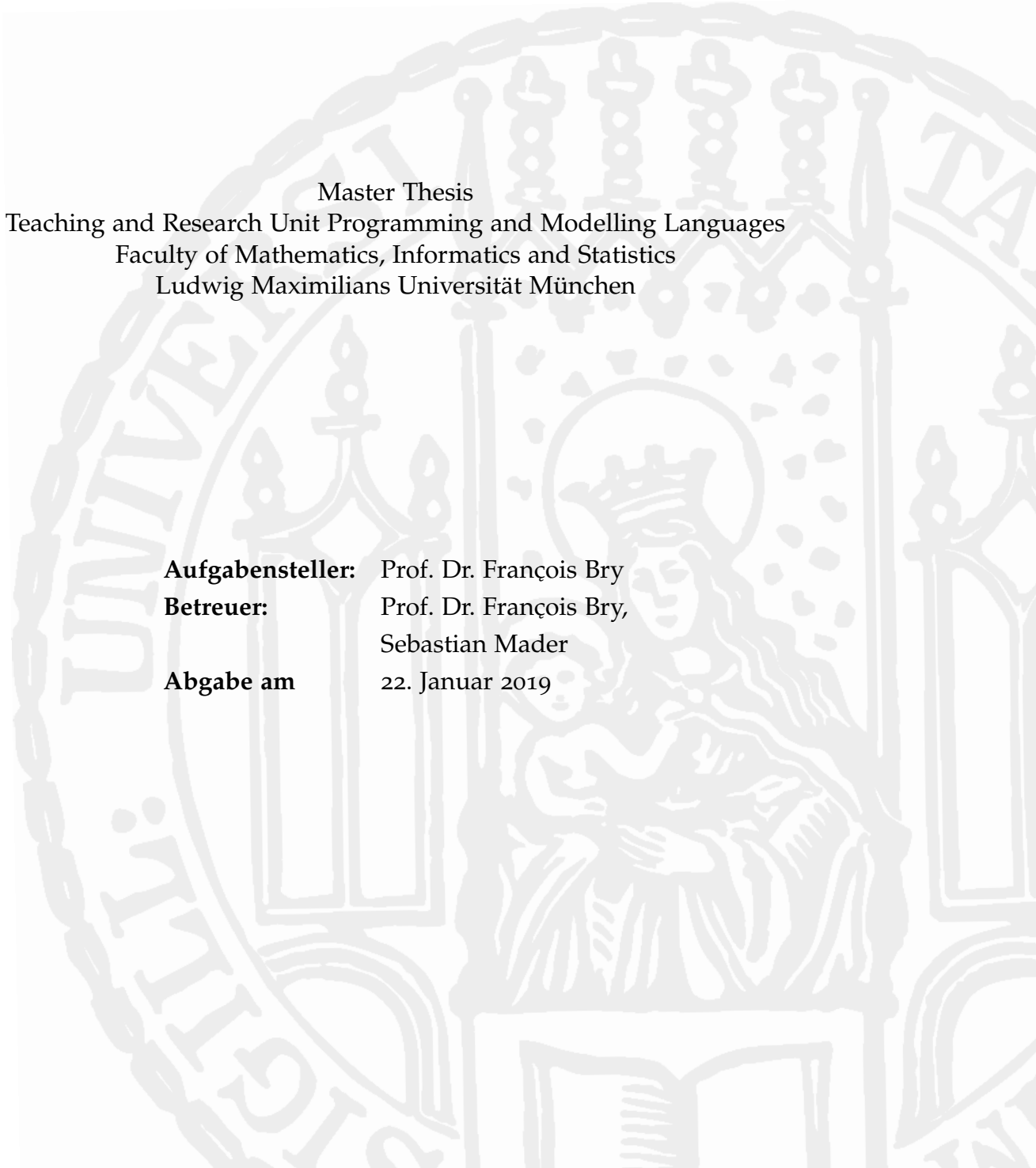
A BROWSER-BASED DEVELOPMENT ENVIRONMENT FOR JAVASCRIPT LEARNING AND TEACHING

MAXIMILIAN MEYER



Master Thesis
Teaching and Research Unit Programming and Modelling Languages
Faculty of Mathematics, Informatics and Statistics
Ludwig Maximilians Universität München

Aufgabensteller:	Prof. Dr. François Bry
Betreuer:	Prof. Dr. François Bry, Sebastian Mader
Abgabe am	22. Januar 2019

DECLARATION / ERKLÄRUNG

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstandig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, Januar 2019

Maximilian Meyer

Ohana means family.
Family means nobody gets left behind, or forgotten.

— Lilo & Stitch, film produced by Walt Disney Feature Animation, 2002

Dedicated to Amma and Antonia

2019

ABSTRACT

Web development is becoming increasingly important in a world, that is more and more interconnected. JavaScript and its frameworks empower developers to complete nearly every task and create applications that rival JavaScript's native counterparts, making the language ubiquitous. In tertiary STEM education, it is becoming progressively more difficult to manage growing numbers of students and to provide enough practical training both in and outside of lectures. A solution to this problem is to implement an automated process in order to facilitate revision processes for teachers and grant instant On-screen feedback to students. This project firstly identifies the pedagogical and technological needs of potential users for a learning tool and puts them into perspective within existing scientific advances. Secondly, it analyzes existing online tooling. After evaluation of the current environment, an in-browser editor is proposed with the aim of teaching web development. Using HTML, CSS and JavaScript, users can learn online, work with code promptly without the need to setup an extensive programming environment and receive automated test results. In conjunction with this thesis, a concept for such an editor has been designed and implemented. Afterwards, a usability study of the editor was conducted, and its results evaluated. The results show, that tools such as the one proposed in this project help to mitigate some of the problems that arise when starting with coding.

ZUSAMMENFASSUNG

Web Entwicklung wird immer wichtiger in einer Welt, die zunehmend vernetzt ist. JavaScript und dessen Programmiergerüste befähigen Entwickler dazu, nahezu jede erdenkliche Aufgabe zu erledigen und Applikationen zu erzeugen, die ihren nativen Gegenstücken gleichkommen. Das macht Javascript allgegenwärtig. In tertiärer STEM Ausbildung wird es zunehmend schwierig, wachsende Studentenzahlen zu verwalten und für diese Studenten adäquate praktische Übungen in- und außerhalb von Vorlesungen bereitzustellen. Eine Lösung für dieses Problem ist es, einen Automatisierungsprozess einzuführen, um Korrekturprozesse zu vereinfachen und Nutzern sofortige Rückmeldung zu gewährleisten. Als erstes identifiziert dieses Projekt pädagogische und technologische Bedürfnisse potenzieller Nutzer eines Lernwerkzeuges und setzt diese in Zusammenhang mit aktuellen wissenschaftlichen Erkenntnissen. Dann werden bereits vorhandene Onlinewerkzeuge analysiert. Nach Auswertung der aktuellen Umgebung wird ein Konzept für einen im Browser laufenden Code Editor vorgestellt. Dieses Werkzeug hat das Ziel, für die Lehre besonders geeignet zu sein. Durch die Anwendung von HTML, CSS und JavaScript können Nutzer online lernen und schnell anfangen zu coden, ohne eine Entwicklungsumgebung aufsetzen zu müssen. Automatiserte Testresultate geben schnelles Feedback zum geschriebenen Code und bieten Hilfestellung für Anfänger. Verknüpft mit dieser wissenschaftlichen Arbeit wurde ein Konzept für solch einen Editor gestaltet und implementiert. Danach wurde eine Nutzbarkeits-Studie anhand dieses Editors durchgeführt und deren Resultate ausgewertet. Befunde zeigen, dass Werkzeuge wie das in dieser Ausarbeitung Vorgeschlagene dabei helfen, Probleme zu lösen, die auftreten wenn man neu mit der Programmierung anfängt.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [47]

ACKNOWLEDGEMENTS

The biggest eternal thanks go to you, Amma Frimpong, my loving fiancé, for putting up with my stressed out self, providing food and shelter when I most needed it.

To my life-coach, my mother Carmen: because I owe it all to you. And to Ulrich & Thorsten, my loving fathers - always there to support me. Many Thanks!

Also Sebastian Mader, my energizing supervisor and Francois Bry, my professor. Thank you for the continuous stream of wisdom and encouragement. I could not have done it without you.

I am grateful to my siblings, who have provided me with moral and emotional support in my life. I am also grateful to my other family members and friends who have supported me along the way.

With a special mention to Johann Arendt, Johann Rottenfußer, Michael Thanei, Marcel Sebald, Max Karadeniz, Leon Busse, Patrick Nagel, Robert Frimpong, Carl Luis Pöhl and Max Hünemörder. Thank you guys!

It was fantastic to have had the opportunity to do the majority of my research in the Bayerischer Rundfunk facilities. What a cracking place to work!

Thanks for all your encouragement!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

STEM Science, Technology, Engineering, Mathematics

ZPD Zone of Proximal Development

HTML Hyper Text Markup Language

CSS Cascading Style Sheets

JS JavaScript

PHP Hypertext Preprocessor

DOM Document Object Model

HAML HTML Abstraction Markup Language

IDE Integrated Development Environment

UI User Interface

SPA Single Page Application

JSX Syntax Extension to JavaScript

RTA Retrospective Think Aloud

CTA Concurrent Think Aloud

CP Concurrent Probing

RP Retrospective Probing

API Application Programming Interface

I/O Input/Output

ECMA European Computer Manufacturer's Association

1 INTRODUCTION

1.1 MOTIVATION

Nowadays, the internet is one of the most used mediums. Therefore it is not only important to understand how it works, but it is also crucial for people to be able to work with and develop for it. University courses in Web Development can be a great help for students to improve their development skills by learning about the different building blocks of the web and how they interact. Setting up a development environment often proves to be difficult. Making sure everything runs smoothly for each and everyone of the students is very time consuming due to different operating systems, the user's personal preference and individual skills. Having students work with code as quickly as possible and therewith bringing web development closer to them is the main motivation behind this thesis. This can be enhanced by making the learning environment browser-based. By being web-based, new forms of learning and teaching are enabled and starting out with programming is easy.

Universities are lacking in educated staff that would be sufficient to deal with the influx of new enrollments [4]. Teaching in higher education often benefits from alternative teaching and learning formats, such as flipped classrooms [81], learning circles and others that can be easily implemented for classes with small sizes [92]. In order to effectively teach web development, students have to start coding as quickly as possible. Teachers should be able to set coding tasks without delay, have students work on them, and see results in real time during the course. Additionally, users need tools which let them learn about web development and enable them to get as close to the experience as possible. Future students should have an easy time to start coding. Building upon a pedagogical foundation, a system is created which helps students navigate the development landscape and facilitates their first steps in coding.

1.2 CHALLENGES WHEN STARTING OUT WITH CODING

Learning how to write clean code and understanding the languages is a great challenge. Not only is it difficult to grasp all the theoretical concepts, but it is also difficult to find the extensive amount of time needed to enter the field. Finding adequate studying material can be arduous. Learning about the different concepts of coding, for example

Callbacks[1] or the Stack [20], takes immense amounts of time due to the complexity of underlying theory. Learning to piece these concepts together into working programs takes even more time. There is an amount of frustration involved in learning to program. Programs may fail for no apparent reason or the results may be unexpected - many programmers who are starting share this experience.

JavaScript is a fragmented, rapidly growing language, which makes it hard to gain an overview. Overcoming frustration takes much effort. Taking into account the above mentioned challenges, it can be concluded that students and anyone learning to code would benefit from a tool that guides them through the process in an efficient manner and mitigates the frustrating aspects of learning how to code.

1.3 GOALS OF THIS PROJECT

The goal of this project is to make it easier to start coding by solving some of the problems that arise when setting out. When starting, users have to worry about setting up their system for coding. They usually have to install an development environment or an editor, a compiler, manage version control and more. These challenges can be solved by having a fully functional system within the browser. Another goal of this project is to design a user interface concept for a system, which is intuitive and which features high usability. Traditional sit-and-listen lectures can be less effective than other formats [68]. Therefore, it is important to use other formats once in a while, like flipped classrooms [68] or learning circles [17]. In this project, a tool is developed, which can be used in flipped classrooms [68] and supports teachers in their work. Educators can be relieved of part of their burdens by introducing some automation and feedback through automated testing.

1.4 WHAT HAS BEEN DONE IN THE THESIS?

A web-based editor for the programming language JavaScript has been conceptualized and implemented. This editor focuses on web development. By letting users see the results of their coding instantly and by providing an educational scaffolding [10], some of the challenges of JavaScript and starting with coding in general are resolved. A usability study using Think Aloud, first introduced by Lewis et al. [48], has been performed to evaluate the prototype. The study has shown, that the prototype is usable and similar systems in general could be desirable.

1 https://stackoverflow.com/questions/824234/what-is-a-callback-function/7549753#7549753

1.5 OVERVIEW

Chapter 2 of the thesis focuses on fundamentals. Firstly, it encompasses other works related to the project, and secondly the analysis of some of the most well known online web development tools. This way, the most important features are understood and extracted for use within the prototype. In Chapter 3, "Concept", requirements and design goals are established, the technological, UI and testing concepts are presented in detail, and the advantages and complications of such a system are examined. Chapter 4, "Implementation", offers an in-depth explanation of the technological decision making process, the various mechanics that were put into place and the different features presented within the prototype. Chapter 5, "Study and Results", reports on details about the design of a Think Aloud usability study meant to analyze the prototype. Afterwards, preparation, realisation and results of the study are explained and evaluated.

2 FUNDAMENTALS

This chapter addresses the educational aspects and corresponding technological support proposed, while also giving an overview of Web technologies. Additionally it presents and compares existing JavaScript editors in order to establish the status quo.

2.1 EDUCATIONAL SCIENCE

Regarding educational methods, the following section addresses flipped classrooms [81], educational scaffolding [63], and fading [34]. A flipped classroom reverses traditional learning environments by delivering educational content, often online, outside of the classroom. It is an instructional strategy which moves activities, including those that may have traditionally been considered to be performed at home, like for example solving math exercises, into classrooms. In a flipped classroom, students watch online media, discuss in online sessions, or carry out research at home while engaging in the classroom with guidance of a teacher [34].

In the old-school educational concepts, teachers would be the center of attention and primary source of information to students. They would pose questions and the teacher would directly answer. A lecture-style type of teaching would be used in this traditional method [71]. Students would only be able to either work in small groups or independently on tasks designed by the teacher. The flipped classroom intentionally shifts focus away from the teacher and onto the exploration of topics in a learner focused manner. This way, a greater depth of understanding and more meaningful experiences can be facilitated. Educational media such as videos and quizzes can be used at home to deliver content that would otherwise be explained by the teacher [1, 70, 81].

The usage of a flipped classroom changes activities that can be conducted inside the class, because more time is available for other activities besides lecture. Such activities can include: in-depth laboratory experiments, using emerging mathematical technologies and mathematical manipulation, original document analysis, debate or speech presentation, discussion of current events, peer reviewing, project-based learning and skill development or concept practice [12]. The use of these techniques allows for more higher order tasks to be done by the students such as collaborative work, creative exercises - such as designing tasks for other students, problem finding, and problem solving [5]. In this way, interactions of educators with students in a

flipped classroom can be more personalized and didactic. Students are actively involved in knowledge finding and construction as they participate in and evaluate their own learning [5].

Active learning strives to more involve the student into the teaching session. Students participate by doing something other than passively listening. In order to learn effectively, students must do more than just listen: They must read, write, discuss, or be engaged in solving problems [15]. Students should specifically engage in higher order thinking tasks such as analysis and evaluation. Active learning is the opposite of passive learning as it attempts to put the learner at the center, instead of the teacher. Active learning aims to convert students from passive listeners to active participants and helps students understand the subject through asking questions, gathering data and analyzing data in order to solve higher order cognitive problems [7]. Using class time for active learning, rather than just lectures, provides opportunities for greater teacher-to-student mentoring, peer-to-peer collaboration and cross-disciplinary engagement [68]. As can be seen in figure 1, retention of knowledge is much higher when using active learning strategies, for example, discussions or collaborative learning groups [64]. The pyramid from Figure 1 shows different tasks and how they correlate to memory performance. The illustration in Figure 1 was adapted from a similar one by Igor Kokcharov[1].Some tasks are stronger in this regard than others. Traditional lectures rank at the bottom of the pyramid exhibiting a retention of under 20%, while active learning tasks, such as practice with a coach rank above 75% [15, 78].

Another important concept to look at in educational sciences is **instructional scaffolding**. It refers to the help given to students during their learning process. This guidance is crafted to the needs of students with the intention of supporting them, so that they may achieve their learning goals [74]. Instructional scaffolding is meant to support students when they are first introduced to the material. Teachers help the students conquer a task or concept by providing this assistance. These supports may include compelling tasks, resources, templates, helping materials, and guidance on the development of social and cognitive skills. Instructional scaffolding can be employed through modeling a task, giving advice, or qualified personnel providing coaching. As soon as students start to develop their own autonomous learning strategies and skills, these supports can be gradually removed.

There are a number of features to consider when employing instructional scaffolding. There should be strong interaction between learners and teachers. Only their close relationship can make sure an exchange of knowledge takes place [10]. Learning should take place in the students Zone of Proximal Development (ZPD). The ZPD is

1 https://commons.wikimedia.org/wiki/File:Learning_Retention_Pyramid.JPG

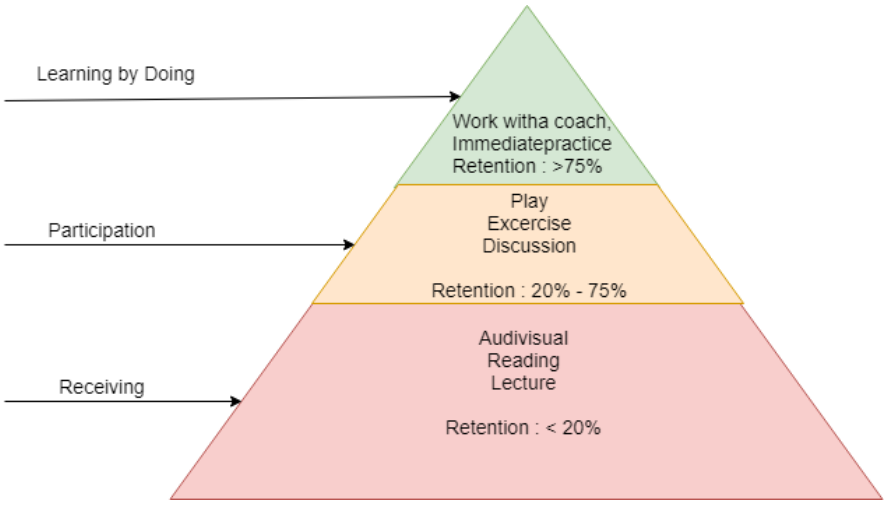


Figure 1: This figure shows a learning retention pyramid, according to Bloom's taxonomy [15]. Bloom's taxonomy is a set of three hierarchical models used to classify educational learning objectives into levels of specificity and complexity [15]. The pyramid shows, that active learning tasks, such as working with a coach, are more effective compared to traditional lectures when it comes to memory retention.

the difference between what a learner can do without help, and what they can't do without help. It is believed, that the role of education is to give students experiences that are within their ZPD, thereby encouraging and advancing their individual learning, such as their skills and strategies [13]. If the task is too easy, learners will be unmotivated, since they already know the solution. If it is too hard, they will not understand what to do. When using instructional scaffolding, teachers need to be aware of the learners current level of knowledge. The final feature of instructional scaffolding is the scaffold, which is often compared to scaffolding used in building construction. It is meant to only be temporary, until the "building" can support itself [89]. The support and guidance provided to learners facilitates internalization of the knowledge they need to complete the task. This support is gradually taken away until the learner is independent [61]. This taking away of the scaffolding is referred to by many as "fading" [63].

2.2 WHY IS THE PROTOTYPE USING JAVASCRIPT?

The question remains why the editor is required to be JavaScript and why JavaScript is good for students. In today's world, JavaScript is worthwhile to learn, since the trend of usage worldwide has increased continuously [41]. JavaScript can be compiled client-side, directly in the browser. As a multi-paradigm language, JavaScript supports functional, imperative and event-driven (including object-oriented and

prototype-based) programming styles. It has a capable API for working with arrays, booleans, strings, dates, regular expressions, and manipulation of the Document Object Model[2], but JavaScript itself does not include any I/O, such as storage, networking, external devices or graphics facilities, often relying for these upon the hosting system in which it is embedded [21]. It does however provide APIs like fetch or the canvas, that provide many of these features. JavaScript can be instantly compiled and run within browsers. No compilation on an external server is required for the code to run. JavaScript is the required programming language for this project.

2.3 WHY IS JAVASCRIPT COMPLICATED?

It remains to be seen why JavaScript is a difficult language to pick up compared to other languages. First of all, it is an asynchronous[3] language. Normally, a given program's code runs with only one thing happening at once. If a function relies on the result of another function, it has to wait for the other function to finish and return. This is not the case with JavaScript. Its programs can run asynchronously. This is useful in the context of web browsers. When a web app runs in a browser and it executes code without returning control to the browser, the browser can appear to stand still. This is called blocking. When the call stack[4] is blocked, the browser prevents user's interrupts and other code statements from executing until the blocking statement is executed and the call stack is freed. Asynchronous code does not block execution.

