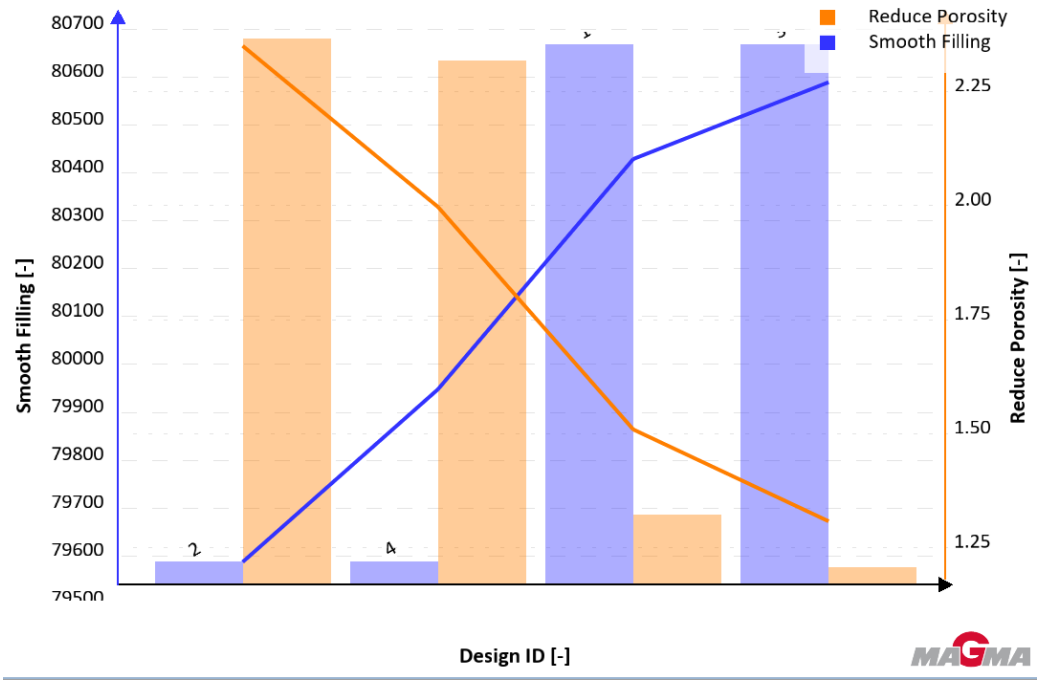


Abb. 2 Gemischtes sortiertes Säulendiagramm mit entsprechenden Kurven aus MAGMASOFT®

Um die Konvergenz der Berechnung einer automatischen Optimierung zu bestimmen, sind die darauffolgenden History Charts gedacht. Hierzu gab es in der Befragung keine Anmerkungen, sie sollten jedoch in jedem Fall weiterhin unterstützt werden.Unter allen internationalen befragten Anwendern erhält das Parallelkoordinatendiagramm[2] den größten Zuspruch. Besonders wichtig ist hier das interaktive Verschieben der Achsen, um die besten Designs herauszufiltern. Als zusätzliches Feature ist hier lediglich gewünscht, dass die Filterwerte der einzelnen Spalten per Tastatureingabe gesteuert werden können.

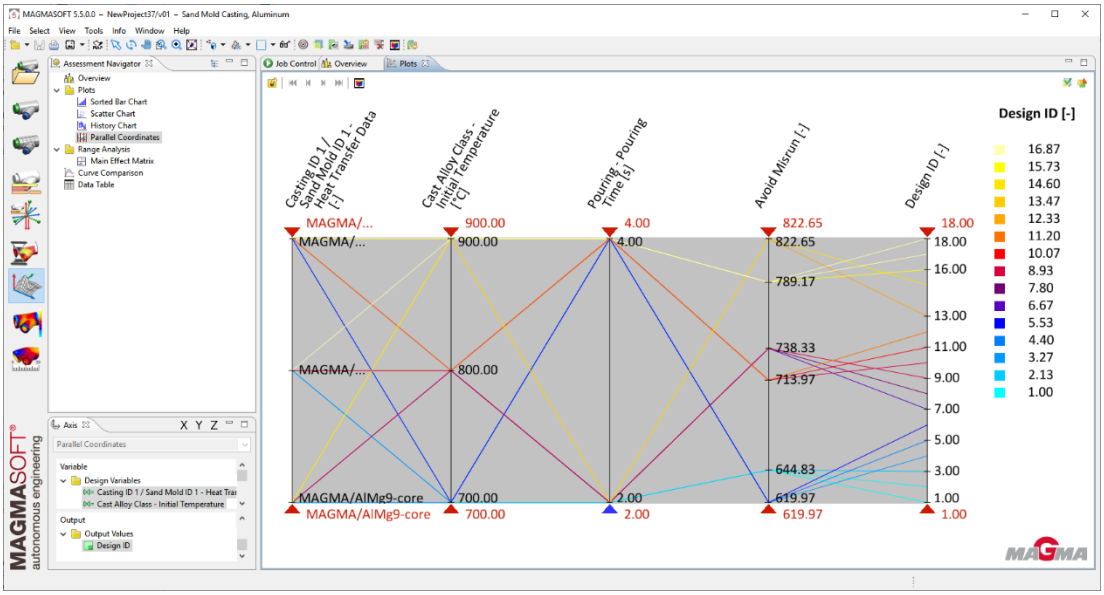


Abb. 3 MAGMASOFT Parallelkoordinatendiagramm

[2] Vergleiche Abb. 3

Durch die Zusammenführung von Korrelationsmatrix und Main Effect Diagrammen zur „Main Effect Matrix“[3] ist in der neuen Softwareversion bereits ein Teil der Kundenwünsche in diesem Bereich umgesetzt. Aus Großbritannien kam jedoch der Vorschlag, anstelle der Korrelationswerte die Übereinstimmung mit dem Optimierungsziel farbig zu markieren.

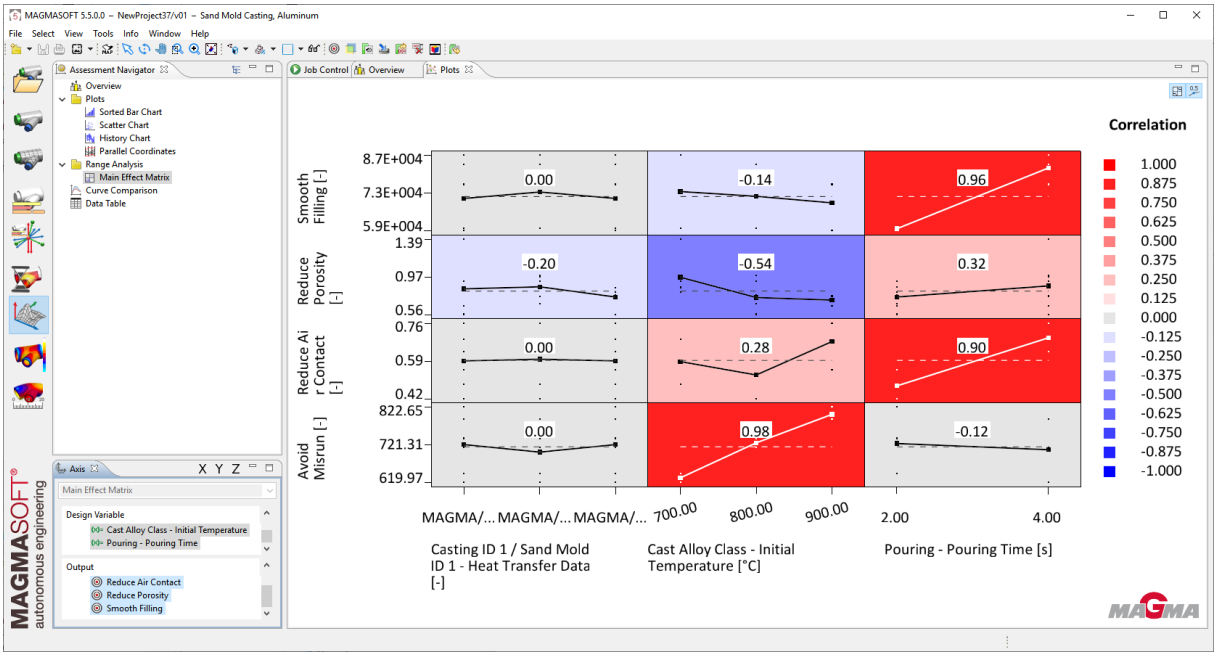


Abb. 4 MAGMASOFT "Main Effect Matrix"

Zusammengefasst möchte also der Großteil der Anwender die aktuellen Funktionen behalten und einige Features hinzufügen.

