



OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

University of Galway

College of Science and Engineering, School of Computer Science

Department Head: Professor Michael Madden

Chips and Code: Moving a Learning Tool to the Web for Usability and Inclusion

Jakub Duras

Supervisor: Dr. Frank Glavin

A thesis submitted in partial fulfilment for the degree
MSc in Software Engineering and Database Technologies

September 2023

Final Thesis Submission

MSc in Software Engineering and Database Technologies

*School of Computer Science
College of Science and Engineering
University of Galway*

Student Name:	Jakub Duras
Telephone:	+421 917 432 974
E-mail:	jakub@duras.me
Date of Submission:	09-09-2023
Title of Submission:	Chips and Code: Moving a Learning Tool to the Web for Usability and Inclusion
Supervisor Name:	Dr. Frank Glavin

Certification of Authorship:

I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for the purpose of partial fulfilment of requirements for the Degree Programme.

Signature:

Date: 09-09-2023

Acknowledgements

First of all, I would like to thank my advisor, Dr. Frank Glavin, and the university staff, who patiently answered my questions and provided me with helpful feedback. Then, I would like to thank my brother Jan Duras for his help with the graphic design and adaptation of the learning material. Moreover, I would like to thank everybody who generously offered their time to participate in the comparative study. Last but not least, I would like to thank my family, friends, and colleagues, who encouraged me and gave me the time and space needed to focus on this thesis.

Contents

Acknowledgements	i
List of Tables	vi
List of Figures	vii
List of Acronyms	ix
Abstract	xii
1 Introduction	1
1.1 Overview and Motivation	1
1.2 Research Questions and Objectives	2
1.3 Research Contributions	3
1.4 Thesis Structure	3
2 Literature Review	5
2.1 Human-System Interaction Ergonomics	5
2.1.1 Definition and Distinction	5
2.1.2 Benefits	8
2.1.3 Implementation	9
2.1.4 Evaluation	13
2.2 Learning Computing Principles	20

Contents

2.2.1	State of Remote Learning and MOOCs	20
2.2.2	Low-Level Computing Principles	22
2.3	Device Type Usage	24
2.3.1	Developed Countries	25
2.3.2	Developing Countries	25
2.4	Text and Code	26
2.4.1	Producing Learning Material	26
2.4.2	Writing Code	27
2.4.3	Parsing Code	28
2.5	Creating an Open Web Application	32
2.5.1	Legal Background	32
2.5.2	Essential Steps	35
2.5.3	Development and Deployment	36
2.5.4	Reliability and Security	37
2.5.5	Testing	38
3	Methodology	40
3.1	Software Design	41
3.2	Functional Testing	41
3.3	Comparative Study	43
3.3.1	Controlled Variables	44
3.3.2	Independent Variable	48
3.3.3	Dependent Variables	50
3.3.4	Data Analysis	52
3.3.5	Limitations	53
3.4	Summary	54
4	Design and Implementation	55
4.1	Overall Architecture	55

4.2	Build and Deployment	56
4.3	User Interface	58
4.3.1	Interactions	58
4.3.2	Error Handling	61
4.4	Branding and Presentation	63
4.5	Efficiency	64
4.6	Security	68
4.7	Accessibility	68
4.8	Device Support	69
4.9	Learning Content	71
4.10	Summary	72
5	Evaluation	73
5.1	Functional Testing	73
5.1.1	Unit Tests	73
5.1.2	UI and Accessibility Tests	75
5.2	Accessibility Evaluation	78
5.3	Comparative Usability Testing	80
5.3.1	Efficiency	84
5.3.2	Questionnaire	86
5.3.3	Assignment Bias	88
5.4	Summary	90
6	Discussion	91
6.1	Feasibility	91
6.2	Improvements	93
6.3	Limitations and Further Work	96
7	Conclusions	98

Contents

Bibliography	100
A HDL Railroad Diagrams	115
B HDL Token, Grammar Definition	119
C HDL Example	123
D TST Railroad Diagrams	124
E TST Token, Grammar Definition	128
F TST Example	132
G Study SUS Scores	133

List of Tables

3.1	Partial timestamp data collected from one of the sessions. . . .	51
5.1	Group A participant profiles.	81
5.2	Group B participant profiles.	82

List of Figures

2.1	Relation between Accessibility, Usability, and UX.	8
2.2	Usability in the context of System Acceptability (C. Wilson 2009).	12
2.3	Web content relation to Accessibility guidelines, users, and developers (WAI 2021b).	14
2.4	Sample size vs “correct” conclusions (Tullis and Stetson 2004).	19
2.5	Two-sample <i>t</i> -test formula (Sauro and Lewis 2016b).	20
2.6	Confidence interval formula (Sauro and Lewis 2016b).	20
2.7	Example disconnected graph showing different concepts.	29
2.8	Example directed rooted tree.	30
2.9	License type comparison and examples (Duras 2020).	34
2.10	Testing pyramid originally proposed by Cohn (2010).	39
3.1	Screenshot of Proposed Software within text—Group A.	49
3.2	Screenshot of Existing Software’s input within text—Group B.	49
4.1	High-level architecture diagram.	56
4.2	Comparison of steps needed to iterate on an HDL assignment.	60
4.3	Existing Software’s Text Comparer (top) and Proposed Software’s visual output diff (bottom).	61
4.4	Existing Software’s (top) and Proposed Software’s (bottom) syntax error handling.	62

List of Figures

4.5	Existing Software's (top) and Proposed Software's (bottom) pin connection error handling.	62
4.6	HDL processing from text to chip connection graph.	63
4.7	Size comparison between Proposed Software and Existing Software.	66
4.8	Lighthouse performance report for Proposed Software.	67
4.9	Example use of aXe DevTools to verify focus order.	69
4.10	Hardware IDE viewed from a smartphone.	70
4.11	Probable percentage of supported devices.	71
4.12	Example of learning content written in Markdown.	72
5.1	Example unit test.	74
5.2	Testing framework run output and coverage report.	75
5.3	Example simplified Cypress test.	77
5.4	Output from Cypress run.	78
5.5	Lighthouse report summary.	79
5.6	aXe report summary.	80
5.7	Cumulative profile points between groups.	83
5.8	Age characteristics of participants.	83
5.9	Platforms used by participants.	84
5.10	Time needed to prepare software and perform tasks.	85
5.11	Cumulative time needed to prepare software and perform tasks.	85
5.12	Number of times participants got confused.	86
5.13	SUS score comparison.	87
5.14	SUS score percentile.	88
5.15	Relation between total time and SUS score.	88
5.16	Relation between education and dependent variables.	89
5.17	Relation between occupation and dependent variables.	90
5.18	Relation between age and dependent variables.	90

List of Acronyms

ALU	Algorithmic Logic Unit.
API	Application Programming Interface.
ARIA	Accessible Rich Internet Applications.
AST	Abstract Syntax Tree.
BFS	Breadth First Search.
CAGR	Compound Annual Growth Rate.
CC	Creative Commons.
CD	Continuous Deployment.
CI	Continuous Integration.
CLI	Command Line Application.
CS	Computer Science.
CSS	Cascading Style Sheets.
CST	Concrete Syntax Tree.
CUE	Components of User Experience.
DSR	Design Science Research.
EU	European Union.
HCI	Human-Computer Interaction.
HDL	Hardware Description Language.
HTML	HyperText Markup Language.

List of Acronyms

HTTP	Hypertext Transfer Protocol.
ICT	Information and Communication Technology.
IDE	Integrated Development Environment.
ISO	International Organisation for Standardisation.
JS	JavaScript.
JVM	Java Virtual Machine.
KPI	Key Performance Indicator.
meCUE	Modular Evaluation of Key Components of User Experience.
MOOC	Massive Open Online Course.
OS	Operating System.
OSS	Open Source Software.
PSSUQ	Post-Study System Usability Questionnaire.
PWA	Progressive Web Application.
RAM	Random Access Memory.
REST	Representational State Transfer.
ROI	Return on Investment.
SPA	Single Page Application.
SSR	Server-Side Rendering.
SUS	The Software Usability Scale.
TST	Testing Script Language.
UEQ-S	User Experience Questionnaire Short.

List of Acronyms

UI	User Interface.
UMUX	Usability Metric for User Experience.
URL	Uniform Resource Locator.
US	United States.
UX	User Experience.
WAI	Web Accessibility Initiative.
WCAG	Web Content Accessibility Guidelines.
WCAG-EM	Website Accessibility Conformance Evaluation Methodology.
WYSIWYG	What You See Is What You Get.
YoY	Year over Year.

Abstract

Older desktop learning tools pose a challenge in a world that increasingly relies on new classes of devices such as smartphones, Chromebooks, and tablets. While the use of new classes of devices is mostly a matter of preference in developed countries, households in developing countries often do not own any other device capable of running desktop applications. This thesis explores the feasibility of efficiently reimplementing the functionality of a popular desktop learning tool as a modern web application accessible to a wide range of users. Additionally, it uses remote moderated usability testing to perform a between-subjects comparison of the existing and proposed software, focusing on quantitative data. The proposed software (chipsandcode.com) mitigates common disadvantages of the web while achieving efficiency improvements (presented at thesis.chipsandcode.com). Data from the conducted comparative study show participants ($n = 12$ per group) completed the work in one-third of the time ($512 \pm 190s$ vs $1497 \pm 326s$, $\alpha = 0.05$, $p < 0.001$) and experienced considerably fewer confusing situations ($p < 0.001$). Considering common reasons for Massive Open Online Course (MOOC) dropouts are lack of time and technical problems, more efficient and less confusing learning tools could be of particular interest. Moreover, web-based learning tools could be easier to incorporate into various learning materials using embedding or linking, particularly if they are easier to pick up.

1 Introduction

1.1 Overview and Motivation

Software usability is a well-researched important part of software quality (Almazroi 2021). Notably, improvements in the area can bring a high, double-digit Return on Investment (ROI) (Nielsen 2008). However, the ROI is smaller if the starting point is better. Data from Nielsen (2008) suggest newer creations are usually better positioned than older ones. Therefore, it appears the key is to identify software that has poor usability but is still used very often.

One of the most popular learning materials on low-level computer principles is the Nand2Tetris course, “taught at 400+ universities, high schools, and bootcamps” (Shocken and Nisan 2017). It introduces multiple computing concepts by building a virtual computer from individual logic gates to a high-level programming language. These concepts can be of interest to people from all walks of life:

- From Computer Science (CS) students covering the relevant undergraduate curriculum.
- Through Information and Communication Technology (ICT) professionals with or without formal education, who can use the knowledge for a deeper understanding of the craft.
- To the general public who would like to understand the mysterious devices they rely on daily.

However, like many other learning materials, Nand2Tetris is supported by an older desktop learning tool with suspected usability and device support limitations. One of the obvious limitations of older desktop software is that it could not count with the diversity of devices used in today's age. With the increasingly common smartphones, tablets, or devices like Chromebooks that do not allow running arbitrary desktop applications, this represents a problem for a growing part of learners. Even worse, households in developing countries often have no devices capable of running desktop applications.

1.2 Research Questions and Objectives

Using the existing desktop tool for learning computing principles as an example, this thesis seeks to discover:

- RQ1: Is it feasible to efficiently reimplement the functionality of a desktop learning tool using web technologies, achieving better device support?
- RQ2: To what extent is it possible to improve usability when reimagining a desktop learning tool as a web application?

In order to answer these research questions, this thesis has the following objectives:

1. Implementation of a reimagined web version of the Nand2Tetris Hardware Simulator with the focus on:
 - Compatibility with the original software.
 - Usability—particularly task efficiency.
 - Broad device support.
2. Collection of quantitative and accidental qualitative data on:
 - Objective usability, such as the time on task or the number of errors.

- Subjective usability in the form of a questionnaire answers.
3. Analysis of the collected data, comparing the data between the old and proposed software and looking for patterns.

1.3 Research Contributions

Considering the objectives of this thesis represent a combination of software engineering and usability testing outputs, we can group the contributions of this thesis into two overlapping groups:

- **Practical:** Working alternative to the Nand2Tetris Hardware Simulator accessible from the web for future learners who cannot or do not prefer running the original software.
- **Academic:** Data on the possible usability (especially efficiency) and device support gains to support decision-making. Solution design in the form of this thesis and Open Source Software (OSS) source code to help reproduce the results.

1.4 Thesis Structure

This thesis is divided into several chapters. Chapter 2 reviews the literature and starts by exploring the relevant literature on Human-Computer Interaction (HCI) ergonomics concepts, how they can be implemented, and especially how the implemented concepts can be evaluated. It examines the relevance of Massive Open Online Courses (MOOCs), common hurdles of online classes, and existing software used to learn low-level computing concepts. Then, it looks for patterns and trends of the types of devices currently used worldwide and by learners in particular. Finally, it ends by covering CS and software engineering concepts like the production of learning material, code editing, parsing, legal aspects of software and learning materials, and design, development, and testing of OSS software.

Once the relevant concepts and improvement opportunities are covered, Chapter 3 introduces the methodology used for designing and evaluating the new web-based learning tool. The evaluation methodology first covers functional testing—whether the functionality works and can do what the desktop tool can. Then, the methodology introduces a controlled comparative study that involves two versions of the same content that differ only in the used software—one using the existing software and the other the new software.

There are two distinctive phases of the primary research: design and implementation of the new learning tool, and evaluation that verifies the results of the first phase. Chapter 4 covers the first phase of the research—design and implementation. It describes the feasibility part of the RQ1, focusing on the differences in the approach, its effects on the mentioned goals, and how it was possible to address them efficiently through technology. Chapter 5 covers the second phase of the research, which answers RQ2—evaluation. It presents the scope and results of the functional testing, the outcome of the accessibility evaluation, and the analysis of the data points collected from the comparative study.

Finally, the achieved results are discussed in Chapter 6. The discussion reflects on the design and implementation process in the context of the specified questions and goals by commenting on the extent to which the goals were achieved and the practical implications. The key results are ultimately summarised in Chapter 7.

2 Literature Review

Considering the theme of this thesis, this chapter starts by taking a closer look at the field of HCI Ergonomics, with special attention paid to the implementation and evaluation. Then, it continues with the exploration of MOOCs, focusing on gaps, common challenges, and existing equivalent software. Next, it briefly gathers data on the device type usage to form a basis for the motivation. After that, it introduces text manipulation principles to support and explain the decisions. Finally, it ends with a quick overview of selected software engineering problems most relevant to this thesis.

2.1 Human-System Interaction Ergonomics

The following section focuses on the selected topics of HCI Ergonomics: Accessibility, Usability, and User Experience (UX). The selected topics are explored in relation to computer software and, where possible, web services specifically. After introducing the concepts, this section focuses on non-obvious benefits and delves into the implementation and evaluation strategies. This section strives to introduce the topics and dive deeper into pragmatic information relevant to this thesis.

2.1.1 Definition and Distinction

Considering there are incompatible definitions for Accessibility provided by the International Organisation for Standardisation (ISO) (Wegge and Zimmermann 2007), the specific definition chosen for this thesis describes Ac-

cessibility as the “extent to which [a service] can be used by people from a population with the widest range of user needs, characteristics and capabilities to achieve identified goals [...]” (ISO 2018). The ISO (2020) adds that Accessibility includes but does not apply exclusively to formally disabled people. That means Accessibility is concerned with the basic ability to utilise the software by the widest possible range of users, including disabled users (Wegge and Zimmermann 2007). Similar concepts include “design for all”, “inclusive design”, or “universal design” (Sauer *et al.* 2020, p. 1210) with a viewpoint of designing services for all types of people, including the disabled. Wegge and Zimmermann (2007) point to existing confusion between the terms Accessibility and Usability. Although Wegge and Zimmermann (2007) admit these terms are related and have potential overlap, Wegge and Zimmermann (2007) stress the importance of their distinction. Importantly, ICT is one of the fields where Accessibility, depending on the country and sector, is mandated by the law (Wegge and Zimmermann 2007; Sauer *et al.* 2020). For example, public sector services within the European Union (EU) have to meet accessibility standards outlined in the Web Accessibility Directive, and there is an ongoing effort to extend this to the private sector (European Commission 2021).

Usability, on the other hand, is defined by the ISO (2018) as the “extent to which [a service] can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction [...]”. As Wegge and Zimmermann (2007) hint, compared to Accessibility, Usability deals with the success—“effectiveness, efficiency and satisfaction” (ISO 2018)—of software interactions which is hard to mandate the way Accessibility is. Therefore, there are few relevant laws and Usability is usually considered more as a competitive advantage that authors are trying to capitalise on (Wegge and Zimmermann 2007).

Similarly to confusion between Accessibility and Usability, researchers point to issues with the distinction between Usability and User Experience (Darin *et al.* 2019; Sauer *et al.* 2020). There are multiple views on the definition of UX and a great amount of disagreement on the topic (Sauer *et al.*

2020). The ISO (2018) defines UX as “user’s perceptions and responses that result from the use and/or anticipated use of [a service]”. Compared to the sole satisfaction mentioned to be a part of Usability, Darin *et al.* (2019) argue that it is incorrectly used as, and often only, UX measurement. UX deals with “users’ emotions, beliefs, preferences, perceptions, comfort, behaviours, and accomplishments” (ISO 2018). Additionally, UX is concerned with a broader timeline—the time spent performing the task *and* the time before and after that (Sauer *et al.* 2020; ISO 2018). Considering the definition by the ISO (2018) and in partial disagreement with Darin *et al.* (2019), we can look at UX as a concept that overlaps with Usability or a whole complex superset of Usability (Sauer *et al.* 2020).

In short, for the purposes of this thesis, Accessibility is the basic ability to use the software regardless of the user’s various limitations, Usability is the extent to which it can be used to achieve delineated goals successfully (ISO 2018), and UX is a more complex extension also concerned with impressions both during and outside of the use of the software. The following subsections always refer to these ideas as Accessibility, Usability, and UX, even if cited literature refers to them differently.

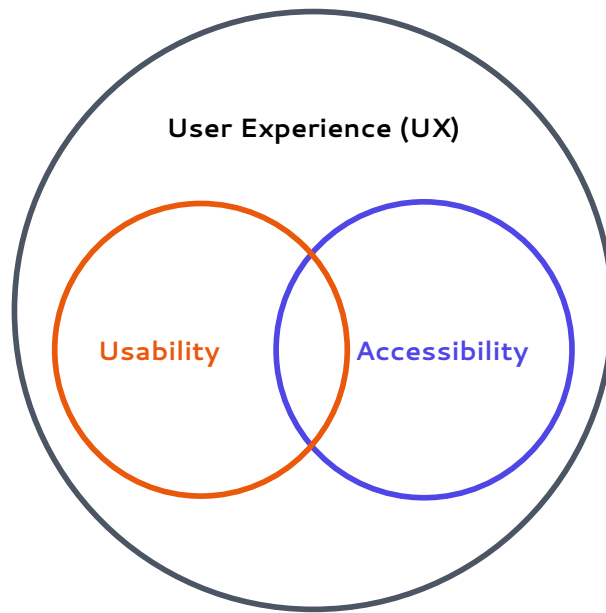


Figure 2.1: Relation between Accessibility, Usability, and UX.