There are many pitfalls that inexperienced users can run into when trying to learn JavaScript in regards to Asynchronicity. JavaScript is single-threaded and powered by an event queue. It has a nature of waiting for an event to trigger the right code at the right time. This can be hard to understand for beginners. As a dynamically typed language, JavaScript contradicts many of the rules established in traditional programming. Dynamic typing implies more succinct code and the absence of a compilation step allows it to be more tolerant to changes [62]. JavaScript is one of the most used programming languages in the world [60], which is why it is ever-changing and developing new features rapidly. Keeping up with this continuous evolution can be daunting.

1. Speed. Being client-side, JavaScript is very fast because any code functions can be run immediately instead of having to contact the server and wait for an answer [21].

2 https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
3 https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous
4 https://developer.mozilla.org/en-US/docs/Glossary/Call_stack

2. Simplicity. JavaScript is relatively simple to learn compared to for example C++. It can be used to write imperative code as most computer science university students learn in their first semesters as part of their coursework. As such it is easy for them to instantly get started with writing code.

3. Versatility. JavaScript plays nicely with other languages and can be used in a great variety of applications. Unlike PHP[5] or SSI scripts [18], JavaScript can be inserted into any web page regardless of the file extension. JavaScript can also be used inside scripts written in other languages such as Perl and PHP. It is also a Full-Stack language, that can be used in the Front-end, in the Back-end, or for application programming [21].

4. Less Server Load. Being client-side reduces the demand on the website server and allows websites to be rendered without waiting for a server to send a lengthy response. In most cases, this is very quick, with average loading times between two and ten seconds [76], depending on the site. Lazy loading[6] and server side rendering can be used to decrease loading times even further. It is known, that most users leave the page if it does not load after approximately four seconds, which is why this feature is very important in web development [49].

5. Functions[7]. They are treated as first class objects within JavaScript code. This means, that they can be handed to other functions or saved to variables just like a string, integer or boolean. This forward thinking feature made JavaScript popular in the early days of its existence and helps its popularity to this day.

6. JavaScript is web native. In 2019, with a few very specialized exceptions, JavaScript comes installed on every modern web browser[8]. Starting with JavaScript programming can therefore happen very quickly. If one is for instance using Google Chrome[9], one can just go to the "View" menu, click on the "Developer" sub-menu, and see an option to open a JavaScript console. The console then allows for JavaScript programming.

2.4 WEB DEVELOPMENT - THE JAVASCRIPT STACK AND ITS FEATURES

JAVASCRIPT This section is dedicated to web development in general. Before the world wide web was conceived and implemented,

5 http://www.php.net/
6 https://developer.mozilla.org/en-US/docs/Web/Apps/Progressive/Loading
7 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions
8 https://en.wikipedia.org/wiki/Comparison_of_web_browsers#JavaScript_support
9 https://www.google.com/intl/de_ALL/chrome/

it was not possible to quickly and easily share digital information between many people. Alongside HTML and CSS, JavaScript is one of the three core technologies that enable the World Wide Web [27]. JavaScript code can be embedded within HTML code to control the behaviour and content on websites. Content could for example be loaded onto the page using JavaScript. Interactivity usually also stems from JavaScript. If the user would click a menu or button, the handling of this event would fall to JavaScript. JavaScript drives interactive web pages and thus is an essential part of web applications. The great majority of websites use JavaScript, and all major web browsers have a dedicated JavaScript engine to execute it and allow users to potentially start coding right away. JavaScript sees universal support for all modern web browsers with built-in interpreters. For example, it supports if statements, do while loops or switch statements. Inheritance is implemented using prototypes in the JavaScript programming language. Many class-based features can be simulated successfully using prototypes. Functions within JavaScript double as object constructors, along with their typical role. The most abundant use of JavaScript is to add client-side behaviour to HTML pages. Scripts are embedded or loaded on demand into HTML pages and interact with the DOM of the page. Since JavaScript code can run locally in the user's browser, the responsiveness of every application, if properly implemented, is in comparison increased dramatically. There is little to no waiting time involved [86], as would usually be the case with a server-client architecture. Whenever a query has to be sent to the server though, waiting times do occur. Techniques such as Persistent Queries[10], Lazy Loading[11] and others can be used to mitigate this issue. JavaScript can also detect some of the users actions, that HTML alone could not, such as individual keystrokes which is widely used (e.g. by Ajax[12] development) as it represents a great strength of the language. JavaScript is used far and wide and is one of the most used programming language in existence, with for example Node.js[13] being one of the most used runtimes in the world. It is allowed as a scripting language in following projects:

- Google's Chrome extensions[14]
- Adobe's Acrobat and Adobe Reader[15]
- OpenOffice.org[16]

10 https://blog.apollographql.com/persisted-graphql-queries-with-apollo-client-119fd7e6bba5
11 https://developers.google.com/web/fundamentals/performance/lazy-loading-guidance/images-and-video/
12 https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX
13 https://nodejs.org/en/
14 https://chrome.google.com/webstore/category/extensions
15 https://get.adobe.com/de/reader/
16 https://www.openoffice.org/

- RPG Maker MV[17]

JavaScript also serves as Application platform for a variety of successful software development projects. Originally it was submitted to ECMA international[18] in 1997 as a scripting language for browsers. At this time, JavaScript is also used for a wide array of other tasks, like for example application programming. Some of the prominent ones written in JavaScript are for example:

- Apache Cordova[19]
- The Electron framework[20]
- Ubuntu Touch[21]
- Open webOS[22]

The Hypertext Markup Language also known as HTML is introduced here. It is a text based markup language for the semantic structuring of digital documents like texts with hyperlinks and images. HTML documents are received by a web browser and sent by a web server or retrieved by local storage. They are one of the core building blocks of the world wide web. HTML serves for semantic structuring, but not its formatting. HTML provides a means to display this structuring by designating elements using so called tags such as `<img />` which designates a image or `<p>text in here</p>` which marks the content as a text block. These tags tell the browser what certain elements are. Based on this information, the CSS styling language can be used to style only e.g. certain aspects such as font-size or color of text elements as will be explained later on. Browsers explicitly do not display this tag information on the page. Instead they use this information to interpret what can be seen on the page. The latest version of HTML, that is HTML5, has introduced an array of interesting new features to the markup language. Web workers can now be implemented to load large scripts via parallel threads. The Canvas[23] API can be used to render graphics rapidly. Also, application caches allow the developers to store more information about large web apps on the users device. The Geolocation[24] API has been introduced and is mainly used and best known on mobile devices.

The formatting, that is, how a structured text is rendered on a screen or on paper, is not part of the HTML specification [65]. The

17 http://www.rpgmakerweb.com/products/programs/rpg-maker-mv
18 https://www.ecma-international.org/
19 https://cordova.apache.org/
20 https://electronjs.org/
21 https://ubuntu-touch.io/de_DE/
22 http://webosose.org/
23 https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
24 https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

formatting language Cascading Style Sheets (CSS) is meant for this purpose. Cascading style sheets, also known as CSS, is a styling language for electronic documents. Together with HTML and DOM it creates the main building blocks of the internet as we know it. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts[25]. The name cascading comes from certain rules which determine in which order the code applies to an element. CSS is implemented in such a way, that a lot of power is granted to developers over how elements, which are displayed to the user, behave. In CSS, selectors are used to determine which styles are applied to which items. The identifier `#navigation` selects the id navigation, `.list-entry` selects all elements with a class called `list-entry`. There are many other ways to access elements for styling, but these are the most basic and widely used ones. CSS is also able to animate transitions and 3d transformations[26] of items on the screen. When a user for example hovers over a black link text, the font color could be turned blue and the text could be underlined to provide the user with a visual cue that he may be clicking a hyperlink [36].

2.5 EXISTING EDITORS

Other in-browser code editors exist and are widely used by programmers and designers. A closer look will now be taken at which editors are already broadly in use, what their advantages are, and which design decisions they have introduced in the past. Later on, it is analyzed in detail, how these concepts can be improved upon. Similar projects include, but are not limited to the following:

2.5.1 *JsFiddle*

This editor was one of the earliest code playgrounds and a major influence for all which followed. It offers the features of writing HTML, CSS and JavaScript code and seeing the result in a output window as well as URL sharing. We will from now refer to these as Basic Editor Features. JS Fiddle also lets users add external libraries and CSS pre-processing with e.g. SASS[27] or LESS[28]. Templates can be added to circumvent the need to write boilerplate code as can be seen in Figure 2. JsFiddle also supports Code Forking, Autocompletion and Linting. Code Forking refers to the ability of creating new repositories with unique URLs from an existing repository. JsFiddle is still being actively used by the online community. Especially Sites like **stackoverflow.com** make use of jsfiddle in order to show code snip-

25 https://www.w3.org/TR/CSS/#css
26 https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transforms/Using_CSS_transforms
27 https://sass-lang.com/
28 http://lesscss.org/

pets, illustrate problems or ask questions about code-specific topics. There is no in-browser console support within JsFiddle yet, but the JsFiddle development team has it on their Road-Map[29] and wants to integrate it soon.

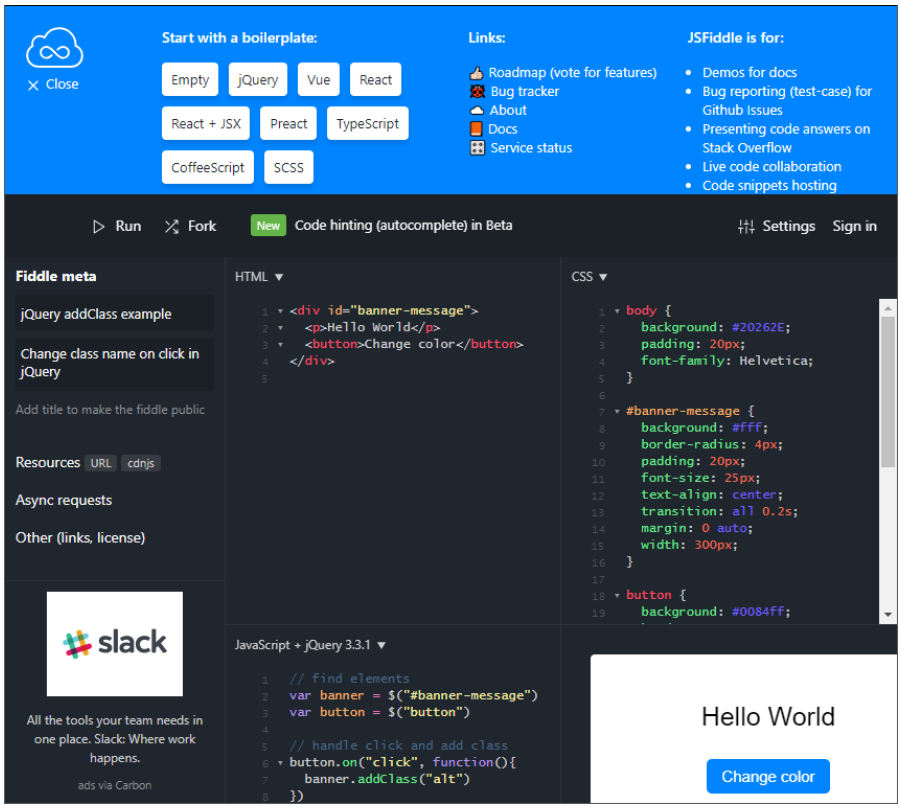


Figure 2: A screenshot from jsfiddle.net

2.5.2 *JsBin*

This editor offers the Basic Editor Features and also many advanced properties like library support, CSS pre-processing, code forking, linting and even an in-browser development console for debugging. It also sports a column-style UI where columns can be added and removed. This feature allow for varying amounts of screen space for the windows and therefore greater flexibility. **JS Bin** offers premium accounts[30] that come with "Pro Features" like for example private bins and embedding functionality. JsBin allows code to be cloned to other bins which are by default always public if no pro account was bought.

29 https://trello.com/b/LakLkQBW/jsfiddle-roadmap
30 https://jsbin.com/upgrade

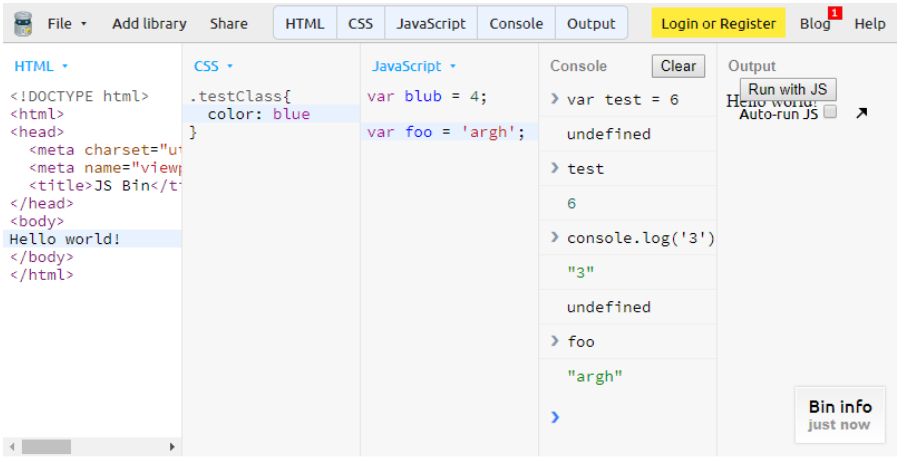


Figure 3: A screenshot from jsbin.com

2.5.3 *CSSDeck*

This editor supports all Basic Editor Features. CSS Deck is capable of more than only CSS. It is an old project, that is not actively maintained anymore. In comparison to other online editors, CSS Deck is very limited in its features. It offers CSS pre-processing and is one of the few editors that supports the aged HAML[31] HTML pre-processing. Other features are not supported as can be seen in Table 1. The styling of the page breaks in multiple locations, which hints at an old code base as can be seen in figure 4. Mobile devices or even smaller screens are not supported at all. There is also a requirement for its code snippets to need moderator approval. One would regularly read *"Page is Under Review. The lab is awaiting moderator review and approval."*. This then denies access to the code in question, further limiting use of this editor.

31 http://haml.info/

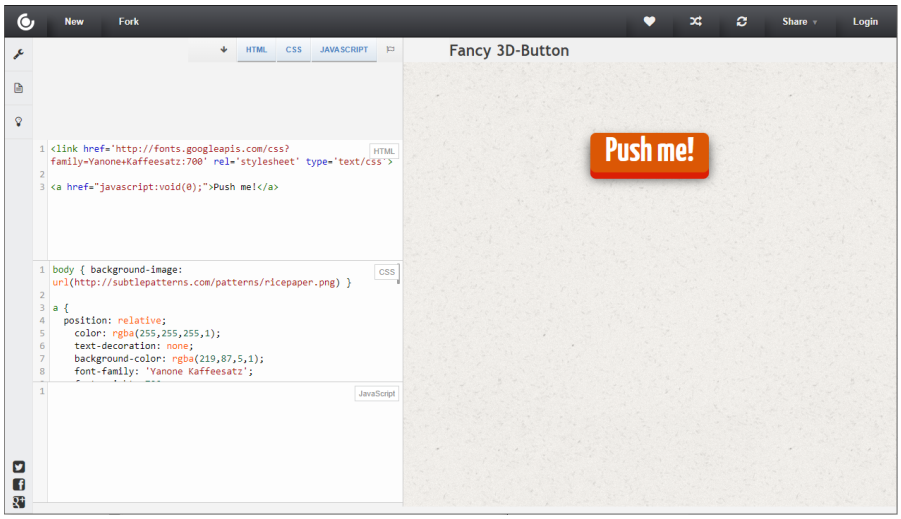


Figure 4: A screenshot from cssdeck.com

2.5.4 *CodePen*

This editor is one of the most wide-spread and famous ones. It is updated regularly and is used heavily by the community. It supports the Basic Editor Features, as well as library support, CSS pre-processing, a developer console for debugging, code forking, templates, autocomplete, linting and multi-file support. Its so called "Pens" account for many examples of scenarios and can be duplicated for quick starting. Additionally, it offers a flexible multi-window UI and settings, that can be adapted for many needs. In figure 5 an example showing React[32] code shows the style and code highlighting Codepen employs. Multiple flavours of HTML, CSS and JavaScript are supported, like for example HAML pre-processing for HTML, SCSS for CSS or jQuery support for JavaScript to name a few. Specific details can be seen in Table 1. Codepen even supports Vim[33] key bindings for developers who are so inclined. The UI offers many different features, that not many other editors can boast, like for example a full page, detail or debug view. The CodePen Pro users can also switch into live view mode, where other users can watch them code live. Collab mode allows real time work on the same pen for different users at the same time like is known from for example Google Docs[34] or MicroSoft Word Online[35].

32 https://reactjs.org/
33 https://www.vim.org/
34 https://www.google.com/docs/about/
35 https://office.live.com/start/Word.aspx

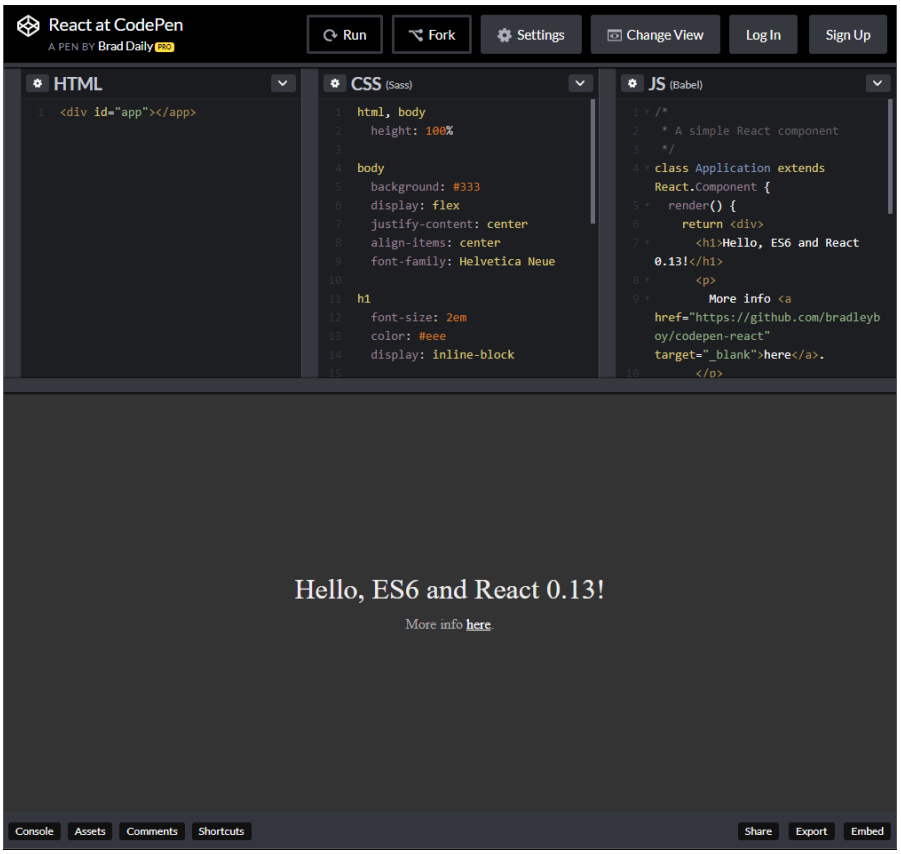


Figure 5: Codepen.io : HTML, CSS, JavaScript and the Output windows are visible in this online development environment.

2.5.5 *Dabblet*

This editor is one of the very early editors of the web. It has been around since 2011, but has not seen many updates or upgrades in the past. As such, many features, that others consider normal, are not present on this site. The page itself breaks on medium sized and smaller screen sizes. **Dabblet** uses a full-page design. Since Dabblet has been a one-woman-project, the reduced features are understandable. Other projects have been developed by large teams. The seven year old project is still maintained, but has been barely touched[36].

36 https://github.com/LeaVerou/dabblet

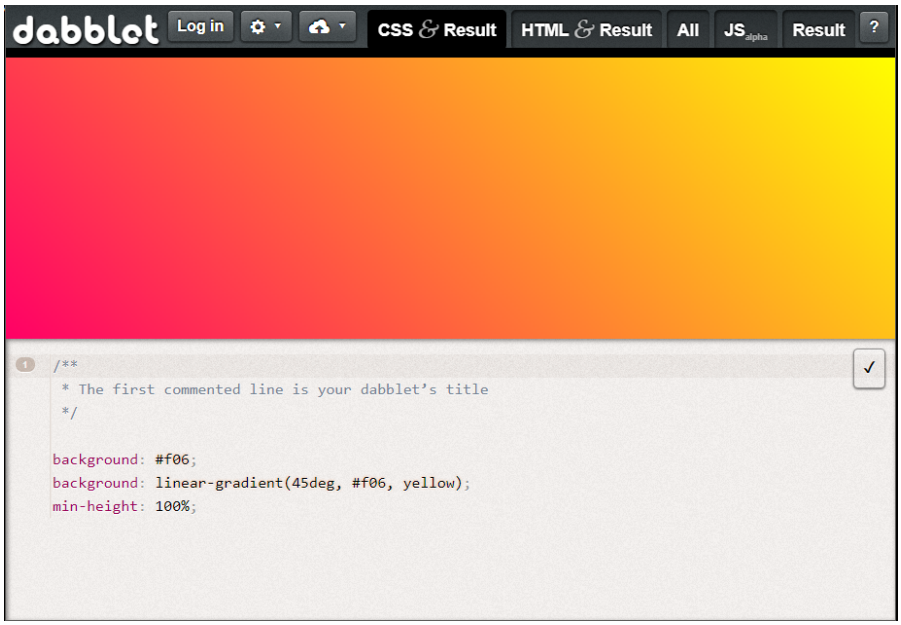


Figure 6: Dabblet: One of the older online JavaScript editors

2.5.6 *Plunker*

This editor is not being used very much anymore. It supports the Basic Editor Features together with library support, CSS pre-processing, code forking, templates, linting and multi-file. Its multi file feature can be seen in Figure 7. It also has a user generated template library. Its repositories can deliver the boilerplate one needs for smaller projects. Since **Plunker's** community is not as large as those of Codepen or JsFiddle, many repositories, called "plunks" in this case, are outdated. The number of repositories in also very limited, which makes it hard to find special scenarios in case of need.