2.2 Notwendigkeiten aus Sicht der Entwicklung

Die Entwicklungsabteilung der MAGMA GmbH stellt im Vergleich zu den vorgenannten Userwünschen andere Anforderungen. So ist hier neben der Benutzerfreundlichkeit für die Anwender vor allem eine übersichtliche Dokumentation für Entwickler sehr wichtig. Dabei sollte auf eine intuitive Struktur der Definition von Datensätzen und Eigenschaften geachtet werden. Dazu gehört auch, dass Funktionen, Style Schemata oder Graphen, die an mehreren Stellen verwendet werden, gekapselt werden können. Über die bereits existierende Schnittstelle in MAGMASOFT® sollte die generelle Einbindung kein Problem darstellen. Zudem wird ein hohes Maß an Individualisierbarkeit gefordert, da zum Beispiel Farbskalen und Icons mit dem Rest der Software übereinstimmen müssen. Ein weiterer Aspekt in dieser Richtung ist die Anpassbarkeit der Interaktivität. Es werden bestimmte Mausaktionen wie das Zoomen mit dem Mausrad oder die Selektion von einzelnen Datenpunkten gefordert, die auch weiterhin möglich sein sollten. Um die Entwicklung zu vereinfachen, sollte eine ausreichende Dokumentation mitgeliefert werden. Ein weiterer wichtiger Punkt ist die benötigte Lizenz. Hier ist es wichtig, dass die ausgewählte Software auch zur kommerziellen Nutzung freigegeben ist. Zudem ist es für Entwicklungsprojekte hilfreich, wenn der Code als Open Source Version verfügbar ist. Um gute Software zu schreiben ist es

[3] Vergleiche Abb. 4

außerdem notwendig, dass „Seperation of Concerns“ [1] möglich ist, also in diesem Fall die inhaltliche Logik von der Oberfläche zu trennen ist.

3. Bibliotheken im Vergleich[4]

3.1 Allgemeines

Um JavaScript Graphen zu implementieren, gibt es hauptsächlich zwei konkurrierende Technologien, die hierfür verwendet werden. Seit HTML5 gibt es entweder die Möglichkeit, skalierbare Vektorgraphiken, im Folgenden SVG genannt, zu erzeugen, oder mit einem HTML Canvas Element auf einer freien Fläche die Graphen zu zeichnen. Während die SVG Variante im XML-Format gespeichert ist, wodurch bestimmte HTML Elemente hierin verwendet werden können, wird beim Canvas via JavaScript auf die einzelnen Pixel zugegriffen. Aus diesen verschiedenen Zugriffsarten ergeben sich verschiedene Vor- und Nachteile. Trotz aller im Folgenden genannten Unterschiede eignen sich beide Methoden dazu, Graphen darzustellen. [2]

SVG Graphiken zeichnen sich durch ihre hohe Druckqualität aus, da sie durch die Speicherung im XML-Format unabhängig von der Auflösung sind. Durch diese Darstellung können die grundlegenden geometrischen Formen einfach als HTML Elemente innerhalb des SVG Tags eingebunden werden. Diese Eigenschaft ermöglicht Anpassungen mithilfe von JavaScript und CSS, was auch von den Graphikbibliotheken ausgenutzt wird. Hier werden die einzelnen, durch JavaScript generierten Teile der Graphen als Elemente des Domain Object Models[5] eingefügt. Da das XML-Format case-sensitive ist, muss auch im HTML Code die Groß- und Kleinschreibung beachtet werden, auch wenn das für reines HTML nicht notwendig wäre. Bei der Positionierung der Elemente ist zu berücksichtigen, dass ein SVG Element intern ein Koordinatensystem hat, dessen Ursprung in der oberen linken Ecke des Elements liegt. Zur weiteren Bearbeitung existieren zusätzlich noch Möglichkeiten Animationen, das Weichzeichnen von Kanten oder Beleuchtungseffekte zu definieren.

Das Canvas Element wird häufig auch als „programmierbares <img>-Element“ [3] bezeichnet, was bedeutet, dass dieses Feld nur durch JavaScript[6] Zugriffe modifiziert werden kann. Um das Canvas Element zu manipulieren, wird ein JavaScript Kontext bereitgestellt, welcher Methoden zum Zeichnen von geometrischen Formen, Linien und Texten enthält. Zudem kann hier auch die Bitmap gelesen und verändert werden, was den Zugriff auf einzelne Pixel ermöglicht.

Der größte Vorteil von SVG liegt in seiner Skalierbarkeit, welche im Unterschied zur Rasterbasis beim Canvas durch die Speicherung in Form von Vektoren erfolgt. Zudem kann der Stil der HTML Seite mithilfe von CSS auch ohne größeren Aufwand im SVG Element beibehalten werden. Da die SVG Elemente Bestandteile des DOM-Baums sind, lassen sich hier JavaScript-Event-Handler verknüpfen, um sie einzeln zu manipulieren und interaktiv zu gestalten. Dafür dauert es verhältnismäßig lange, die DOM Baum Elemente zu erzeugen [4], was dem Canvas Element besonders bei einer großen Anzahl von Figuren eine bessere Performance einbringt.

Die spezifischen Vor- und Nachteile der einzelnen Bibliotheken werden im Folgenden vorgestellt

[4] Zur Veranschaulichung befinden sich im Anhang Abbildungen jeweils des gleichen Datensatzes mit den verschiedenen Bibliotheken und den dafür benötigten Code.
[5] Auch genannt DOM Baum, also das Grundgerüst einer HTML Seite.
[6] Der Richtigkeit halber sei an dieser Stelle erwähnt, dass auch ein Zugriff via PHP möglich ist. Das ist für diesen Kontext allerdings irrelevant.

3.2 SVG basierte Bibliotheken

3.2.1 d3.js [5]

Diese Bibliothek gilt als Basis für viele SVG basierte Bibliotheken. Sie wird seit 2011 von Mike Bostock, Jason Davies und Jeffrey Heer entwickelt und verwendet erstmals das SVG basierte Rendering. Hier werden die Graphen als einzelne HTML5 DOM-Baum Elemente auf einem leeren SVG Element erzeugt. Der Zugriff erfolgt ähnlich wie in jQuery mithilfe von sogenanntem function-chaining. Dabei werden dem zu erzeugenden HTML Element verschiedene Attribute angehängt, um sein Aussehen zu bestimmen. Wichtig zu wissen ist, dass D3JS nicht als reine Graphenbibliothek konzipiert ist, sondern als allgemeine Bibliothek, um mithilfe von dynamischen Daten HTML Dokumente zu manipulieren. Dadurch ist diese Bibliothek hochgradig individualisierbar, aber auch wesentlich aufwändiger zu verwenden als auf das Erzeugen von Graphen spezialisierte Bibliotheken.[7] Zudem sind in Bezug auf den Wartungsaufwand ähnliche Probleme zu erwarten, wie in der aktuellen MAGMASOFT® Implementierung. Sie bietet jeden geforderten Graphentyp an und ermöglicht außerdem die wichtigsten Interaktionen, auch wenn diese explizit implementiert werden müssen und nicht defaultmäßig eingeschaltet sind. Performancetechnisch schneidet d3.js eher schlecht ab. Schon bei einem Scatter Chart mit 100.000 Punkten stößt diese Bibliothek an ihre Grenzen. Trotzdem ist es in jedem Fall sinnvoll sich weiter mit dieser Variante der DOM-Baum Manipulation auseinanderzusetzen, da auch andere Bibliotheken Schnittstellen zu d3.js anbieten, um ihren eigenen Funktionsumfang zu erweitern. Die Bibliothek ist unter der BSD Lizenz [6] veröffentlicht, also ist sie Open Source und kommerziell kostenfrei nutzbar.