2.1.2 Benefits

Sauer *et al.* (2020) point to their earlier research results that indicate implementing Accessibility on the web could provide benefits to users other than the usual primary target of Accessibility enhancements—disabled users (Schmutz *et al.* 2016; Schmutz *et al.* 2017; Schmutz *et al.* 2018). This notion is seconded by Vanderheiden (2000), who mentions that others in similar challenging situations can benefit from Accessibility as well. Edyburn (2010) provides terminology identifying these two groups of users: primary and secondary beneficiaries. Specifically in the educational setting, Edyburn (2021) mentions that Accessibility can improve the ability to use the software for both primary beneficiaries—disabled students—and secondary beneficiaries—all other students.

Examples of secondary beneficiaries concerning the web are mentioned by Henry (2021b):

- People using different devices with smaller screens or “different input modes” (Henry 2021b).

- People challenged by limitations introduced by ageing.
- Temporarily limited people—e.g. by a “broken arm or lost glasses [...or a] bright sunlight” (Henry 2021b), or conditions that do not allow audio playback.
- And people who are limited by their internet connection—i.e. latency or bandwidth.

However, Sauer *et al.* (2020) also mention that benefits from some specific concepts could be limited. For example, “using easy language” (Sauer *et al.* 2020, p. 1210) was shown to have limited benefits and considerable disadvantages like decreased enjoyment and higher required time to consume the content (Schmutz *et al.* 2019).

As for the Usability and UX benefits, as was already mentioned in Section 2.1.1, these can be used as a competitive advantage (Wegge and Zimmermann 2007), or, more specifically, to improve some metric like conversion rate, traffic numbers, user performance, or usage of key features (Nielsen 2008). Surveys summarised by Nielsen (2008) indicate a high, double-digit ROI even though this number is declining yearly, presumably due to the ever-improving starting state.

2.1.3 Implementation

Accessibility

According to Wegge and Zimmermann (2007, p. 296), there are three main approaches when implementing the Accessibility requirements:

- **Universal Design:** Designing software to be usable by the widest range of users without any modifications.
- **Adaptive Design:** Designing software to be adaptable to different types of users.

- **Interoperability with Assistive Technology:** Designing software to work with existing assistive software.

Wegge and Zimmermann (2007) warn about the downsides of Universal Design: the ability to hinder the experience of the majority, stigmatisation, and implementation difficulties due to conflicting requirements. Sauer *et al.* (2020) reiterate this point and add that a compromise may be necessary to satisfy all groups. In contrast, implementing Adaptive Design allows one to opt into alternative, independent representations without influencing other groups of users (Wegge and Zimmermann 2007). Similarly, assuming the software follows the relevant standard Application Programming Interface (API), Interoperability with Assistive Technology allows selected users to use their existing tools, e.g. screen readers (Wegge and Zimmermann 2007). Edyburn (2021) mentions the usefulness of Assistive Technologies like speech-to-text and text-to-speech in relation to the learning environment and argues they are already available on most platforms and can be targeted at both types of beneficiaries.

Sauer *et al.* (2020) mention that the web has received a significant amount of attention in regards to Accessibility thanks to its perceived importance. Web Accessibility Initiative (WAI) is recognised as the relevant source of information used to develop and verify the Accessibility of web services (Henry 2021b). Web Content Accessibility Guidelines (WCAG) by WAI are adopted around the world, including by the ISO (2012) and in the law (WAI 2018), an example of which is the EU's Web Accessibility Directive (European Commission 2021). The current stable version is WCAG 2.1. While version 2.0 is standardised in the ISO (2012) and is the source for many legally binding documents, any minor versions like WCAG 2.1 are backwards compatible as they contain verbatim all requirements from the previous version (WAI 2021b).

According to WAI (2016), there are two kinds of Accessibility requirements: technical, which are mostly fulfilled by properly utilising available APIs; and interaction/visual requirements implemented with the assistance of real users. Implementing both kinds of requirements should mean the ser-

vice is both “technically and functionally usable” (WAI 2016). WAI (2016) warns that Accessibility is ideally implemented during the development as implementing it later can be problematic, but also admits addressing all Accessibility issues is challenging even for large projects.

While Accessibility for web services can be implemented following the mentioned industry-standard WCAG, Vanderheiden (2000) argues that implementing Usability and Accessibility, in general, can be overwhelming. Vanderheiden (2000) points to hundreds of strategies one may choose from without appropriate consideration. Therefore, Vanderheiden (2000) proposes prioritisation using four dimensions:

- **Accessibility:** Does the given feature influence the basic Accessibility of the service? Not implementing the most important ones means the service is completely unusable for some groups, while skipping the less important ones only makes the product harder to use for some groups of users.
- **Independence:** Should the user independently use the feature? Less used and more involved features can be usable and delegated only to more advanced users, while everyday tasks must be usable by all.
- **Efficiency and Urgency:** How much impact does the task have on the efficiency of the given type of user, and is there some time constraint? Tasks performed very frequently and within some time constraints have a higher priority. Tasks that are not reversible and a failure can have significant consequences have a higher priority.
- **Ease of Implementation:** The estimated effort required to implement the Usability enhancement. Enhancements that are easier to implement have higher priority.

Usability and UX

Considering a narrower definition of Usability, Nielsen (1993) looks at Usability as a quality and mentions a service should be both usable and have

utility (provide features users need). The relation between Usability, utility and other acceptability requirements is outlined in Figure 2.2. Assuming service is Accessible and has utility, Nielsen (1993) recognises the following five qualitative components:

- **Learnability:** How easy a service is to pick up for the first time.
- **Efficiency:** How quickly the tasks can be successfully performed.
- **Memorability:** To what extent the efficiency is reproducible after a period of time not using a service.
- **Errors:** How many errors are being made, their significance, and recovery.
- **Satisfaction:** How nice the service feels to use.

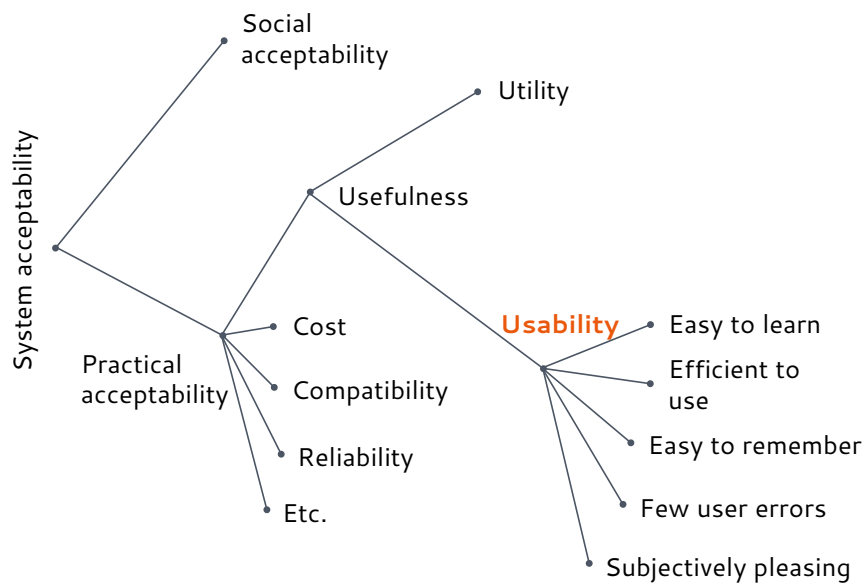


Figure 2.2: Usability in the context of System Acceptability (C. Wilson 2009).

The Components of User Experience (CUE) model introduced by Thüring and Mahlke (2007) discerns between “instrumental” and “non-instrumental” qualities (Sauer *et al.* 2020, p. 1209). Similarly, Hassenzahl *et al.* (2008)

distinguish between “pragmatic” and “hedonic” qualities (Sauer *et al.* 2020, p. 1209). Both former categories match the first four qualities mentioned by Nielsen (1993), while the latter are closer to Satisfaction and encompass, e.g., visual aesthetic, relatedness, or stimulation (Thüring and Mahlke 2007; Hassenzahl *et al.* 2008).

2.1.4 Evaluation

Accessibility

Henry (2021a) acknowledges two ways to evaluate the Accessibility of web services akin to the two kinds of Accessibility requirements outlined in 2.1.3:

- Using tools—software can automatically check a service against standards like WCAG or assist humans but cannot solely determine whether a web service is Accessible (WAI 2017),
- With the assistance of humans as target users—users without disabilities or with different disabilities can be involved in evaluating Accessibility. Still, they alone cannot determine whether a web service is Accessible (WAI 2020).

A comprehensive evaluation of conformance to WCAG can be performed using Website Accessibility Conformance Evaluation Methodology (WCAG-EM) (WAI 2021a). It can be used for self-assessment or assessment by a third party and can involve both tools and users (WAI 2021a). WAI (2021a) provides the five steps that make up the WCAG-EM, and that can be executed with the help of WCAG-EM Report Tool:¹

1. Determine the overall scope—define the evaluated subject, baseline, and objectives, including the mentioned WCAG version and conformance level—A, AA, or AAA—with AAA being the strictest and AA being recommended.

¹WCAG-EM Report Tool is available online at <https://www.w3.org/WAI/eval/report-tool>.

2. Explore the subject—find the used technologies, e.g., HTML, WAI-ARIA, or MathML, and identify essential functionality or types of content, e.g., authentication, check-out, or blog pages.
3. Pick a sample from the subject—pick specific (random) Uniform Resource Locators (URLs) representative of the findings from the previous step or all available URLs if possible.
4. Evaluate the sample—follow the WCAG to gather information about compliance with all of the requirements of the relevant conformance level for each identified URL.
5. Report the results—create a report, e.g. using the provided template, that includes an executive summary, background, scope, details about the person evaluating, review process approach and goal, and results and recommendations (Brewer 2018).

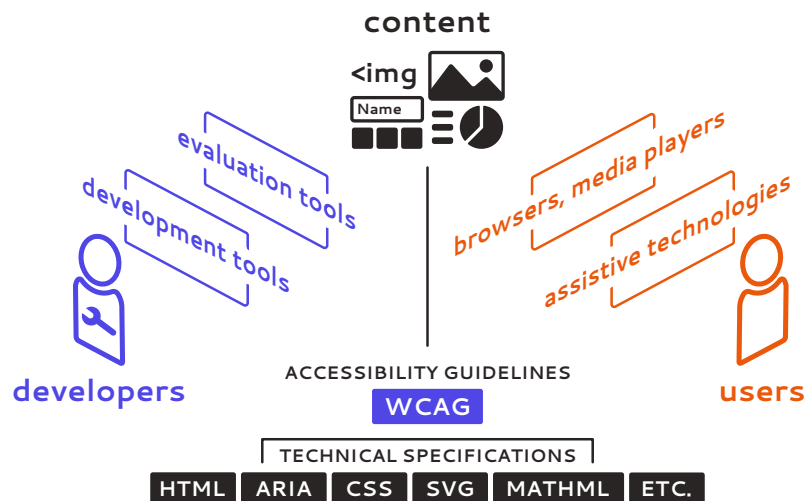


Figure 2.3: Web content relation to Accessibility guidelines, users, and developers (WAI 2021b).

WAI (2017) mentions a wide variety of (semi) automated evaluation tools are available to cover different guidelines, languages, or technologies and the work with different environments, inputs, and outputs. Three examples of types of tools in regard to the mode of operation include APIs, browser plugins, and online services² (WAI 2017). Findings from Frazão and Duarte (2020), who reviewed the currently available most popular browser plugins, support the recommendation from WAI on the need to use multiple tools and perform manual actions on top of the automated checks. For example, the most popular one, Google’s Lighthouse, based on the aXe engine, checks for 19 criteria compared to TotalValidator’s 50, and it seems to be more efficient in one kind of criteria while the others are more efficient at finding errors relevant for other criteria (Frazão and Duarte 2020). As far as evaluation with the assistance of users is concerned, WAI (2020) warns that users with different backgrounds and representing different groups should be ideally involved to capture a wide variety of real potential users and eliminate subjectivity. Wegge and Zimmermann (2007) reiterate Accessibility evaluation involves users from different groups that can tell whether the service can be used with their disability, is compatible with their assistive technology, or if the service behaves the way they expect.

Usability and UX

Wegge and Zimmermann (2007) mention that Usability, in contrast to Accessibility, is not typically tested by adherence to standards but by testing and analysing the impact on the end-users. Edyburn (2021) adds that we can monitor subjective measures like decreased frustration or objective measures like increased productivity.

A distinction is pointed to by Sauer *et al.* (2020), who reiterate two kinds of Usability testing:

- **Formative:** Helps pinpoint and solve problems with services during the development—called “diagnostic usability” by Lewis (2014),

²A list of web accessibility evaluation tools maintained by WAI is available online at <https://www.w3.org/WAI/ER/tools>.

- **Summative:** Evaluates a service to compare it to other services or criteria—called “measurement-based usability” and told to have similarities to experimental psychology by Lewis (2014).

Sauer *et al.* (2020) also argue that the same thinking could be followed when evaluating UX as well, even though such explicit distinction has not been made yet in regards to UX. There is a large number of concrete methods that could be utilised, like questionnaires, observation, interviews, data logging, or user testing (Sauer *et al.* 2020). While expert-based methods are seen as more cost-effective, some argue they do not reflect real use the way user-based do and user-based methods can generate a broader range of data (Sauer *et al.* 2020). Sauer *et al.* (2020) argue a “principal method” to evaluate the Usability of a service is user testing—users use artefacts watched by observers that acquire objective quantitative data, e.g., on efficiency or errors. All while focusing on specific roles and tasks (Wegge and Zimmermann 2007; McCloskey 2014).

UX evaluation that takes into consideration emotion (i.e. not or not only Usability) is argued to be harder to evaluate since it is very individual (Sauer *et al.* 2020). Therefore, expert-based methods are replaced by user-based methods that can be based on more broad but still subjective established questionnaires like Modular Evaluation of Key Components of User Experience (meCUE) (Sauer *et al.* 2020). Considering another angle of subjective UX evaluation that includes Usability, Lewis (2014) asserts practitioners should use standardised Usability questionnaires proven to reliably assess perceived Usability and referenced in national and international standards, for example, The Software Usability Scale (SUS) (Lewis 2018) or the Post-Study System Usability Questionnaire (PSSUQ) (Lewis 1995; Lewis 2002). Both standardised questionnaires are available for free, should produce statistically significant data, and have a relatively low number of items—10 for SUS and 16 for PSSUQ (Lewis 2014). An even shorter alternative that has only slightly lower reliability ($r > 0.8$ compared to $r > 0.9$) and seems to produce results correlated to SUS repeatedly is Usability Metric for User Experience (UMUX) with four items and its shorter variant UMUX-LITE

with two items (Lewis 2014; Lewis 2018). Schrepp *et al.* (2023) suggest that researchers can feel confident choosing SUS or UMUX-LITE based on their findings if the main objective is Usability-focused aspects. That is mainly thanks to the focus on the Usability aspects that makes these variants lend themselves well to professional software (Schrepp *et al.* 2023). The same cannot be said about software used for leisure. As Schrepp *et al.* (2023) mention, only questionnaires like the User Experience Questionnaire Short (UEQ-S) were able to capture hedonic qualities.

Some questionnaires, like the mentioned SUS, have a long history and can offer useful suggestions based on the data from years of use (J. Brooke 2013). For example, J. Brooke (2013, p. 35) points out that the word “cumbersome” used in one of the questions is not easily understood by non-native English speakers. To address this problem, J. Brooke (2013, p. 35) suggests it can be replaced with the word “awkward”, citing several supporting studies. In terms of interpreting the resulting SUS score, J. Brooke (2013, pp. 36-37) points out there are data-backed ways to assign a grade and an adjective to resulting scores the way Bangor *et al.* (2009) and Sauro (2011) did.

In relation to e-learning software, in particular, Darin *et al.* (2019) also mention that commonly used instruments are questionnaires. That being said, Darin *et al.* (2019, p. 60) also point to “a rising trend [of capturing both] self-reported data [and] UX measurement, in quali-quantitative approaches”. A more specific and objective alternative to capturing emotional aspects can be physiological data like body posture (Tan *et al.* 2013). However, these can come with privacy challenges and difficulty in evaluating the data (Sauer *et al.* 2020).

In regards to the design of Usability testing, McCloskey (2014) mentions that user tasks should capture key user goals, prompt users to take action and be engaging by providing context without giving out any clues. Remote Usability testing can, but does not have to, involve interactive sessions between users and facilitators and can be done using automated tools (Moran 2019). Formative Usability testing is likely to be qualitative and even several users can identify most of the problems (Lewis 2014; Moran 2019). On the other

hand, summative Usability testing is more likely to be quantitative and can be used to compare (Macefield 2009; Moran 2019).

Some of the most common quantitative metrics are task success rate and time on task (Macefield 2009; Moran 2019). Macefield (2009) mentions that these are suitable for analysis using statistical methods and can be used to make important decisions like go/no-go for new systems. When processing time on task data, Sauro and Lewis (2016a) suggest using the geometric mean rather than mean or median, as it seems to have a considerably lower error and bias on smaller sample sizes ($n < 25$). However, when comparing two systems, Sauro and Lewis (2016b) do not think it is worth it to use the geometric mean as “two-tailed paired *t*-test is widely considered robust to violations of normality”.

Lewis (2014) contends there is no universal number of participants for summative studies, and one should try to utilise available tools to calculate the required number of participants. Macefield (2009) mentions that a comparative study is essentially a hypothesis test aiming to collect evidence that a software A has better Usability metrics than software B. As such, it is important to ensure the results are statistically significant by having a low probability ($p \leq 0.1$, ideally $p \leq 0.05$) of the *null hypothesis* (Macefield 2009). Since increasing the number of participants should lower this probability, Macefield (2009) suggests one can run the study until statistical significance is reached or the test fails. Based on the previous data across the field, Macefield (2009) hints that a study group should not be smaller than eight participants, with ten to twelve participants being likely to result in statistically significant data. The same suggestion seems to apply specifically to questionnaires examining Usability on the web as well, with SUS achieving the “correct” conclusion 100% of the time with twelve or more participants on data with a large effect size (Tullis and Stetson 2004). The relation between the sample size and the probability a “correct” conclusion is reached involving different questionnaires can be seen in Figure 2.4.

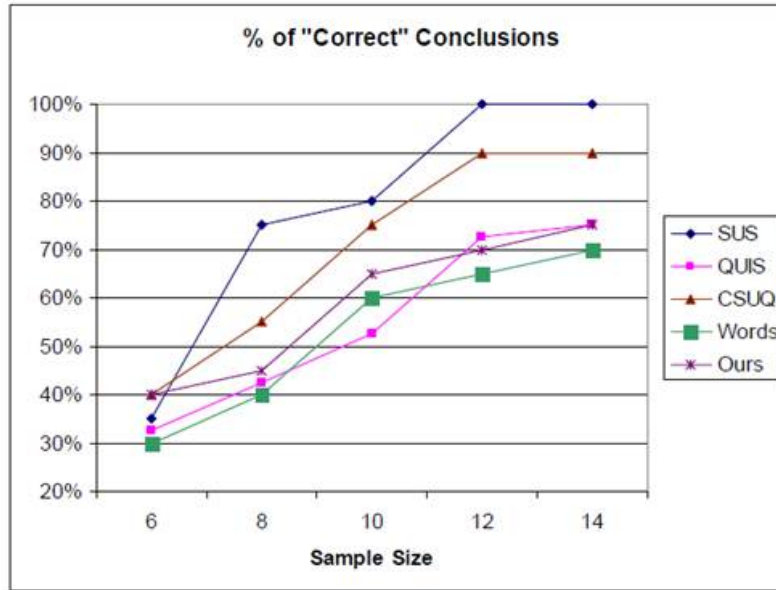


Figure 2.4: Sample size vs “correct” conclusions (Tullis and Stetson 2004).