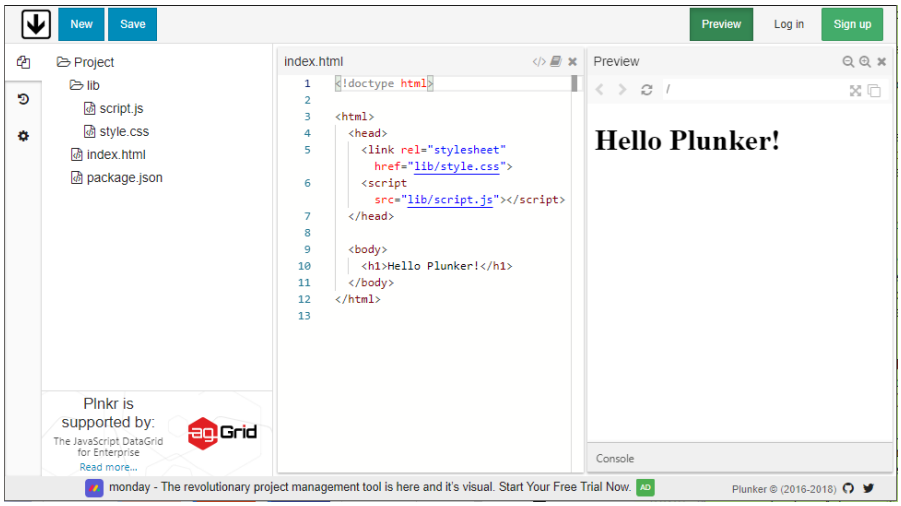


Figure 7: Plunker. A JavaScript online code editor.

Table 1: A comparison of the better known freely available online editors on the web. This Figure shows a table comparing six tools in regards to their features. The Basic Editor Features are supported by each editor.

	CodePen	JsFiddle	JsBin	Plunker	CssDeck	Dabblet
HTML, CSS and JS	✓	✓	✓	✓	✓	✓
Color coding	✓	✓	✓	✓	✓	✓
URL Sharing	✓	✓	✓	✓	✓	✓
Preview	✓	✓	✓	✓	✓	✓
Library Support	✓	✓	✓	✓	x	x
CSS pre-procssing	✓	✓	✓	✓	✓	x
Developer Console	✓	x	✓	x	x	x
Code Forking	✓	✓	✓	✓	x	x
Templates	✓	✓	x	✓	x	x
Auto-complete	✓	✓	x	x	x	x
Linting	✓	✓	✓	✓	x	x
Multi File Support	✓	✓	x	✓	x	x
HTML pre-processing	✓	✓	x	x	✓	x
Modern UI	✓	✓	✓	x	x	x

2.6 LEARNING MANAGEMENT SYSTEMS

A learning management system is a software application for the administration, documentation, tracking, reporting and delivery of educational courses, training programs, or learning and development programs [25]. Many higher education institutions have implemented a learning management system (LMS) to manage online learning and teaching. Widely spread in the US are "Blackboard" [8], "Moodle" [23] and "Canvas" [42]. The question remains what the advantages of using a learning management system are over traditional teaching. A LMS usually manages all types of information, including documents, videos, files and courses. In higher education, LMS will usually have some other features, like quizzes, rubrics, facilitated teaching for educators and a syllabus. A discussion board may also be included for the students to talk about concurrent teaching materials as was done for example by Wang et al (2012) [85]. Through LMS, teachers may create and integrate course materials, articulate learning goals, align assessments and content, create customized test for students and track studying progress [25]. LMS also allows for the organization of learning time-lines and communication of objectives. Teaching in higher education often benefits from different teaching and learning formats, such as flipped classrooms [81], learning circles [17] and others that can be easily implemented for small class sizes [92]. Sadly these formats often become impossible to implement due to limitations in available staff and increased complexity. Large classes and lectures are not necessarily problematic, but they can promote poor results [9, 54] and make it harder to use alternative teaching methods like for example the above mentioned flipped classroom. Learning management systems can be used as part of a strategy to cope with missing staff and reduce complexity. LMS bring their own problems though. Data shows the existence of challenges, such as insufficient time, technical problems, students and teachers not being familiar with technology, and integration of systems into courses, which are already filled with material [6].

Advantages of LMS are among other things the following:

- Accessibility. LMS can be used from many devices in many locations. They do not require students to be physically present in class to be used. Content can be modified at any times and students will receive the updates instantly. This feature can be used by teachers to update learning material.

- Interoperability. Learning management systems offer the functionality of cross-department and cross-faculty collaboration. This may decrease overall work-load, since teachers can share tasks.

- Reusability. Courses, videos and other materials within an LMS can be reused indefinitely. Once prepared, it can be used for all

classes thereafter [55]. This also potentially leads to an accumulation of knowledge over time.

- Durability and maintenance ability. Since most LMS are deployed off-campus, these will work, even when the electricity is down. In such a case, users can still access the system via browsers at home, since clients would be unavailable. It is also improbable for the system to crash or not work. Many of these systems are easy to maintain and User interfaces like Text-to-speech or visual aids exist to make them usable by most students, also by those with disability [32].

- Adaptability. Learning management systems can be applied to a variety of tasks and are highly adaptable within them. They can be used e.g. for high schools, middle schools, universities or even corporations and can be tailored to fit the task.

- Strong content. LMS can provide all kinds of content, that a computer can provide. Quizzes, videos, images, soundtracks and many more.

- Evaluation of students is easier and arguably fairer, since it can be partly automated through online quizzes and attendance can be tracked automatically. This is prone to errors though and should be examined carefully before use.

Disadvantages of LMS are on the other side these:

- Acceptance. Teachers and also students have to accept and use the infrastructure provided by the LMS. Especially older teachers are used to their ways of educating, without using LMS. These persons have to be convinced and studies have shown this is not an easy or quick task [84, 85].

- The implementation of LMS requires a well-built technological infrastructure. Students need screens in order to access the clients, while teachers need equipment like laptops or tablets in order to upload their materials and access the system. Not all schools and universities are financially equipped to handle these requirements.

- Current studies contradict themselves in whether technology and LMS truly reduce the workload of teachers. On one side, it surely reduces bureaucratic burdens by automation of certain tasks. Then again on the other side, teachers are always connected to the system. Students who spot errors may immediately notify the teacher and expect changes to be made. Such systems may produce stress and work for the teacher even after class and during time off.

2.7 RELATED WORK

In the following section, we will specifically look at related work from educational sciences and technology-related topics alike. These works are of interest to the development of this project and the scientific analysis thereof. This project builds upon insights and successes seen within prior works. Facebook itself was used as a learning management system by Wang et al. (2012). In their experiment, they have taken two university courses and employed Facebook for the sharing of exercises and resources, organization of meetings, setup of tutorials and for giving out announcements to students [85]. Since Facebook provides social, pedagogical and technological affordances [59] through its built-in functionalities it was chosen for the task. The study explores how the students perceive Facebook as am LMS throughout their course. The study's results indicate that students were satisfied with the experience that Facebook provides as an LMS. A problem was, that some students did not feel comfortable about their identities being revealed by Facebook and its privacy in general.

WAINWRIGHT ET AL. (2009) introduced a new Learning Management System called Moodle at the Lewis and Clark College. Moodle is a free and open-source learning management system (LMS) written in PHP [69]. In their paper, they talk about the difficulties in getting staff and students to use the new system. They had to start a mission of evangelism about the new system, since the old one had barely been used and mostly ignored before these changes. Things to consider when introducing a new LMS include the technical and pedagogical training of teaching staff and maintenance implications. Introducing a LMS into a college proved to be very time consuming. A lot of patience and effort was required to make it work and raise awareness of staff and students [84].The change in culture also requires much finesse and care on the part of those spearheading it. When Moodle was developed, Dougiamas et al. (2003) contributed to it by researching what they call *social constructionism* and *connected knowing* in their own online classes. Since then, Moodle has been translated into twenty-seven languages and being used all around the world by educators [23].

WEAVER ET AL. (2008) have looked at LMS in the academic world. They have analyzed a LMS called "WebCT" via surveying of academic staff and students. It was believed that student feedback would relate to technical and infrastructure issues. Instead, the survey returned responses primarily on how WebCT was used in teaching and learning, indicating that quality control is a major issue for the University. They found out, that student opinions mostly reflect the material created by the teachers - students who have seen a well-designed unit laden with

resources, timely feedback and good interaction with staff reported a positive experience with the technology. Staff feedback was concentrated on technical and management aspects of using WebCT, rather than educational issues. This paper may have implications about how to use LMS in general and how students and staff view them. They concluded, that learners should be more actively engaged by the systems. They concluded that staff and students are ready to engage more into e-learning and online approaches [87].

RÖSSLING ET AL. (2008) have examined how LMS can be used to improve education in computer science. Many instructors and universities tend to increase the amount of comprehensive LMS usage, such as Blackboard Learn[37] or Moodle[38]. Blackboard Learn is an LMS system and a Web-based program, which features course management, customizable architecture, and scalable design that allows integration with student information systems and authentication protocols [16]. It is good for managing courses and enhancing student learning. Computer science educators keep developing tools that help in management, teaching and learning in computer science courses. LMS that are made specifically for computer science are abbreviated as CALMS - Computer Augmented Learning Management Systems. Rößling et al. (2008) have concluded, that novel computer science education technologies can be integrated in some scenarios and that this would improve the learning and teaching process. If their goal of technical integration takes place, new pedagogical models will emerge. For example, the chance of real-time automatic evaluation within a LMS supported classroom environment.

A NOVEL INVESTIGATION by Saw et al. (2018) has shown, that in this case Moodle was a great support to teaching and assessment evaluation at higher education facilities in Myanmar. With growing numbers of students, learning management systems have become a necessity at universities in Myanmar. Moodle is used there to manage teaching and studying. A total of 318 respondents answered a questionnaire about learning management systems. It was not shown however, that LMS have any positive effect on students grades or productivity [73].

TSUKAHARA ET AL. (2007) have made an extensive three year study at the Tokyo University of Agriculture, where they where challenged to replace over 30 courses with e-learning courses. They developed their own LMS by expanding and modifying an existing LMS called WebClass. This existing LMS makes it easy to add new functions. In their paper [82], they discuss the process of developing

37 https://www.blackboard.com/
38 https://moodle.org/

and introducing their new system. They analyzed effects on administration, authoring of tasks and communication through the system. Their system enables registration, authoring and importing information from their educational affairs section. This enables effective mentoring and coaching by the use of digital portfolios.

In conclusion, learning management systems are a positive addition to the worlds teaching landscape. Many schools and universities around the world that currently use them, could not do without them. The advantages by far outweigh the disadvantages and few have yet regretted the decision to transition into LMS supported teaching. As we have seen, the biggest problems stem from affordability and insufficient technological advancement of some institutions.

3 CONCEPT

Previously, properties of existing online editors have been analyzed and the meaning of LMS in the educational world has been introduced in Chapter 2. In this chapter, the concept for a JavaScript online editor running in the browser is introduced. In Section 2.5, details about features supported by existing editors were analyzed. Based on these existing capabilities and didactic fundamentals from Section 2.1 an editor for the Backstage system is proposed. From now on this editor may be referred to by its project name: QuestJs.

3.1 GOALS

The aim of the proposed editor is to assist in the study of coding in JavaScript. In order to successfully do this, certain criteria need to be taken into account. Users of the editor are primarily students who are unfamiliar with JavaScript code or who have formerly not done much web development. It is essential to not overload users' cognitive abilities with an overly complicated editor [22, 77]. This goal may be partially reached by not having to set up a local development environment. It is therefore the intention of this editor to provide a positive, encouraging and easy-to-use development experience. The user interface must be intuitive, so that users can start to write code as soon as the editor is opened in the browser. In addition to this, another explicit goal is not only the ability to write HTML, CSS and JavaScript code, but also to see the compiled result in the browser. It must also be possible to debug and test the code in a manner that comes close to developing code inside an integrated development environment in order to introduce learners to this side of programming as well.

3.2 TECHNOLOGICAL CONCEPT

A potential JavaScript editor has input and output fields. Initially it would support a number of features corresponding to the needs of its future users. In Chapter 2, a set of Basic Editor Features has been established. These features are supported by all the editors that were analyzed previously. They are therefore present in this concept as well. The editor will run client-side only, without the need of a server. This brings with it a number of advantages, such as fast user interface response times and therewith improved usability. Since JavaScript is a dynamic browser native language, it does not have to be compiled on a server in order to run. JavaScript runs instantly. This feature may

be powerful from an educational point of view - users can write code, instantly run it and see the results. The driving technological idea is to use an inline frame[1] in order to bring together and render the HTML, CSS and JavaScript code the users write in the editor.

3.2.1 *Input*

The editor has these input fields:

- **HTML.** An input view where HTML code can be written.
- **CSS.** An input view where CSS code can be written.
- **JavaScript.** An input view where JavaScript code can be written.

3.2.2 *Output*

The editor has these output fields:

- **Output** window. A HTML inline frame is to be used to show the output generated by the written HTML, CSS and JavaScript code.
- **Testing window.** This window will show results of tests written by teachers. These tests automatically run on the code the users write and provide feedback to learners.

3.2.3 *Additional Features*

The editor supports these features:

- **Syntax highlighting.** The above mentioned HTML, CSS and JavaScript Input windows support colored code highlighting for better readability. Syntax highlighting also has positive effects on code comprehension as was shown by Reijers et al. [66].
- **Linting.** The above mentioned HTML, CSS and JavaScript Input windows support basic code linting. Linters are static analysis tools, that warn developers about possible code errors or violations to coding standards. They are therefore part of White-box Testing [88]. By introducing this feature, errors can be spotted in real time without the need to run the program at all. Users are able to see their syntax mistakes directly in the browser and are able to correct them accordingly. Using linters improves readability, enables code reviews and provides a way to find errors before execution [80, 91].

1 https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe

- **Library support.** is implemented in order to allow users to add broadly available open-source libraries such as JQuery[2], React[3] or Require.js[4]. Users should not be overwhelmed with possibilities on their first day of coding JavaScript. On the other hand, libraries may also provide advantages. They enable users to employ shortcut functions, which potentially save time and make it possible to do more with less code as can be seen in Listing 1. This stands in direct conflict with the aim of acquiring a deeper understanding of the language. Therefore, this feature is optional as required by the teacher. The corresponding dropdown menu may or may not be rendered as needed.

  Listing 1: The JQuery library versus normal JavaScript. JavaScript can achieve the same, but sometimes needs more lines of code compared to JQuery. JQuery is declining in popularity due to JavaScript catching up rapidly.

```
1 //standard JavaScript
2 document.getElementbyId('mydiv').style.color = '#f00';
3
4 //the Library JQuery
5 $('mydiv').setStyle('color','#foo');
```

- **Templates** are provided by the teacher through the editor in order to give some starting help to the users employing the concept of scaffolding [63]. More advanced students or classes can be given less help. Support can gradually be taken away using the concept of Fading [63] in order to keep students within their ZPD at all times.

- **Developer Console.** In order to give users the possibility to experience dynamic JavaScript development, they are given the ability to debug their code. For this purpose, a developer console is proposed, that is fully integrated into the development experience in the browser. This console operates on the inline frames JavaScript context. Users are therefore able to receive information from this context while the program is running, potentially from each execution step. They can use it to output debugging information in real time via for example `console.log()`[5]. The console can also be used to compare objects, handle exceptions and errors, monitor events, measure and count executions and evaluate expressions to only name a few possible ways of use. As a two-way communication terminal it can receive input and provide output back to the users.

2 https://jquery.com/, https://insights.stackoverflow.com/survey/2018/
3 https://reactjs.org/
4 https://requirejs.org/
5 https://developer.mozilla.org/en-US/docs/Web/API/Console/log

- **Testing.** Tests are included using the Chai.js assertion library[6]. It can run in the browser since it only requires JavaScript. This feature lets teachers write tests for their users. These tests will then run on their code and provide a way of automated feedback [51].
- **An Exercise view** is developed that supports markdown script in order to pass information about the exercise that is to be done directly to the user.
- **A Modern User Interface** is implemented taking into account the users requirements and general usability rules.
- **Autocomplete.** Having this feature would be advantageous. Time constraints and complexity may not allow for its implementation. It is optional.

The goals defined in Section 3.1 may be partially reached once the above features are successfully implemented. The following features are omitted from development for a variety of reasons. The main one being cognitive load theory. Based on the presumption that working memory is limited, as well as the fact that users are mostly programming for the first time, they should not be overloaded with input or their learning may be hampered [22, 77]. Additionally, the need for didactic reduction [28] has to be taken into account when developing a prototype for young users. Since the implementation is going to be client-side only, some of the following features are not viable, because some of them would require a server:

- **Multi-file support.** The editor is mainly going to be used in live lectures. As such, there is not a lot of time to develop complex systems or programs with it. Also, multi file support would provide an additional layer of abstraction and complexity to the learning process. The primary target group are first and second semester students, who may not have had a lot of experience with programming. In order to not overly strain their cognitive abilities with too much load, multi file support is not going to be supported [77].
- **CSS Pre Processing.** This feature would require the users to learn an additional syntax. SCSS or LESS use their own optional dialect of CSS. For beginners, this is not the way to go, as CSS itself offers sufficient content. At the same time, it allows for the same outcome.
- **Code Forking.** Since the code that is being written is not saved to any repository, forking will initially not be implemented.

6 https://www.chaijs.com/

- **A Collaboration mode**, where users can stream their screen to others would require a different architecture. It is therefore omitted.

Users will then be able to write web development related code and instantly see the compiled and combined result in the output window inside an inline frame. The instant feedback of this approach may provide positive learning outcomes. The proposed usage inside lectures provides advantages associated with active learning [11, 43, 45].

3.3 USER INTERFACE CONCEPT

Users have over time become familiar with interface elements acting in a certain way. This is why the editor's user interface is consistent and predictable. Doing so will help with task completion, efficiency and satisfaction [29]. There are many interesting user interface concepts available and already in use as can for example be seen in Figures 2, 3 and 5. Before a user interface for the QuestJs editor is developed, a scientific foundation needs to be established. Many resources are available when it comes to user interface and user experience design. Inspirational for the concept phase of QuestJs is Peter Morville's usability honeycomb [56]. QuestJs users are mainly young students who want to learn JavaScript development. As can be seen in Figure 8, Morville recommends a product to be useful, desirable, usable, findable, credible and accessible in order to be valuable. The question surrounding the meaning of these words, given the relation to the problem at hand still remains.

Figure 8: Peter Morville's Usability Honeycomb [56]. Each facet of user experience design can be defined by this diagram. The honeycomb helps to identify all the areas that are important to a strong user experience [57].

The concepts from Figure 8 can be explained as follows:

- **Usability.** The editor needs to be simple and easy to use. It is designed in a way that is familiar and easy to understand. The learning curve a user must go through to understand the user interface and its features should be as short and painless as possible [56].
- **Usefulness.** The editor provided needs to be useful and fill a need. If the editor is not as useful or fulfilling as the users needs, there is no real purpose for the product itself [56].
- **Desirability.** The visual aesthetics of the service need to be attractive and easy to translate. Design is minimalist and to the point [56].
- **Findability.** Information needs to be easy to find and navigate. If the user has a problem, they are able to quickly find a solution. The navigational structure is set up in a way that makes sense [56].
- **Accessibility.** The QuestJs editor is designed in such a way, that even users with disabilities like for example some forms of visual impairment can have the same user experience as others [56].
- **Credibility.** The service needs to be trustworthy. Studies have shown users to be reluctant to enter any information if the service is not credible [56] as was also shown in a study by Wang et al. (2012) [85].

If all the above mentioned points are taken into account, a valuable and meaningful product can be created. User interface design places its focus on anticipating what users might need to do and makes sure that the interface has elements that are easy to access, easy to understand and easy to use in order to promote those actions [29]. User interface design brings together concepts from interaction design, information architecture, and visual design [29]. Unnecessary elements are avoided in favour of a clear layout and concise, but understandable labels [29]. However, labels are avoided wherever possible in favour of clearly implied usability. The goal here is to create consistency and use common, well known elements in order to make users feel comfortable and have them find their way around the interface intuitively. Spatial arrangement within the editor is based on importance, with navigation elements at the top and views below. For directing the attention of users between elements, strategic use of color is implemented. In order to for example promote interactivity, inactive buttons are greyed out and hover effects for buttons are applied to focus attention and deliver visual cues [29, 67]. It is important for a

system to always communicate clearly what is happening at the time. All buttons and other user interface elements used in the QuestJs prototype are, where necessary, appropriately labeled and colored. All animations used in the prototype conform to patterns already known to users [29].