3.2.2 Highcharts.js [7]

Erstmals im Jahre 2009 erschienen und damit einer der ersten ihrer Art, ist die JavaScript Bibliothek Highcharts.js. In ihrer Anfangszeit gehörte sie noch zu den Canvas basierten Bibliotheken, durch den weiteren technischen Fortschritt und die Ausarbeitung des neuen HTML5 wurde sie jedoch bereits im Jahr 2010 auf die SVG Technologie umgestellt. Optisch ist diese Bibliothek sehr ansprechend[8], da hier defaultmäßig viele Elemente, die bei der Auswertung helfen implementiert sind. Highcharts.js basiert auf der übergeordneten Bibliothek d3.js und bietet auch eine Schnittstelle zu dieser an. Zudem werden alle geforderten Graphen zur Implementierung angeboten. Bei der Verwendung dieser Bibliothek müssen allerdings Abstriche bei der Interaktion, vor allem beim Parallelkoordinaten Diagramm gemacht werden. Hier ist nur das Hovern einzelner Datenreihen möglich. Der Hauptanwendungsfall in MAGMASOFT® liegt allerdings darin, mithilfe der Parallelkoordinaten durch Eingrenzen der Achsen das am besten geeigneten Design zu finden. Diese Form von Interaktivität wird durch Highcharts.js allerdings nicht unterstützt. Zudem kommt diese Bibliothek auch bei der Performance trotz verschiedener Optimierungen für große Datenmengen schnell an ihre Grenzen. Dafür ist das Verschachteln von Graphen sehr einfach möglich. Die Verwendung von Highcharts.js erfolgt mithilfe von JSON Objekten, die durch ihre Attribute den Inhalt und das Aussehen des Graphen steuern. Zudem ist diese Bibliothek gut dokumentiert, sodass die Nutzung intuitiv erfolgen kann. Zur kommerziellen Verwendung ist es notwendig, eine Lizenz kaufen.

[7] Vgl. Tabelle 1
[8] Vgl. Tabelle 1

3.2.3 Anychart.js [8] und Chartist.js [9].

Diese beiden Bibliotheken weisen eine ansprechende Optik und leichte Bedienbarkeit mit vielen Möglichkeiten zur Individualisierung auf. Besonders das Framework Anychart.js stellt eine große Bandbreite an verschiedenen Diagrammtypen zur Auswahl, die sich über ein objektorientiertes System in JavaScript erzeugen lassen. Es bietet außerdem viele Möglichkeiten zur Interaktion über Zoom, Slider an den Achsen, mit denen der angezeigte Datenausschnitt variiert werden kann, Tooltips und Klickfunktionen auf einzelnen Elementen. Dennoch fehlt leider die unbedingt geforderte Möglichkeit, ein Parallelkoordinatendiagramm zu erstellen. Außerdem ist es nicht möglich, mehrere Graphen in ein Koordinatensystem bzw. übereinander zu zeichnen, sodass auch das Erzeugen einer „Main Effect Matrix“ nicht umsetzbar ist. Zudem muss hier zur kommerziellen Nutzung mit hohen Lizenzkosten gerechnet werden.

Chartist.js stellt wesentlich weniger Arten von Graphen zur Verfügung, ist dafür aber unter der MIT Lizenz frei zu benutzen. Das Aussehen kann hier lediglich über CSS-Strukturen angepasst werden. Die Erstellung des Graphen erfolgt mithilfe eines JSON Objektes, in welchem sämtliche Informationen zur Erzeugung bereitgestellt werden. Für jegliche Individualisierung wie Titel oder Achsenbeschriftung ist es notwendig, ein weiteres Plugin hinzuzufügen, wodurch die Verwendung der Bibliothek sehr unübersichtlich wird. Einige wenige Möglichkeiten zur Interaktion werden über die gleiche Schnittstelle angeboten. Zudem ist hier die Skalierung der Achsen nicht immer korrekt.

Da diese Bibliotheken nicht alle geforderten Graphen zur Verfügung stellen, sind sie leider nicht zur Anwendung in MAGMASOFT® geeignet, auch wenn sie an vielen Stellen als häufig verwendete Bibliotheken auftauchen.

3.2.4 Plotly.js [9]

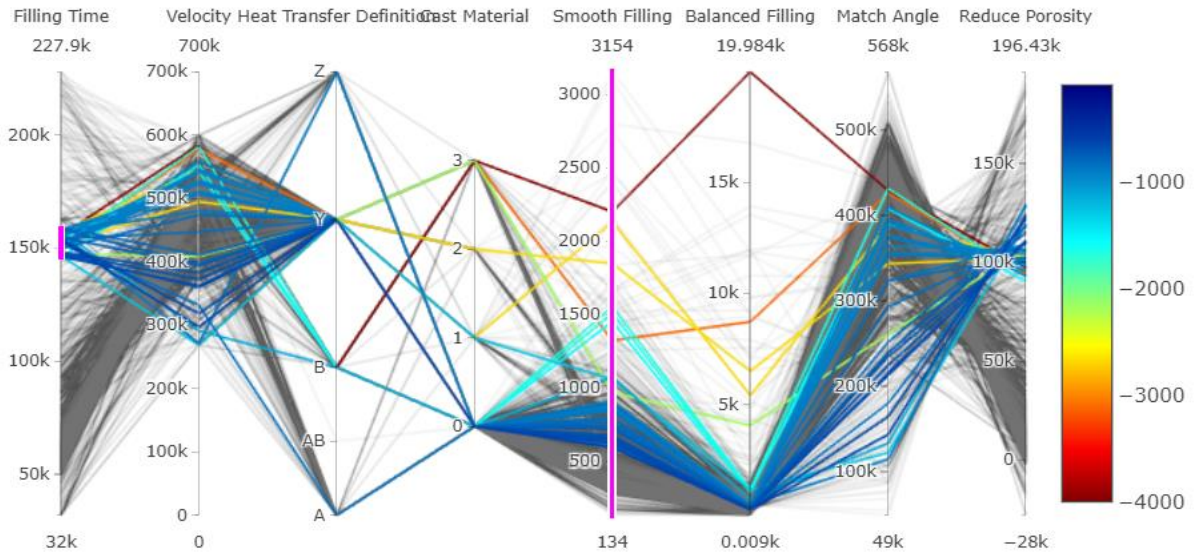


Abb. 5 Parallelkoordinatendiagramm von plotly.js

Dieses Framework schneidet im Vergleich besonders gut ab. Neben den geforderten Graphen stellt es noch viele weitere bereit, die möglicherweise für ein zukünftiges Dashboard interessant sind. Die einfache Überlagerung mehrerer Graphen ermöglicht die Darstellung der „Main Effect Matrix“ und

eine interaktive Variante des Parallelkoordinatendiagramms[9] wird auch mitgeliefert. Diese bietet sogar erweiterte Möglichkeiten zum Filtern der Daten, indem auf den einzelnen Achsen mehrere Positionen ausgewählt werden können. Dafür lässt sich allerdings der Feature Wunsch, die Skala durch die Eingabe von Werten einzustellen schwieriger umsetzen. Dennoch bietet diese Bibliothek durch zusätzliche Events weitere Möglichkeiten, die Interaktion zu erweitern. Als Default wird bereits eine Mode Bar angeboten, die verschiedene Varianten beinhaltet, Punkte zu selektieren, zu zoomen, Achsen zu verstellen und den Graphen zu bewegen. Zudem besteht die Möglichkeit Slider einzubinden, um die angezeigten Daten zu variieren. Für große Datenmengen wird das Rendern mithilfe von WebGL angeboten, um eine bessere Performance beim Laden zu gewährleisten. Hier wird ein Canvas Element im DOM-Baum verwendet, um effizientes Zeichnen zu ermöglichen, sodass hier die Vorteile sowohl von SVG als auch von Canvas basierter Programmierung berücksichtigt werden. Plotly.js verwendet intern d3.js und bietet auch eine Schnittstelle dazu an, um noch weitere Individualisierung zu ermöglichen. Definiert werden die Graphen auch hier in JSON Format, wodurch eine einfache Integration gewährleistet ist. Das Erzeugen eines Graphen erfolgt mit bis zu drei Parametern, die Informationen über Daten und Graph, Layout und weitere Konfigurationen beinhalten. Plotly.js hat leider eine etwas sparsame Dokumentation, sodass nicht alle möglichen Features erkenntlich sind. Für die wichtigsten existieren allerdings viele Beispiele und Code Snippets, sodass es möglich ist, einen Überblick über die meisten benötigten Funktionen zu erhalten. Die Bibliothek ist Open Source und unter der MIT Lizenz freigegeben, wodurch sie kostenlos kommerziell verwendet werden kann.