Macefield (2009) mentions that the size of the effect has a strong influence on the required number of participants, and *power analysis* can be done at different stages to verify the sample size required for statistically significant data or significance in general. However, Dziak *et al.* (2020) argue that using power analysis to decide significance during or after the study (post hoc) should never be done. Alternatives suggested by Dziak *et al.* (2020) for this purpose are *confidence intervals* and *Bayesian analysis*. That being said, Dziak *et al.* (2020) admit Bayesian analysis is not yet very commonly understood. Sauro and Lewis (2016b) concur with the use of confidence intervals when comparing SUS scores or time-on-task times. If a between-subjects comparison is performed, Sauro and Lewis (2016b) suggest the use of a two-sample *t*-test to take into consideration both variation within the group and between the groups. The relevant formula can be seen in Figure 2.5, where \hat{x}_1 and \hat{x}_2 are means of data for systems 1 and 2, s_1 and s_2 are respective standard deviations, and n_1 and n_2 are respective sample sizes.

$$t = \frac{\hat{x}_1 - \hat{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Figure 2.5: Two-sample t -test formula (Sauro and Lewis 2016b).

The confidence interval can then be calculated using the formula in Figure 2.6, where t_a is “the critical value from the t -distribution for the specified level of confidence and degrees of freedom” (Sauro and Lewis 2016b).

$$(\hat{x}_1 - \hat{x}_2) \pm t_a \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Figure 2.6: Confidence interval formula (Sauro and Lewis 2016b).

2.2 Learning Computing Principles

The goal of the following section is to explore the state of remote learning and MOOCs in general concentrating on the problems or potential challenges, the most popular content available for learning low-level computing principles, and its supporting software. Learning low-level computing principles, in this case, means introduction to topics like logic gates, chip design, computer architecture, or low-level code. The assumed demographic is senior secondary or postsecondary students, computing professionals without a formal background in computer science, or the general public interested in computers.

2.2.1 State of Remote Learning and MOOCs

Data gathered by the U.S. Department of Education (2021) show roughly 60% of U.S. postsecondary students took at least some of their classes online in 2021. While this number is down from the COVID-19 pandemic level of

75% in 2020, it shows a clear upward trend compared to 36% in 2019 and 25% in 2012 (U.S. Department of Education 2021). More broadly, the global remote learning market was valued at about 300 to 400 billion U.S. dollars in 2022 and is expected to grow at about 10% Compound Annual Growth Rate (CAGR) by some market research companies (Global Industry Analysts (GIA) 2023; Global Market Insights (GMI) 2023). One of the largest MOOC providers, Coursera, reported the total number of registered users grew 61% Year over Year (YoY) during the pandemic year of 2020 and 30% the year after that (Yu *et al.* 2021).

The effects of the accelerated widespread shift to remote learning were especially apparent during the onset of the pandemic that, at one point, UNESCO (2022) estimates caused more than a billion children ($> 70\%$) to be out of the classroom globally. European Commission (2023) shows disadvantaged children and children in countries with lower adoption of ICT were more likely to be negatively affected. Taking into consideration research from Tadesse and Muluye (2020), these challenges were especially evident in developing countries. Tadesse and Muluye (2020) point out the lack of stable internet connection and devices suitable for remote learning as some of the challenges. In order to support all learners, Ali (2020) mentions, among else, that it is necessary the content is “available on a wide variety of devices and mobile friendly” and accessible despite limited bandwidth or even offline.

Although remote learning has seen a major uptick in students, MOOC dropout rates can be up to 90% (Goopio and Cheung 2021). Common problems with MOOCs leading to a high rate of dropouts include lack of time, difficulty, and bad past experience (including technical problems) with MOOC platforms (Onah *et al.* 2014). Goopio and Cheung (2021) mention several software-related recommendations based on their systematic review of research focused on dropouts:

- Increase interactivity by providing activities like quizzes, games, or video interactions.
- Improve course design by using smaller chunks of content with visualisation of abstract content and real examples.

- Use various media formats and ensure accessibility from mobile devices and with limited internet connectivity.

From the viewpoint of teaching institutions, MOOCs are said to require significant time to both create and integrate (Stikkolorum *et al.* 2014). That being said, Stikkolorum *et al.* (2014) mention the use of MOOCs for software engineering education within higher education is argued to broaden student knowledge and is successfully integrated by some universities into their courses and programmes. Furthermore, data from the U.S. Department of Education (2021) show that, at least in the U.S., online classes serve a considerably higher number of racially diverse students compared to conventional classes.

2.2.2 Low-Level Computing Principles

One of the most popular, if not the most popular, learning materials for learning computing principles is the Nand2Tetris, “taught at 400+ universities, high schools, and bootcamps”, which explains how to build a computer from individual logic gates to a working virtual computer (Shocken and Nisan 2017). In the process, Nisan and Schocken (2021) cover a wide variety of computing principles via practical projects:

1. Construction of simple Boolean gates.
2. Design of more advanced 16-bit chips.
3. Incremental construction of an Algorithmic Logic Unit (ALU) starting with simpler chips like incrementer or half and full adder.
4. Introduction of a clock to chips and construction of registers and Random Access Memory (RAM).
5. Programming using simplified assembly.
6. Construction of the final computer utilizing previously built parts.

7. Writing of an assembler that translates custom assembly language into binary runnable on the built computer.

The learning material is offered as a book written by Nisan and Schocken (2021), as a MOOC³, and in a PDF format⁴. The latter is available freely under a Creative Common Attribution-NonCommercial-ShareAlike 3.0 Unported License. However, it does not contain additional material that builds on the mentioned projects and adds more software-related topics revolving around a Java-like Virtual Machine, a custom high-level programming language and a standard library for a custom Operating System (OS). Regardless of which form of learning material learners choose, they are expected to use the accompanying desktop Java software.⁵

Edybrun (2021) argues the industry now focuses on web-based curricula as the web already has essential accessibility tools, and frameworks like Depth of Knowledge (DOK) lend themselves better to the web environment. More specifically, Edybrun (2021) takes note of the development of more interactive experiences than just “text on a screen” adapted from textbooks. For example, web-based curricula can contain interactive “embedded supports”, and Edybrun (2021) mentions these should be context-sensitive. However, the only kind of examples Edybrun (2021) provides are a tool that provides a breakdown and planning of subtasks and “virtual pedagogical agents” similar to virtual assistants like Alexa or Siri but specific to the pedagogical environment.

The recently created web-based WepSIM (García-Carballeira *et al.* 2019) provides microdesign, microprogramming and assembly language educational simulator and was implemented by authors as an interactive way to practice learned concepts. Compared to Nand2Tetris, which takes a more broad approach and goes through many concepts throughout several layers of abstraction, WepSIM is focused only on a thorough simulation of the CPU and instruction processing. García-Carballeira *et al.* (2019) argue the in-

³Available at <https://www.coursera.org/learn/build-a-computer>

⁴Available at <https://www.nand2tetris.org>

⁵Available at <https://www.nand2tetris.org/software>

troductioin of WepSIM increased the percentage of students taking the final exam and improved grades in assembly and microprogramming exercises. García-Carballeira *et al.* (2019) also add that WepSIM is usable not only as a supplementary tool but also as a standalone learning tool thanks to the bundled examples and help material. Notably, García-Carballeira *et al.* (2019) mention students were able to use WepSim on a wide variety of devices and operating systems, from Windows-NT, Linux, and macOS to Android, iOS, and Windows-Phone.

2.3 Device Type Usage

The following section focuses on the current device type usage and possible trends both in developed and developing countries. The aim is to look at statistics both among the general population and in the education sector. Web clients are of particular interest due to the focus of this thesis. This section is broken down into two sub-sections for developed and developing markets, as the gathered data are considerably different.

Globally, mobile devices account for around 56% of browser market share, up from only 11% in 2012 (StatCounter 2023e). The shift to mobile devices slowed significantly around the year 2017 when they reached a market share of 52%, and there has been no apparent significant change in data since then (StatCounter 2023e). Tablet devices, on the other hand, seemed to have reached a peak of almost 7% in 2014, and their market share has been steadily declining since (StatCounter 2023e).

In terms of OS and screen resolution, the market is now also very fragmented (StatCounter 2023f; StatCounter 2023g). While in the past, just 10 years ago, Windows had a commanding position, now no OS has a majority of the market share and the top four make up the same market share as Windows alone did in 2009 (StatCounter 2023f). StatCounter (2023g) also shows how there is no major screen resolution with a market share larger than 10% the way it used to be and how the variety of screen resolutions has been growing over time.

2.3.1 Developed Countries

Contrary to the global web traffic data, data for Europe and North America still show a gradual increase in the mobile device market share with parity reached within the last few years (StatCounter 2023a; StatCounter 2023b). Notably, Chromebooks have seen considerable growth in North America and Western Europe and accounted for the majority of the upcoming learners from the US K12 market in 2020 and 2021 (Boreham 2019; IDC 2021). However, tablet and Chromebook shipments have since slowed down noticeably (IDC 2022).

In terms of internet smartphone dependency, about 15% of U.S. adults—up from 8% in 2013—report a mobile device is the only way they access the internet (Pew Research Center 2021). Importantly, this group is made up mostly of the 18–29 age group and that is the only age group where we can see a clear trend of increasing smartphone dependency (Pew Research Center 2021). That being said, data collected by Chen *et al.* (2022) at the University of Central Florida show smartphone use for learning has not increased during the years 2016–2021 with laptops still dominating.

2.3.2 Developing Countries

In sharp contrast to Europe and North America discussed before, developing markets like Africa and Asia have a much higher market share of mobile devices at about 70% (StatCounter 2023c; StatCounter 2023d). Additionally, mobile devices overtook desktop devices much sooner, around the year 2015 (StatCounter 2023c; StatCounter 2023d). As far as tablet devices go, there is no considerable difference we can observe with developing markets (StatCounter 2023c; StatCounter 2023d).

Similarly, while developed countries were seeing noticeable growth in Chromebook and tablet shipments, developing countries did not (Boreham 2019). Internet smartphone dependency is very high in emerging economies, with only about 34% of households having access to a desktop, laptop, or tablet device (Silver *et al.* 2019). Importantly, this number varies greatly

between individual developing countries, with India being at the lower end with just 11% and Lebanon being at the high end with 57% (Silver *et al.* 2019). That being said, smartphone ownership is prevalent mainly in the age group of 18–29, with typically less than 40% of people aged 50+ owning a smartphone in developing countries (Silver *et al.* 2019). As far as use in education goes, data seem harder to find, but surveys say the majority of people think smartphones have a positive impact on education and educated people are more likely to own smartphones (Silver *et al.* 2019).

2.4 Text and Code

This section starts with a look at the text as a learning material that humans produce and consume continues with the production of a source code from the viewpoint of a learner, and ends with a short subsection on how the code is processed by a machine.

2.4.1 Producing Learning Material

For the purposes of this thesis, we can simplify the choice of a way to produce learning material by considering two approaches:

1. What You See Is What You Get (WYSIWYG) editors like Microsoft Word and PowerPoint or Adobe Dreamweaver and Captivate.
2. Plain text formats like L^AT_EX, HTML, or Markdown and the tools that are necessary to make them usable.

The first group of tools is commonly mentioned when presenting authoring tools for learning content (Khademi *et al.* 2011). While most of these tools are said to be easy to use and often concentrate on creating courses, the majority of them are also paid, and there are concerns about interoperability (Khademi *et al.* 2011; Shieber 2014). One particular concern mentioned by Shieber (2014) is the lack of output consistency when different users use the large range of functionality of these tools differently.

Compared to the problems with the input of WYSIWYG editors mentioned above, plain text input is by nature backwards compatible and interoperable as it is largely a UTF-8 encoded text (Shieber 2014). However, the complexity of plain text formats can differ considerably. LaTeX, which is used extensively within the scientific community, is commonly understood to be more powerful and complex, carries larger overhead on the input, and can potentially take longer to write (Baramidze 2013; Knauff and Nejasmic 2014; Shieber 2014). One of the most common specifications of Markdown, Commonmark, features a two-page long self-explanatory reference guide⁶ while LaTeX distributes a 31-page long user guide⁷. Moreover, Markdown is said to have a lower overhead than both LaTeX and HyperText Markup Language (HTML) (Shieber 2014) while still permitting extensibility with LaTeX-like features (Shieber 2014) and allowing for potentially better accessibility (Voegler *et al.* 2014). The output format LaTeX and Markdown focus on seems clear from the respective websites. While the LaTeX project website⁸ commonly mentions PDF as the output format, Commonmark specification⁹ directly mentions HTML as the expected output. HTML output is possible with LaTeX as well, but the efficiency of the most commonly used tool¹⁰ is limited as the produced HTML is said to be often substandard (Voegler *et al.* 2014).

2.4.2 Writing Code

According to a survey conducted by Stack Overflow (2023), more complex Integrated Development Environments (IDEs) like Visual Studio Code, Visual Studio, or IntelliJ IDEA are used by more developers than simpler text editors like Notepad++ or Vim. The most popular IDE used by more than 70% of professionals and learners is Visual Studio Code (Stack Overflow

⁶Available at <https://commonmark.org/help/>.

⁷Available at <https://www.latex-project.org/help/documentation/usrguide.pdf>.

⁸Available at <https://www.latex-project.org>.

⁹Available at <https://spec.commonmark.org/0.30/>.

¹⁰LaTeX2HTML available at <https://www.latex2html.org/>.

2023).

Research focusing on the ease of use of sophisticated IDEs, especially among learners, is inconclusive. While a survey from Owoseni and Akanji (2016) suggests learners find sophisticated IDEs difficult to use when learning programming, Vihavainen *et al.* (2014) use session traces to back the opinion that there is no evidence learning to use an IDE is hard.

As far as a web-based IDE goes, Valez *et al.* (2020) explored the idea of using a pre-configured custom IDE that is a low-threshold alternative to common IDEs. While most participants found the web IDE useful and found the fact that it was web-based and did not require installation useful, it was also found to often lack the functionality learners were looking for and was used mostly by more junior students (Valez *et al.* 2020).

2.4.3 Parsing Code

This subsection starts with a mention of a subset of basic graph terminology relevant to this thesis and some basic algorithms and graph properties. After a brief introduction to graph theory, it is possible to talk about tokenizers and parsers from the practical point of view of a software developer.

Graph Theory

A graph is a data structure that consists of a non-empty finite set of *vertices* and a possibly empty finite set of *edges* (R. J. Wilson 2009). Each edge connects exactly two vertices, and if the edges have a direction, we call the graph a *directed* graph (R. J. Wilson 2009).

There are many different ways to traverse a graph. One such way is to use the Breadth First Search (BFS) algorithm. BFS follows the following logic (R. J. Wilson 2009):

1. Take a vertex v .
2. Visit all vertices that have an incoming edge from v .

3. Repeat step 1. with the least recently visited v from which we did not yet visit other vertices.

If we can find a path through the graph that starts and ends with the same vertex, it is said it contains a *cycle* (R. J. Wilson 2009). In case some vertex has no associated edges, it is an *isolated vertex*, and if it is not possible to visit all vertices via edges from each vertex, the graph is called *disconnected* (R. J. Wilson 2009). Figure 2.7 shows an example graph that is disconnected as there is no edge connecting J with the rest of the graph. Additionally, it is cyclic, as vertex DF creates a cycle that could be removed by removing one of the edges DE , EF , or DF .

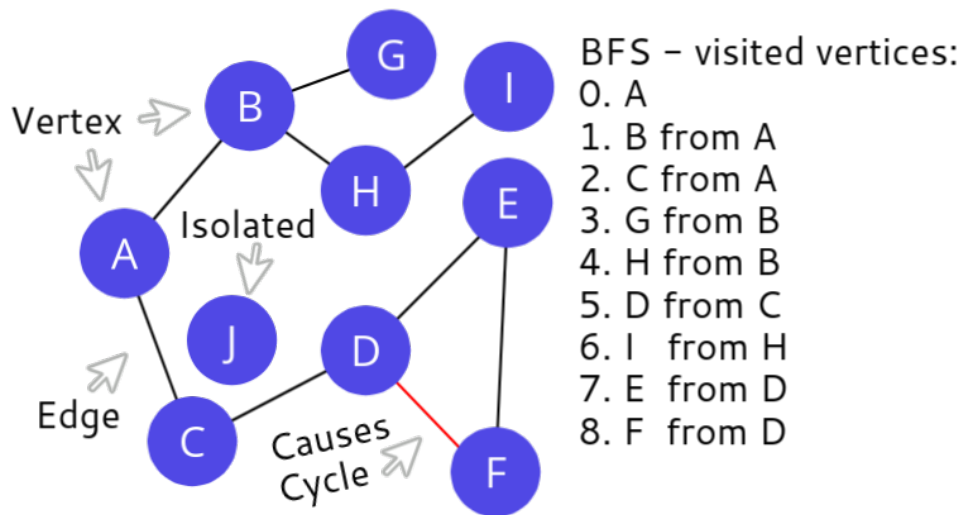


Figure 2.7: Example disconnected graph showing different concepts.

A graph is considered a *tree* if it has exactly one path between any of the vertices, does not contain cycles, and is not disconnected (R. J. Wilson 2009). A tree is called *rooted* if exactly one vertex is chosen as the root and all edges are “directed away from the root” (Baek and Sugihara 2015). Figure 2.8 shows an example of such a tree. Vertices that have at least one outgoing edge are called *internal vertices*, while vertices that have no outgoing edge are called *leaf vertices*. *Children* of an internal vertex are all

vertices that have an incoming edge from that internal vertex (Baek and Sugihara 2015).

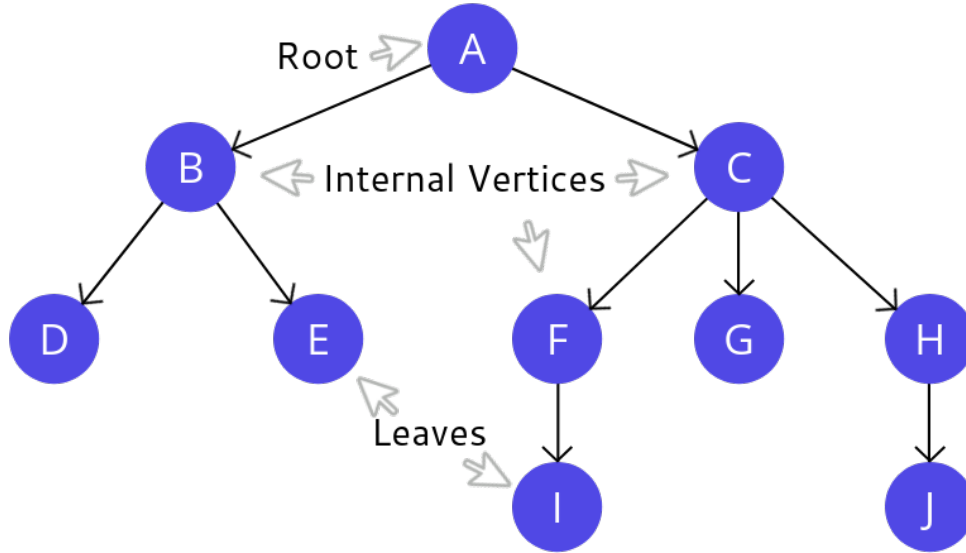


Figure 2.8: Example directed rooted tree.