3.3.1 *Full Screen View*

For the QuestJs prototype, initially a large full screen design was chosen, as can be seen in Figure 9. On figure 9, a multi-column layout is conceptualized. It has one row of buttons on the upper part of the screen and several corresponding input and output windows on the middle lower part of the screen. The buttons are meant for activating and deactivating the column-like windows of the lower part of the screen, thereby hiding or revealing them within the user's field of view. This technique gives users full flexibility in choosing their own layout depending on their needs. On Figure 9, a wireframe of the QuestJs editor can be seen. On it, from left to right, the following views are active: HTML, CSS, JavaScript, Console and Output, as can be seen by inspecting the navigational buttons. The Column below the buttons matches the label on the button. In this case, users have four input windows (HTML, CSS, JavaScript and Console) and one Output window, where the compiled code can then be viewed. Arguably, the console is also an output window, since it is able to for example to output debug information. It is a two-way view. On large screens, the flexibility of this layout provides a direct overview of every part of the development process involved. The advantages of this layout become apparent in Figure 10 : Views can be opened and closed at will via a press of the button. The button then changes to a darker color in order to provide a visual cue for the columns hidden status. Users are nowadays already conditioned to associate darker colors with inactivity [29].

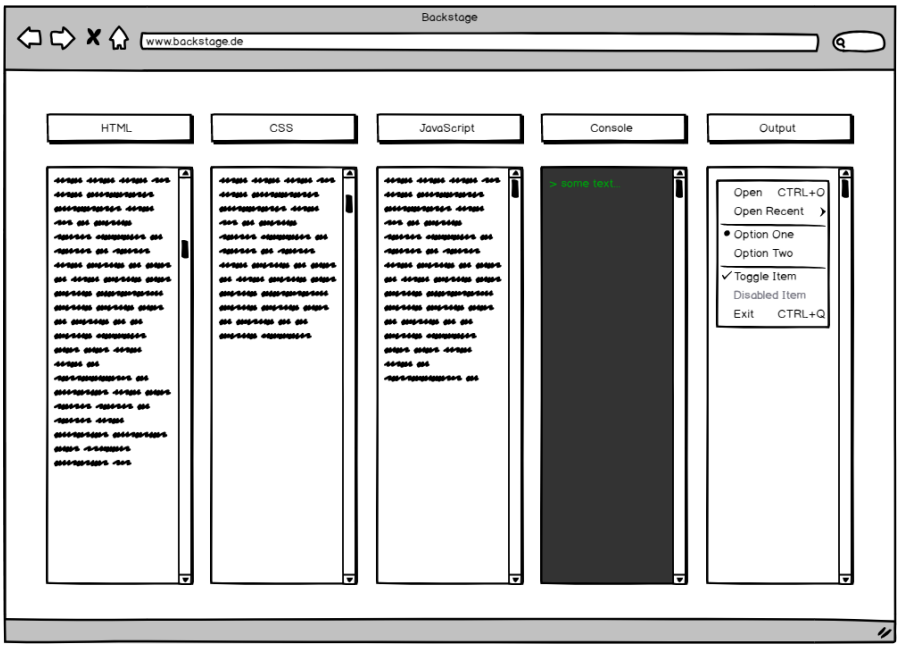


Figure 9: A high-fidelity wireframe for the QuestJs editor.

In some situations for example, if one of the exercises requires only JavaScript code and its results to be viewed, windows from the editor can be hidden at will by users by pressing the corresponding buttons. An example of this can be seen in Figure 10. As the button is pressed, the corresponding window is hidden from view. The remaining windows then take up all the remaining space improving visibility for the user.

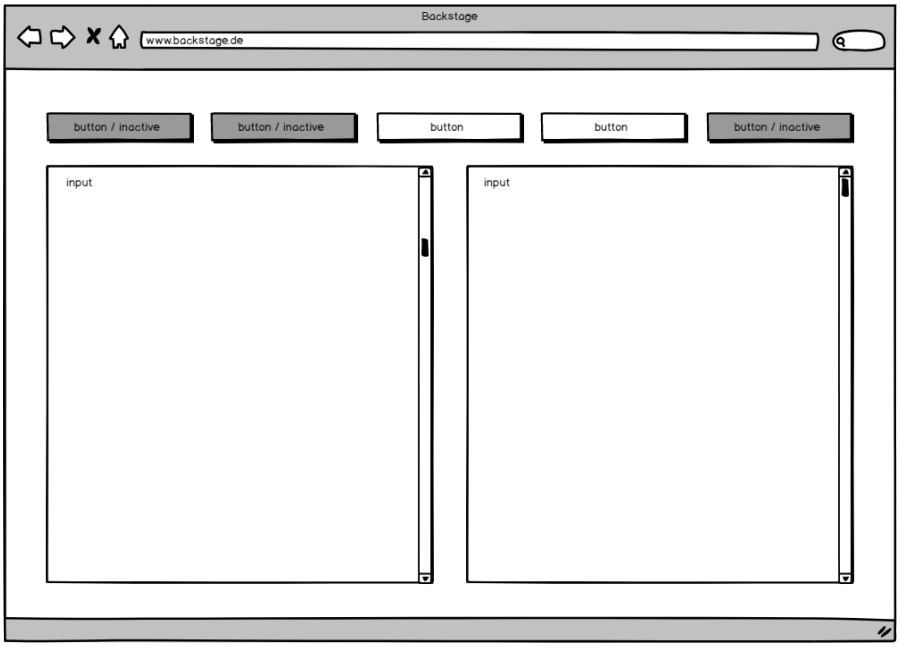


Figure 10: A view of the QuestJs editor with some windows inactive.

Teachers are able to decide which features of the editor they want to make available to the students. Depending on which views are necessary for the current task, the others are removed from the site.

As an example, users could have the task to write some HTML code and style it via CSS in a certain way. For this task, only the HTML and CSS input views, as well as the output view, would be needed. Educators could then initialize the editor in this way for the users as can be seen in the wireframe in Figure 11. This approach brings a number of advantages related to didactic reduction [28]. Since only the views needed are shown, users would not be overloaded with unnecessary information and possibilities. It is therefore easier for them to focus on the tasks at hand [22, 77]. This reduction can be applied to a number of scenarios in the web development context. Other prominent examples would be a JavaScript view and the console in a two-window scheme for JavaScript training or the JavaScript, HTML and output view in order to learn better on-site scripting.

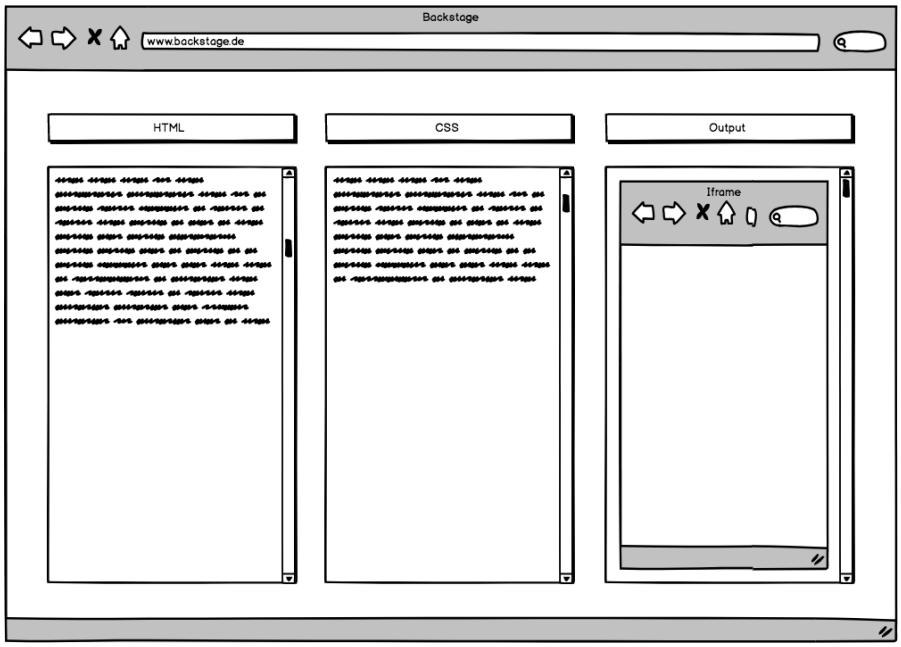


Figure 11: A view of the QuestJs editor with reduced views.

3.3.2 *Compact View*

In compact mode, the navigation changes from a navbar-multi-column-layout, to a single column layout using navigational tabs. This way, only one view is visible at a time, making it suited for smaller devices. Figure 12 shows a wireframe of this compact view. The navigation bar is reduced to tabs. The only way to view content in this mode is inside of the one-column layout. Users can switch between different input and output views by clicking the tabs at the top. Clos-

ing a tab permanently is impossible. The content is always preserved and users can switch between the different views.

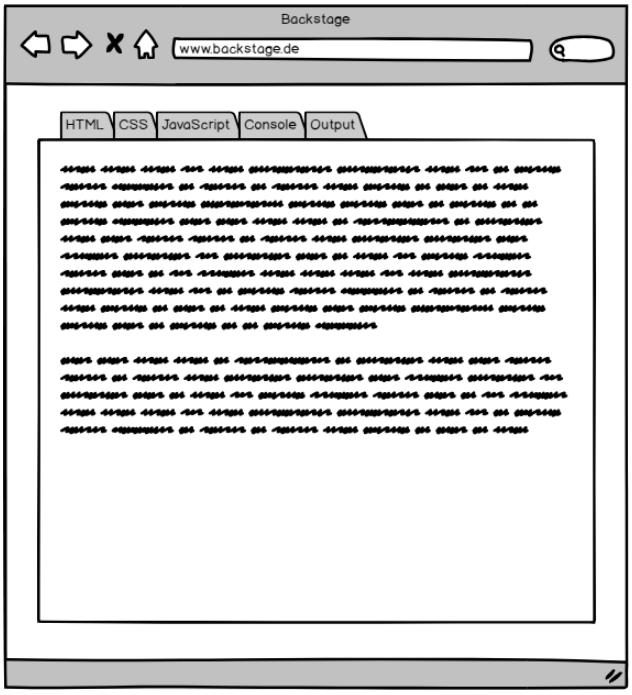


Figure 12: A view of the QuestJs editor in compact mode.

Since the semantic behaviour of both the large and compact view are identical, a button is added to the user interface, that lets users switch freely between the two. This way, users on all devices can use the view that best fits their requirements.

3.3.3 *Buttons*

A number of buttons are introduced to the editor. Buttons are needed for navigation in large mode as can be seen for example in Figure 11. A button to navigate between the large and compact view is added as well. A drop-down-menu allows users to choose and integrate a pre-defined assortment of external JavaScript-based libraries. Another button lets users run the code and therewith the tests. The libraries drop-down-menu is optional. Educators decide if it is shown to the users or not, depending on their level of knowledge and the context of their tasks.

3.4 TESTING CONCEPT

Finally, a concept is needed for educators and also users to see if the code written by them fulfills the functional requirements of the exercises. A mix of manual marking and automated testing is proposed. Teachers are able to write tests for the code and pass them on to users

together with the exercises. The Chai.js assertion library is used for this purpose. The tests run automatically whenever the code is executed and the inline frame containing the output view is refreshed. A view containing test results is also present for users to see their results as shown in Figure 13. Chai.js is an assertion testing library that runs in the browser. For QuestJs, the implementation in the browser is used, since the editor runs client-side only. It was chosen from an array of possible testing libraries because of its capability to run inside the browser. As a simple example, the users could be given some numbers and the task to write a JavaScript program, which multiplies these numbers a certain amount of times and then stores them to an array. The assertion tests seen in Listing 2 could be passed to the editor by the teacher to be run on their code.

Listing 2: Chais.js assertion testing examples.

```
1 assert.isArray(myArray, 'is array of numbers');
2 assert.include(myArray, 7, 'array contains 7');
3 expect(myArray).to.be.an('array').that.includes(2);
4 expect(myArray).to.have.lengthOf(5);
```

In the tests from Listing 2, the teacher first asserts that the users have created an array using the method `assert.isArray()`. Then, they could use the method `assert.include()` to make sure the correct result, in this case seven, is stored inside the array. This can be repeated if there were multiple results. Depending on the return values of the tests, a positive or negative color-coded user interface signal will appear inside the testing window with a corresponding message as can be seen in Figure 13. The text next top the symbol is describing the nature and requirements of the test.

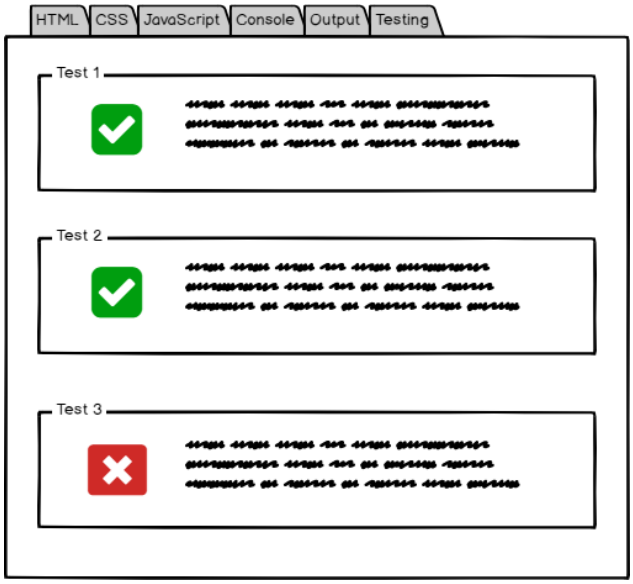


Figure 13: A view of the editor in compact mode showing the testing tab. The green symbols indicate successful tests, while the red symbol indicate tests, that have not yet run successfully.

3.5 ADVANTAGES IN REGARDS TO LOCAL IDE'S

This concept has a number of advantages when compared to traditional integrated development environments. Since it is in the browser only, users do not have to set up their own local programs. Setting up ones own environment has some intricacies and may take a long time when done for the first time. Users may use the QuestJs prototype instantly and without any possible setup complications. It is therefore very beginner-friendly and may prove a positive experience for our target group. Since the editor runs client-side only, very quick UI feedback is possible. This may provide a comfortable working experience and can prove adequate for the upcoming programmer, as remains to be seen.

3.6 USE CASES

The question which use cases exist for the QuestJs editor still remains. Naturally, it can be used by first or second semester students within web development courses, for example to solve exercises posed by teachers in class or as homework. This way, educators can directly see the work that has been done and, if they want to, provide feedback to students. Students can work on coding challenges in flipped classroom scenarios [68, 81] and perform peer-review or pair programming [58] to assess their performance. In another use case, other users wanting to learn JavaScript can use the editor to work on coding challenges. Educators or content creators can develop powerful

assertion tests in order to provide exact feedback and mostly automate the experience for users. It requires some preparation, but this way, JavaScript and web development can be taught to an audience remotely.

3.7 POSSIBLE COMPLICATIONS AND LIMITATIONS

The QuestJs in-browser development environment has many advantages compared to traditional integrated environments as mentioned before. However, there are also a number of problems that may arise when using it. One limitation is program length. Once programs become too long, users have to do a lot of scrolling in order to find their way around their programs, since only one file can be used. Especially in compact view, with a limited window width, long programs could become a problem. Another problem is posed by experienced coders. Users who already bring advanced programming experience to the lessons may feel incomplete with this editor, since functionalities such as Multi-File-Support have been purposefully omitted. They may be used to their own setups and plug-ins and would rather use those. They may therefore have trouble adapting to the straightforward approaches of the QuestJs editor. Since it is a prototype, it is expected to produce any number of other technical or user interface problems.

4 IMPLEMENTATION

This chapter discusses the implementation of a prototype. It provides technical information about the system, including design decisions taken, and an outline of structures, data flows and underlying technology.

4.1 TECHNOLOGICAL DECISIONS

Before developing QuestJs, a series of technological decisions had to be taken. Nowadays, we live in a world with a plethora of development languages with between 700 and 25,000 active programming languages in use depending on whom one asks [2, 60]. Every year, new development tools, frameworks, libraries and programming languages are developed an released. As such, it is difficult to make informed decisions. When deciding which technologies to use, there are many sources for information, like for example medium.com[1], stackoverflow.com[2] and other high quality blogs, publishing platforms and informative websites. For the QuestJs prototype editor, only JavaScript is an option, because of its ability to dynamically compile in the browser and provide instant feedback to users. Also, the Backstage system is implemented using React.js[3]. Using React.js is therefore a requirement for the system reported about in this thesis.

4.2 SINGLE PAGE APPLICATIONS AND THEIR FRAMEWORKS

A Single-Page Application (SPA) is a web application that keeps in touch with users by dynamically changing the current page rather than loading entire new pages from a server [53]. This approach averts interruption of the action between consecutive pages, making the application behave more like a desktop application. In an SPA, either all necessary code – HTML, JavaScript, and CSS – is fetched with a single page load or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions [75] or because or large load sizes. The page does not reload at any point in the process. Interaction with single page applications often involves communication with the web server behind the scenes. This process is possible because of JavaScript adding and removing elements from pages. Websites have been using more and

1 https://medium.com/
2 https://stackoverflow.com/
3 https://reactjs.org/

more JavaScript since its introduction in 1995. Because of growing network speeds and machine power, it is now in many cases possible to stay on one page and let JavaScript manage the complete UI [75]. In the case of QuestJs, there is no server communication implemented, because it is not needed. Backstage is implemented as a single page application, which is why QuestJs is also an SPA. Many successful frameworks exist for JavaScript. For example AngularJs[4], Vue.js[5] or Ember.js[6]. For the QuestJs prototype, React.js[7] is a requirement set by the project and is thus being used for its implementation.

4.3 REACT.JS

React.js is a JavaScript based library revolving around components for building user interfaces. It is developed and maintained by Facebook and an active community of developers. React.js was created in 2011 by Jordan Walke and has since then grown into one of the most used web development libraries worldwide. As a front-end library, React.js is used to build reusable encapsulated components for the web [26]. These components can then be combined to build complex user interfaces. Notable characteristics of React.js include the following:

- **Declarative.** Since React.js uses a declarative component composition style, code is predictable and easier to debug.
- **Component-based.** React.js components are encapsulated and manage their own state. They can then be composed into more complicated user interfaces [26].
- **React.js is a library.** React.js does not make presumptions about the rest of the stack in use. As such it is usable together with all kinds of software. This makes it very reusable as a library and attractive for developers to learn.
- **Virtual DOM.** In React.js, for every DOM object, there is a fitting virtual DOM object. A virtual DOM object is a representation of a DOM object. A virtual copy of the original DOM is created. The advantage to the real DOM is that nothing has to be rendered in order to update the virtual DOM - it is a one-way data binding [26].
- **JSX** is a way of writing HTML code mixed with JavaScript. JSX can best be thought of as a markup syntax that very closely resembles HTML itself [26].

4 https://angularjs.org/
5 https://vuejs.org/
6 https://www.emberjs.com/
7 https://reactjs.org/

React.js is supported by Facebook and therefore is expected to have an extended lifetime. Since every framework requires some getting into and learning, React.js is a good choice, because it is one of the most popular current frameworks. As such, it is a good thing to learn how to use this framework.

4.3.1 *React.js Props*

Props[8] in React.js are used to pass information from a parent component to a child component. This information can for example be objects in JSON format, strings, numbers or functions. The "props" object is immutable. Props are useful because they enable reusability. When data or content should change inside a component, props are used to enable this behaviour [26].

4.3.2 *React.js State*

The State[9] property is very important in many React.js class components. React.js monitors components for changes and re-renders whenever an update is detected. React.js lifecycle methods can be employed to manage this process more effectively. The state is usually used as a private state. It manages and encapsulates the state of that particular component. The idea behind this concept is to make components reusable and independent [26].

4.4 INLINE FRAME

The HTML Inline Frame element[10], also known as iframe or `<iframe>`, is an important part of the editor. These names all refer to the same HTML element. The inline frame element represents a nested browsing context that can be embedded into the current HTML page. Each embedded browsing context has its own active document and session history. It is effectively a website inside a website. The use of inline frames in general has to be done very carefully, since it can be resource hungry. Because each embedded browsing context created by `<iframe>` is itself a complete document environment, every use of `<iframe>` within a page can cause substantial increases in the amount of memory and other computing resources required by the document overall [27]. The more iframes, the more strained the resources of the enveloping system. For the editor, only one iframe is used to render and display the combined output. Inline frames are known to pose

8 https://reactjs.org/docs/components-and-props.html
9 https://reactjs.org/docs/state-and-lifecycle.html
10 https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe

significant security risks[11][12][13] as can be confirmed from several entries in, for example, the American national vulnerability database[14]. [90]. Iframes can be used to load malicious payloads onto sites from so called distribution sites [50]. Injected content can then be hidden using zero-pixel-iframes [50]. In a typical attack, the user would first visit the page and download the initial exploit script via, for example, an inline frame. The exploit script then targets a vulnerability in the browser, one of its plugins or one of its expansions. If the vulnerability is successfully exploited this results in the automatic execution of the exploit code, thereby triggering a drive-by download of further malicious code [50].

4.5 THE FUNCTION EVAL()

The function `eval()` passes a string to the JavaScript compiler and executes the result as JavaScript code. It is used within the QuestJs editor's code to execute assertion tests and to run JavaScript code that has been written in the console by users. It is therefore responsible for key features. The function `eval()` genereally compromises the security of applications because of it granting too much authority to code passed to the function `eval()`. It threatens the performance of code as a whole since code passed to `eval()` cannot for example be cached [21]. Uses of the function `eval()` are very flexible. The function `eval()` also posses a significant number of dangers. It is not recommended in general to use it often. Its practice is known to pose security risks [52, 79]. Especially in conjunction with inline frames, the function `eval()` has some risk to be abused. In attacks such as "clickjacking", a users session can be hijacked by attackers using inline frames [72]. Since `eval()` can be used to dynamically create new scripts within the page, potential for abuse by attackers is very high [90].