3.3 Canvas basierte Bibliotheken

3.3.1 canvas.js [10]

Zu den Kunden von Canvas.js gehören einige der größten Unternehmen der Software und Computerbranche, was für eine sehr gute Leistung spricht. Der größte Vorteil von canvas.js liegt in der Performance bei vielen Datenpunkten. Die Dokumentation ist sehr ausführlich und übersichtlich gegliedert. Zudem werden editierbare Beispiele zum sofortigen Ausprobieren der dokumentierten Eigenschaften bereitgestellt. Mithilfe von [11]Einstellungen zu Tooltips, Zoomen und Verschieben der Darstellung wird eine gewisse Interaktivität ermöglicht. Dennoch werden die Eigenschaften durch die vorgegeben JSON Objekte bestimmt und können daher nur in einem sehr begrenzten Rahmen angepasst werden. Durch die Spezifikation als JSON Objekte können mehrere Datenreihen und Diagrammtypen intuitiv miteinander kombiniert werden. Canvas.js bietet zusätzlich Funktionen an, um nachträglich Datenreihen oder Eigenschaften anzupassen und hinzuzufügen.

Das größte Problem dieser Bibliothek liegt in der eingeschränkten Verfügbarkeit von Graphen. So existieren hier weder Möglichkeiten um ein Parallelkoordinatendiagramm noch eine „Main Effect Matrix“ umzusetzen. Außerdem ist die kommerzielle Nutzung mit hohen Lizenzkosten verbunden.

3.3.2 chart.js [11]

Mit ca. 40.000 Sternen auf GitHub ist chart.js eine der am meisten verwendeten Open Source Bibliotheken für JavaScript Graphen. Durch die Verwendung des HTML canvas Elements ist performantes Rendern möglich. Die Implementierung von Interaktivität erfolgt mithilfe von zusätzlichen Plugins. Diese beinhalten das Zoomen und das Verschieben von einzelnen Elementen oder dem ganzen Graphen. Zusätzlich können Annotationen und Bezeichner an die einzelnen Punkte

[9] Vergleiche Abb. 5.

angefügt werden. Durch die Verwendung von JSON Objekten zur Spezifikation der Graphen, kann die Einbindung ohne größere Schwierigkeiten erfolgen. Da die Datensätze jeweils als einzelnes Objekt mit eigenem Typ definiert werden, können die Graphen auch miteinander kombiniert werden. Die Dokumentation ist übersichtlich und gut strukturiert, sodass die wichtigsten Funktionen schnell gefunden werden können. Außerdem gibt es viele Beispiele, die die konkrete Verwendung veranschaulichen. Ein weiterer Vorteil ist die Möglichkeit zur kostenfreien kommerziellen Nutzung dank der MIT Lizenz.

Das größte Problem dieser Bibliothek liegt darin, dass nur sechs Diagrammtypen angeboten werden, sodass die Anforderungen durch das fehlende Parallelkoordinatendiagramm nicht erfüllt werden können.

3.3.3 canvasXpress [12]

Die Bibliothek canvasXpress stellt eine deutlich größere Auswahl von Graphen bereit als die anderen Bibliotheken, die das canvas Element zum Zeichnen verwenden. Hier werden neben dem unbedingt geforderten Parallelkoordinatendiagramm und der „Main Effect Matrix“ noch viele weitere Graphen angeboten. Diese Vielfalt beinhaltet allerdings trotz der Verwendung des JSON Formats eine hohe Unübersichtlichkeit bei der Eingabe der Datensätze. Die Mausoperationen können überschrieben werden, was eine große Interaktivität ermöglicht. Dennoch können die einzelnen Elemente nicht separat angesteuert werden, was zum Beispiel das Filtern des Parallelkoordinatendiagramms unmöglich macht. Außerdem ist die standardmäßige Interaktivität nicht sehr intuitiv. So bewirkt das Klicken der linken Maustaste und bewegen der Maus, dass im Graph gezoomt wird anstatt, dass der gezeigte Ausschnitt sich verschiebt. Das Überlagern mehrerer Datenreihen mit verschiedenen Diagrammtypen ist auch möglich, auch wenn hier etwas Aufwand nötig ist, um die Eingabe der Daten richtig zu erkennen. In den Beispielen, die als Erklärung dienen sollen, werden zu viele Einstellungen übergeben, sodass nicht mehr ersichtlich ist, welche benötigt werden, um die gewünschten Daten anzuzeigen. Generell ist die Dokumentation eher unübersichtlich. Es werden zwar alle möglichen Eigenschaften aufgelistet, die Beschreibung, wie diese verwendet werden ist allerdings eher dürftig. Zudem beinhaltet die kommerzielle Verwendung der Bibliothek hohe Kosten. Dank der Verwendung des canvas Elements ist auch bei dieser Bibliothek die Performance beim Rendern gut.

4. Fazit

Zusammenfassend kann gesagt werden, dass alle Bibliotheken für ihren Anwendungsfall durchaus geeignet sind. Da im Anwendungsfall MAGMASOFT® besondere Anforderungen gelten, sind hierfür nicht alle Bibliotheken geeignet. So entfallen bereits alle, die kein Parallelkoordinatendiagramm zur Verfügung stellen, da dieses in der Anwenderbefragung eindeutig als meistgenutztes Feature bestimmt wurde. Auch wenn die anderen Bibliotheken andere Features enthalten, die zum Beispiel eine intuitivere Bedienung ermöglichen, überwiegt dieser Punkt. Die Frameworks, die diesem Kriterium entsprechen sind d3.js, Highcharts.js, plotly.js und canvasXpress.

D3.js weist wie oben genannt eine sehr hohe Fehleranfälligkeit mit einem hohen Wartungsaufwand auf, sodass die Verwendung dieser Bibliothek lediglich eine Verlagerung des bestehenden Problems in eine andere Programmiersprache wäre. Da die anderen beiden SVG basierten Bibliotheken, die in der engeren Auswahl sind auf d3.js aufbauen, besteht hier durchaus die Möglichkeit, dass diese auch später zur weiteren Individualisierung ergänzend verwendet wird.

Bei der Bibliothek Highcharts.js ist ein großes Problem, dass die Performance bereits bei geringen Datenmengen deutlich zu schwach ist. Außerdem müssen hier große Abstriche bei der Interaktivität gemacht werden. Dies ist ein Problem, da den Benutzern ein möglichst intuitives und angepasstes User Interface zur Auswertung ihrer Daten zur Verfügung gestellt werden soll. Dennoch bietet die die Bibliothek eine ansprechende Darstellung ohne, dass viele Einstellungen gemacht werden müssen. Die Responsivität des Designs ist gut gelöst, da Schriften und Abstände jeweils mit skalieren, sodass auch bei kleiner Fenstergröße alle Beschriftungen noch klar lesbar sind.

Das Framework canvasXpress zeigt auch bei größeren Datenmengen eine wesentlich bessere Performance, weist aber die gleichen Schwächen im Hinblick auf die Interaktivität auf. Zusätzlich ist die Bedienung der Zoom- und Scroll-Funktionen nicht immer intuitiv und klar verständlich. Außerdem werden viele Animationen vorgeschrieben, die für die alltägliche Betrachtung von Daten eher störend und unseriös wirken. Dafür stellt es eine noch breitere Auswahl an Graphen bereit, sodass hier besonders das Ziel, ein Dashboard mit neuen Funktionen zu entwickeln gut umgesetzt werden kann.

Insgesamt am besten kann plotly.js überzeugen. Diese Bibliothek bietet ein interaktives Parallelkoordinatendiagramm, welches intuitiv zu bedienen ist und alle Funktionen beinhaltet, die von den Usern gewünscht werden. Durch das Hinzufügen von Klick Events kann hier auch die oft gewünschte Eingabe der Filterwerte unterstützt werden, um nah beieinanderliegende Werte richtig auszuwerten. Durch die Möglichkeit zur Verwendung von WebGL und canvas Elementen bei großen Datenmengen oder aufwändigen Graphen ist auch die Performance in einem guten Bereich. Als Default werden hier interaktive Elemente wie das Zoomen oder Verschieben über eine Toolbar angeboten, die der Bedienung in MAGMASOFT® sehr ähnelt, sodass hier keine große Umstellung für die Benutzer nötig ist. Da diese Bibliothek dank der MIT Lizenz zur kommerziellen Nutzung freigegeben ist, kann sie ohne Bedenken eingebunden werden.