Tokenizers and Parsers

Parsing is a process of turning a text in a given *language*, like code, into some meaningful object, for example, a program that can be executed (Tomassetti 2017; Kjolstad 2023). A naive approach to parsing is to go through the program character by character using a loop. However, the logic of our language, called *grammar*, would have to be captured within the procedural code, which would make it rather verbose, inflexible, and relatively hard to understand. Instead, the grammar of the language is commonly defined separately and serves as an input for a *parser generator*, which creates the parser. The parser can operate on individual characters, in which case it is said to be *scannerless* as it lacks a *scanner* that would pre-process the text. The scanner groups individual characters into more meaningful tokens and, for the purposes of this thesis, is also referred to as a *lexer* or *tokenizer*.

Therefore, we can say that the processing of the text on the input goes

through three distinct consecutive phases:

1. **Lexing:** A string of characters on the input is broken down using a lexer into a string of tokens.
2. **Parsing:** A string of tokens on the input is turned into a Concrete Syntax Tree (CST) using a parser—if the input is valid.
3. **Processing:** A valid CST, which is mostly just a set of tokens represented using a tree, is simplified into an Abstract Syntax Tree (AST), which is a convenient logical representation of the described object.

From a practical point of view, lexing is typically done using regular expressions as it is the simplest sufficient way to define a regular language used for tokens. Moreover, the processing of parsed tokens can usually be done during the parsing, rather than after, skipping the extra tree traversal. Therefore, parser generators can take: 1. one or more regular expressions defining tokens, 2. grammar, and 3. processing rules on the input and produce encapsulated parser abstraction capable of performing all three mentioned steps. As such, the produced parser takes text on the input and provides the AST on the output.

Since the job of a lexer and parser generator is rather generic, these are abstracted away by a library or a set of libraries. The specification of grammar is left to a human, and the supported features differ between different types of parser generators. A commonly used feature is recursion within the grammar rule. A special case is a left and right recursion when the rule refers to itself on the leftmost or rightmost part of the rule.

Whether a given grammar can be written concisely and the performance of the generated parser is sufficient are typical deciding factors when choosing a parser generator. Conciseness allowed by using features like left and right recursion may require a higher complexity that can mean a worse performance depending on the use case. Simpler and faster parsers called *LL* parse from left to right and produce a tree from the largest logical blocks to the smallest logical blocks called *terminals*. More flexible but potentially slower

are parsers using the *Earley* algorithm. Earley algorithm uses top-down dynamic programming and enables support of both left and right recursion, and allows for ambiguous grammar. While the average performance of an Earley parse is $\theta(n^3)$, where n is the number of tokens on the input, there are multiple ways to achieve better performance (and complexity class): by using left recursion, unambiguous grammar, or an implementation that features optimizations from Leo (1991) if using right recursion in an LL grammar.

2.5 Creating an Open Web Application

Taking into account the vehicle of this research is an open web application that takes inspiration from an existing tool and reuses various open content, this section starts with the exploration of legal basis. Then, it looks at some basic necessities of open software to improve the chances the software succeeds. Afterwards, it covers the most relevant software engineering topics regarding the development and deployment, reliability and security, and testing, that are used later in this thesis.

2.5.1 Legal Background

The following subsection deals with the legal matter required for someone who produces software that is based on an existing work and is assumed to be similarly reused again. Moreover, it looks specifically into exclusions of the copyright law and, even more specifically, law relevant to the reuse of Nand2Tetris mentioned in Section 2.2.2.

Free Versus Open

In relation to software, there are two ways to define “free” (Fogel 2022, Chapter 1). The first definition most people think of is free as in “zero-cost” (Fogel 2022, Chapter 1). The second definition, on the other hand, deals with “the freedom to run, copy, distribute, study, change and improve” (The Free Software Foundation 2023). Because of the ambiguity of the English

language in this case, an alternative way to refer to the software that follows the second definition is to call it an OSS (Fogel 2022).

Licensing

There are significant legal implications of licensing both when producing and consuming third-party software and other assets (Fogel 2022, Chapter 9; Duras 2020, pp. 11-12). The range of licenses one can apply or come across is rather wide, but the main aspect relevant to this thesis is the “openness”—to what extent one can apply the mentioned freedoms. As the author summarised before in Duras (2020), we can group these into three categories:

- **Proprietary:** The least open. Under the complete control of the author, usually confidential and paid.
- **Copyleft Open Source:** Technically open. Users have all the mentioned freedoms. However, they are severely restricted in how they can modify and distribute software. The modified work must be licensed under a compatible Copyleft license—i.e. not proprietary or permissive—and changes often have to be tracked. While it is considered a desired feature, it also greatly influences reuse.
- **Permissive Open Source:** The most open. Users have all the mentioned freedoms. Licenses are usually very short and do not place any restrictions—the users are free to do whatever they want as long as the author is attributed, not held liable for damage, and it is clear what was the original license.

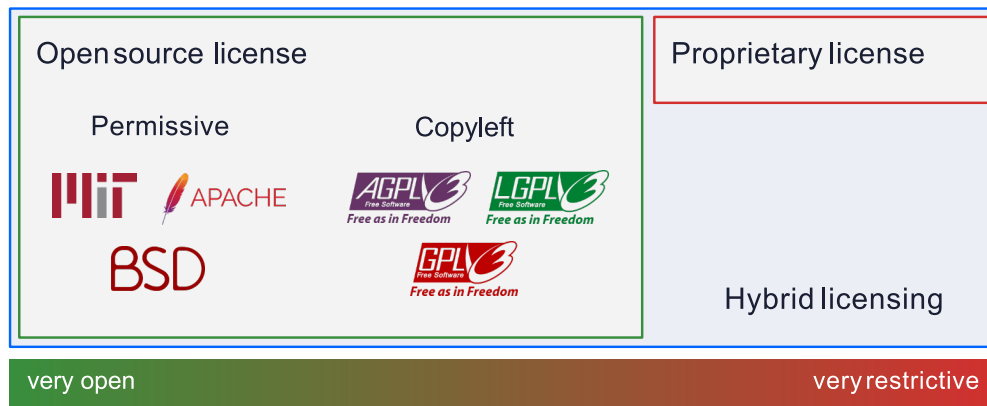


Figure 2.9: License type comparison and examples (Duras 2020).

While the above-mentioned licenses are commonly used to license software, one notable group of licenses commonly applied to other creative works are the Creative Commons (CC) licenses (Hagedorn *et al.* 2011). These can differ considerably in restrictions they impose on users, from just requiring attribution—CC BY¹¹—through disallowing commercial use and mandating the use of the same license—CC BY NC SA¹²—to even prohibiting any adaptation—CC BY NC ND¹³ (Hagedorn *et al.* 2011). As such, they move almost across the whole spectrum of “openness” outlined in Figure 2.9. The non-commercial variant is especially controversial as there are differing opinions on what constitutes commercial use (Hagedorn *et al.* 2011). While for-profit companies using a specific creative work to generate profit are a clear case of commercial use, a work by non-commercial organizations or individuals that is not directly compensated may or may not be classified as commercial use (Hagedorn *et al.* 2011).

Copyright Exclusions

In relation to software, there are some aspects that may not constitute a copyrightable work. A common requirement for work to be copyrightable is

¹¹Available at <https://creativecommons.org/licenses/by/4.0/>.

¹²Available at <https://creativecommons.org/licenses/by-nc-sa/4.0/>.

¹³Available at <https://creativecommons.org/licenses/by-nc-nd/4.0/>.

for it to be original (Fisher 2016). While the level of originality (creativity) required in the United States (US) or Japan is considered to be low, within the EU, it is higher (Fisher 2016). Under the EU law, the work “must reflect the author’s own intellectual creation” and the author must be able to make their own choices (Fisher 2016, p. 443). The latter alludes to external limitations—like when the author has to make certain choices to ensure interoperability (Fisher 2016).

Specifically for programming languages, Rose and Barton (2012) mention the ruling of the EU’s highest court that confirmed the “functionality [...] nor the programming language and the format of data files [...] constitute a form of expression [...] and so [...] are not protected by copyright [...]” (Court of Justice of the European Union 2012). Moreover, it is allowed “to observe, study or test the functioning of that program so as to determine the ideas and principles” (Court of Justice of the European Union 2012). It is based on the argument that any other decision would go against the “necessary innovation and competition” (Rose and Barton 2012). However, it was made clear this does not apply to the actual source code, object code, or the supplied manual (Rose and Barton 2012).

Israel’s copyright law is thought to be originally largely influenced by British and European law, with recent influences from the more lenient approach of the US (Greenman 2007). In relation to software, as is the case in the EU, “the idea behind software is not protected by copyright” (Suslina and Tarasova 2018).

2.5.2 Essential Steps

Fogel (2022, Chapter 2) mentions the following as the first steps relevant to this thesis when creating OSS:

1. **Name:** Choose a name that, among else, can be easily branded, is easy to use even for non-native speakers, and is available as a .com or .org domain.
2. **Mission Statement:** Provide a concise description of what the soft-

ware does so that the user knows “within 30 seconds” (Fogel 2022) if they are interested.

3. **Free:** Make sure to always make it clear the software is free.
4. **Features and Requirements:** Make it clear what the software offers and requires.
5. **Version Control and Bug Tracking:** Use a publicly accessible place like Github to host the code and manage feature requests and bug reports.
6. **Communication:** Provide ways for the users to communicate with the author and between themselves and keep it open and friendly.
7. **Guidelines and Documentation:** Offer developer guidelines and (even if short) documentation.
8. **Demos:** Show the software works by using videos or images.

When it comes to involving others, Fogel (2022, Chapter 8) suggests keeping in mind most OSS participants are intrinsically motivated and “credit is the primary currency”, should be prevented from “territoriality”, and repetitive tasks should be automated as much as possible. Notably, this includes automated testing, which is important for any modern software project (Sommerville 2019, Chapter 9), but Fogel (2022, Chapter 8) argues it is especially important for OSS. Regression tests allow others who are completely unfamiliar—as is often the case with OSS—to contribute to the code without both authors and contributors fearing breakage.

2.5.3 Development and Deployment

Sommerville (2019, Chapter 4) argues one has to pay attention to the architecture of the software as it has a direct influence on the “performance, usability, security, reliability, and maintainability”. While all of the mentioned aspects are important, each software differs in its needs, and so the

prioritisation should follow that (Sommerville 2019, Chapter 4). Regardless, Sommerville (2019, Chapter 4) mentions the ability to break the application down into smaller meaningful parts (components), which are:

1. **Separated:** Components should have a single purpose.
2. **Stable:** Interfaces should be stable and coherent.
3. **Unique:** Avoid duplication of the same functionality.

Specifically in relation to the cloud, Sommerville (2019, Chapter 5) argues the main advantages and features one should be looking for are scalability, elasticity, and resilience. These commonly allow for lower costs, lower time to market, and distribution for users from different parts of the world (Sommerville 2019, Chapter 5). Some notable considerations when choosing cloud that Sommerville (2019, Chapter 5) mentions include cost, developer experience, privacy and data protection, and portability. One of the more recent trends is to utilise serverless computing, where the application runs in a completely abstracted away environment with virtually unlimited computing power and a pay-as-you-go pricing model.

In relation to the delivery of the software, Sommerville (2019, Chapter 10) and Fogel (2022, Chapter 7) concur with the idea the goal is to ship reliable software and to ship often. DevOps, in particular, is highlighted by Sommerville (2019, Chapter 10) as the preferred modern approach to integration and delivery. The basic idea is to utilise automation and data to, among else, integrate and deploy the code—Continuous Integration (CI) and Continuous Deployment (CD).

2.5.4 Reliability and Security

Sommerville (2019, Chapter 8) claims that “high-quality” software not only does what it is expected to do, but is also, among else, reliable, secure, and resilient. Simple examples of how to achieve reliability offered by Sommerville (2019, Chapter 8) are problem avoidance, input validation, and management of problems.

When it comes to security, the main motivation is to prevent reputation, privacy, or monetary damage done to the users and authors Sommerville (2019, Chapter 7). In the case of serverless computing, one significant potential problem is monetary damage from Denial of Service attacks (Kelly *et al.* 2021). Considering the virtually endless scalability and the pay-as-you-go model, instead of endangering availability, the attacks can create an artificial load that can mean significant costs—especially for small projects (Kelly *et al.* 2021).

2.5.5 Testing

Sommerville (2019, Chapter 9) maintains code reviews and (automated) software testing are “essential”. While there are many types of testing, the most relevant ones for this thesis are functional—the system does what it should—and user—the system brings value—tests. The following subsection focuses on functional testing alone as one way of user testing in the form of a comparative study was already covered in detail.

Functional testing involves testing on different levels of abstraction. The lowest level is testing of the smallest encapsulated parts of the software (units) like functions (Sommerville 2019, Chapter 9). A level higher in abstraction are feature tests that cover testing the introduced or modified feature (Sommerville 2019, Chapter 9). The highest level of tests relevant to this thesis are system tests that involve testing the whole product—the combination of its features (Sommerville 2019, Chapter 9). While lower-level testing is almost always done by developers and is easier to automate, highest level testing is usually done by testers and harder to automate (Sommerville 2019, Chapter 9). Additionally, experience from the field suggest lower-level tests are more reliable, easier, and cheaper to maintain than higher-level tests (Cohn 2010; Greenman 2015; Sommerville 2019). As such, a common recommendation is to have a higher proportion of unit tests and a lower number of feature and system tests, as illustrated in Figure 2.10.

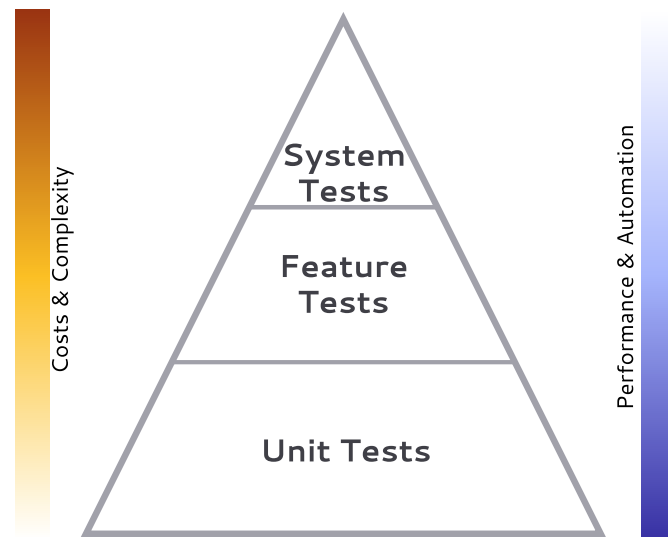


Figure 2.10: Testing pyramid originally proposed by Cohn (2010).

A common question to answer, especially when it comes to unit tests, is how much of the code should be covered. While at first thought we could argue for the highest possible test coverage, research suggests the relation between the code coverage and benefits like lower number of errors is not linear while the added effort is the highest at very high percentages (Antinyan *et al.* 2018). Looking at a large data set of projects at Google, Ivankovic *et al.* (2019) report an average coverage of a little over 80%.

3 Methodology

The main paradigm used for this thesis is Design Science Research (DSR), as currently outlined by Brocke *et al.* (2020). The author of this thesis used existing theories, frameworks, and models to produce and evaluate a new innovative artefact—instantiation, as categorised by Hevner *et al.* (2004).

The created artefact is a web application—referred to as “Proposed Software” for the rest of this thesis. Proposed Software improves on a Java desktop application supplied with a popular learning content called Nand2Tetris¹—referred to as “Existing Software” for the rest of this thesis.

As the first objective is to efficiently implement Proposed Software usable by a wide range of people, this chapter starts with a software design methodology. After that, it covers the evaluation of Proposed Software consisting of the following:

- **Functional Testing**, to verify Proposed Software can perform the selected functions Existing Software can (compatibility and feature parity of selected parts) and that it works as expected (correctness).
- **Comparative Study**, to collect and analyse mainly quantitative and partially qualitative data on the efficiency and usability of Proposed Software and Existing Software while performing the same set of tasks under controlled conditions.

¹Hardware Simulator, available at <https://www.nand2tetris.org/software>.

3.1 Software Design

The design of the software tries to maximise the usability and accessibility of the software based on knowledge from sections on HCI Ergonomics: Section 2.1.2 and Section 2.1.3. As such, the chosen approach to technology choice kept in mind access from as many devices as possible, taking note of data from Section 2.3 and for people with as many limitations as possible. Moreover, considering some common problems when learning are lack of time and technical difficulties (see Section 2.2.1), the chosen path was to streamline all interactions as much as possible. To draw on the knowledge from Section 2.4 and Section 2.5, the design consciously incorporated covered computer science and software engineering practices. Whenever possible, the design and implementation of Proposed Software is compared to Existing Software in terms of the chosen approach, its effects on the mentioned goals, or by using metrics.

The author chose to reuse existing OSS if it fulfilled the requirements mentioned above and if the short-term investment into custom solution would be greater than the time saved long-term. To ensure Proposed Software is legally sound and reusable by others, it did not use any proprietary or copyleft parts (including the GPL-licensed Existing Software). At the same time, it was determined all choices must allow for publication under a permissive Open Source license to maximise possible further reuse. In the case of the adapted learning content specifically, since the original is available under the CC BY-NC-SA 3.0 license², the adapted content had to use the same license. Any chosen technologies, tools, and services had to be free (at least free of charge if not open) and relatively easy to use to improve reproducibility.

3.2 Functional Testing

Multiple types of automated tests were utilized to verify the functionality of Proposed Software.

²See <https://www.nand2tetris.org/license>

Unit Tests were used for the verification of granular parts of the software with clear inputs and outputs. That includes all proposed parsers that accept a text on the input and provide an AST on the output; a factory that accepts an AST on the input and produces chips on the output; elementary built-in and simple custom chips that have a clear interface and a definite number of combinations of inputs and outputs; and other similar reusable abstractions. The exact unit test framework chosen for this purpose was Vitest³, as it is well-integrated with the code bundler Vite⁴ the chosen SvelteKit framework⁵ is based on. The expected coverage on all files providing the mentioned functionality should exceed 80%, which corresponds to the median coverage on projects at Google and is set to prevent an exponential increase in effort required to increase coverage further—see Section 2.5.5.

End-to-End UI Tests verified the software from the viewpoint of a user. In contrast to the Unit Tests, that meant designing automated tests that covered interaction between multiple components and even multiple screens. Examples include navigation through Proposed Software with checks for the existence of essential information, interaction with the IDE, or error handling. The tool of choice for this purpose was Cypress.⁶ No automated coverage as a percentage was calculated for this group of automated tests, as many lines of code would be marked as covered (given the nature of these tests) even though they were not actively tested. Instead, all major paths through the software (use cases) were identified and captured in an automated test unless the return on the effort to do so would not be considered worth it. The reasoning was captured individually for each use case that was not covered.

Accessibility Tests checked automatically for common errors with incorrectly implemented APIs—aiming for “technical accessibility” as mentioned in Section 2.1.3. The chosen conformance target was “generally accepted and recommended” WCAG Level AA (WAI 2014) and WCAG version 2.1 as it is the latest supported version within the WCAG-EM Report Tool as

³Available at <https://vitest.dev/>.