4.6 DATA FLOW

This section explains the data flow within the editor as can be seen in Figure 14. In the default scenario, users write code in the three input fields present: HTML, CSS, and JavaScript. The written code is then appended to the inline frames document object model and rendered using React.js. The HTML code is directly added to the document body using innerHTML, the CSS code is appended to the DOM as a style sheet, and the JavaScript code is joined using a script tag. Within the inline frame component, it is then executed using the function `eval()`.

11 https://nvd.nist.gov/vuln/detail/CVE-2018-0355
12 https://nvd.nist.gov/vuln/detail/CVE-2017-12258
13 https://nvd.nist.gov/vuln/detail/CVE-2017-7830
14 https://nvd.nist.gov/

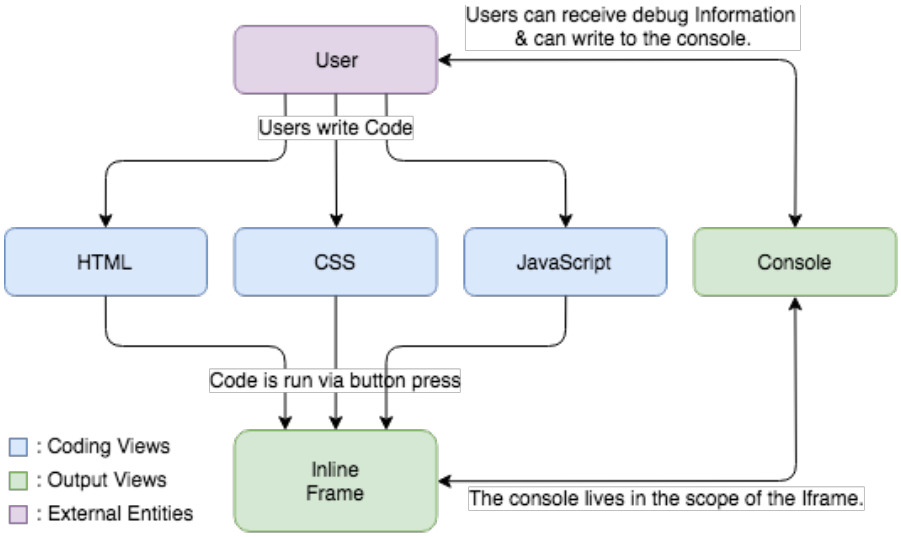


Figure 14: Data flow within the editor. The flow starts from the user who writes code and flows downward to the point where the output is displayed within the inline frame. The console can be used to interact with the inline frame or to output debugging information.

Figure 15 depicts the way the inline frame is assembled step by step. When the user clicks the Run-Code button, the re-render is triggered, and the code written within the editor windows gets transferred. At first, a inline frame element is newly created as a parent. Its innerhtml[15] property is set to the HTML code received by the editor windows. A loop runs through an array of chosen library scripts and appends them to the head of the document. Afterwards, the style sheet, if already existent, is replaced with the CSS code received from the CSS input editor window. If non-existent, it is newly created, based on the CSS code received. It is then appended to the inline frame's document head[16]. Finally, a script element is created. Its innerHTML is set to the JavaScript code received from the JavaScript editor and appended to the head of the inline frame as well.

15 https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML
16 https://developer.mozilla.org/en-US/docs/Web/API/Document/head

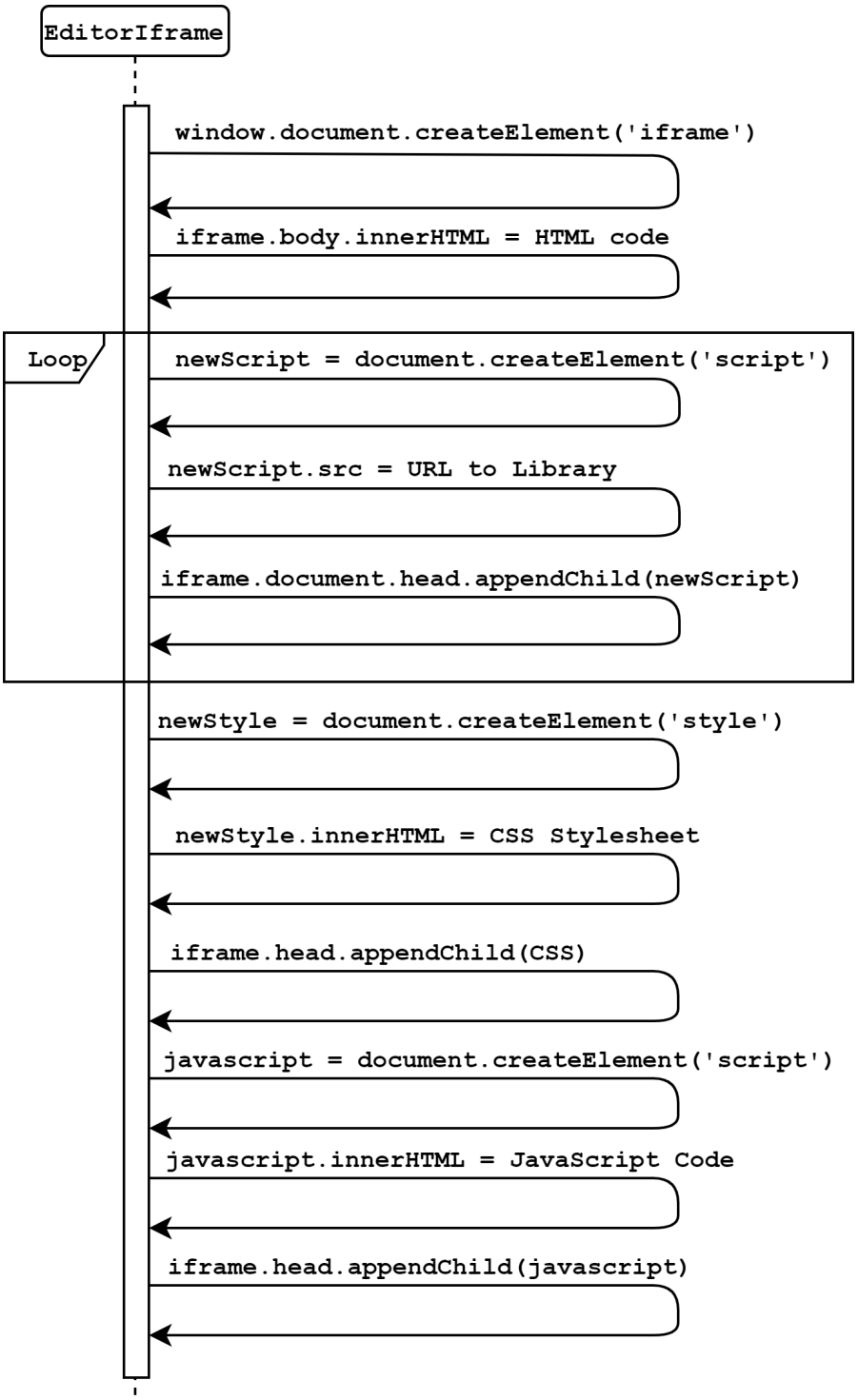


Figure 15: The sequence of execution within the inline frame. HTML, CSS, JavaScript code and library scripts are assembled in this order. The inline frame then gets rendered into the output window.

4.7 FEATURES

In this section, the implementation of the features for the QuestJs editor are described.

4.7.1 *Developer Console*

Implementing the developer console was a challenge, since it has to function on the inline frame's scope. With the current implementation, the console retains its history, even if the viewing mode is changed. Tabs in compact mode are hidden and therewith invisible but are actually always present in the background. The developer console is an approximation of the Chrome development console[17] and supports some of its features. Since it operates within the scope of the inline frame it can be used to access and debug within this scope. Errors occurring in the code written by users can be logged and displayed on the console. Additionally, it is capable of displaying nested objects, executing functions and generally operating on the inline frame's scope.

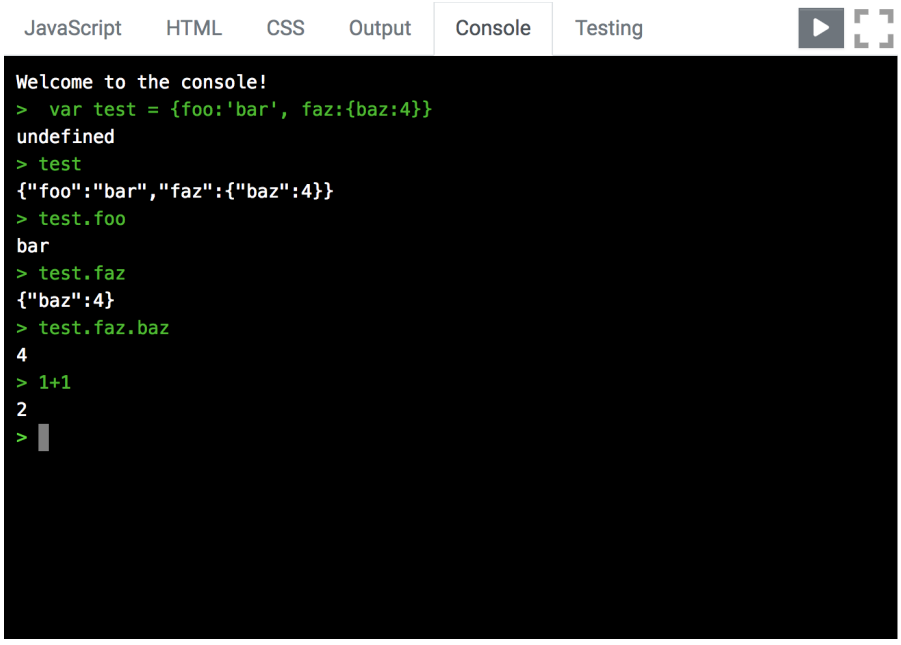


Figure 16: The developer console feature. It allows users to write input queries in some ways similar to the Chrome Development console. Since it works on the scope of the inline frame, it can be used to debug JavaScript code. `Console.log()` statements can for example be written within the JS editor and output can be generated via the console.

17 https://developers.google.com/web/tools/chrome-devtools/console/

Figure 16 shows the developer console tab within the compact editor. The green text represents user input. As can be seen in the picture, dot notation[18] can be used to access nested values. They are then printed to the console in white letters. Red script is used to print out errors happening in the JavaScript code. For the implementation, jq-console[19] is used as a base. It provides methods to write and receive from the console. It also provides a console-like user interface. The communication between inline frame and console is implemented using React.js. The parent editor window receives all user input from the console and sends it to the inline frame. There, it is executed using the `eval()` function. Error handling is implemented here as well. If an error occurs, the error message will be printed to the console in red. If the message received by the inline frame was an object, an additional step to enable readability is performed. If not, the answer will be returned as a normal string.

4.7.2 *Error/Success Logging*

One especially complicated feature to implement was the error logging. When an error occurs inside of the inline frame's context, it is not easy to transfer it to the console for the user to see. The way that makes it possibly is using the `postMessage()`[20] function of the Window API. It safely enables cross origin communication between windows - in this case between the actual window and the inline frame. This way, either a 'success'- or 'failure'-message is dispatched whenever the inline frame renders. This feature allows error messages to then be displayed in the console. Additionally, a UI element appears at the top of the screen to indicate an error and show its detailed stack trace (S. Mader 2018, personal communication, 19 December).

4.7.3 *Full Screen Mode*

The full-screen mode, as can be seen in Figure 17, initially shows every view enabled by the teacher. In this case, from left to right, it would be three input windows for HTML, CSS and JavaScript code. They are followed by console, the UI for the automated tests and finally the output view showing the compiled results. Views can then be disabled using the blue navigation buttons in the top navigation bar as described in Section 3.3.1. The editor windows for coding can be scrolled sideways. The scroll bars can be seen on the bottom of the picture.

18 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_accessors
19 https://github.com/replit-archive/jq-console
20 https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

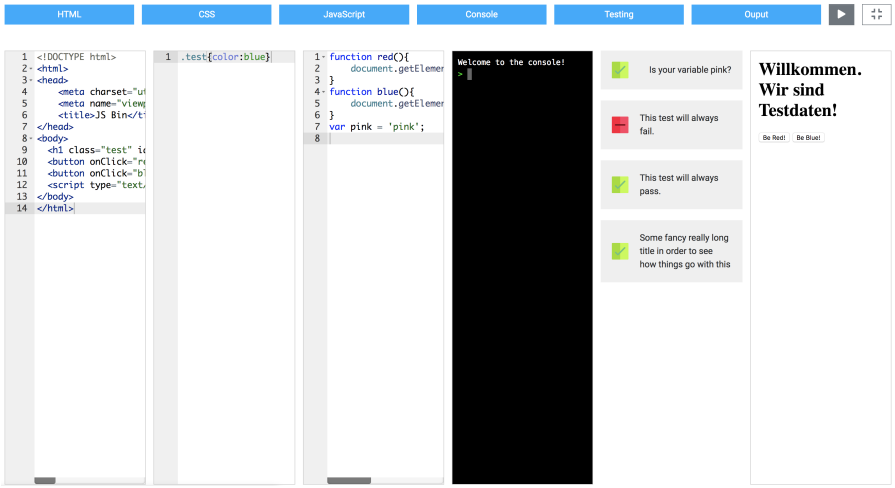


Figure 17: The QuestJs prototype in fullscreen mode.

As can be seen in figure 17 and 18, syntax highlighting has been successfully implemented for the HTML, CSS and JavaScript editor input windows. Figure 18 shows the full screen mode with every column collapsed except JavaScript and Console, exactly as it was conceptualized in Figure 10 from Chapter 3.

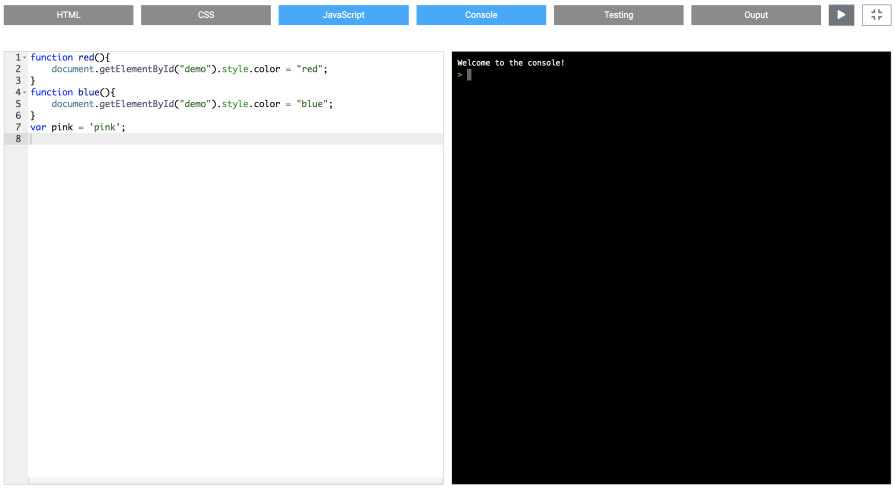


Figure 18: The QuestJs prototype in fullscreen mode, with two views active.

4.7.4 *Compact Mode*

In compact mode, tabs are managed via the top navigation bar. Whenever a tab is clicked, all the other views are hidden via CSS[21]. On Figure 19, the compact mode can be seen. In the top right corner on the image, the implementation of the "Run Code"- and "Go to Full Screen"-Button can be seen. Well known designs have been chosen

21 https://developer.mozilla.org/en-US/docs/Web/CSS/display

as icons for these buttons. Most users are conditioned to instinctively understand what these buttons do.

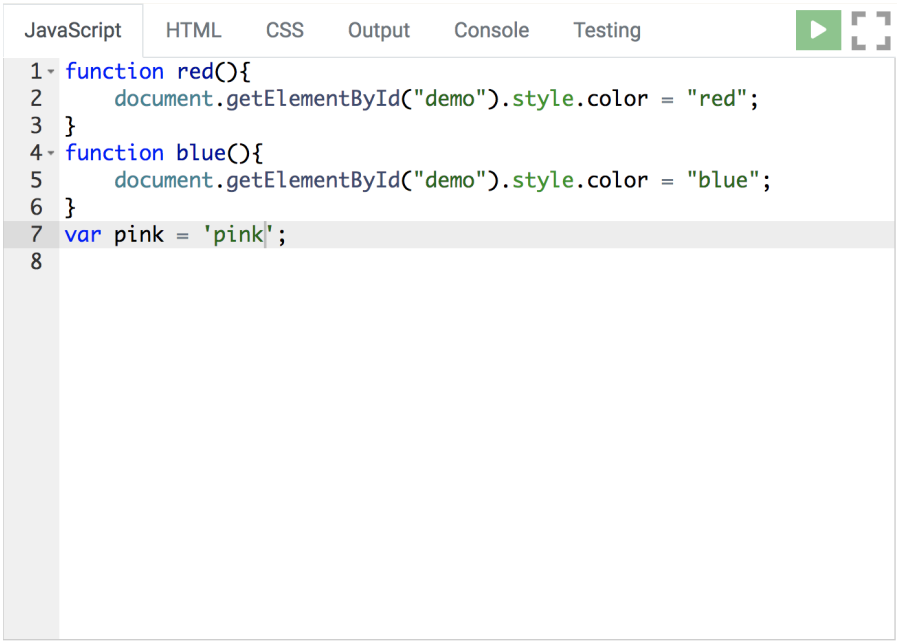


Figure 19: The QuestJs prototype in compact mode, with the JavaScript editor active.

4.7.5 *Library Loading*

Libraries, chosen by the users via the drop-down menu, are loaded using JavaScript Promises[22]. Promises can be used when writing asynchronous code and represent the eventual completion or failure of an asynchronous operation and its value. A promise guarantees that callbacks will never be called before the completion of the current JavaScript event loop[23]. The method `then()` can be used to chain promises in a way that is easier to read as deeply nested callback functions. Libraries that have been selected by users via the drop-down menu are wrapped into promises. Once they all finished loading, they are appended to the DOM. This is guaranteed to happen before the inline frame is rendered. This method guarantees libraries to be successfully loaded into the inline frame. A corresponding code snippet showing the implementation can be seen in Listing 3. In this case, `Promise.all()`[24] is used in order to await the successful load of every library. An Array called libraryScripts, which is assembled by the user's choice of libraries, is converted into promises. The corresponding JavaScript scripts are then appended to the head of the

22 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
23 https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop
24 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

document. The execution only moves on to the statement `.then()`[25], once all scripts are loaded.

Listing 3: This code loads external library scripts before rendering. It waits for each library script chosen by the user to be successfully loaded and only then moves on with the execution.

```
Promise.all(
  libraryScripts.map(
    mappedScript =>
      new Promise((resolve, reject) => {
        const newScript = document.createElement('script');
        newScript.async = true;
        newScript.src = mappedScript;
        // trigger fulfilled state when newScript is ready
        newScript.onload = resolve;
        // trigger rejected state when newScript is not found
        newScript.onerror = reject;
        document.head.appendChild(newScript);
      })
  )
)
  .then(() => {
    this.props.handleScriptsLoaded();
    // success : all promises fulfilled!
  })
  .catch(error => {
    // error : some error occurred in the promises.
    console.log(error.message);
  });
```

4.7.6 *Automated Assertion Tests*

Testing has been implemented using the Chai.js assertion library[26]. As can be seen in Figure 20, a user interface has been implemented, that can show the results of assertion tests passed to the editor by teachers.

25 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then
26 https://www.chaijs.com/

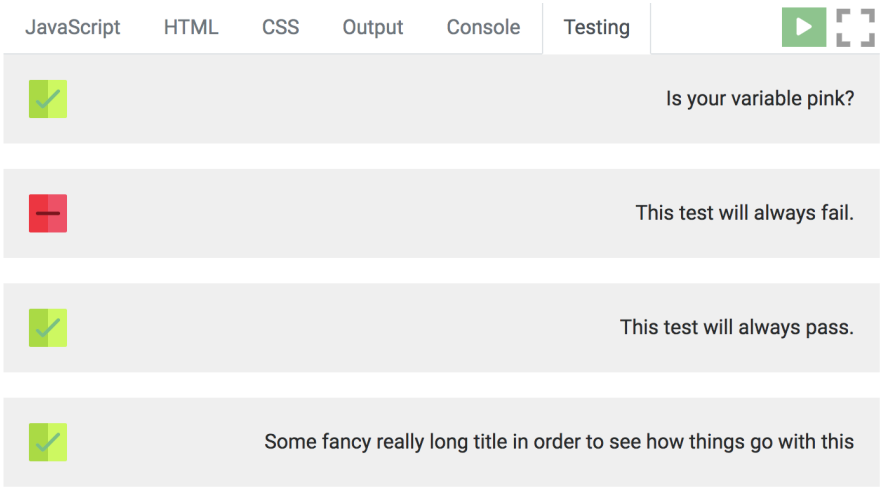


Figure 20: The testing view in compact mode. Green check marks indicate successful test runs. Red boxes indicate failed tests. Tests can be refreshed by running the code.