5. Beispielhafte Integration in MAGMASOFT® und Ausblick

Als Vorbereitung auf die voraussichtlich folgende Bachelorarbeit, sollte bereits untersucht werden, inwieweit die bestehenden Strukturen in MAGMASOFT® angepasst werden müssen, um die Simulationsergebnisse auf einer HTML Seite anzuzeigen. Dazu werden zunächst die aktuellen Software Strukturen zur Anzeige von Graphen erklärt, um dann exemplarisch die Integration eines Scatter Plots zu beschreiben und zum Abschluss der Ausarbeitung einen Ausblick auf die Themen der Bachelorarbeit zu geben.

Die aktuelle Struktur zeichnet sich durch eine Implementierung mithilfe einer JNI Schnittstelle aus.[10] Das User Interface wird mit dem SWT Framework[11] erstellt. Dieses verwendet zur Darstellung eine Baumstruktur, in der sogenannte Composites mit verschiedenen Typen ineinander verschachtelt werden. Zu den Typen gehören zum Beispiel einfache Container, Textfelder, Eingabefelder, Combo Boxen oder, ähnlich wie in HTML, canvas Elemente. Zusätzlich gibt es die Möglichkeit, spezielle

[10] JNI, also das Java Native Interface ist eine Schnittstelle der Java Standardbibliothek, um in einem Java Programm mit anderen Programmiersprachen oder dem Betriebssystem zu kommunizieren. In diesem Fall wird sie zum Aufrufen von C++ Code verwendet.
[11] Standard Widget Toolkit, von eclipse bereitgestelltes Framework zur GUI Programmierung.

canvas Elemente einzufügen, die mithilfe von einem JNI aus einem C++ Aufruf mit OpenGL[12] gerendert werden, die auch in MAGMASOFT® verwendet werden. Die Assessment Perspektive ist als Reiter mit verschiedenen Tabs strukturiert. Der generelle Tab ist in diesem Fall der *PlotEditor*[13]. Hierin eingebettet liegt ein Objekt der Klasse *Plot*, welches ein wie oben beschriebenes OpenGL Widget enthält. Über die JNI Schnittstelle *OptimizationChartInput* werden die zu zeichnenden an den C++ Code weitergegeben, in dem dann das OpenGL Widget befüllt wird. Jeder Graph wird in einer eigenen Klasse erzeugt und die Daten werden mehrfach zwischen Java und C++ hin und her transportiert. [14]

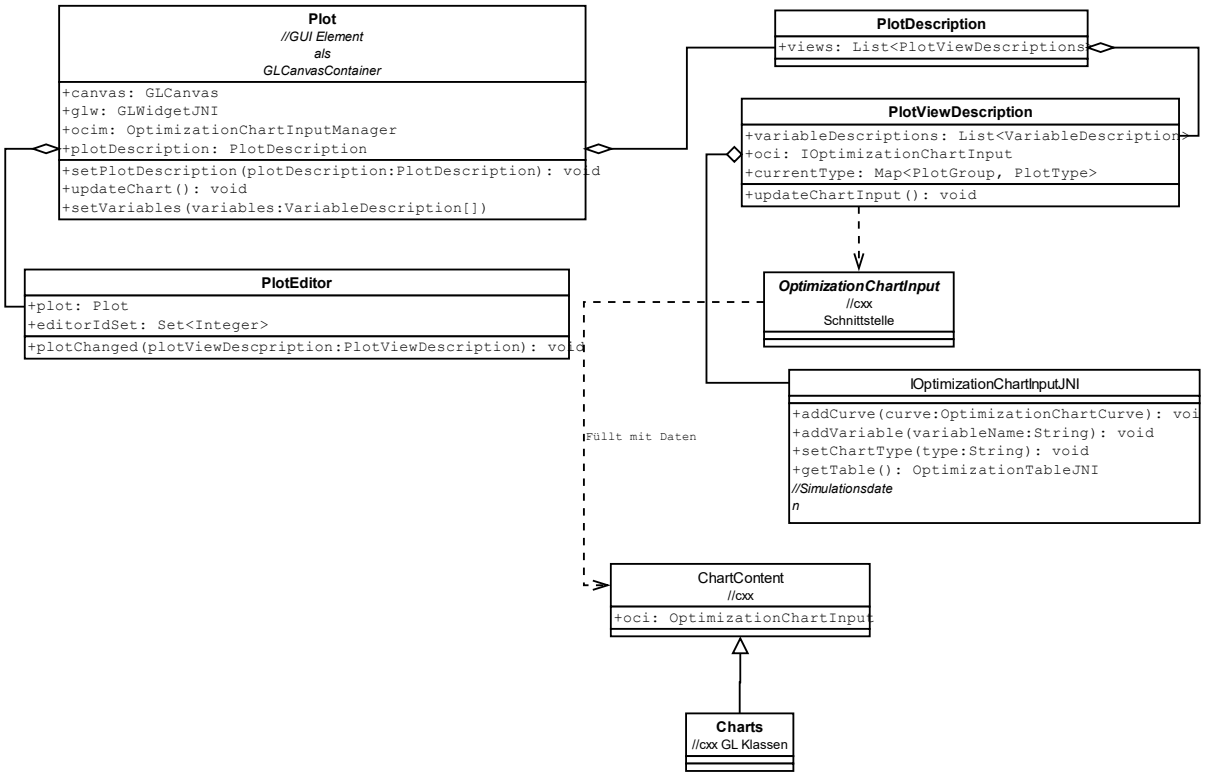


Abb. 6 Aktuelle Struktur der Assessment Perspektive

Dieser Verwaltungsaufwand kann deutlich reduziert werden. Ziel ist es zu testen, wie gut HTML und JavaScript sich in die aktuellen Strukturen einfügen lassen. Die Navigation und Einstellung der betrachteten Achsen sollen hierbei zunächst bestehen bleiben. Um eine Implementierung neben der aktuellen zu ermöglichen, habe ich zunächst einen neuen Reiter in der Assessment Perspektive erstellt. Zudem habe ich die Klasse „Plot“ abstrahiert, um das OpenGL Canvas durch ein Browser Fenster zu ersetzen. Mithilfe der Methode *createHtmlContent*(…), in welcher eine Liste mit HTML Elementen erzeugt wird, wird in der Elternklasse eine HTML Seite als *String* erzeugt, die dann über den verwendeten Jetty Server[15] im Browser Fenster angezeigt wird.[16] Um den Graphen hinzuzufügen,

[12] OpenGL steht für Open Graphics Library und ist eine standarisierte Bibliothek zur effizienten 2D und 3D Graphikprogrammierung.
[13] Die folgenden kursiv beschriebenen Ausdrücke sind die Klassen- und Methodennamen, die in MAGMASOFT® verwendet werden, wie in Abb. 6-9 gezeigt.
[14] Vergleiche Abb. 6
[15] Jetty ist ein von Eclipse unterstützter Webserver.

muss der bisherige Funktionsumfang lediglich um ein HTML Script Tag erweitert werden, damit der JavaScript Code aufgerufen werden kann. Beispielhaft ist die JavaScript Funktionalität zum Zeichnen eines Scatter Plots hier als Methode gekapselt[17].

```
@Override
public List<HtmlBlock> createHtmlContent(ServerRequestData requestData) {
    HtmlBlock div = new Div(StylePropertyName.PADDING.of("5px"));
    HtmlBlock dataDiv = new Div(StylePropertyName.PADDING.of("5px"), StylePropertyName.HEIGHT.of("750px"));
    dataDiv.addAttributes(new Attribute("id", "plot"));
    HtmlBlock script = new HtmlBlock() {
        private static final String TAG = "script";
        @Override
        protected String getTagName() {
            return TAG;
        }
    };
    String xValues = Arrays.toString(plot.getVariableValue(plotViewDescription.getVariableDescriptions().get(0)));
    String yValues = Arrays.toString(plot.getVariableValue(plotViewDescription.getVariableDescriptions().get(1)));
    String xAxisName = plotViewDescription.getVariableDescriptions().get(0).getUIName();
    String yAxisName = plotViewDescription.getVariableDescriptions().get(1).getUIName();
    script.addTextContent(getScatterPlot(xValues, yValues, xAxisName, yAxisName));
    return Arrays.asList(div, dataDiv, script);
}
```