⁴Available at <https://vitejs.dev/>.

⁵Available at <https://kit.svelte.dev/>.

⁶Available at <https://www.cypress.io/>.

of the time of writing. Checks were run on multiple screens in multiple states entered during the execution of Cypress tests to test for possible violations continually and in more scenarios than can be realistically achieved using one-off evaluation. This was accomplished using the JavaScript (JS) package `cypress-axe` that integrates `aXe`—an accessibility testing engine that powers Google’s Lighthouse as mentioned in Section 2.1.4. Considering the limited amount of time and resources, the human testing involved was limited to testing done by the author to produce the WCAG-EM report.

The actual extent and output from all mentioned types of tests, including additional information, where relevant, was captured in Section 5.1.

3.3 Comparative Study

The central hypothesis is that using Proposed Software instead of Existing Software will increase efficiency. Therefore, the main objective of the comparative study was to perform summative testing of Existing Software and Proposed Software in a way that produces comparable data. To do that, the author conducted a remote moderated usability testing and performed a between-subjects comparison.

The chosen methodology considers three kinds of variables to achieve conditions that would lead to the collection of comparable data:

- **Controlled** variables: for example, presented learning content, participant’s profile, and actions performed outside of the measured software.
- **Independent** variable: the used software—Existing Software or Proposed Software.
- **Dependent** variables: mainly the time spent on the tasks and the perceived usability.

Since the only independent variable is the used software and there are two options, participants were split into two groups:

- **Group A** using Proposed Software.
- **Group B** using Existing Software.

Target users of the software are expected to come from different backgrounds corresponding to three target demographics mentioned in Chapter 1: CS students, practitioners with/without formal education, and people interested, but not directly involved, in Computer Science. Participants were recruited from the social circle of the author, and they were subsequently asked if they knew any other potentially interested participants. To control for external motivation and help with the recruitment, one-half of participants within both groups were paid to participate at a rate of 10 euros per hour, as suggested by Prolific.⁷ However, Prolific or a similar crowdsourcing method was not used to recruit participants. All participants were based in an advanced economy country as defined by the International Monetary Fund⁸. Almost all participants were non-native English speakers.

Participants were pre-screened to make sure they:

- Are capable of reading English text of an academic nature.
- Are at least somehow interested in ICT and either want to pursue a career in the field or are already active in the field.
- Have a suitable device to run desktop software.
- Can join a Google Meet voice call and share their screen.

3.3.1 Controlled Variables

Both groups used the same simplified content based on the CC-licensed Nand2Tetris content from Nisan and Schocken (2021). After several rounds of pilot studies, the content and assigned tasks were simplified significantly

⁷Archived Prolific calculator: <https://web.archive.org/web/20221221162903/https://app.prolific.co/calculator.html>

⁸Available at <https://www.imf.org/external/pubs/ft/weo/2022/01/weodata/groups.htm>.

to reduce the time needed to complete the session and to control for the differences in the ability to learn new concepts. As a result, the tasks were mostly mechanical in nature and could be performed just by using the same set of steps that were shown in an example. In the end, participants were expected to spend more time with the software and less time learning and thinking about the presented problems.

The learning material consisted of the same short introduction to Boolean algebra and Boolean gate logic and design. As part of the content, there was a video showing how to solve the first example that differed only in the necessary parts that involved the tested software.

The study was conducted by the author using the same online meeting software, Google Meet. Each participant took the comparative study individually so that the author could fully focus on only one person, and other people did not influence the results.

All participants used their own devices, with only laptops and desktop computers allowed due to the limitations of Existing Software—a desktop Java software. The use of the participant’s device aims to reflect how target users are likely to practice, work on assignments, or explore the content on their own.

To lower the chances the participant would associate the content presented with the author that conducted the study:

- All mentions of the author’s name and website header with logo and navigation were removed from the modified version.
- Both videos featured a voice-over from the same professional voice actor with a neutral American accent, slow rate of speech, and clear pronunciation.

The communication between the participant and the author was kept to a minimum and consisted of the following:

1. The participant is greeted and asked whether they are aware of the author’s research or if they have heard any details about the comparative

study from other participants.

2. The author explains the participant will read a short text, watch a video, and will be expected to complete all presented tasks.
3. The participant is asked to share their screen, and they are informed the screen recording will be kept only until all data are captured and that it will not be shared with any third party.
4. The participant is asked to work on their own and to ask questions only if necessary.
5. The author answers any questions the participant has after the learning content has loaded but before the participant dedicates their attention to the content.
6. The author is strictly muted anytime they are not needed. If necessary, the author answers questions or offers help while the participant solves assigned tasks. This is typically noted in the captured data, as explained when describing dependent variables.
7. After the participant finishes all of the assigned tasks, the author reminds the participant to use the link to open the administered SUS questionnaire. The participant is asked to be as objective as possible, reminding them there are no correct answers and that since the participant does not know which group they belong to, they should not try to knowingly influence the results either way.

Since it is impossible to keep the profile of the participant the same if different groups of target users are to be represented, the author captured the following variables forming the participant's profile:

- **Prior Experience** with the relevant topics on a scale of 0 to 2:
 - 0: No prior experience.
 - 1: Came into contact with the mentioned topics for a short period of time.

2: Has either relevant professional experience or was exposed to the topics for a long period of time.

- **Education** in the field of Computer Science or Computing on a scale of 0 to 2:

0: No relevant education.

1: Taken modules included Boolean logic or programming in general, but not logic gates or chip design.

2: Learnt about Boolean gates and chip design specifically.

- **Relevancy of Occupation** performed by the participant currently or recently on a scale of 0 to 2:

0: No relevant occupation.

1: Works as a part-time hardware designer, software developer or similar.

2: Works as a full-time hardware designer, software developer or similar.

- **Intrinsically Motivated** to participate, as observed by the author on a scale of 0 to 2:

0: Shows no significant interest in the topic. Is hesitant to start reading.

1: Shows some interest in the topic and appears mostly focused.

2: Is visibly engaged and explicitly confirms their interest in the topic.

- **Paid** to participate:

1: Yes.

0: No.

- **Age** on a scale of 18 to 65+:

18–29

30–49

50–64

65+

- **Platform:**

Windows: One of the commonly used desktop Windows NT versions—Windows 7, Windows 8, Windows 10, or Windows 11.

Linux: One of the currently supported versions of any commonly used distribution that uses a floating window manager like X11 or Wayland.

macOS: One of the currently supported versions, regardless of the CPU architecture—x86 or arm64.

An effort was made to distribute participants among Group A and Group B so that their profiles are distributed equally. Any correlations between these profile variables and dependent variables were analysed and revealed during the analysis of the results to hint at possible data skew.

3.3.2 Independent Variable

The link shared with Group A was `side-a.chipsandcode.pages.dev`⁹ and the link shared with Group B `side-b.chipsandcode.pages.dev`¹⁰.

The use of Proposed Software by Group A and Existing Software by Group B was reflected mainly in the way tasks were assigned. Group A had Proposed Software embedded within the same page with the assignment preloaded, as can be seen in Figure 3.1. Group B, on the other hand, considering the limitations of Existing Software, had to start by copying three files representing the same assignment, as can be seen in Figure 3.2.

⁹Archived at <https://web.archive.org/web/20230810031115/https://side-a.chipsandcode.pages.dev/>

¹⁰Archived at <https://web.archive.org/web/20230810031153/https://side-b.chipsandcode.pages.dev/>

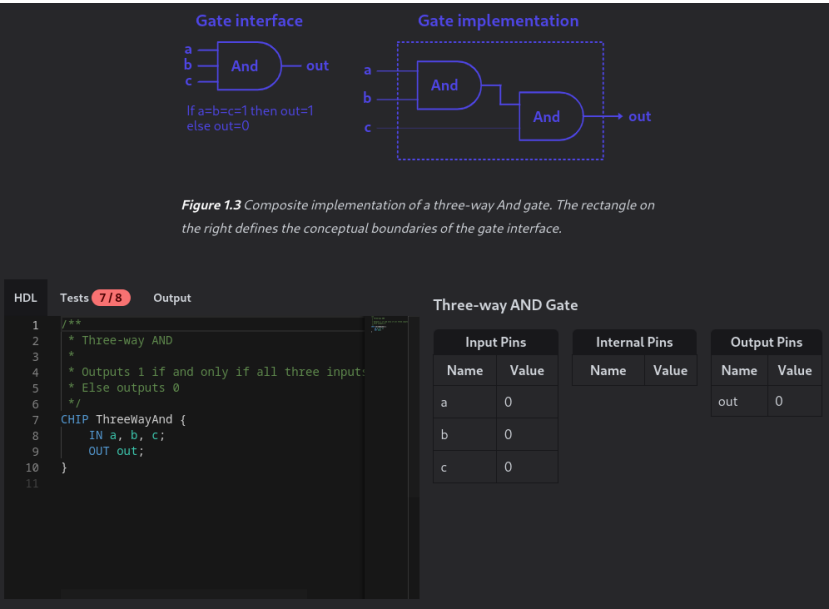


Figure 3.1: Screenshot of Proposed Software within text—Group A.

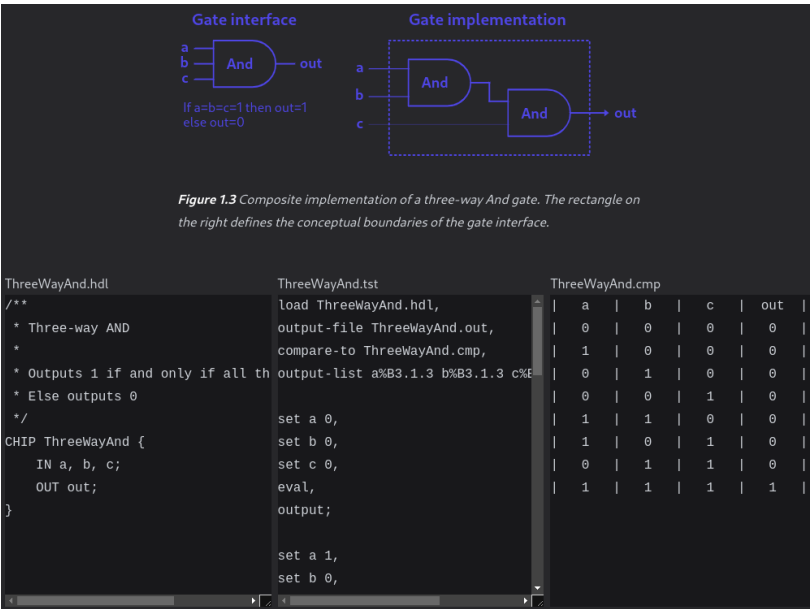


Figure 3.2: Screenshot of Existing Software’s input within text—Group B.

3.3.3 Dependent Variables

The following metrics were measured:

- **Success**—Did the participant complete all assigned tasks—1/0.
- **Preparation Time**—Time required to start working with the hardware simulator in seconds. This includes the download and installation time in the case of Existing Software.
- **Time for 1st Task**—Time required to finish the implementation of a three-way And Boolean gate in seconds.
- **Time for 2nd Task**—Time required to finish the implementation of a Nand Boolean gate in seconds.
- **Time for 3rd Task**—Time required to finish the implementation of a Nor Boolean gate in seconds.
- **Time for 4th Task**—Time required to finish the implementation of a three-way Or Boolean gate in seconds.
- **Confusions**—Number of times:
 - (a) The participant asked for help because they were confused.
 - (b) The author offered help to the participant because the participant had been stuck on the same problem for more than about one minute without any progress.
 - (c) The participant did not perform the needed action in the User Interface (UI) due to a misunderstanding and could not immediately correct themselves.

The time it took the participant to initially go through the learning material and the embedded video was not measured and was excluded from the metrics mentioned above. The time to perform a task was measured from the moment the participant started focusing on the part of the website that

contains the task to the time they left the hardware simulator after all comparisons and optional manual testing was done. These times were manually extracted by writing down the timestamp ranges from the screen recording after the session ended, an example of which can be seen in Table 3.1.

Table 3.1: Partial timestamp data collected from one of the sessions.

Prep		
From	To	Time
0:10:45	0:12:25	0:01:40
0:13:32	0:15:06	0:01:34
Total:		194
First Task		
From	To	Time
0:23:13	0:25:11	0:01:58
0:25:35	0:25:55	0:00:20
0:29:10	0:30:33	0:01:23
Total:		221

While there was no hard time limit, the participant was informed the session would take one hour and was given the option to quit once the allocated time had passed. The chosen approach to time limit aims to represent the way learners at home either have an expectation of how long the task will take or have allocated some time to the task but are willing to go past it if they feel like they are making progress and are close to finishing.

In addition to the mentioned metrics, participants were also asked to fill out the SUS (j. Brooke 1996) questionnaire using a 5-point Likert scale from Strongly agree to Strongly disagree. This questionnaire was chosen based on the amount of supporting data combined with the relatively short length of 10 questions and the relevance of questions to the tested software. The word “cumbersome” was replaced with the word “awkward” to help non-native speakers, see Section 2.1.4.

In the process, unstructured qualitative data were collected. While the comparison did not primarily concentrate on qualitative data, it was mentioned when discussing relevant outcomes or theories in Chapter 6. The

collected qualitative edata included:

- Explanations of confusing situations experienced by participants,
- Overall description of the experience of participants using the system,
- Various feedback.

3.3.4 Data Analysis

To decrease the likelihood of manual errors when evaluating SUS scores, the author obtained the SUS Guide & Calculator Package.¹¹ Based on Section 2.1.4, it was suggested at least 12 participants per group would need to participate in the study if the assumption is the effect size is relatively large.

The measured efficiency is represented by calculating the arithmetic mean of the measured times—individually for each task and added together, and the mean number of confusions. The subjective usability is analysed as a mean of SUS scores for each group. Analysed data are assumed to follow standard distribution and all relevant calculations are done with two-tailed tests.

To determine the statistical power of the gathered data, the results were evaluated using the two-sample *t*-test, where relevant, and included:

- *p*-value, with expected $p \leq 0.1$, ideally $p \leq 0.05$,
- and confidence intervals.

Since the make-up of the groups was not identical, any profile variables that differed considerably were analysed for correlation with the dependent variables using Pearson's correlation coefficient. As the assigned software is assumed to have a strong influence on the dependent variables, correlations are determined for each group individually with:

- $|r| \leq 0.2$ considered as no correlation,

¹¹Available at: <https://measuringu.com/product/suspack/>.

- $0.2 < |r| \leq 0.4$ considered as a weak correlation,
- $0.4 < |r| \leq 0.6$ considered as a moderate correlation,
- $0.6 < |r|$ considered as a strong correlation.

Visually, results are presented using a series of charts and plots:¹²

- A linear chart with the time each group needed to perform subsequent tasks.
- A stacked bar chart showing the difference in measured efficiency.
- A box plot representing the mean and dispersion of the number of times participants got confused using each software.
- A box plot and a position on a percentile curve for SUS scores inspired by visualisations from Blattgerste *et al.* (2022).
- A scatter plot showing the relation between the measured efficiency and SUS score, including colour-coding representing the group to reveal clusters.
- Scatter plots that show possible correlations between the various data points in participant profiles and dependent variables.

3.3.5 Limitations

The comparative study had several notable design limitations which could not be completely addressed:

- **Sampling Bias:** The study participants consisting primarily of the extended social circle of the author and only includes participants who agreed to participate (self-selection bias).

¹²Created using Plotly.js available at <https://plotly.com/javascript/>.

- **Additional Step:** While Existing Software bundle does include a starting point for projects, it does not include the simple exercises chosen for this study. That means Group B participants had to take one extra step to create starting files.
- **Moderator Bias:** The study was moderated by the author rather than a blind data collector.

3.4 Summary

In this chapter, we outlined the two-phase approach chosen for this thesis based on the two distinctive primary research steps of DSR. The first phase involves designing and implementing Proposed Software with the aim to allow the use by as many people as possible and to optimise for usability (and, transitively, performance). The second phase represents evaluation. First, that means functional testing using mainly automated tests to verify and validate Proposed Software. Second, that means a between-subjects controlled comparative study involving Existing Software and Proposed Software.

4 Design and Implementation

The following chapter goes through the most important design considerations and their implementation. Where appropriate, it includes a comparison to Existing Software. Following a top-down approach, it starts with the overall architecture and the build and deployment process, continues with UI and branding/communication aspects, and ends with a description of efficiency, device support, security, accessibility, and learning content details. The source code is hosted at Github¹—referred to as Repository from now on.

4.1 Overall Architecture

As can be seen in Figure 4.1, Proposed Software is primarily a client-side web application. This decision was made for two reasons:

- **Connectivity:** To make sure the application does not depend on an internet connection and is available with poor or even no connectivity. This also means the design should not be disadvantaged in this regard compared to a desktop application.
- **Reproducibility:** No dependence on the cloud means it should be much easier to deploy to any (free) CDN, any simple web hosting, or even run locally or bundled as an Electron²—desktop—or a Capa-

¹Up-to-date state: <https://github.com/durasj/chipsandcode/>. At the time of writing: <https://github.com/durasj/chipsandcode/tree/4576a0b>.

²Available at <https://www.electronjs.org>.

citor³—mobile—application.

To double down on this approach, it is also a Progressive Web Application (PWA), which means it can be installed on any combination of a major internet browser and OS. Thanks to a service worker⁴ that is configured to cache all assets as soon as the user visits the website, all functionality should be available at any point, even once the user loses the connection.

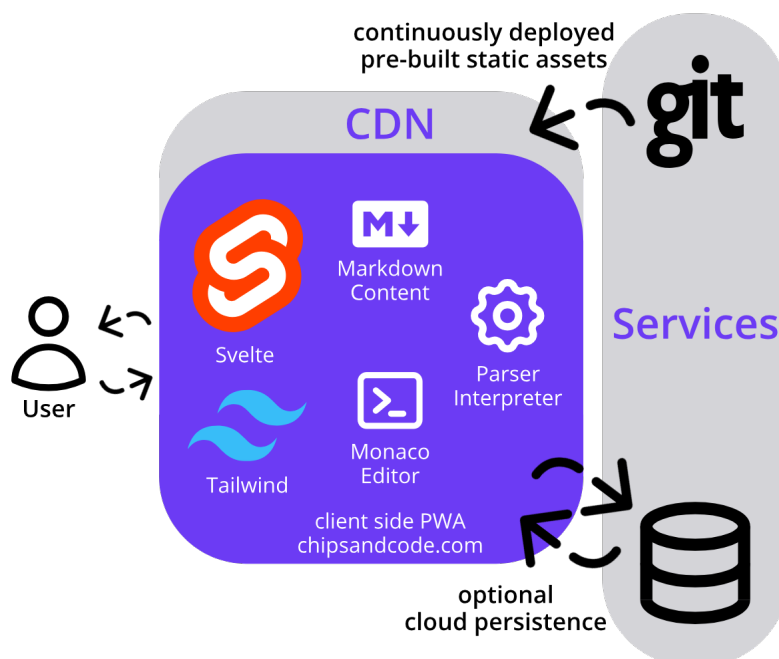


Figure 4.1: High-level architecture diagram.

4.2 Build and Deployment

To enable the static file approach, the whole application has to be built ahead of time during the building phase instead of relying on Server-Side Rendering (SSR). Although Sveltekit⁵ used for this project defaults to platform-

³Available at <https://capacitorjs.com/>.

⁴See https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.