The corresponding class will accept an array of tests made up from an id, a title, a description and the result, which can be "successful" or "failed", represented as a boolean. The tests themselves are written by educators in a simple string format. The function `eval()`[27] is used to execute the tests on the inline frames context. The tests update whenever the "run-code" button is pressed. This is where the assertion tests run on the code written by users. The UI is then updated accordingly. Whenever a test fails, an error is thrown within the function `eval()`. Such testing errors can then be caught and the testing view can be refreshed respectively.

Concluding this chapter, a prototype has successfully been implemented with most of the features aimed for well completed.

27 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval

5 STUDY AND RESULTS

Previously, a prototype for a JavaScript online learning tool was successfully implemented. Its implementation process was reported on in chapter 4. In this chapter, a method for testing the usability of the QuestJs editor is chosen. Results of a study about its user interface are shown and explained. Insights from the study are analyzed and interpreted.

5.1 METHODOLOGY

In this section, current methods for usability studies are explained and one is chosen for the study. Usability problems that are to be solved are introduced and examples are provided.

5.1.1 *Testing Methods*

In order to successfully test the prototype in regards to its usability and performance, a testing method was chosen. There were several scientific methods to establish the usability of software. The following methods for testing exist:

- **Retrospective Think Aloud (RTA)** studies ask participants to recount their voyage through the prototype after the actual testing is done. This allows users to reflect on what they experienced. They sometimes view a video of their testing session and may comment on it while doing so [83].
- **Concurrent Think Aloud (CTA)** is used while the participants of the study are interacting with the prototype. They would then report on their impressions during the testing session. The idea is to encourage test subjects to talk aloud, so that a maximum amount of valuable feedback can be gained [14, 19].
- **Retrospective Probing (RP)** is used in a way that researchers wait until the testing is done and then ask questions afterwards [14].
- **Concurrent Probing (CP)** is used while the participants of the study are interacting with the prototype. Researchers are then allowed to ask questions in between whenever the participants say or do anything of interest.

- **Moderated Usability Testing** is related to live discussions. Testers and researchers meet in person and discuss the prototype. Anyone may ask and answer questions. This method is recommended during the design phase and is therefore not suited for completed prototypes [24].
- **Focus Groups.** This method of usability testing brings together a group of testers between six and twelve in size. Their task is then to discuss and evaluate the features of the prototype. There may also be a moderator present who probes further. This method is very helpful in assessing users needs, feelings and preferences [46].

Think Aloud studies are the dominant method in usability testing [48, 83]. The CTA method usually results in significantly more problems detected by means of observation only. The RTA method, on the other hand, proves significantly more rewarding in revealing problems that were not observable, but could only be detected by means of articulation [83]. Results [33, 83] indicate, that the CTA method reveals valuable task-oriented feedback. One drawback of the CTA method is its interference with usability metrics such as timings and speed. When users talk aloud, they are not as concentrated and therefore not as quick. The CTA method was chosen to analyze usability of the QuestJs prototype, because of its capability to collect a maximum amount of qualitative feedback.

5.1.2 *Typical Usability Problems*

The question which typical problems can be solved with usability tests still remains. Here are some examples for usual problems, that give hints to what was tested within this study [30]:

1. Users don't know where to start.
2. Users don't know which possibilities are at their disposal.
3. Users look for features they would expect to be present, but they do not exist.
4. Users don't know how to progress along their path at some point.
5. Users are unable to see important UI elements on certain pages of the site.
6. Users are not sure about where exactly they are on the site at a certain moment in time.

These problems played an important role for evaluating the results of the Think Aloud study. In Chapter 3, a concept was shown, that

tries to be as accessible and intuitive as possible. Users were expected to not run into many difficulties due to the design principles that were taken into consideration.

5.2 STUDY DESIGN

The goal of this user study was to evaluate the QuestJs editor and its features. For the study, participants had to solve three coding challenges with increasing difficulty. A total time of 45 minutes was planned for each participant, with a time limit of 30 minutes for coding challenges. This way, there was still time to fill out the questionnaire and receive feedback. The screens of users were recorded using the QuickTime player on a MacBook Pro and voice was recorded using a LG Samsung Galaxy S7 smart phone. Before the study began, participants were quickly briefed about the concept of a Think Aloud study and that they were supposed to talk about their experience. Additionally, the purpose of the study and the QuestJs editor was briefly explained to participants. Participants were allowed to use Search Engines and the internet freely during the study in order to look up syntax and other knowledge that could help them solve the exercises. They were, however, not allowed to use other online coding tools such as CodePen[1] or IDEs in order to solve their tasks.

5.2.1 *Exercise 1*

The first exercise was a simple "Hello world!" style task. It required participants to use the JavaScript coding window. Additionally, the console view was introduced here and used with `console.log()`. The full exercise can be seen in Appendix A. Participants had to write one function and declare one variable to pass the automated tests defined for this exercise. Assertion tests using `Chai.expect()` for the task were designed as follows:

Listing 4: Assertion tests for Exercise One.

```
1 Chai.expect(helloWorld).to.be.a('function');
2 Chai.expect(parameterVariable).to.exist;
3 Chai.expect(parameterVariable).to.equal('Hello World!');
```

5.2.2 *Exercise 2*

The second exercise introduced more HTML manipulation as can be seen in Appendix A. Users needed to use HTML tables[2] to generate a Sudoku board from a given multi-dimensional array and render it

1 https://codepen.io/
2 https://developer.mozilla.org/en-US/docs/Web/HTML/Element/table

to the page. Usually this task required some knowledge of the document[3] interface, since HTML elements needed to be manipulated using JavaScript. Errors were expected to occur here. Users could then naturally use the development console to debug and remove errors. Syntax errors were noted by the static syntax checking included in the QuestJs editor.

5.2.3 *Exercise 3*

Participants needed to solve a coding challenge involving nine buttons arranged in a three-by-three grid for this exercise as can be seen in Appendix A. The buttons were numbered from one to nine. When the middle one of the nine buttons was pressed, the outer buttons were supposed to rotate their numbers clockwise. This exercise required participants to use all of the three editors, since they needed to write HTML, CSS and JavaScript code. In order to solve this puzzle, many approaches could have been chosen. One of the most straightforward solutions would have been to store the numbers on the buttons as an array and insert them into the HTML buttons via innerHTML[4] using the Element API. Errors were expected to occur and to be solved using the static syntax checking and the development console.

5.2.4 *Questionnaire*

A Questionnaire was designed to supplement the Usability study with insights into the feelings of the participants. The Questionnaire was filled out after the exercises had either been completed or the 30 minute time limit had run out. It contained twelve questions regarding usability as can be seen in Table 3 and Appendix B. Another four questions allowed participants to answer with a couple of sentences of free text and give specific feedback in regards to the prototype they were using. A five point Likert scale was used in question one to twelve [3] within this survey. Participants rated with statements ranging from "Strongly Disagree" to "Strongly Agree" as can also be seen in Figure 3. Two of the statements within the survey, number two and 10, were negated in order to make sure participants read everything carefully.

5.3 PARTICIPANTS

For the study, a total number of six participants had been chosen to use the editor. All of them had been specifically selected for the study

3 https://developer.mozilla.org/en-US/docs/Web/API/Document
4 https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML

due to their pre-existing knowledge about JavaScript and web development in general. All of them are experienced developers. Users without prior knowledge would not have been able to solve the coding challenges or use core features of the editor, like for example the developer console. Having had advanced users also provided better feedback, since they had a direct comparison to other IDE's, were used to development tools and could provide valuable feedback in regards to usability in terms of real software development.

5.4 MEASUREMENT CRITERIA

For measuring the success of the study, a set of criteria was defined. According to research by Hornbæk et al., certain measurement criteria for usability are especially effective and used a lot in evaluating usability [35]. Among them are the following:

1. **Effectiveness**: Number of tasks completed in the space of 30 minutes.
2. **Errors**: Number of bugs encountered. These can be conventional errors or usability problems as explained in Section 5.1.2.
3. **Variety and Learning**: Number of features discovered and actively used.

Metrics for measuring usability in software products have also been established by the international standard organization in 2004 [39, 44] and have been renewed in 2016 [38, 40]. These metric had been considered, but were proving to go beyond the scope of this usability study. Preliminary tests had shown, that Exercise One could be solved in under five minutes. Exercise Two was expected to take about 10 minutes, with another 15 minutes left for Exercise Three. The results of the questionnaire, filled out by participants after the coding challenges were completed, were used to gather qualitative feedback regarding usability of the prototype. Statements from participants of the Think Aloud study were gathered and presented in compact form.

5.5 EXECUTION

The study was conducted at the facilities of the Bayerischer Rundfunk in Munich in December 2018. Six software developers aged 24 to 32 took part and successfully completed 30 minutes of challenges and 15 minutes of combined preparation and questionnaire. Their Audio and Screen were captured. Beforehand, each of the participants had received a short explanation regarding the goals of the study and what they had to do, that is use the prototype to solve the coding challenges. The study was performed on a MacBook Pro system, with two large 27-inch iMac monitors attached to it.

5.6 RESULTS

This section contains the results of the user study. In detail, it entails usage statistics, results of the multiple choice questionnaire and general feedback from the participants.

5.6.1 *Usage Statistics*

As can be seen in Table 2, Exercise One has been solved on average after 4 minutes and 13 seconds, as expected. The large mode has been discovered after, on average, 2 minutes and 50 seconds. One user has discovered it after only 13 seconds. This user, S2, also discovered the Run-Code button after 40 seconds and the Large Mode Tabs after 1 minute 20 seconds. They discovered the console very quickly after just 30 seconds. They employed the strategy of first going through all the features of the editor and then starting the exercises. Participant S6 never discovered the console, since they did not use it to solve exercises at all. They employed the strategy of running the code and then analyzing the output window only.

Table 2: Timings of each participants of the study in minutes and seconds. It depicts when they completed tasks or discovered parts of the user interface for the fist time. It also shows the average time for each event.

ACHIEVEMENT:	Task 1 Solved	Large Mode Discovered	Run Code Button Discovered	Error Message Discovered	Compact Mode Tabs Discovered	Large Mode Tabs Discovered	Console Discovered
S1	3:58	2:03	2:40	14:30	1:23	2:30	32:40
S2	3:42	0:13	0:40	26:30	0:30	1:30	0:30
S3	3:30	5:15	2:45	2:50	0:30	6:11	28:25
S4	4:20	3:10	3:01	24:01	2:55	12:13	14:00
S5	6:30	4:02	5:35	5:40	3:12	1:20	16:10
S6	3:20	2:20	3:05	9:30	1:30	3:11	-
Averages	4:13	2:50	2:57	13:50	1:40	4:29	18:21

5.6.2 *Questionnaire Results 1 – Likert Scale*

As can be seen in the results in Table 3 and Appendix B, most users had a positive user experience with the editor. Question number three and ten had been reversed. The reversal can clearly be seen in the pattern of answers. Users disagreed on question 1,2,5,7 and 12 with 2 users disagreeing on question 12. All other answers have been either neutral or positive.

Table 3: Questionnaire results. These are statements from the questionnaire and how participants answered them on a Likert scale.

STATEMENTS:	I quickly understood the user interface.	I found the elements i was looking for.	Switching between editors was hard.	The large editor helped me gain an overview.	Interaction with the console feels natural.	Error messages helped me find errors.	I find the editor suitable for beginners.	Beginners have an easier time getting into coding with this editor.	The omitted file management reduces complexity.	The missing setup process makes starting out more difficult.	I would have had an easier time getting into JavaScript, if I had had an editor with reduced complexity.	I would recommend this editor for first semester students.
Strongly Disagree			2							4		
Disagree	1	1	3		1		1			2	1	2
Neutral			1	2	2	1	1	2	1		2	1
Agree	4	4		2	2	2	2	2	3		3	3
Strongly Agree	1	1		2	1	3	2	2	2			

5.6.3 *Questionnaire Results 2 – Text*

This section explores the free text answers collected within the Questionnaire as can also be seen in original form in Appendix C.

Regarding question number 13, "Which features did you find especially useful?", users have positively mentioned:

- Using the Large Mode on Big Screens ($n = 2$)
- Error Messaging ($n = 2$)
- Multi-Cursor Support ($n = 1$)
- Indentation Handling ($n = 1$)
- Static Error Handling ($n = 1$)
- The Preview Feature ($n = 1$)

- Lack of file management ($n = 1$)
- The UI is not cluttered ($n = 1$)

For question number 14, "Please note this editor is built for education. Do you think any other features could have been useful in this context? Please explain.", users have positively mentioned:

- Auto-complete ($n = 2$)
- See generated HTML source code ($n = 2$)
- Clear Console feature ($n = 1$)
- Step-by-Step Walk-through ($n = 1$)

Regarding question number 15, "Do you have any thoughts to improve upon the QuestJs editor? Please explain.", user have written:

- Clear Console Feature. ($n = 2$)
- Auto-complete or something like Intellisense[5]. ($n = 2$)
- Improve Large Mode Column Toggle. ($n = 2$)
- Add HTML Source View for generated Code. ($n = 2$)
- Rework Pop-up errors into something better. ($n = 1$)
- Better Warnings within static code checking (e.g. '===' instead of '=='). ($n = 1$)

5.6.4 *General Feedback*

Since the participants of the study were seasoned software developers, they were qualified to commentate on the usability and possible measures to improve the editor. During the Think Aloud study, many suggestions and ideas could be collected. The most interesting comments have been the following:

- "I would expect visual feedback when I click the Run-Code button."
- "I would like to be able to clear the console history."
- "The error message immediately told me what was wrong. That was nice!"
- "This error logging should not move the UI around so much."
- "I am used to a multi column layout from other editors."

5 https://code.visualstudio.com/docs/editor/intellisense

- "I miss plugins from my IDE."
- "Drag and Drop would be cool to re-size the columns of the large editor."
- "I enjoy using arrow functions and ES6 within this editor."
- "I do not like the magical assembly of the code in the output window. I like to know what is going on."
- "Dark mode would be nice."

The original German comments can be found in Appendix C.

5.7 DISCUSSION

All participants could solve Exercise One in approximately five minutes as anticipated, but became highly delayed on Exercise Two until their 30 minutes had run out. One possible conclusion to draw from this is that Exercise Two was too difficult to be completed in 10 minutes as was assumed possible for the study. Another problem that became apparent during the study was that some of the participating developers were used to their own development environments and plugins. For example: Hitting `CTRL-S` to save, format and autocompletion for the written code was missed by participants. Having to do everything "manually" took some getting used to for some participants and considerably slowed them down. Additional slow-down was generated by nature of the Think Aloud study, as expected. Two participants used methods to generate most of their HTML code using JavaScript with for example the function `document.createElement()`[6]. Some of the participants then complained about a missing source code view. This problem could be solved in the future with a button in the output window that lets users toggle between normal output and source code view. One user suggested, as mentioned in Section 5.6.3, to have an interactive tutorial or feature tour. This could be implemented in the future using for example Joyride[7] or other similar implementations. Such a feature could decrease the time it takes users to gain a grasp of the system and its features.

Some parts of the editor, like the compact mode navigation tabs, were easy to understand and were used by participants intuitively as can be seen by the low time to first use in Table 2. The large mode itself was also quickly discovered by most users and used extensively by all except one user, who preferred to use the compact mode. On the large screens, where the study was conducted on, the large mode of the prototype made a positive difference and was welcomed by users. The large mode navigational tabs were harder to spot and not

6 https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement
7 https://zurb.com/playground/jquery-joyride-feature-tour-plugin

used as intuitively as the timings of Table 2 clearly indicate. This may have been due to missing visual cues. One user suggested a drop shadow or something similar that would identify the navigational elements as clickable buttons. This way, users could spot them more comfortably in the future and make use of the improved usability provided by hiding the a couple of views.

One of the most difficult parts to understand and access for users was the developer console. Participants of the study struggled to see that it could be used not only to output data via for example the function `console.log()` but could also be used to execute statements directly in the inline frame's context. Such functionality was not required for the exercises. Asking users revealed that some of them were not used to having a console that works in both ways, as other online editors did not have this feature. This problem could also be solved by introducing a feature tour as stated earlier. On the other hand, the console could initialize with a short greeting and very basic set of displayed functionality. As a last resort, a "help" command could be implemented to allow users to gain an overview or documentation.

Concluding this chapter, it can be said, that the QuestJs editor has proven to be largely usable by experienced programmers. Users were able to experience a majority of features without delay. The fact that Exercise Two was only completed by one participant is attributed to differing individual skills and predictions about timings that were too optimistic. Most participants of the study recommend this tool for students starting out with web development. With advantages like the missing setup process, users are able to commence coding quickly and evade some of the early frustrations that usually accompany users when starting in software development.

6 CONCLUSION AND PERSPECTIVES

Previously, a prototype for a JavaScript online learning tool was successfully implemented. A usability study was conducted in order to assess its usability and its results were reported on and analyzed in chapter 5. In this chapter, possible future improvements and directions are shown and the project is concluded with a summary.

6.1 PERSPECTIVES

The QuestJs editor can be improved upon in the future in a number of ways. Participants of the user study already provided various suggestions for enhancement. From a development viewpoint, syntax auto-completion, a HTML source view and other features could be introduced to improve the system. From a user interface point of view, features like drag-and-drop could be implemented in order to further increase usability by letting users rearrange editor windows as they please. Rearranging windows is a feature that was also requested by participants. Looking at the broader picture, other programming languages could be integrated into the editor to increase its applicability outside of the web development context. A client-server architecture would be required to allow compilation of other programming languages on servers. This would slow down the process, but increase its capability. A Back-end could be used to allow real time monitoring of the student's progress by teachers. Teachers could for example monitor the amount of tests that return positive results compared to the total number of students. This way, teachers could decide when to continue the lesson based on how many students have successfully completed exercises.

Additionally, the editor could be made available for the general public. It could be used as a tool for web development novices to start out with coding. Training exercises could be made available. Looking even further ahead, motivated content creators could be allowed to conceive open source exercises for other users as a form of user generated content.

6.2 SUMMARY

This project set out with the goal of making it easier for students to start with web development by solving some of the problems that arise when doing so. Challenges were identified, such as frustrations when starting with coding and troubles with the setup of traditional

integrated development environments. Learning a programming language and finding the extensive amounts of time needed to start with programming is also demanding.

With reduced numbers of staff, teachers have to find new ways to educate larger groups of students outside of traditional settings. A possible solution is the use of LMS and tools like the prototype developed in this project, so that a degree of automation is introduced. In order to reach the aim of having students start coding earlier, state of the art educational techniques were analyzed and evaluated. Teachers are supported in conducting exercises with larger groups, if they do not have to worry about many of the previously mentioned issues which their students could face.

A concept for a browser-based JavaScript learning tool was realized by analyzing existing online editors and expanding on their strengths. This concept has a number of advantages when compared to traditional integrated development environments. Since the prototype runs in the browser only, users do not have to set up their own local programs and tools, which makes it beginner-friendly and saves time. The editor runs client-side only, which enables very quick user interface feedback. Users can start coding instantly.

A series of features were developed, that expand upon existing online tools. A developer console was introduced to allow live-debugging in the browser alongside automated testing. Teachers are now able to create coding challenges and deploy automated assertion tests. Users can then receive instant feedback on their progress within the current task by looking at the results of the tests. Various features, like the included library support, turn learning into a positive experience. Many were positively mentioned in the usability study. The two modes of usage, compact or full-screen, make it possible to use the editor on a small device or on a big screen, depending on the need of the user. The custom error message handling lets users see mistakes in a pop-up window and sets this project apart from established tools. Using this feature, errors can be resolved faster. Another feature lets teachers write assertion tests, which can then provide valuable feedback for students. This way, larger groups of students can be taught employing teaching methods such as flipped classroom [81], active learning [43] and scaffolding [63], which are traditionally harder to implement with bigger groups.

After establishing a Think Aloud study as the preferred testing method, a usability study was conducted in order to evaluate the prototype's performance. Results of the study indicate a positive reception by participants. Five out of six users reported to have quickly understood the user interface. The usability study has shown that the editor is usable by experienced developers. Participants of the study recommend this tool for students starting out with web development. With advantages like omission of the setup process, users are able to

commence coding quickly and evade some of the frustrations that often appear in the early stages of starting with software development.

In conclusion, this project shows that technologies such as the QuestJs prototype appear to be able to solve some of the aforementioned problems and create favorable conditions for setting out with web development.

A

USER STUDY – EXERCISES

QuestJs Usability Study

1) Beginner Level (JS code editor, console, compilation)

Perform the following tasks to complete this challenge:

1) Declare a variable called *parameterVariable* and give give it the string “Hello World!”.
2) Write a greeting function called “helloWorld” in the editor.
    a) Use console.log to print *parameterVariable.*

2) Sudoku Generator :

Given a multi-dimensional Array, generate a Sudoku-style board. Each Array within the Array represents one tile of the sudoku with its 9 numbers.

E.g. [5,2,8,9,5,3,1,2,6] would look like this:

5	2	8
9	5	3
1	2	6

[[5,2,8,9,5,3,1,2,6] , [7,2,8,9,5,3,1,2,1] , [9,2,8,9,5,3,1,2,1] , [3,2,8,9,5,3,1,2,3] , [1,2,8,9,5,3,1,2,3] , [2,2,8,9,5,3,1,2,5] , [8,2,8,9,5,3,1,2,9] , [4,2,8,9,5,3,1,2,8] , [6,2,8,9,5,3,1,2,7]]

- Render this Array to a Sudoku Puzzle game board.
- Use little to no CSS.
- Use a HTML table with the unique id “sudokuMap”.
- Give your <tr> elements nine id’s “row1” to “row9”

8	2	5	4	7	1	3	9	6
1	9	4	3	2	6	5	7	8
3	7	6	9	8	5	2	4	1
5	1	9	7	4	3	8	6	2
6	3	2	5	9	8	4	1	7
4	8	7	6	1	2	9	3	5
2	6	3	1	5	9	7	8	4
9	4	8	2	6	7	1	5	3
7	5	1	8	3	4	6	2	9

3) Advanced Level (3 editor tabs, console)

We want to create nine buttons enclosed in a *div*, laid out so they form a 3x3 grid. Each button has a distinct label from 1 to 9, and the labels on the outer buttons must rotate in the *clockwise* direction each time we click the middle button.