Abb. 7 Methode zum Erzeugen des HTML Codes

```
public String getScatterPlot(String xValues, String yValues, String xAxisName, String yAxisName) {
    String data = "{x: " + xValues + ","
            + "y: " + yValues + ","
            + "mode: 'markers',"
            + "  type: 'line',"
            + "marker: {"
            + "size: 14"
            + "}"
            + "},";
    String layout = "{title: 'Assessment',"
            + "xaxis:{"
            + "title: '" + xAxisName + "',"
            + "},"
            + "yaxis:{"
            + "title: '" + yAxisName + "',"
            + "},"
            + "font: {"
            + "size: 18"
            + "}"
            + "}";
    String config = "{"
            + "responsive : true,"
            + "displaylogo: false,"
            + "}";
    return "TESTER = document.getElementById('plot');"
            + "Plotly.newPlot( TESTER, [" + data + "], " + layout + "," + config + ");";
}
```

Abb. 8 Methode zum Erzeugen des JavaScript Codes

[16] Vergleiche Abb. 7
[17] Vergleiche Abb. 8 und Abb. 9

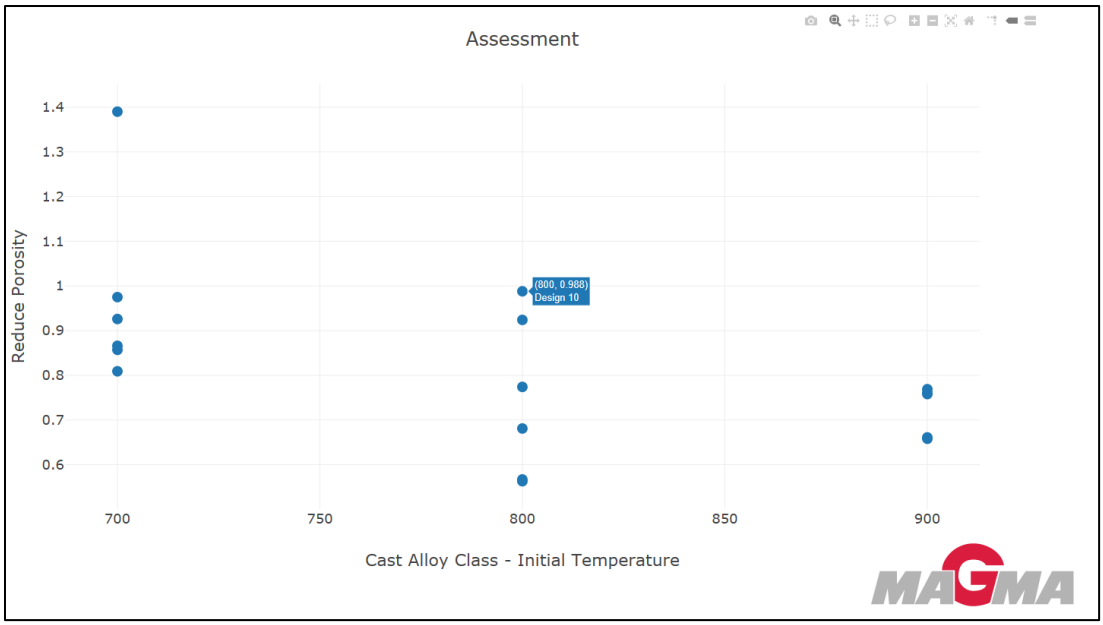
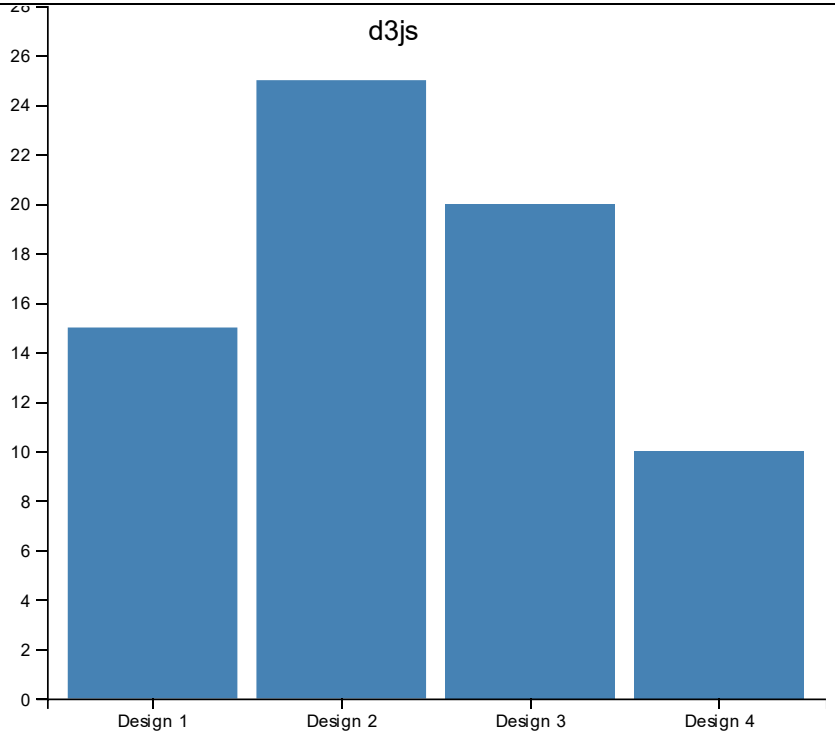
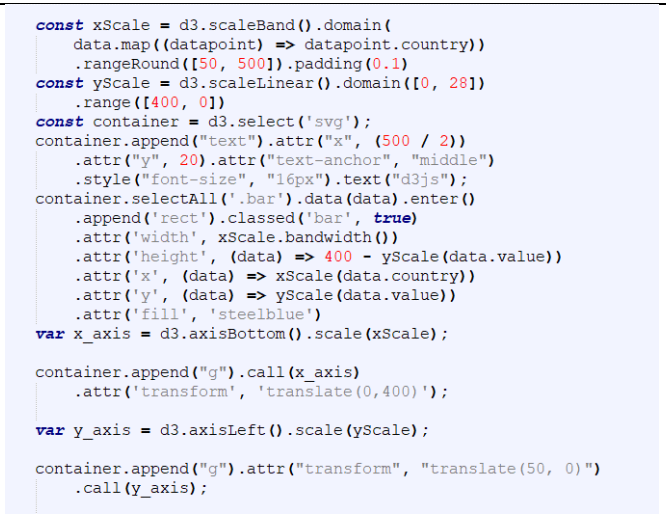