⁵Available at <https://kit.svelte.dev>.

specific SSR⁶, it is possible to configure it to generate static assets and work as a Single Page Application (SPA) for pages that are not completely static.⁷ In this case, the resulting bundle of static files is deployed via Cloudflare Pages,⁸ but there are many equivalent services that can easily host static assets for free. For a proper CD implementation, the build is triggered on push to Repository and every branch is deployed as *branch.chipsandcode.pages.dev*. The *main* branch is mapped to the primary domain name chipsandcode.com.

Optionally, it is possible to persist files to the cloud—in this case, a simple custom-made Representational State Transfer (REST) API written as a serverless application running on Cloudflare Workers.⁹ The API is consciously kept separate from the client-side application, although it does re-use code where it makes sense, e.g. validation schemas.

The same branch-level deployment applies to the server-side code. However, one limitation is the storage is not separated for each branch. Rather, there are only two key-value namespaces¹⁰ for production and development, with the latter used for branch previews as well. Even though the cloud functionality is optional, it was still kept to a minimum (it is only about 200 lines of code) so that it is easily replaceable. Additionally, local development does not depend on Cloudflare as it uses a simulated local environment with a SQLite database.¹¹

All in all, while Cloudflare deployment is mentioned in the supplied README, anybody who wants to contribute or adapt the project does not need to worry about it if they do not want to. The cloud functionality is opt-in even for developers and can be enabled by setting a single *.env* variable.

⁶See <https://kit.svelte.dev/docs/adapters>

⁷See <https://kit.svelte.dev/docs/adapters-static> and <https://kit.svelte.dev/docs/single-page-apps>.

⁸See <https://pages.cloudflare.com/>.

⁹See <https://workers.cloudflare.com>.

¹⁰Using Cloudflare KV, see <https://www.cloudflare.com/developer-platform/workers-kv>.

¹¹Using Miniflare, see <https://github.com/cloudflare/miniflare>.

4.3 User Interface

This section covers key UI aspects. It always starts with design decisions from the user’s perspective before moving on to an explanation of how technology enabled that.

4.3.1 Interactions

In line with the premise the time and difficulty of using a learning tool are important (see Section 3.1), Proposed Software minimises the friction when the user transitions from learning content to learning tool and shortens the cycle of “trial and error”. Figure 4.2 shows a comparison between Existing Software (left) and Proposed Software (right) in terms of steps needed to iterate on an assignment to implement the chip’s Hardware Description Language (HDL).¹² As we can see, with Proposed Software, the flow is linear and with only simple steps like “scroll down” that are natural. This is in contrast to Existing Software, which has a higher number of steps that can introduce opportunities for confusion and mistakes. Some of the simplification comes from the web, as the learning tool can be easily embedded within the content. On top of that, there are two other relevant features:

- **Code Editor Integration:** The tool has a built-in code editor so that the learner does not have to switch contexts. Specifically, it integrates Monaco Editor,¹³ which is the same library underneath the most popular IDE (see Section 2.4). It is additionally configured with custom themes and languages to fit within the content and provide syntax highlighting.¹⁴ Moreover, one more feature that is integrated right into the tool is a visual diff of the output. As can be seen in Figure 4.3, while Existing Software requires the learner to leave the UI and use a Command Line Application (CLI) utility with a confusing

¹²See video for context: <https://youtu.be/mnSuqmvn4hg>.

¹³Available at <https://microsoft.github.io/monaco-editor/>.

¹⁴See <https://github.com/durasj/chipsandcode/blob/4576a0b/src/lib/editor/index.ts>.

output, Proposed Software provides a clear indication of all rows and even individual values that are wrong.

- **Automatisation:** If there is no benefit in having the user perform a manual action, the interface reacts in real time to lower friction. In this case, it means on any change in HDL, Testing Script Language (TST), output, or values of input pins, the interface recalculates the outputs. Also, all output comparisons are evaluated at once so that the user has more information to base their decision on.

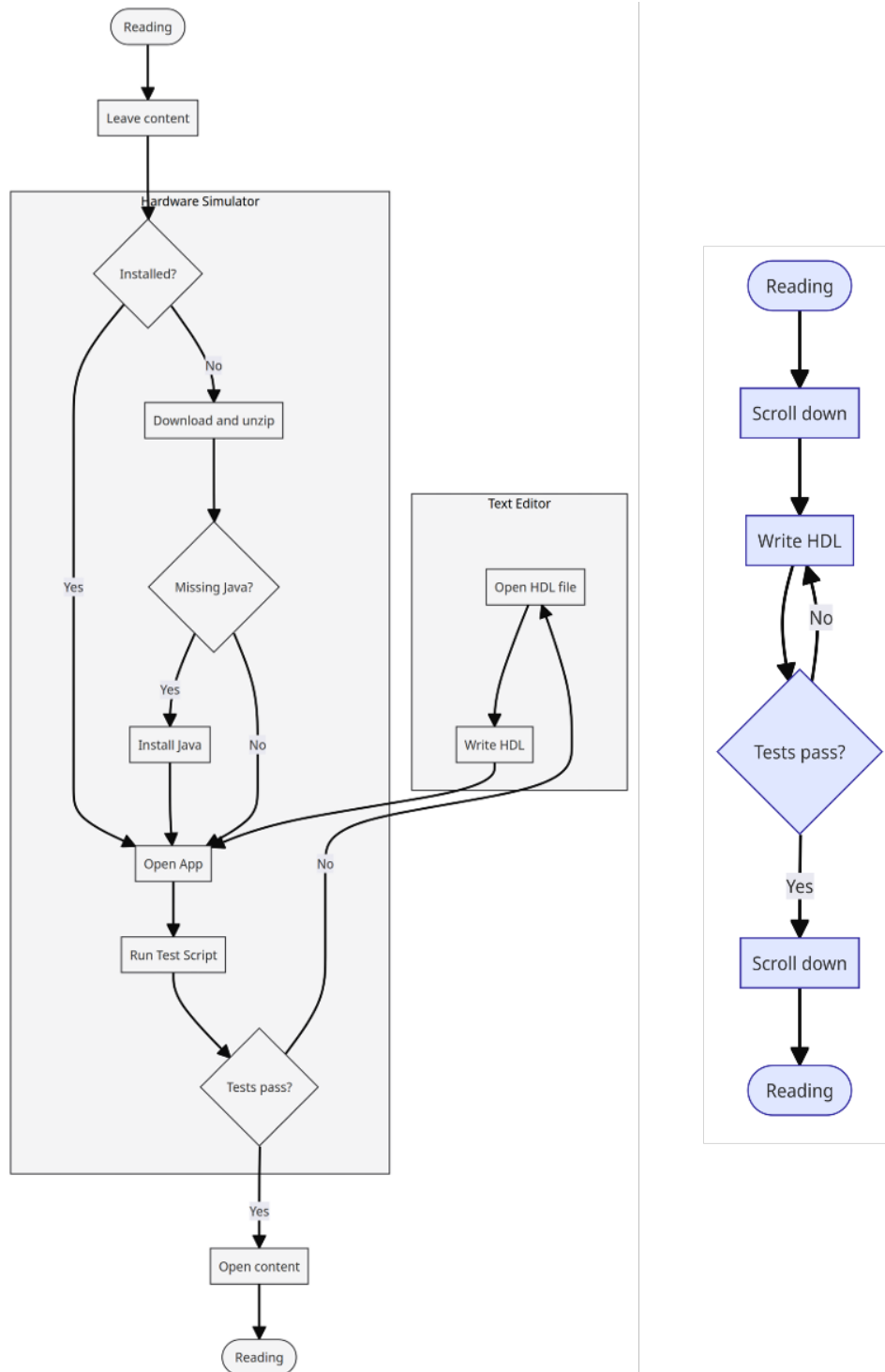


Figure 4.2: Comparison of steps needed to iterate on an HDL assignment.

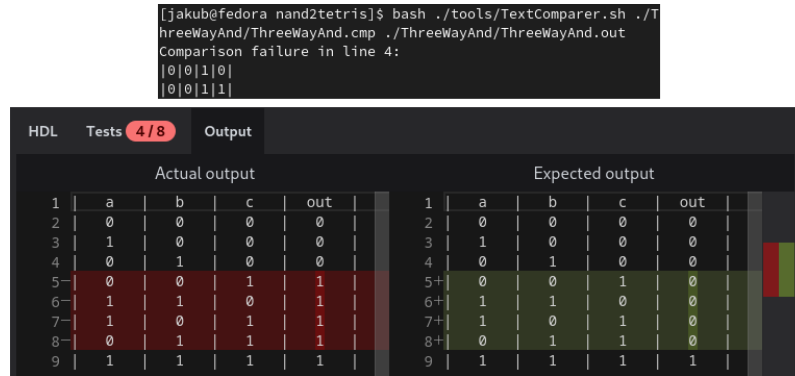


Figure 4.3: Existing Software’s Text Comparer (top) and Proposed Software’s visual output diff (bottom).

4.3.2 Error Handling

Since it is assumed user errors are likely, especially when introducing new concepts, with Proposed Software, the intention is to prevent user errors when possible and try to make them helpful when not. This is achieved by combining:

- **Robust Parser:** Some syntax errors (e.g. due to extra white space) are prevented, while others are streamlined using real-time inline errors with feedback suggesting expected token, as can be seen in Figure 4.4. While Existing Software uses procedural code for parsing, Proposed Software utilises a Regular Expression-based lexer generator¹⁵ and an Earley parser generator¹⁶. Instead of using hundreds of lines of procedural code, Proposed Software uses one file with less than a hundred lines of transparent token definition, grammar, and post-processing code—see Appendix A, B, D, and E.
- **Simple Error Messages:** Similarly to syntax errors, other errors, like errors in logic, are simple and provide details and suggestions. Figure 4.5 shows Existing Software and Proposed Software are comparable,

¹⁵Moo, available at: <https://github.com/no-context/moo>.

¹⁶Nearley, available at: <https://github.com/kach/nearley>.

but Proposed Software offers a more focused and detailed (notice the mention of input pins) error message and especially a suggestion. As can be seen in Figure 4.6, Proposed Software creates a graph with pins as vertices and edges that can reference chips. Then, it is possible to traverse the graph from inputs using BFS and determine which connections are invalid or if the implementation contains cycles.

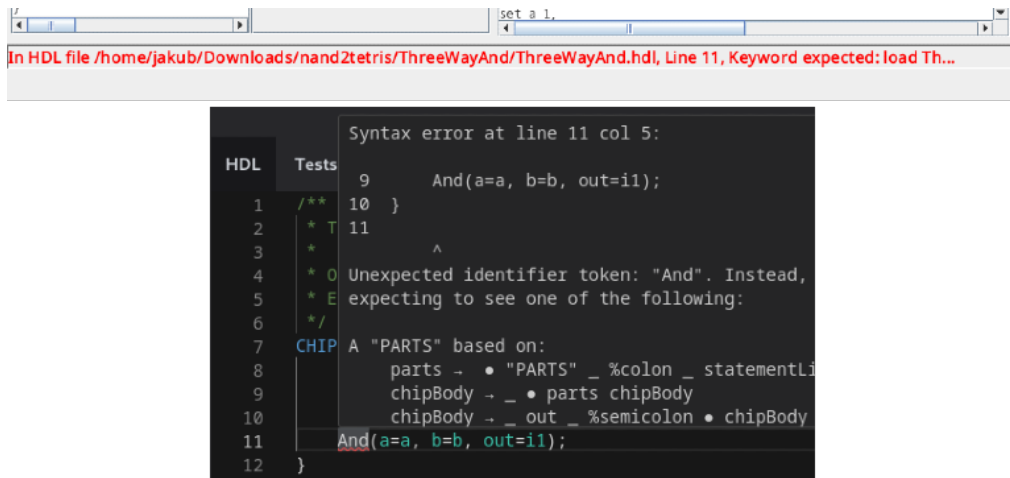


Figure 4.4: Existing Software’s (top) and Proposed Software’s (bottom) syntax error handling.

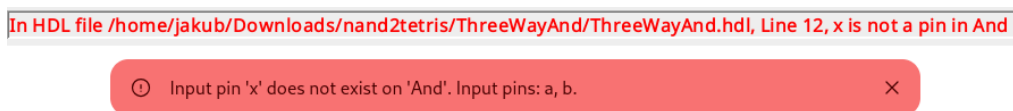


Figure 4.5: Existing Software’s (top) and Proposed Software’s (bottom) pin connection error handling.

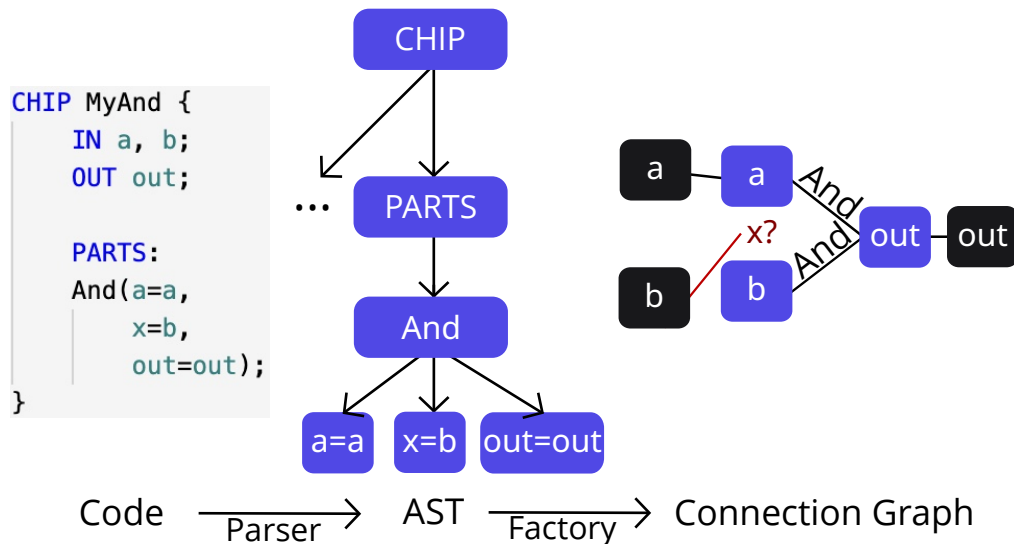


Figure 4.6: HDL processing from text to chip connection graph.

4.4 Branding and Presentation

In line with the advice explored in Section 2.5.2, the project is available under the name Chips and Code at <https://chipsandcode.com>. It makes use of a consistent brand identity—logo, colours (accessible combination of violet and orange), and fonts. There are two different mission statements depending on the target audience:

1. **Wide Audience:** “Wondering how computers work? Find out by embarking on the journey of building your own computer from scratch, from chips to code. No prerequisites—start for free from the browser.”
2. **Technical Audience:** “Chips and Code—reimagined Nand2Tetris tools powered by the modern web with better User Experience.”

Both <https://chipsandcode.com> and Repository mention the project is in a “preview” or “work in progress” state. Repository also features a list of current and planned features and very simple documentation on how to get

started with the development and how to deploy. Moreover, Repository contains an animated demo of the software. Features and bugs are managed in Repository, which also offers a simple discussion forum¹⁷ prepared for communication. Basic information and links are also available on the website's About page.¹⁸

4.5 Efficiency

One of the aims was to keep the application as small and efficient as possible. This brings benefits not only for less equipped users (e.g. from developing countries) with lower power devices and limited network connection but also for users with powerful client devices and fast internet connection for whom the application loads and responds instantly. There are several technologies that enabled this:

- **Svelte and Tailwind:** Although each has a different responsibility—the former is a front-end framework¹⁹ and the latter a Cascading Style Sheets (CSS) framework²⁰, both share the same philosophy: no overhead of the library. Svelte achieves this by generating tailor-made optimised code during the build instead of using a run-time rendering library. Tailwind uses atomic styles—a subset of available styles is reused throughout the app, and the rest is removed during the build.
- **Tree Shaking, Chunking, Prerendering:** All imports are automatically analysed by Vite,²¹ unused parts of the code are removed (tree shaking), and the rest is split into chunks for faster initial loads so that larger features like Monaco Editor do not block the initial load. Prerendering done at build time also speeds up the initial load as the page loads instantly as a simple HTML and then is hydrated via JS.

¹⁷See <https://github.com/durasj/chipsandcode/discussions>.

¹⁸See <https://chipsandcode.com/about>.

¹⁹Available at: <https://svelte.dev>.

²⁰Available at: <https://tailwindcss.com>

²¹See <https://vitejs.dev/guide/features.html>.

- **Small Assets:** Any hard-earned efficiency gains could be easily ruined with unoptimised multimedia assets like images. Therefore, whenever possible, Proposed Software uses vector graphics (SVGs), and when not, modern efficient AVIF or WEBP image formats with JPEGF fall-back using the picture element.²² This is done automatically and includes resizing and low-quality variants used for the initial load.²³ Proposed Software also does not use custom fonts but instead relies on a system font stack.²⁴
- **Preload:** Apart from the pre-caching done using the previously mentioned service worker, all crucial modules are marked for preload²⁵ so that the browser can process them ahead of time in parallel. Additionally, anytime the user hovers or starts a touch gesture on an internal link, it is preloaded with high priority in a split second before the tap/click actually happens.²⁶

The result of these optimisations can be not only observed subjectively but also, to some extent, captured in the size of the page or the whole application and using performance tests. Figure 4.7 features a chart showing the size of Proposed Software compared to Existing Software. Keep in mind that Proposed Software includes learning content and a code editor with syntax highlighting and does not require any additional third-party runtime like a Java Virtual Machine (JVM).

While the zip file with Existing Software is 700 kB, we can also consider the size of the website (1.7 MB²⁷) and the Google Drive link (700 kB²⁸) we likely have to visit to download it. In case the user has to download Java

²²See <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/picture>.

²³Using *svelte - img*, available at: <https://github.com/zerodevx/svelte-img>.

²⁴See <https://bitsofco.de/the-new-system-font-stack/>.

²⁵See <https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel/modulepreload>.

²⁶See <https://kit.svelte.dev/docs/link-options>.

²⁷Transferred over the network. Assuming we land directly on <https://www.nand2tetris.org/software>.

²⁸Transferred over the network. When visiting linked https://drive.google.com/file/d/1xZzcMIUEtv3u3sdpM_oTJSTetpVee3KZ/view without a Google account.

SE to run Existing Software, it adds from 62 MB on Windows to 93 MB on Linux. If we assume a poor internet connection with 1 Mb/s download speed, both Existing Software and Proposed Software can be downloaded within 30 seconds. Java SE 8, on the other hand, would take up to 12 minutes. However, in terms of size, to put it into perspective, even within developing countries, 100 MB of data costs at most 1% of average monthly income (Rodriguez and Woodhouse 2022).

Although the actual size of Proposed Software is about 5 MB (most of which is Monaco Editor), only about 1.5 MB is actually transferred over the network.²⁹ That being said, since Proposed Software does not yet contain all of the expected learning material and functionality, we can assume the final size to be 2 MB. That is comparable to an average website (Indigo and Smart 2022)—a single page of a website, not a whole PWA with a code editor, etc. The Lighthouse performance report shown in Figure 4.8 captures a score of 100/100 for the most complex page with long learning content and embedded Hardware IDE.