Complete the code in the editor so that it satisfies the following criteria:

- *Initial State.* The initial layout looks like this:

1	2	3
4	5	6
7	8	9

Element IDs. Each element in the document must have an `id`, specified below:

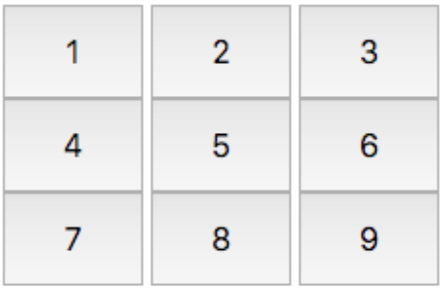
- The button container *div*'s `id` must be `btns`.
- The initial `innerHTML` labels must have the following button `id`s:

innerHTML	id
1	btn1
2	btn2
3	btn3
4	btn4
5	btn5
6	btn6
7	btn7
8	btn8
9	btn9

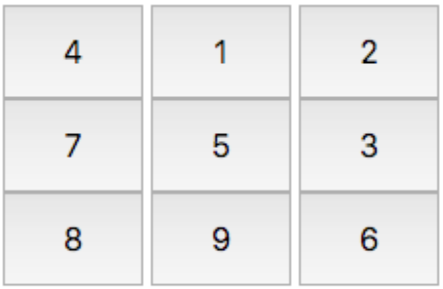
- *Styling.* The document's elements must have the following styles:
    - The `width` of `btns` is 75%, relative to the document body's width.
    - Each button (i.e., `btn1` through `btn9`) satisfies the following:
        - The `width` is 30% , relative to its container width.
        - The `height` is `48px`.
        - The `font-size` is `24px`.
- *Behavior.* Each time `btn5` is clicked, the `innerHTML` text on the grid's outer buttons (i.e., `bt1`, `btn2`, `btn3`, `btn4`, `btn6`, `btn7`, `btn8`, `btn9`) must rotate in the *clockwise* direction. *Do not* update the button `id`'s.

Explanation

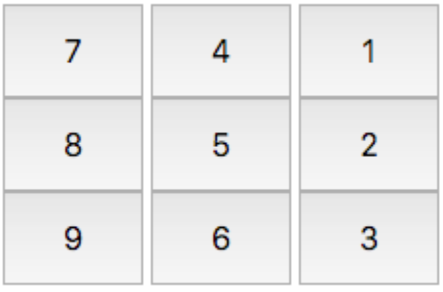
Initially, the buttons look like this:



After clicking `btn5` 1 time, they look like this:



After clicking `btn5` 1 more time (for a total of 2 clicks), they look like this:



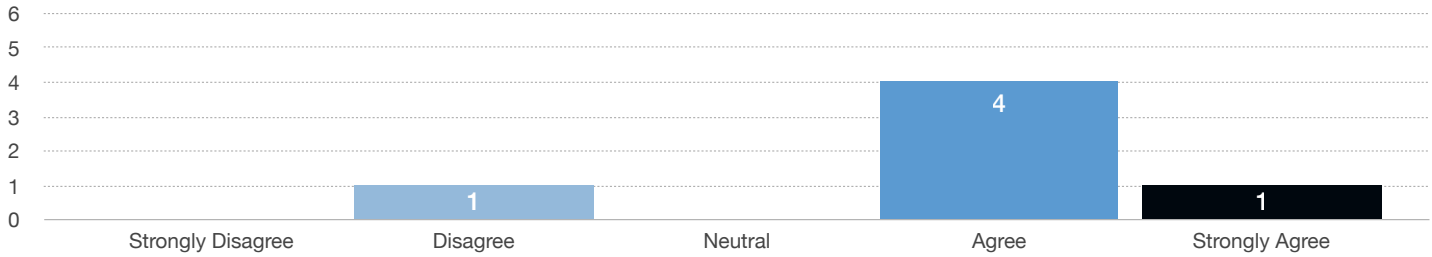
B

USER STUDY – QUESTIONNAIRE RESULTS

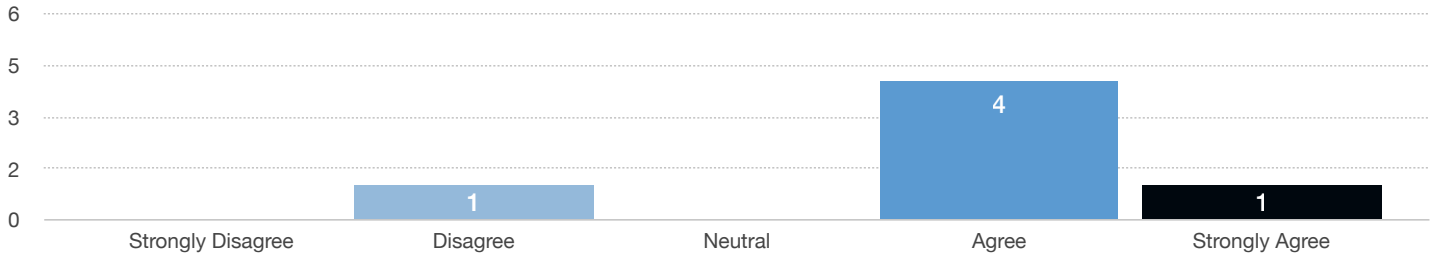
QuestJs Usability Study

QUESTION NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Strongly Disagree			2							4		
Disagree	1	1	3		1		1			2	1	2
Neutral			1	2	2	1	1	2	1		2	1
Agree	4	4		2	2	2	2	2	3		3	3
Strongly Agree	1	1		2	1	3	2	2	2			

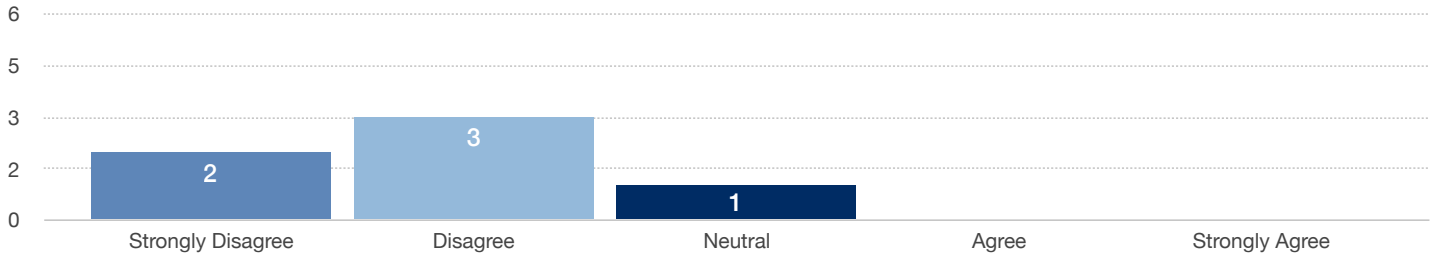
1) I quickly understood the User Interface.



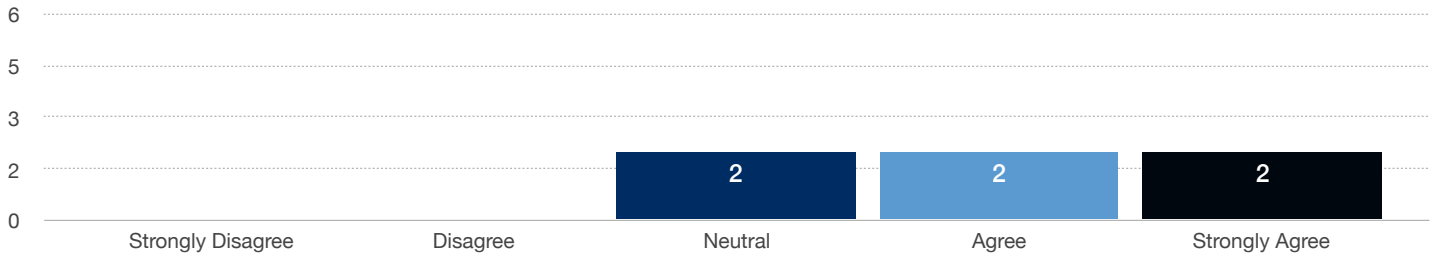
2) I instantly found the elements I was looking for.



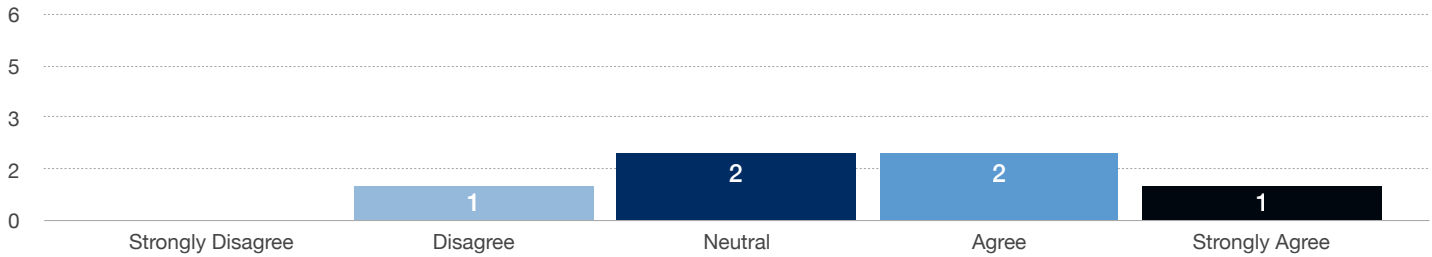
3) Switching between code editors was difficult.



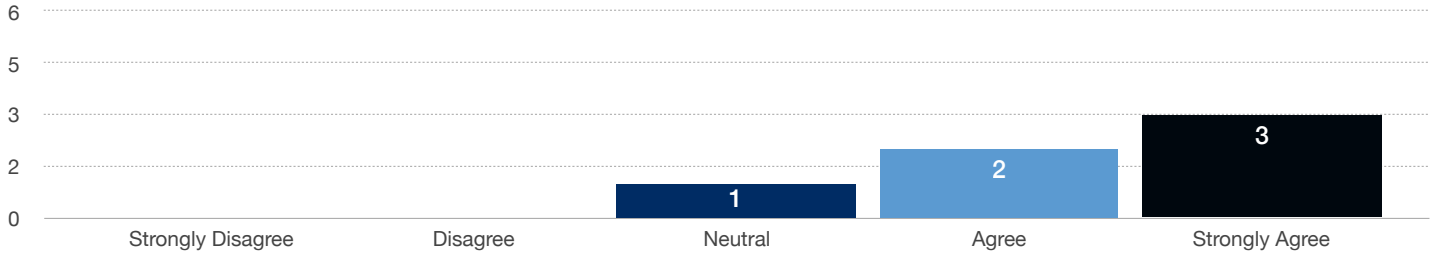
4) The enlarged editor helped me in gaining a quick overview.



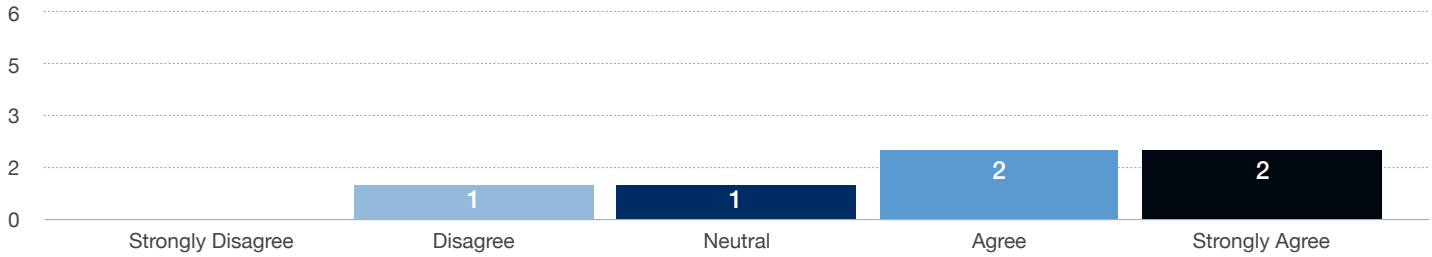
5) Interaction with the console feels natural. It feels similar to the Chrome development console.



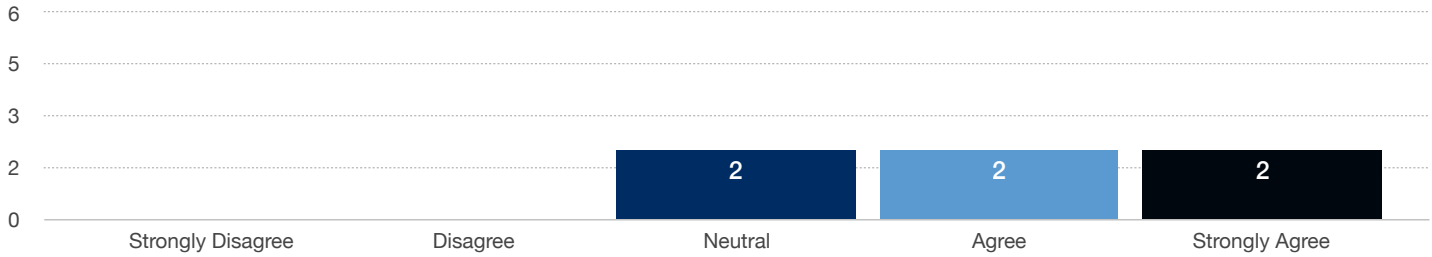
6) Error messages helped me identify errors within code.



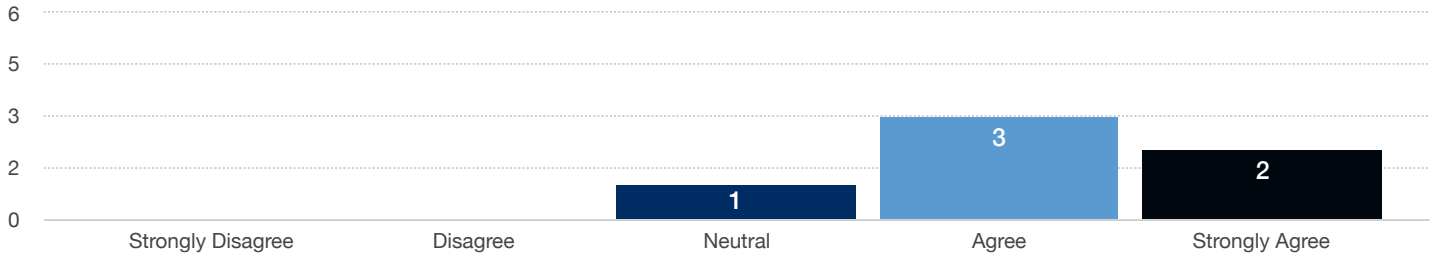
7) I find this editor suitable for beginners.



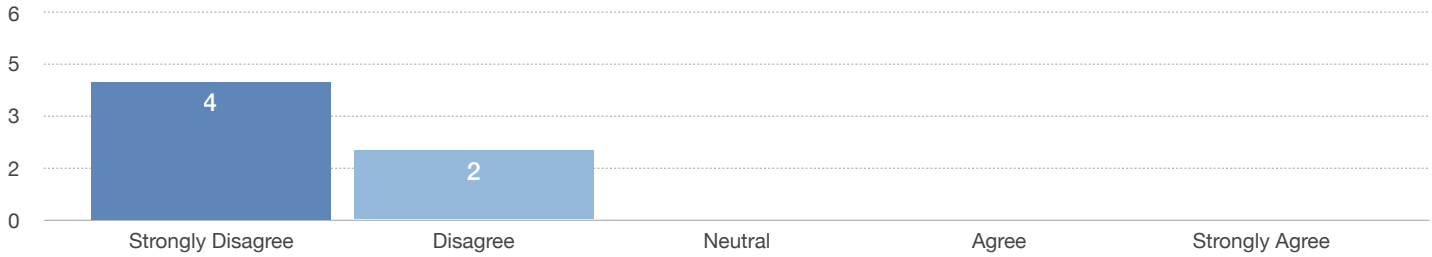
8) Beginners have an easier time getting into coding with this editor.



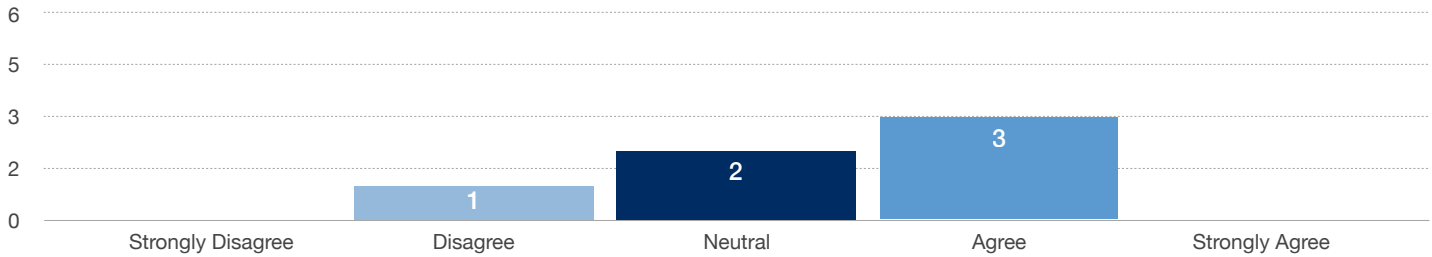
9) The omitted file management reduces complexity.



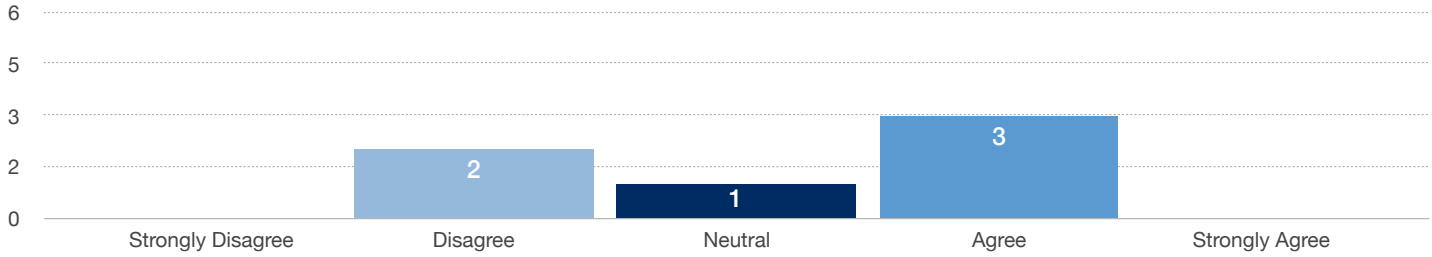
10) The missing setup process of traditional IDE's makes starting out more difficult.



11) I would have had an easier time getting into JavaScript, if I had had an editor with reduced complexity.



12) Compared to traditional IDE's, I would recommend this editor for first semester students getting into web development.