Abb. 9 Mit Plotly.js erzeugter Scatter Chart in MAGMASOFT®

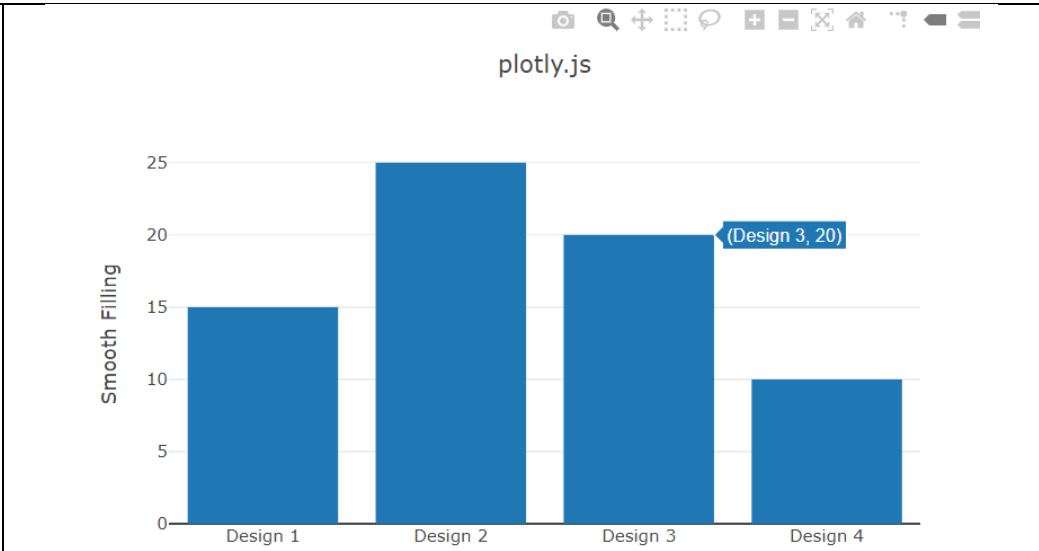
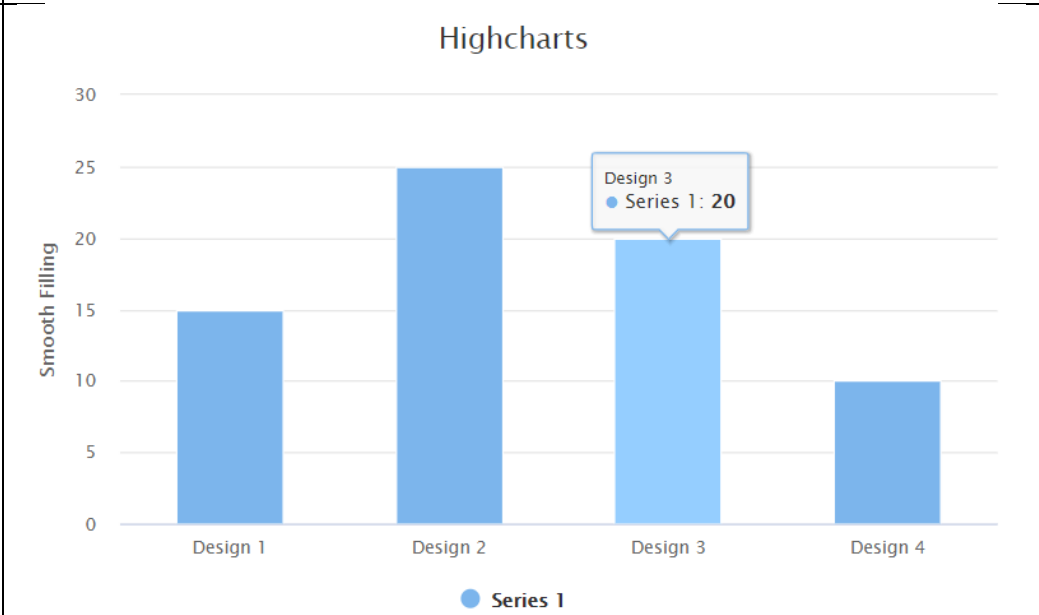
Im weiteren Verlauf und beim Hinzufügen von weiteren Graphen muss hier überlegt werden, ob die einzelnen Graphen nicht in eigene Klassen ausgelagert werden. Indem die Eigenschaften der Graphen mithilfe von Funktionen dynamisch gesetzt werden, können Individualisierbarkeit und Erweiterbarkeit erhöht werden. Weiterhin muss ich noch herausfinden, wie die Einstellungen der Graphen bis jetzt miteinander verknüpft sind. Wenn der Benutzer zum Beispiel im Scatter Chart ein Design selektiert, wird dieses auch in allen anderen Graphen und Tabellen markiert. Dies soll auch in der neuen Variante umgesetzt werden. Das Ziel ist es, nicht nur die aktuelle Implementierung vollständig zu ersetzen, sondern auch die Möglichkeiten für den User zu erweitern und zu vereinfachen. Dabei ist es wichtig, auf die Bedürfnisse der Anwender einzugehen und Lösungen für ihre alltäglichen Probleme anzubieten, anstatt viele neue Features zu entwickeln, die für den User unbrauchbar sind. Um die Benutzerfreundlichkeit zu erhöhen, muss außerdem überlegt werden, wie die Eingangsseite zur Assessment Perspektive effektiver und attraktiver gestaltet werden kann. Es besteht hierzu die Idee, eine Übersicht der Designs und Zielfunktionen als Dashboard anzuzeigen, in der nur angezeigt wird, ob die Werte sich in die gewünschte Richtung verändern. Zudem soll der Nutzer selbst bestimmen, welche Graphen er zur Auswertung benötigt. Diese sollen auf dem Dashboard mit gespeicherten Einstellungen vorgeschlagen werden. Außerdem werden noch Überlegungen angestellt, wie die Auswertung und Benutzerführung noch intuitiver gestaltet werden kann.

Diese Aspekte weiter zu recherchieren und umzusetzen soll Thema einer zukünftigen Bachelorarbeit werden.

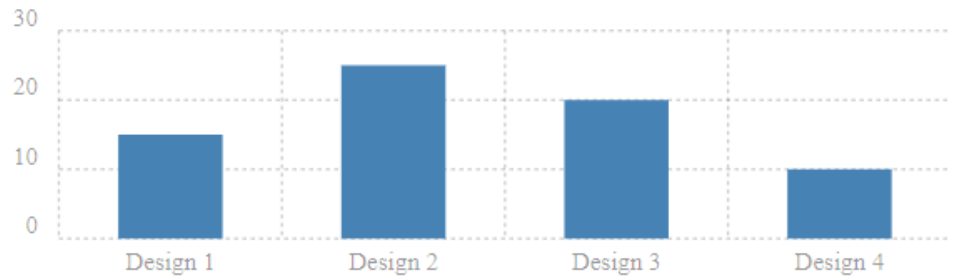
6. Anhang

	Abbildung	Zugehöriger Code
d3.js		


```
const xScale = d3.scaleBand().domain(
    data.map((datapoint) => datapoint.country))
    .rangeRound([50, 500]).padding(0.1)
const yScale = d3.scaleLinear().domain([0, 28])
    .range([400, 0])
const container = d3.select('svg');
container.append("text").attr("x", (500 / 2))
    .attr("y", 20).attr("text-anchor", "middle")
    .style("font-size", "16px").text("d3js");
container.selectAll('.bar').data(data).enter()
    .append('rect').classed('bar', true)
    .attr('width', xScale.bandwidth())
    .attr('height', (data) => 400 - yScale(data.value))
    .attr('x', (data) => xScale(data.country))
    .attr('y', (data) => yScale(data.value))
    .attr('fill', 'steelblue')
var x_axis = d3.axisBottom().scale(xScale);

container.append("g").call(x_axis)
    .attr('transform', 'translate(0,400)');

var y_axis = d3.axisLeft().scale(yScale);

container.append("g").attr("transform", "translate(50, 0)")
    .call(y_axis);
```

plotly.js		``` Plotly.newPlot('plotly', [{ x: data.map(i => i.country), y: data.map(i => i.value), type: 'bar' }], { title: "plotly.js", yaxis: { title: { text: 'Smooth Filling' } }, }, {responsive: true}); ```
highcharts.js		``` Highcharts.chart('Highcharts', { title: {text: 'Highcharts'}, xAxis: {categories: data.map(i => i.country)}, yAxis: { title: { text: 'Smooth Filling' } }, series: [{data: data.map(i => i.value), type: 'column'}] }) ```