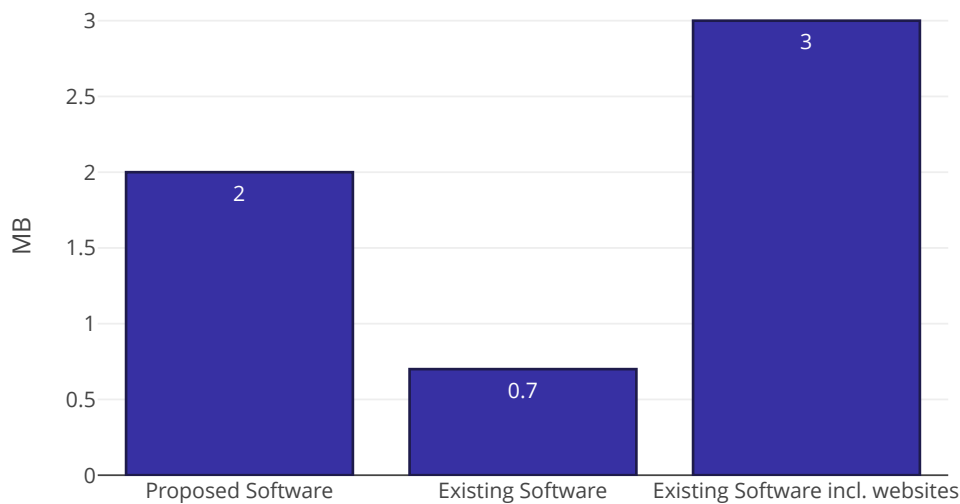


Figure 4.7: Size comparison between Proposed Software and Existing Software.

²⁹When using Brotli supported by over 95% web clients globally, see <https://caniuse.com/brotli>.

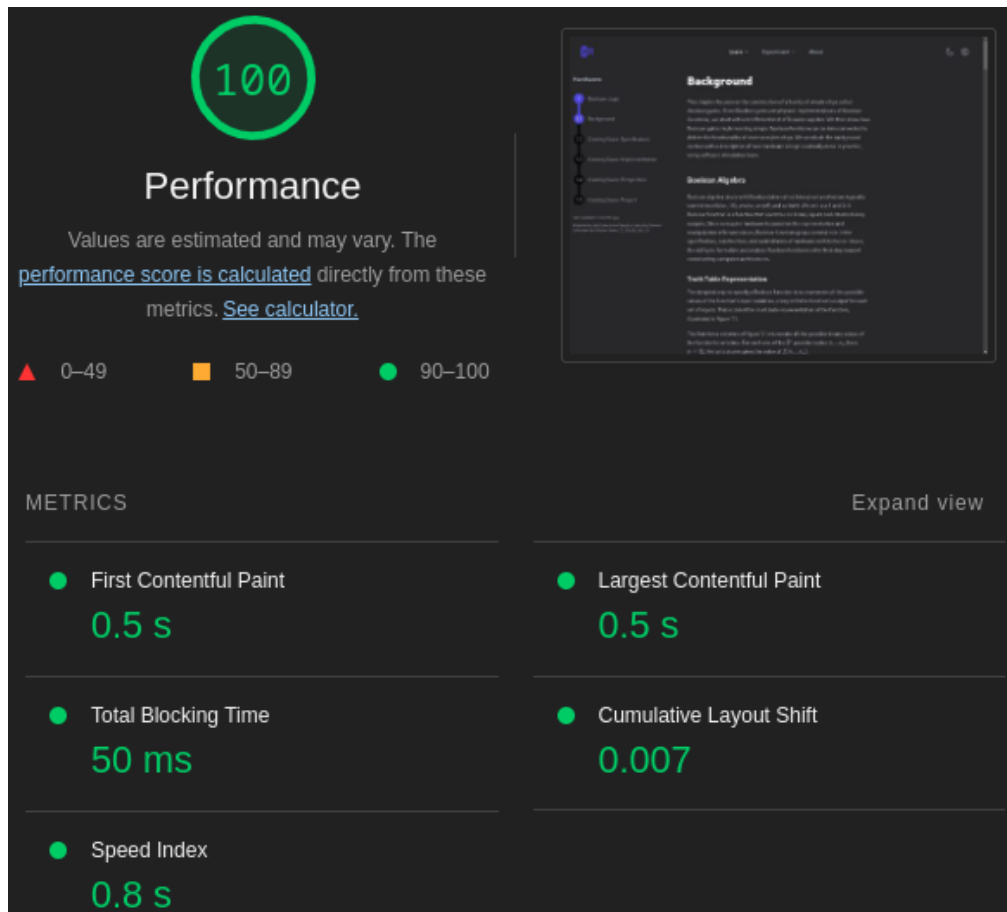


Figure 4.8: Lighthouse performance report for Proposed Software.

As far as the parser goes, despite the fact it uses the Earley algorithm with a heavy use of right recursion and even considers white space during parsing (as opposed to disregarding it during lexing), it still takes within 1 ms per parse. This was measured³⁰ on Appendix C—a medium complexity (in a learning context) example with AST output of 3 kB (JSON). The time varied between 0.09 ms on a high-end desktop with i7 13700 and 0.78 ms on an older midrange smartphone with Snapdragon 662.

³⁰Using the high-precision Performance API, see <https://developer.mozilla.org/en-US/docs/Web/API/Performance/measure>.

4.6 Security

Even though security was not the highest priority for Proposed Software, it includes essential security measures. Deployment-level features include DNSSEC, HTTPS, and mandatory GPG-signed commits that prevent automated deployment of spoofed commits from the main branch. As for Denial of Service attacks, Proposed Software relies primarily on Cloudflare Pages that offer unlimited free bandwidth and requests. As far as the optional cloud functionality goes, in the worst-case scenario, it would stop working for the day after reaching 100k requests.

Only anonymous sessions were implemented, which means risks related to sensitive data were minimal. Sessions were using *HttpOnly* and *Secure* cookies to mitigate session fixation. Parsers and interpreters were not using *eval* or similar techniques that would allow the execution of arbitrary code.

4.7 Accessibility

Proposed Software was built from the beginning to be accessible to the widest range of users, and this was reflected during the design, implementation, and testing. During the design and implementation, that meant focusing on the simplicity of the UI, paying attention to proper use of semantics, use of Accessible Rich Internet Applications (ARIA) APIs if necessary, and continuous use of automated checks³¹ and semi-automated tools like aXe DevTools³².

³¹Using svelte-check, available at: <https://www.npmjs.com/package/svelte-check>.

³²Available at: <https://chrome.google.com/webstore/detail/axe-devtools-web-accessib/lhdoppojpmngadmndnejefpokejbdd>.

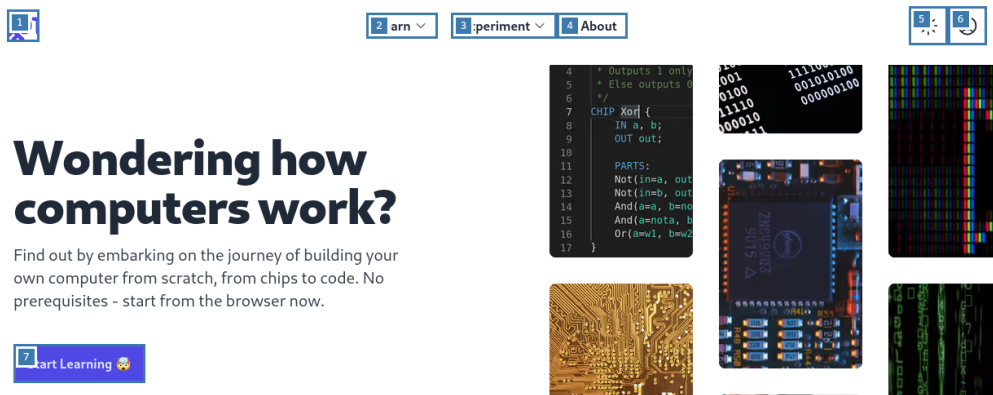


Figure 4.9: Example use of aXe DevTools to verify focus order.

All accessibility-related features of Monaco Editor³³ were bundled, and the editor should be reasonably accessible despite its complexity. Similarly, despite the fact Proposed Software uses client-side routing, navigation triggers an announcement of new pages and handles the focus.³⁴ Proposed Software also contains an accessibility statement, which, among else, prompts users to make use of the accessibility tools built into the browser. During the automated testing, the used approach follows the premise the UI must be easy to target and control using the same semantics utilised by accessibility tools. For example, all elements are findable using their text content or labels³⁵ instead of using “technical” selectors like identifiers or identifiers dedicated to automated tests only.

4.8 Device Support

Designing for a wide range of users means also designing for a wide range of client devices. Thanks to the design decision to implement Proposed Software as a client-side web application, it can be run on a rather wide range of devices by default, even without any optimisations. To make the

³³See <https://github.com/microsoft/monaco-editor/wiki/Monaco-Editor-Accessibility-Guide>.

³⁴See <https://kit.svelte.dev/docs/accessibility>.

³⁵Using Testing Library packages, see <https://testing-library.com/docs>.

using these different devices comfortable, during the development all pages were tested to ensure they worked properly on a realistic range of screen widths by slowly scaling the width from 400px (or equivalent) to 1920px (or equivalent). Figure 4.10 shows an example of the Hardware IDE page on a smartphone.

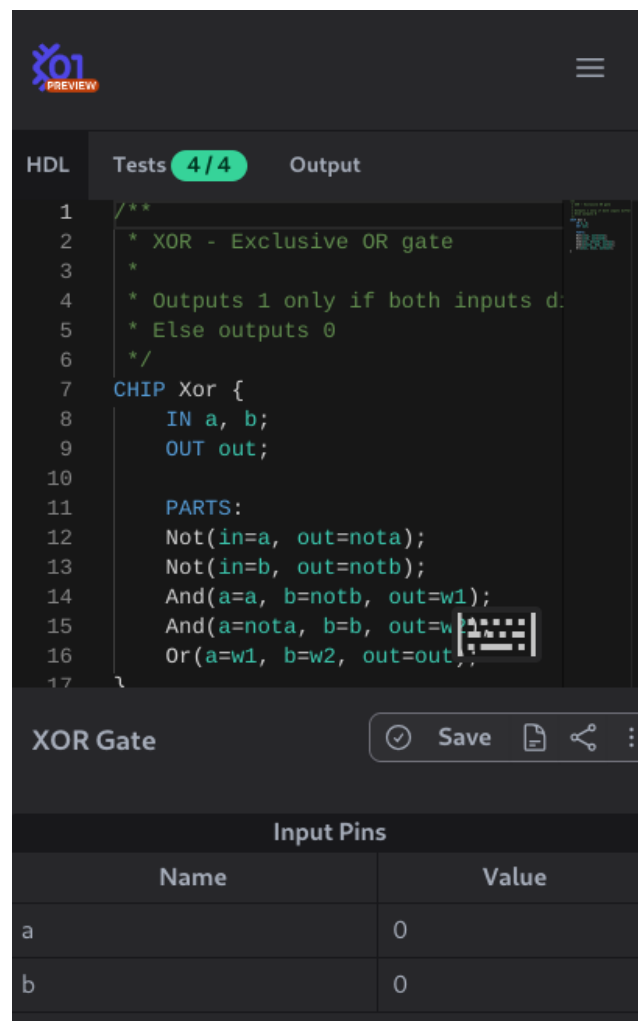


Figure 4.10: Hardware IDE viewed from a smartphone.

The high market share of modern web browsers and the UI responsiveness mean that Proposed Software should be usable by the vast majority of internet users—excluding only users of unsupported browsers like Internet

Explorer that now account for $< 1\%$.³⁶ Compared to that, based on the statistics gathered in Section 2.3, Existing Software would be globally accessible only from 44% of devices used to browse the web. However, if we consider only users that rely on smartphones within developing countries, this means one-fourth of the population is unable to access Existing Software.

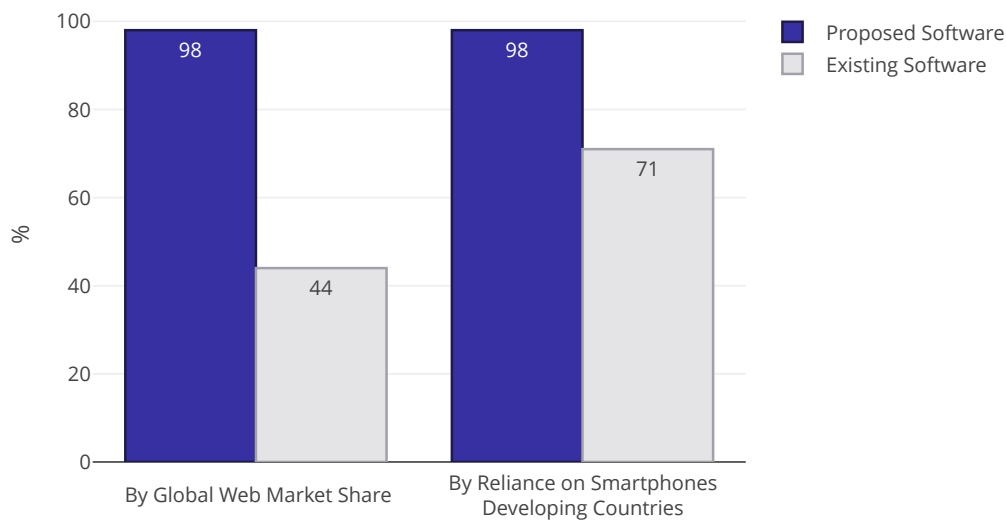


Figure 4.11: Probable percentage of supported devices.

4.9 Learning Content

Proposed Software includes a way to produce learning content. This is done simply using version-controlled Markdown files that can be seen in Repository.³⁷ Based on the review done in Section 2.4, Markdown documents should be easier to write while still preserving the advantages of the plain-text input format. Since Markdown does not allow for any custom styling, all of the content is presented consistently and can be used long-term in different forms. Moreover, the Markdown content implementation still allows for powerful additions, namely:

³⁶See <https://gs.statcounter.com>.

³⁷See <https://github.com/durasj/chipsandcode/tree/4576a0b/src/routes/learn/hardware/boolean-logic> for examples.

- **Math Expressions:** Proposed Software supports the use of TeX-like inline mathematical notation.
- **Embeds:** Videos or developed tools like the Hardware IDE can be easily embedded.

From the technical point of view, this is made possible thanks to Markdoc,³⁸ which is used in a custom Vite plugin that parses and transforms Markdown documents to a renderable tree usable with Svelte. Math support is made possible thanks to KaTeX,³⁹ which uses MathML and custom fonts with CSS to render processed input. Embeds, specifically, are supported via custom tags—an extension to the Markdown syntax.⁴⁰ These features can be seen in the Markdown example in Figure 4.12.

Example: Building a Nand Gate

```
Let _it_ be: `Nand(a, b) = Not a And b`.  
Just some random math expression:  $f(x, y)$ .  
  
{% EmbeddedHardwareIDE id="nnDG6JRQjL0aNVb7AJHnmZrv02pHIINF" %}
```

Figure 4.12: Example of learning content written in Markdown.

4.10 Summary

This chapter has shown the overall architecture of Proposed Software, focusing on the benefits of client-side implementation, use of modern APIs and acceleration of development using OSS. It explained some important user-facing design decisions and how technology enabled them. Notably, it has shown how Proposed Software has support for the vast majority of devices as a web application.

³⁸Available at: <https://markdoc.dev>.

³⁹Available at: <https://katex.org>.

⁴⁰See <https://markdoc.dev/docs/tags>.

5 Evaluation

Closely following Chapter 3, the following chapter shows the extent and results of the performed tests. It starts with the functional tests checking for correctness, continues with the accessibility evaluation, and ends with the comparative usability study.

5.1 Functional Testing

This section is divided into two parts reflecting significant differences in the types of tests used: low-level unit tests and high-level UI tests. Both sections outline the implementation, scope, and achieved results.

5.1.1 Unit Tests

To cover all important reusable parts (units) of the software, all files within the `/src/lib` directory¹ had a corresponding `.test.ext` file. The only exceptions were: Svelte components that would be better suited for Cypress Component Testing²—which did not yet have stable support for Svelte, type definition files, and files that were tested indirectly.

Figure 5.1 shows an example unit test that belongs to the test suite for output processing. It checks the output object can be created from text input, allows observing the loaded values, and can be serialized back.

In total, 181 unit tests spread over 16 test suites covered over 95% of the

¹Available at github.com/durasj/chipsandcode/tree/4576a0b/src/lib.

²Available at <https://docs.cypress.io/guides/component-testing/overview>.

1500 targeted lines as can be seen in Figure 5.2.

Notably, excluded from this report is the coverage of all HDL and TST files making up the built-in and Project 1 chips from Nand2Tetris. The built parsers were tested by using all (dozens) built-in and Project 1 files on the input with the expectation the output matches snapshots. For example, for HDL, these can be seen in the directory `/src/lib/editor/hdl/testing`. The same applies to the introduced TST parser.

Moreover, all produced AST files were used to test the chip factory to verify it can process them. This input was the same output parser tests produced, to prevent duplication. One important limitation of this approach is these chip factory tests are no longer isolated as they require to be run after the parser tests. However, in practice, after the first run, parser snapshots are persisted and do not change, which makes it, at most, a possible source of frustration for the developer adding new relevant tests.

```
it('Can be created from existing output text', () => {
  const output = Output.fromText(`|  a  |  b  | out |
|  0  |  0  |  1  |
|  1  |  0  |  1  |`);

  expect(output.getRow(0)).toStrictEqual(
    [false, false, true]
  );
  expect(output.getRow(1)).toStrictEqual(
    [true, false, true]
  );

  expect(output.getText()).toBe(`| a | b |out|
| 0 | 0 | 1 |
| 1 | 0 | 1 |`);
});
```

Figure 5.1: Example unit test.

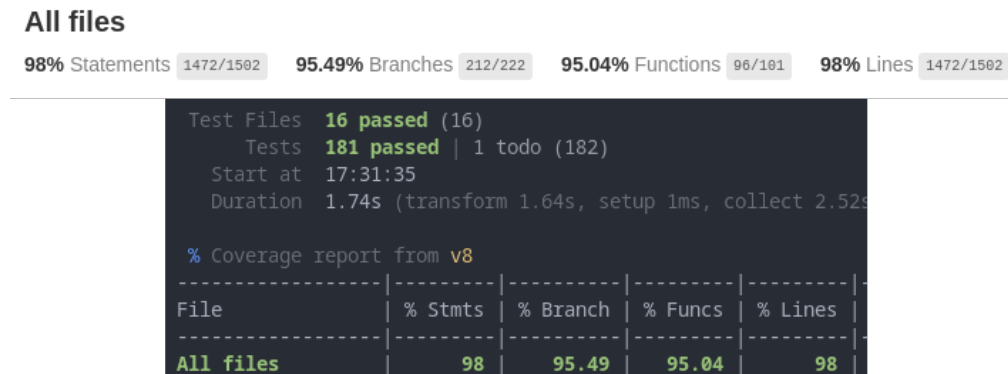


Figure 5.2: Testing framework run output and coverage report.

5.1.2 UI and Accessibility Tests

Following the chosen methodology, the first step to writing UI and accessibility tests was to identify all important use cases:

1. User visits the home page, reads the welcome text, and uses the call-to-action button.
2. User navigates to the learning content using the main menu.
3. User navigates to the next section of the learning content using the side menu.
4. User reads the learning content and navigates to the subsequent section using the “next” button.
5. User navigates to the previous section using the “previous” button.
6. User switches from the light theme to the dark theme while looking at the learning content.
7. User interacts with the embedded Hardware IDE.
8. User navigates to the Hardware IDE using the main menu.
9. User loads the example Xor chip in the Hardware IDE.