C

USER STUDY – COMMENTS IN THE ORIGINAL LANGUAGE GERMAN

Comments during the Study (German Original)

- „Ich würde visuelles Feedback erwarten wenn ich im kompakten Modus den[RUN CODE] Button drücke“
- „Die Columns sind nicht gutangeordnet.“
- „Wünschenswert wäre Ausblendbarkeit [der Columns im Large Mode] … Ach das gibt es ja schon, war aber schlecht erkennbar“
- „Es wäre schön, wenn man die Konsole leeren könnte bzw. es eine Historie gäbe“
- „Ich verwende den large Mode, weil ich gelernt habe, dass es so übersichtlicher ist“
- „Praktisch wäre ein Shortcut um zwischen Columns hin und her zu Switchen]“
- “Session storage wäre gut”
- “Error Nachrichten sorgen für UI Fehler. Das ist unangenehm. Alles geht runter.”
- “Warum bleibt dieser [UI] Fehler bestehen?”
- “Dieser Fehler bezieht sich auf nicht existenten Code. Das muss ein Fehler sein.”
- “Fenster via Drag and Drop anordnen wäre cool. Andere können das nicht.”
- „Vielleicht einen box Schadow für large Mode Navigation Buttons verwenden?“
- 33:22 “Ich habe erwartet, dass ich nicht in der Konsole schreiben kann. Ich kannte von anderen Editoren nur reine Output Konsolen.”
- “Die Fehler Nachricht hat mir direct gezeigt was schief life. Das war sehr hilfreich.”
- “Ich wünschte ich hätte ein Fenster um den generierten HTML Code direkt zu sehen."
- “Logging für die Konsole funktioniert.”
- “Verwirrende Aufgabenstellung.”
- “Tests haben schönes visuelles Feedback.”
- „Konsole funktioniert wie erwartet.“
- „Switch zwischen kompakt und large ist intuitiv.“
- “Diese Fehlermeldung hilft mir sehr!”
- “Das war verwirrend, da der Bug ja weg war.”
- „Das Error Logging sollte nicht die ganze UI verschieben. Vielleicht statisches Fenster verwenden.“
- „Auf dem großen Display ist der Large Mode sehr angenehm.“
- „Auto-complete wäre cool!“
- „Ein Console-Clear Button wäre gut!“
- “Der Side-Scroll-Bug im Code sollte gefixt werden.”
- “Ich bin an solche Views gewöhnt von anderen online Editors.”
- „Ich finde es toll, dass Arrow Funktions und ES6 gehen!“
- „Ich finde es toll, dass mir die Tests direkt Feedback geben!“
- „Ich wünsche mir Auto-Completion.“
- „Ich vermisse Prettier aus meiner IDE“
- „Inline Static-Error-Checking ist hilfreich!“
- „Drag and Drop oder größer ziehen im large Mode wäre toll!“
- „Ich finde compact mode gut um nur in JavaScript zu arbeiten, wenn ich die anderen windows nicht brauche.“
- „Error Messages sind sinnvoll!“

- „Ich würde gerne sehen welches HTML erzeugt wird. Ein HTMLQuelltext Fenster wäre gut zum Debuggen!“
- „Im Output Fenster könnte man einen Switch Button einbauen. Um zwischen HTML Quelltext Code und normalem Output zu wechseln.“
- „Ich verwende am liebsten die Chrome Dev Console.“
- „Ich finde es verwirrend, dass ich nicht weiß, wie der Code zusammengefügt wird.“

From the Questionnaire :

13. Which Features did you find especially useful?
- “Fehler Nachrichten.”
- “Large Mode.”
- “No clutter.”
- “Fehler Benachrichtigungen waren hilfreich.”
- “Es war gut sich nicht mit dem Dateien Management herumärgern zu müssen.”
- “Das hat mir besonders gefallen: Multi-Cursor, Indentation, code-completion”
- “Seite-an-Seite Anzeige der verschiedenen Datei Typen. E.g. HTML, CSS, JS”
- “Das Vorschau Feature. Also der Output.”

14. Which other features could have been good in an educational context?
- “Ein Button um die Konsole zu leeren.”
- “Auto complete.”
- “Auto complete / Intellisense”
- “Ein Schritt für Schritt Tutorial durch den Editor mit Dokumentation wäre nett.”
- “Die Möglichkeit generierten Code zu sehen”
- “Man könnte die Coding Challenges innerhalb des Editors anzeigen.
- “Ein HTML Output Fenster. So können Studenten den generierten Code sehen.

15. How to improve?
- “Man sollte den Large Mode Naviations Balken verbessern. Man brauch einen Hinweis, dass die Nav Buttons clickbar sind.”
- “Auto complete / Intellisense.”
- “Vielleicht automatische Zeilenumbrüche. Warnungen wie “==” instead of “===”.”
- “Error logging in der Konsole.”
- “Pop up errors sind nervig.“
- “Console Autocomplete”
- “Prettier plugin”
- “HTML output Fenster”

16. Other
- “Dark Mode!”

GLOSSARY

Basic Editor Features Every online JavaScript editor in the tested set offers the feature to write HTML, CSS and JavaScript code and show the result in a output window. Each editor has the feature of sharing these projects via URL sharing. Therefore this set of features is called "basic editor features". xv, 12–15, 17, 18, 25

Document Object Model The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. This is an overview of DOM-related materials here at W3C and around the web [37]. 8

Full-Stack language A Full stack programming language work with both the front and back end of a website. As an example, JavaScript cam be used for the Front-end with the React framework or as a Back-end server language with Node.js. 9

HAML HAML (HTML abstraction markup language) is based on one primary principle: markup look good. It is not just good looks though; HAML accelerates and simplifies template creation. It is aged though and not being used a lot anymore.. 14, 15

HTML HTML, also known as hypertext markup language, is the standard Markup language for websites. Together with CSS and JavaScript, it is one of the three cornerstones of the world wide web. "To publish information for global distribution, one needs a universally understood language, a kind of publishing mother tongue that all computers may potentially understand. The publishing language used by the World Wide Web is HTML (from HyperText Markup Language)" -Raggett et al. [31, 65]. 10

LMS A learning management system is a software application for the administration, documentation, tracking, reporting and delivery of educational courses or training programs or learning and development programs. The learning management system concept emerged directly from e-Learning. 19–22

QuestJs The title of the JavaScript editor developed in this thesis. The main reason this editor is developed, is teaching users

JavaScript and web development in general. 25, 29–31, 35–37, 40, 42, 45, 51–53, 61

STEM Acronym for Science, Technology, Engineering, Mathematics. vii, viii

UI Acronym for User Interface. 3

White-box Testing White-box testing is a method of testing software. It tests the internal structure of an application as opposed to its functionality. White-box testing can be applied at the unit, integration and system levels of the software testing process. 26

Zone of Proximal Development (ZPD) The zone of proximal development, often abbreviated as ZPD, is the difference between what a learner can do without help, and what they can't do. 6

BIBLIOGRAPHY

[1] Lakmal Abeysekera and Phillip Dawson. "Motivation and cognitive load in the flipped classroom: definition, rationale and a call for research." In: *Higher Education Research & Development* 34.1 (2015), pp. 1–14.

[2] Jean-François Abramatic, Roberto Di Cosmo, and Stefano Zacchiroli. "Building the universal archive of source code." In: *Communications of the ACM* 61.10 (2018), pp. 29–31.

[3] I Elaine Allen and Christopher A Seaman. "Likert scales and data analyses." In: *Quality progress* 40.7 (2007), pp. 64–65.

[4] Philip G Altbach, Liz Reisberg, and Laura E Rumbley. *Trends in global higher education: Tracking an academic revolution*. 2009.

[5] Brenda Alvarez. "Flipping the classroom: Homework in class, lessons at home." In: *The Education Digest* 77.8 (2012), p. 18.

[6] Michael Barnett, William Harwood, Thomas Keating, and Julie Saam. "Using emerging technologies to help bridge the gap between university theory and classroom practice: Challenges and successes." In: *School Science and Mathematics* 102.6 (2002), pp. 299–313.

[7] John C Bean. *Engaging ideas: The professor's guide to integrating writing, critical thinking, and active learning in the classroom*. John Wiley & Sons, 2011.

[8] Brian Beatty and Connie Ulasewicz. "Faculty perspectives on moving from Blackboard to the Moodle learning management system." In: *TechTrends* 50.4 (2006), pp. 36–45.

[9] Kelly Bedard and Peter Kuhn. "Where class size really matters: Class size and student ratings of instructor effectiveness." In: *Economics of Education Review* 27.3 (2008), pp. 253–265.

[10] Penny L Beed, E Marie Hawkins, and Cathy M Roller. "Moving learners toward independence: The power of scaffolded instruction." In: *The Reading Teacher* 44.9 (1991), pp. 648–655.

[11] Yoany Beldarrain. "Distance education trends: Integrating new technologies to foster student interaction and collaboration." In: *Distance education* 27.2 (2006), pp. 139–153.

[12] Jonathan Bergmann and Aaron Sams. *Flip your classroom: Reach every student in every class every day*. International society for technology in education, 2012.

[13] L Berk and Adam Winsler. "Vygotsky: His life and works" and" Vygotsky's approach to development." In: *Scaffolding children's learning: Vygotsky and early childhood learning* (1995), pp. 25–34.

[14] Julie H Birns, Kristen A Joffre, Jonathan F Leclerc, and Christine Andrews Paulsen. "Getting the Whole Picture: Collecting Usability Data Using Two Methods—-Concurrent Think Aloud and Retrospective Probing." In: *Proceedings of UPA Conference*. Citeseer. 2002, pp. 8–12.

[15] Benajmin S Bloom, DR Krathwohl, and BB Masia. *Taxonomy of educational objectives: The classification of educational goals. New York, NY: David McKay Company*. 1956.

[16] Dave Bremer and Reuben Bryant. "A Comparison of two learning management Systems: Moodle vs Blackboard." In: *Proceedings of the 18th Annual Conference of the National Advisory Committee on Computing Qualifications*. 2005, pp. 135–139.

[17] Michelle Collay, Diane Dunlap, Walter Enloe, and George W Gagnon Jr. *Learning circles: Creating conditions for professional development*. Corwin Press, 1998.

[18] Jody Condit Fagan. "Server-side includes made simple." In: *The electronic library* 20.5 (2002), pp. 382–389.

[19] Lynne Cooke. "Assessing concurrent think-aloud protocol as a usability test method: A technical communication approach." In: *IEEE Transactions on Professional Communication* 53.3 (2010), pp. 202–215.

[20] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[21] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

[22] Ton De Jong. "Cognitive load theory, educational research, and instructional design: Some food for thought." In: *Instructional science* 38.2 (2010), pp. 105–134.

[23] Martin Dougiamas and Peter Taylor. "Moodle: Using Learning Communities to Create an Open Source Course Management System." In: *Proceedings of EdMedia + Innovate Learning 2003*. Ed. by David Lassner and Carmel McNaught. Honolulu, Hawaii, USA: Association for the Advancement of Computing in Education (AACE), 2003, pp. 171–178. URL: https://www.learntechlib.org/p/13739.

[24] Joseph S Dumas and Beth A Loring. *Moderating usability tests: Principles and practices for interacting*. Elsevier, 2008.

[25] Ryann K Ellis. "Field guide to learning management systems." In: *ASTD Learning Circuits* (2009), pp. 1–8.

[26] Artemij Fedosejev. *React. js Essentials*. Packt Publishing Ltd, 2015.

[27] David Flanagan. *JavaScript: The definitive guide: Activate your web pages*. " O'Reilly Media, Inc.", 2011.

[28] Gerald Futschek. "Extreme didactic reduction in computational thinking education." In: *X World Conference on Computers in Education*. 2013, pp. 1–6.

[29] Jesse James Garrett. *Elements of user experience, the: user-centered design for the web and beyond*. Pearson Education, 2010.

[30] Uni Gießen. *Guide for User Studies*. https://www.inst.uni-giessen.de/usability/downloads/Leitfaden.pdf. Accessed: 2018-12-29.

[31] Ian S Graham. *The HTML sourcebook*. John Wiley & Sons, Inc., 1995.

[32] Mari Luz Guenaga, Dominique Burger, and Javier Oliver. "Accessibility for e-learning environments." In: *International Conference on Computers for Handicapped Persons*. Springer. 2004, pp. 157–163.

[33] Maaike J Van den Haak and Menno DT De Jong. "Exploring two methods of usability testing: concurrent versus retrospective think-aloud protocols." In: *Professional Communication Conference, 2003. IPCC 2003. Proceedings. IEEE International*. IEEE. 2003, 3–pp.

[34] Jesse M. Heines, Jeff L. Popyack, Briana Morrison, Kate Lockwood, and Doug Baldwin. "Panel on Flipped Classrooms." In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. SIGCSE '15. New York, NY, USA: ACM, 2015, pp. 174–175.

[35] Kasper Hornbæk. "Current practice in measuring usability: Challenges to usability studies and research." In: *International journal of human-computer studies* 64.2 (2006), pp. 79–102.

[36] Roya Hosseini and Peter Brusilovsky. "A comparative study of visual cues for annotation-based navigation support in adaptive educational hypermedia." In: *CEUR Workshop Proceedings*. Vol. 1618. 2016.

[37] Philippe Le Hégaret. *Document Object Model (DOM)*. https://www.w3.org/DOM/. Accessed: 2018-12-21. 2009.

[38] *ISO Usability standards ISO/IEC 25022:2016*. https://www.iso.org/obp/ui/fr/#iso:std:iso-iec:25022:ed-1:v1:en. Accessed: 2019-01-02.

[39] *ISO Usability standards ISO/IEC TR 9126-4:2004*. https://www.iso.org/obp/ui/fr/#iso:std:iso-iec:tr:9126:-4:ed-1:v1:en. Accessed: 2019-01-02.

[40] *ISO/IEC 25023:2016*. https://www.iso.org/standard/35747.html. Accessed: 2019-01-02.

[41] *JavaScript Usage Statistics*. https://w3techs.com/technologies/details/cp-javascript/all/all. Accessed: 2018-10-16.

[42] Ryan John. *Canvas LMS course design*. Packt Publishing Ltd, 2014.

[43] David W Johnson, Roger T Johnson, and Karl A Smith. *Active learning: Cooperation in the college classroom.* ERIC, 1998.

[44] H. Jung, S. Kim, and C. Chung. "Measuring Software Product Quality: A Survey of ISO/IEC 9126." In: *IEEE Software* 21 (Sept. 2004), pp. 88–92. ISSN: 0740-7459.

[45] Eva Kassens-Noor. "Twitter as a teaching practice to enhance active and informal learning in higher education: The case of sustainable tweets." In: *Active Learning in Higher Education* 13.1 (2012), pp. 9–21.

[46] Jenny Kitzinger. "Qualitative research: introducing focus groups." In: *Bmj* 311.7000 (1995), pp. 299–302.

[47] Donald Ervin Knuth. *The art of computer programming: sorting and searching*. Vol. 3. Pearson Education, 1997.

[48] Clayton Lewis. "Using the'thinking-aloud'method in cognitive interface design." In: *Research Report RC9265, IBM TJ Watson Research Center* (1982).

[49] Chao Liu, Ryen W. White, and Susan Dumais. "Understanding Web Browsing Behaviors Through Weibull Analysis of Dwell Time." In: *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '10. New York, NY, USA: ACM, 2010, pp. 379–386.

[50] Niels Provos Panayiotis Mavrommatis and Moheeb Abu Rajab Fabian Monrose. "All your iframes point to us." In: *USENIX Security Symposium. USENIX*. 2008, pp. 1–16.

[51] Ali Mesbah. "Advances in testing JavaScript-based web applications." In: *Advances in Computers*. Vol. 97. Elsevier, 2015, pp. 201–235.

[52] Leo A Meyerovich and Benjamin Livshits. "ConScript: Specifying and enforcing fine-grained security policies for Javascript in the browser." In: *2010 IEEE Symposium on Security and Privacy*. IEEE. 2010, pp. 481–496.

[53] Michael Mikowski and Josh Powell. *Single page web applications: JavaScript end-to-end*. Manning Publications Co., 2013.

[54] James Monks and Robert M Schmidt. "The impact of class size on outcomes in higher education." In: *The BE Journal of Economic Analysis & Policy* 11.1 (2011).

[55] Hannelore Montrieux, Ruben Vanderlinde, Tammy Schellens, and Lieven De Marez. "Teaching and learning with mobile technology: A qualitative explorative study about the introduction of tablet devices in secondary education." In: *PloS one* 10.12 (2015), e0144008.

[56] Peter Morville. "User Experience Honeycomb." In: *Web Log Post Retrieved from Http://semanticstudios.com/publications/semantics/000029* 21 (2004), pp. 09–12.

[57] Peter Morville and L Rosenfeld. "Information Architecture 3.0." In: *Semantic studios* 29 (2006).

[58] Matthias M Müller. "Two controlled experiments concerning the comparison of pair programming to peer review." In: *Journal of Systems and Software* 78.2 (2005), pp. 166–179.

[59] Donald A Norman. "Affordance, conventions, and design." In: *interactions* 6.3 (1999), pp. 38–43.

[60] *Online Historical Encyclopaedia of Programming Languages*. http://hopl.info/. Accessed: 2018-11-05.

[61] Annemarie Sullivan Palincsar. "The role of dialogue in providing scaffolded instruction." In: *Educational psychologist* 21.1-2 (1986), pp. 73–98.

[62] Linda Dailey Paulson. "Developers shift to dynamic programming languages." In: *Computer* 40.2 (2007).

[63] Janneke Van de Pol, Monique Volman, and Jos Beishuizen. "Scaffolding in teacher–student interaction: A decade of research." In: *Educational psychology review* 22.3 (2010), pp. 271–296.

[64] Michael Prince. "Does active learning work? A review of the research." In: *Journal of engineering education* 93.3 (2004), pp. 223–231.

[65] Dave Raggett, Arnaud Le Hors, Ian Jacobs, et al. "HTML 4.01 Specification." In: *W3C recommendation* 24 (1999).

[66] Hajo A Reijers, Thomas Freytag, Jan Mendling, and Andreas Eckleder. "Syntax highlighting in business process models." In: *Decision Support Systems* 51.3 (2011), pp. 339–349.

[67] David Robins, Jason Holmes, and Mary Stansbury. "Consumer health information on the Web: The relationship of visual design and perceptions of credibility." In: *Journal of the American Society for Information Science and Technology* 61.1 (2010), pp. 13–29.

[68] Amy Roehl, Shweta Linga Reddy, and Gayla Jett Shannon. "The flipped classroom: An opportunity to engage millennial students through active learning strategies." In: *Journal of Family & Consumer Sciences* 105.2 (2013), pp. 44–49.

[69] Patricia L Rogers. *Encyclopedia of distance learning*. IGI Global, 2009.

[70] Marco Ronchetti. "Using video lectures to make teaching more interactive." In: *International Journal of Emerging Technologies in Learning (iJET)* 5.2 (2010).

[71] David Ryback and Joann J Sanders. "Humanistic versus traditional teaching styles and student satisfaction." In: *Journal of Humanistic Psychology* 20.1 (1980), pp. 87–90.

[72] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. "Busting frame busting: a study of clickjacking vulnerabilities at popular sites." In: *IEEE Oakland Web* 2.6 (2010).

[73] Thinzar Saw, Kyu Kyu Win, Zan Mo Mo Aung, and Myat Su Oo. "Investigation of the Use of Learning Management System (Moodle) in University of Computer Studies, Mandalay." In: *International Conference on Big Data Analysis and Deep Learning Applications*. Springer. 2018, pp. 160–168.

[74] R Keith Sawyer. *The Cambridge handbook of the learning sciences*. Cambridge University Press, 2005.

[75] Emmit Scott. *SPA design and architecture: understanding single page web applications*. Manning Publications Co., 2015.

[76] Steve Souders. *High Performance Web Sites: Essential Knowledge for Front-End Engineers. sl*. 2007.

[77] John Sweller. "Cognitive load during problem solving: Effects on learning." In: *Cognitive science* 12.2 (1988), pp. 257–285.

[78] Sirous Tabrizi and Glenn Rideout. "International Journal for Cross-Disciplinary Subjects in Education (IJCDSE), Volume 8, Issue 3, September 2017." In: *http://infonomics-society.org/* (2017).

[79] Ankur Taly, John C Mitchell, Mark S Miller, Jasvir Nagra, et al. "Automated analysis of security-critical javascript apis." In: *2011 IEEE Symposium on Security and Privacy*. IEEE. 2011, pp. 363–378.

[80] Kristín Fjóla Tómasdóttir, Mauricio Aniche, and Arie Van Deursen. "The Adoption of JavaScript Linters in Practice: A Case Study on ESLint." In: *IEEE Transactions on Software Engineering* (2018).

[81] Greg Topp. "Flipped classrooms take advantage of technology." In: *USA Today* 6 (2011).

[82] Wataru Tsukahara, Fumihiko Anma, Ken Nakayama, and Toshio Okamoto. "A Development of Learning Management System for the Practice of E-Learning in Higher Education." In: *Knowledge Management for Educational Innovation*. Ed. by Arthur Tatnall, Toshio Okamoto, and Adrie Visscher. Boston, MA: Springer US, 2007, pp. 145–152. ISBN: 978-0-387-69312-5.

[83] Maaike Van Den Haak, Menno De Jong, and Peter Jan Schellens. "Retrospective vs. concurrent think-aloud protocols: testing the usability of an online library catalogue." In: *Behaviour & information technology* 22.5 (2003), pp. 339–351.

[84] Kelly Wainwright. "The Care and Feeding of a Moodle Campus." In: *Proceedings of the 37th Annual ACM SIGUCCS Fall Conference: Communication and Collaboration*. SIGUCCS '09. New York, NY, USA: ACM, 2009, pp. 267–270.

[85] Qiyun Wang, Huay Lit Woo, Choon Lang Quek, Yuqin Yang, and Mei Liu. "Using the Facebook group as a learning management system: An exploratory study." In: *British Journal of Educational Technology* 43.3 (2012), pp. 428–438.

[86] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. "Demystifying Page Load Performance with WProf." In: *NSDI*. 2013, pp. 473–485.

[87] Debbi Weaver, Christine Spratt, and Chenicheri Nair. "Academic and student use of a learning management system: Implications for quality." In: *Australasian Journal of Educational Technology* 24.1 (2008). URL: https://ajet.org.au/index.php/AJET/article/view/1228.

[88] Laurie Williams. "White-Box Testing." In: *PDF): 60–61* 69 (2006).

[89] David Wood and Heather Wood. "Vygotsky, tutoring and learning." In: *Oxford review of Education* 22.1 (1996), pp. 5–16.

[90] Chuan Yue and Haining Wang. "Characterizing insecure javascript practices on the web." In: *Proceedings of the 18th international conference on World wide web*. ACM. 2009, pp. 961–970.

[91] Nicholas C Zakas. *Maintainable JavaScript: Writing Readable Code*. " O'Reilly Media, Inc.", 2012.

[92] Sebastian Mader et al. *More than the Sum of its Parts: Designing Learning Formats from Core Components*. LMU, 2018.

COLOPHON

This document is set in *Minion Pro*. The choice of font and layout is quite deliberate since I believe that the chosen layout improves readability and provides enough room in the margins for your notes.

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Final Version as of January 23, 2019 (`classicthesis` version 1.0).