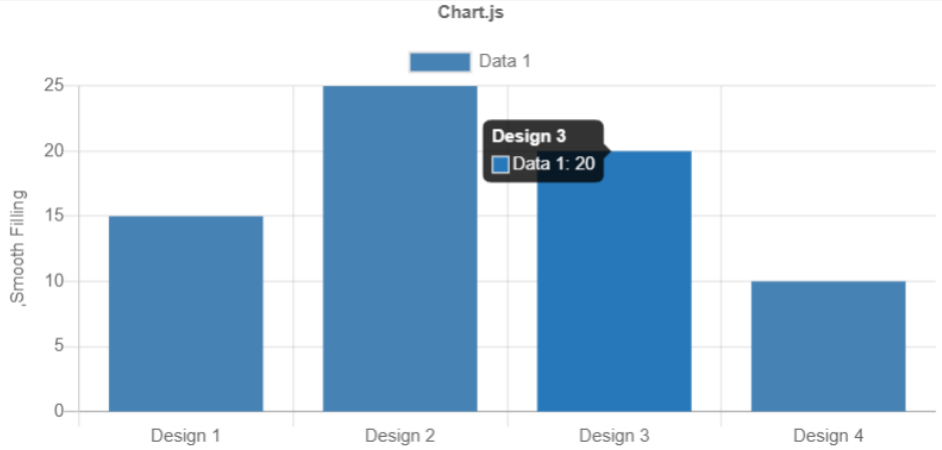
chartist.js



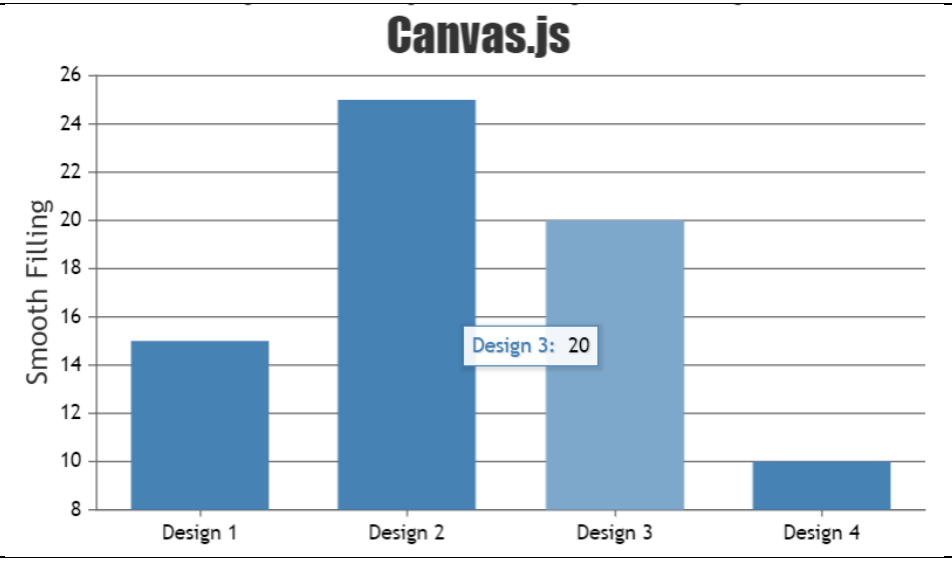

```
new Chartist.Bar('#chartist',
    {
        labels: data.map(i => i.country),
        series: [data.map(i => i.value)]},
    {});
```

```
<style>
    .ct-series-a .ct-bar {
        stroke: steelblue;
        stroke-width: 50px;
    }
</style>
```

chart.js




```
var ctx = document.getElementById('chartjs').getContext('2d');
var chart = new Chart(ctx, {
    type: 'bar',
    data: {
        labels: data.map(i => i.country),
        datasets: [{
            label: 'Data 1',
            backgroundColor: 'steelblue',
            data: data.map(i => i.value)
        }]
    },
    options: {
        title: {
            display: true,
            text: "Chart.js"
        },
        scales: {
            yAxes: [{
                ticks: {
                    beginAtZero: true
                }, scaleLabel: {
                    display: true,
                    labelString: ["", "Smooth Filling"]
                }
            }]
        },

    }
});
```

canvas.js		``` var chart = new CanvasJS.Chart("canvasjs", { title: {text: "Canvas.js"}, axisY: {title: "Smooth Filling"}, data: [{ type: "column", showInLegend: true, dataPoints: [{y: data[0].value, label: data[0].country}, {y: data[1].value, label: data[1].country}, {y: data[2].value, label: data[2].country}, {y: data[3].value, label: data[3].country},], color: 'steelblue' }] }); chart.render(); ```

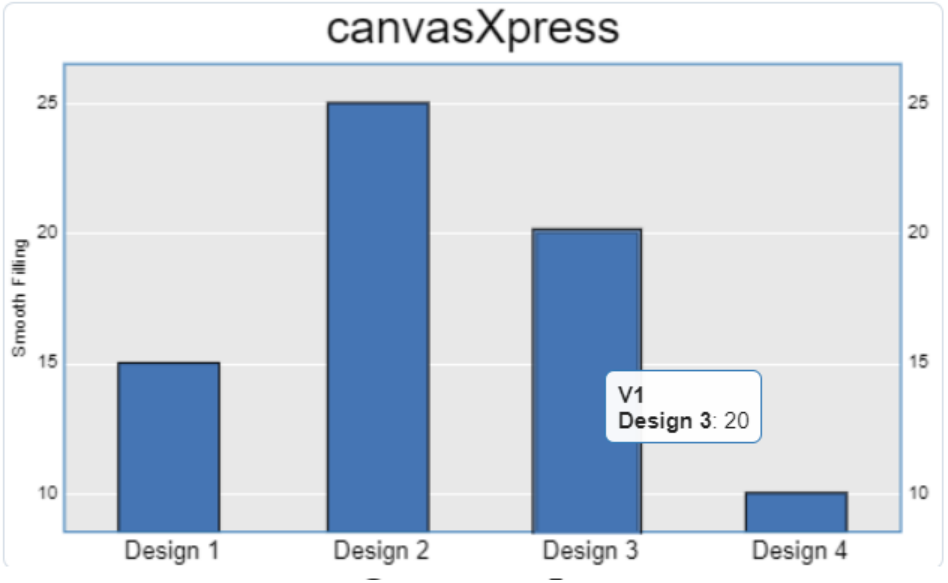
canvasXpress		``` new CanvasXpress("canvasXpress", { "y": { "vars": ["V1"], "smps": data.map(i => i.country), "data": [data.map(i => i.value)] } }, { "graphOrientation": "vertical", "loessDegree": 1, "plotBox": true, "plotBoxColor": "#337AB7", "showContourLevel": true, "showLegend": false, "showScatterPlotMatrixLabels": true, "sizeByContinuous": true, "smpLabelRotate": 90, "theme": "CanvasXpress", "title": "canvasXpress", "toolbarType": "under", "transitionStep": 30, "xAxis": ["V1"], "xAxisTitle": "Smooth Filling", "varsTitle": "Smooth Filling" }, false); new Chartist.Bar('#chartist', { labels: data.map(i => i.country), series: [data.map(i => i.value)]}, {}); ```

Tabelle 1 Vergleich von einfachen Graphen und deren Code Menge

7. Literaturverzeichnis

[1] E. W. Dijkstra, „On the role of scientific thought," in *Selected Writings on Computing: A personal Perspective*, New York, Springer, 1982, pp. 60-66.

[2] F. Nelli, Beginning JavaScript Charts, Apress, 2013.

[3] J. Wolf, HTML5 und CSS3 Das umfassende Handbuch, Bonn: Rheinwerk Computing, 2018.

[4] G. John, „tutorialspoint," 20 03 2018. [Online]. Available: https://www.tutorialspoint.com/What-is-the-difference-between-SVG-and-HTML5-Canvas.

[5] „d3.js," [Online]. Available: /web/20201215143737/https://github.com/d3/d3/wiki. [Zugriff am 15 12 2020].

[6] M. Bostock, „github d3.js," [Online]. Available: http://web.archive.org/web/20201207104101/https://github.com/d3/d3/blob/master/LICENSE. [Zugriff am 2020 12 7].

[7] „Highcharts," [Online]. Available: http://web.archive.org/web/20201215151156/https://www.highcharts.com/demo. [Zugriff am 15 12 2020].

[8] „AnyChart.js Overview," [Online]. Available: http://web.archive.org/web/20201207105217/https://www.anychart.com/products/anychart/overview/.

[9] „plotly JavaScript," [Online]. Available: http://web.archive.org/web/20201215144019/https://plotly.com/javascript/. [Zugriff am 15 12 2020].

[10] „canvas.js," [Online]. Available: /web/20201215151839/https://canvasjs.com/javascript-charts/. [Zugriff am 15 12 2020].

[11] „chart.js," [Online]. Available: /web/20201215152134/https://www.chartjs.org/samples/latest/. [Zugriff am 15 12 2020].

[12] „canvasXpress," [Online]. Available: /web/20201215152548/https://canvasxpress.org/docs/overview.html. [Zugriff am 15 12 2020].

[13] S. Machlis, „Beauty and brains: Plotly combines dataviz and serious statistical analysis," 02 12 2013. [Online]. Available: https://web.archive.org/web/20131202232439/http://blogs.computerworld.com/business-intelligenceanalytics/23078/beauty-and-brains-plotly-combines-dataviz-and-serious-statistical-analysis-cloud.

Eidesstattliche Erklärung

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

"Evaluation von JavaScript Bibliotheken zur Darstellung

statistischer Ergebnisdaten von Gießsimulationen"

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Christine Ophoven

Aachen, den 15.12.2020

C. Ophoven

Unterschrift der Studentin / des Studenten