10. User changes the input for the Xor chip in the Hardware IDE and observes the result.
11. User changes the Xor chip's definition in the Hardware IDE and observes the result.
12. User changes the Xor chip's test script in the Hardware IDE and observes the results.
13. User changes the Xor chip's expected output in the Hardware IDE and observes the results.
14. User switches from the light theme to the dark theme while at the example Xor chip.
15. User creates a new Experiment.
16. User changes an Experiment name.
17. User copies the link to the Experiment using the share functionality.
18. User downloads and deletes an Experiment.
19. User opens the About page using the main menu, reads the content, and opens links.
20. User opens the sign-in page and reads the text.

These use cases were grouped by the screen they were related to and implemented as Cypress tests. As part of the test, all pages are checked for accessibility violations.³ Additionally, the same pages were also checked for visual regressions using a plugin⁴ that calculated the difference between the captured and persisted screenshots. Importantly, all tests are completely isolated, and all API calls were stubbed to ensure there are no side effects.

³Using aXe, see <https://github.com/dequelabs/axe-core/blob/0316e72/doc/rule-descriptions.md> for the list of performed checks.

⁴A Cypress plugin, available at: <https://github.com/FRSOURCE/cypress-plugin-visual-regression-diff>.

Cypress automatically clears any state produced within the browser, but any state other side effects would create would not be cleared.

Figure 5.3 shows an example Cypress test that belongs to the test suite for the home page. It opens the page, checks it contains the heading, performs visual regression tests and accessibility checks, and verifies the call-to-action button redirects correctly.

Altogether, the software passed 19 tests spread over five test suites as can be seen in Figure 5.4.

```
describe('Home', () => {  
  it('Contains welcome message and link to content', () => {  
    cy.visit('/');  
    cy.injectAxe();  
  
    cy.contains('Wondering how computers work?');  
  
    cy.matchImage();  
  
    cy.checkA11y();  
  
    cy.contains('Start Learning').click();  
  
    cy.location('pathname').should('eq',  
      ↪ '/learn/hardware/boolean-logic');  
  });  
});
```

Figure 5.3: Example simplified Cypress test.

Spec		Tests	Passing	Failing	Pending	Skipped
✓ about.cy.ts	00:04	2	2	-	-	-
✓ experiment.cy.ts	00:38	9	9	-	-	-
✓ home.cy.ts	00:04	1	1	-	-	-
✓ learn.cy.ts	00:20	6	6	-	-	-
✓ profile.cy.ts	00:02	1	1	-	-	-
✓ All specs passed!	01:10	19	19	-	-	-

Figure 5.4: Output from Cypress run.

As mentioned in the methodology, not all use cases were covered. These consisted of the following known use cases:

1. **Chip Download:** This functionality was not covered as it is not trivial to test downloads in Cypress, especially if we consider it is a zip file that would have to be opened and inspected.
2. **Dark Mode:** While this functionality is covered using visual regression tests to some extent, since it can be activated anywhere, it would require testing the same functionality in both modes everywhere. As this functionality is considered a “nice to have” opt-in feature, it was not considered worth it.
3. **Responsivity:** Similarly to the previous point, the application can be used in all kinds of environments with a basically unlimited range of screen sizes. While testing the same functionality on a large range of screen sizes could be valuable, it was not possible with the limited time allocated to this thesis.

5.2 Accessibility Evaluation

First of all, accessibility of the website was tested using aXe core⁵ during the Cypress test run. In total, seven invocations to test accessibility of various

⁵Available at <https://github.com/dequelabs/axe-core>.

pages in several states were performed, and all passed. There were three exceptions where accessibility checks had to be disabled:

- **Empty Table Header** rule, which was confused by the use of MathML in table headers.
- **Landmark Unique** rule, which was getting triggered on Monaco editor that is independent.
- **Colour Contrast** rule, which was triggered on Monaco Editor, but the editor provides an opt-in high contrast mode.

Secondly, an automated evaluation using browser extensions—Lighthouse⁶ and aXe DevTools⁷ built into the Chrome DevTools—was performed. Both tools reported no problems with accessibility, as can be seen in Figure 5.5 and Figure 5.6.

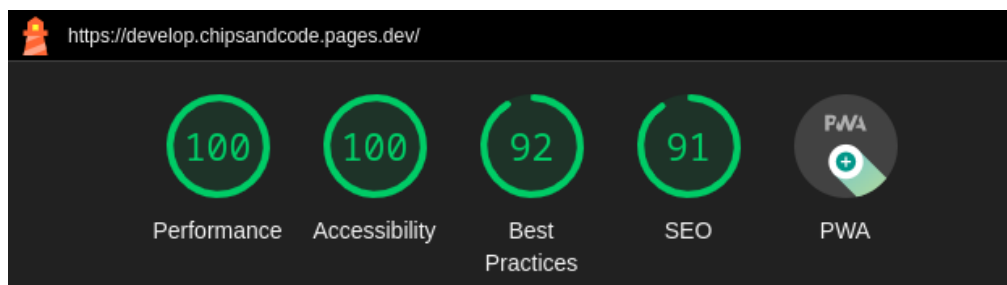


Figure 5.5: Lighthouse report summary.

⁶Available at <https://chrome.google.com/webstore/detail/lighthouse/blipmdconlcpinefehnmjammfjpmbjk>.

⁷Available at <https://chrome.google.com/webstore/detail/axe-devtools-web-accessib/lhdoppojpmngadmndnejejpokejbdd>.

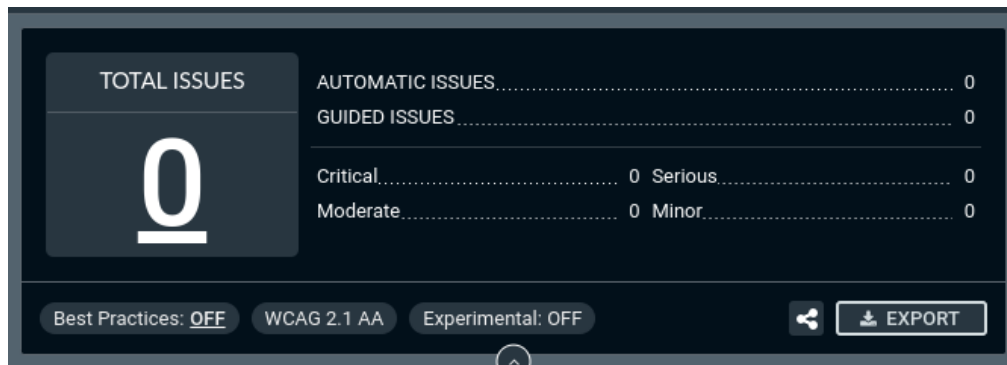


Figure 5.6: aXe report summary.

Lastly, a manual WCAG test, following the WCAG-EM report tool, was performed.⁸ The only potential problem identified during the testing involves dynamic updates to the UI. The author was unable to determine whether the automatically updated pin UI is usable when using assistive technology. For example, if the user changes some code in the editor and the list of pin updates, is this properly communicated using assistive technology? Similarly, if the input on the pins changes, is it clear the output pin value is updated automatically, and can the user navigate to it easily?

Importantly, the software was not tested by third-party users who are either disabled or experienced in using assistive technologies. While the author made the best effort to take into consideration accessibility, it is possible some of the checks were misunderstood.

5.3 Comparative Usability Testing

The following section contains results from the comparative study. All figures are also available at thesis.chipsandcode.com with the advantage of being interactive.

The total number of participants who joined the scheduled comparative study session was $n = 26$. Out of that, the data of two participants had to

⁸Report can be found at https://github.com/durasj/master-thesis/blob/main/assets/WCAG_Evaluation_Report.pdf

be removed—Participant *F*, who experienced technical difficulties unrelated to the software and Participant *J*, who mentioned they tried the Proposed Software in the past. The resulting sample size that was analysed was $n = 24$, with an equal distribution of $n_1 = 12$ and $n_2 = 12$ among both groups.

The list of participants and their profiles within Group A can be seen in Table 5.1, while Group B participant profiles can be seen in Table 5.2.

Table 5.1: Group A participant profiles.

Person	Prior Exp.	Education	Occupation	Age Group	Paid	Intr. Motiv.	Platform
M	1	2	2	30–49	0	0	Ubuntu Linux desktop
LU	0	0	0	18–29	1	1	Windows 10 laptop
Y	1	0	0	30–49	0	2	Windows 10 laptop
MI	0	1	0	30–49	0	0	Ubuntu Linux laptop
MX	1	2	1	30–49	0	1	Windows 10 desktop
R	0	2	0	18–29	1	0	Windows 10 desktop
MK	0	1	2	30–49	0	1	Windows 11 laptop
L	0	2	2	18–29	0	1	macOS 13 arm64 laptop
MA	0	0	0	18–29	1	1	Windows 10 laptop
KY	0	1	1	18–29	1	1	Windows 10 laptop
KA	0	0	0	18–29	1	0	Windows 7 laptop
E	0	0	2	30–49	1	2	Windows 10 desktop

Table 5.2: Group B participant profiles.

Person	Prior Exp.	Education	Occupation	Age Group	Paid	Intr. Motiv.	Platform
P	0	1	1	18–29	0	2	Windows 10 laptop
S	0	1	0	30–49	0	1	Windows 10 desktop
MM	1	1	2	18–29	0	1	Ubuntu Linux laptop
SA	0	1	2	18–29	0	0	Windows 11 laptop
JO	0	0	0	18–29	1	0	Windows 11 desktop
PE	0	1	0	18–29	1	1	Windows 11 laptop
MS	0	2	2	18–29	0	1	Windows 11 laptop
B	1	2	2	18–29	0	1	macOS 13 arm64 laptop
V	0	2	2	18–29	1	1	Windows 10 laptop
MH	1	2	2	30–49	1	1	macOS 13 x86 laptop
LB	0	2	2	18–29	1	1	Windows 10 laptop
MT	0	2	0	18–29	1	2	Windows 10 desktop

As we can see, despite the effort to distribute participant profiles equally, the profiles were not completely equal between the groups. To better capture the differences, Figure 5.7 shows the cumulative “Prior Experience”, “Education”, “Relevant Occupation”, and “Intrinsic Motivation” between the two groups. Additionally, Figure 5.8 shows the age make-up of the groups and Figure 5.9 shows the platforms used by participants of each group.

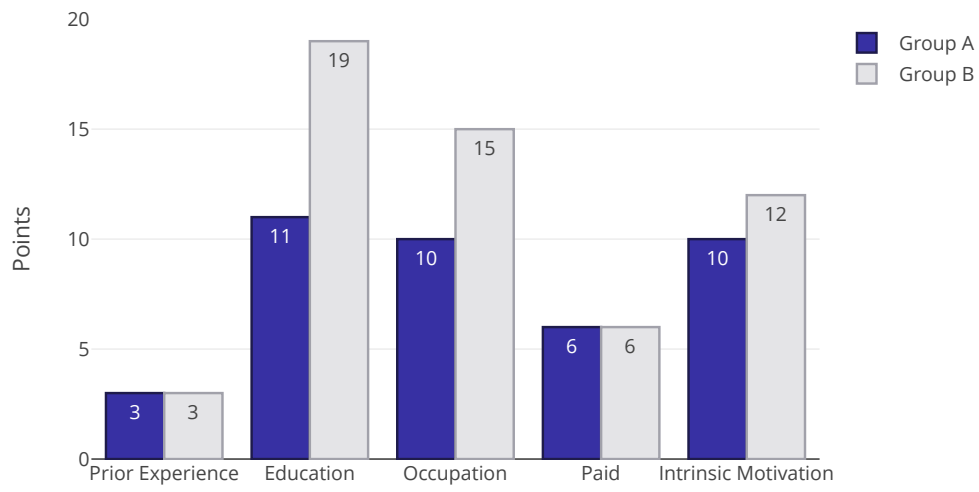


Figure 5.7: Cumulative profile points between groups.

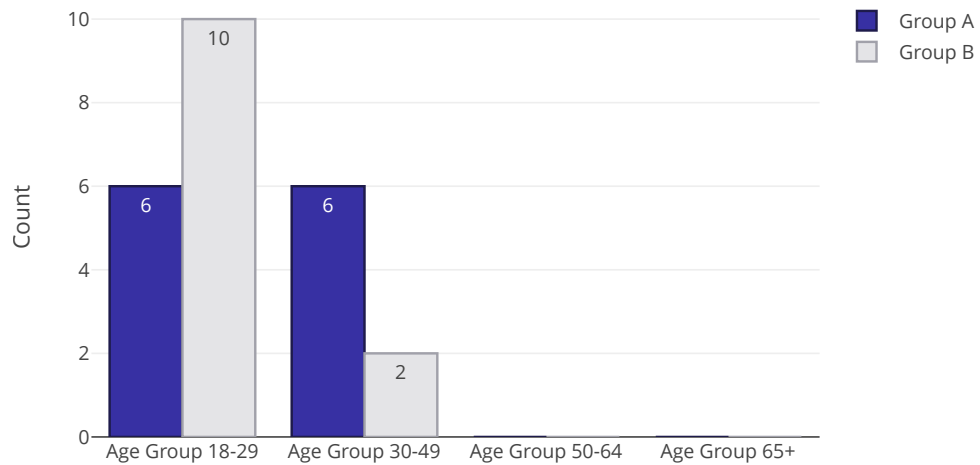


Figure 5.8: Age characteristics of participants.

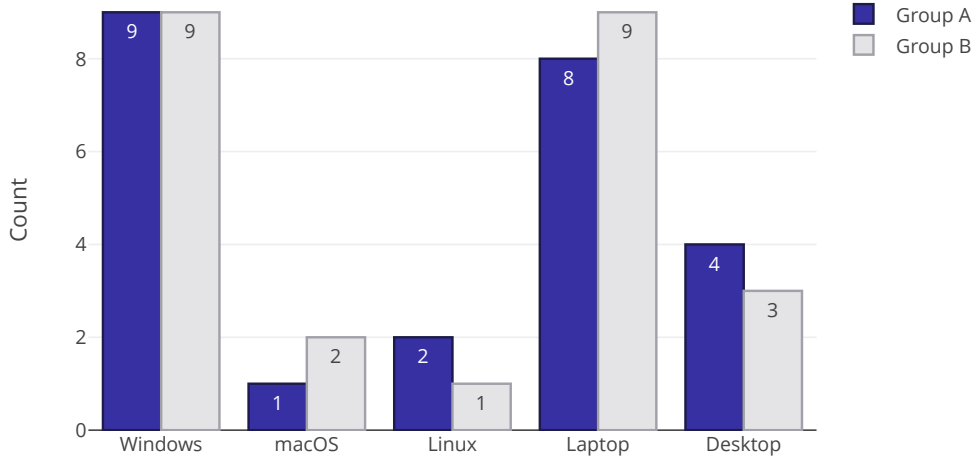


Figure 5.9: Platforms used by participants.

5.3.1 Efficiency

The first kind of collected data on efficiency consisted of time-related data like the time it took to prepare the tool for use and time spent on individual tasks. Figure 5.10 shows a linear graph covering the mean of the time of mentioned steps between both groups. The same data can be seen as cumulative, which is represented by Figure 5.11. The total time spent on all parts was $512.42 \pm 190.01s$ for Group A and $1,496.67 \pm 326.18s$ for Group B ($\alpha = 0.05$, $p < 0.001$). Raw data collected during the study and that served as input to mentioned figures and results can be reviewed online.⁹

The second kind of data on efficiency was the number of times the participant got confused, see Figure 5.12. For Group A, the mean was 0.58 ± 0.42 , while for Group B, it was 2.83 ± 0.85 ($\alpha = 0.05$, $p < 0.001$).

⁹Available at https://docs.google.com/spreadsheets/d/1CGNs1pCKvGJ4QuRDCEXnc_3SZ4pgX3U1TXSKJU8hpa8.

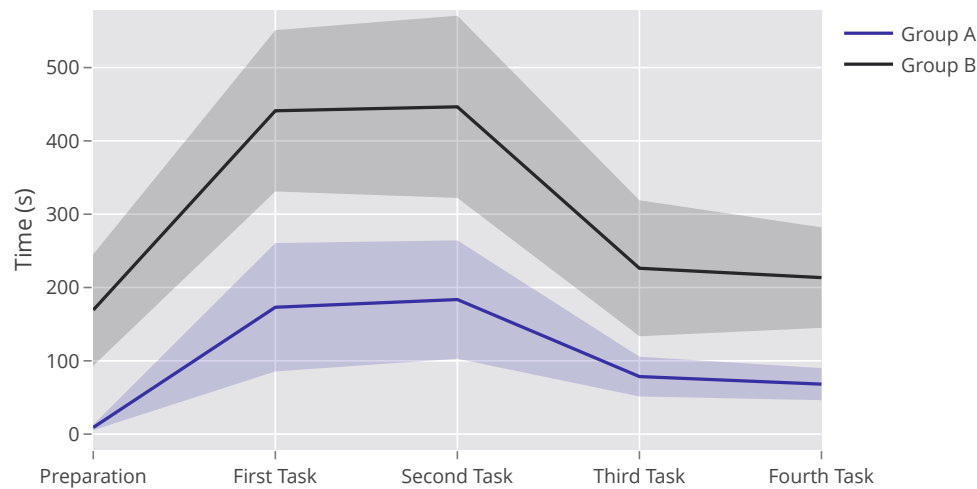


Figure 5.10: Time needed to prepare software and perform tasks.

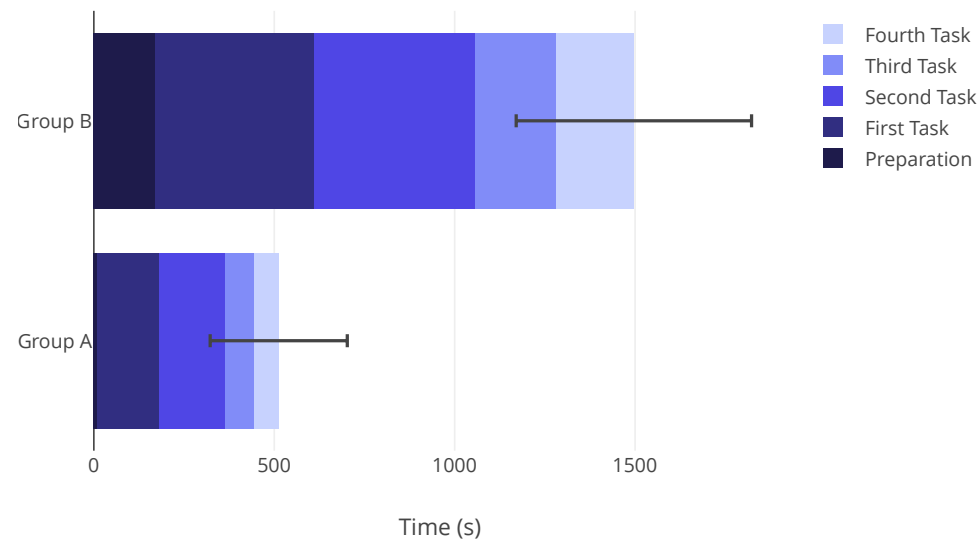


Figure 5.11: Cumulative time needed to prepare software and perform tasks.

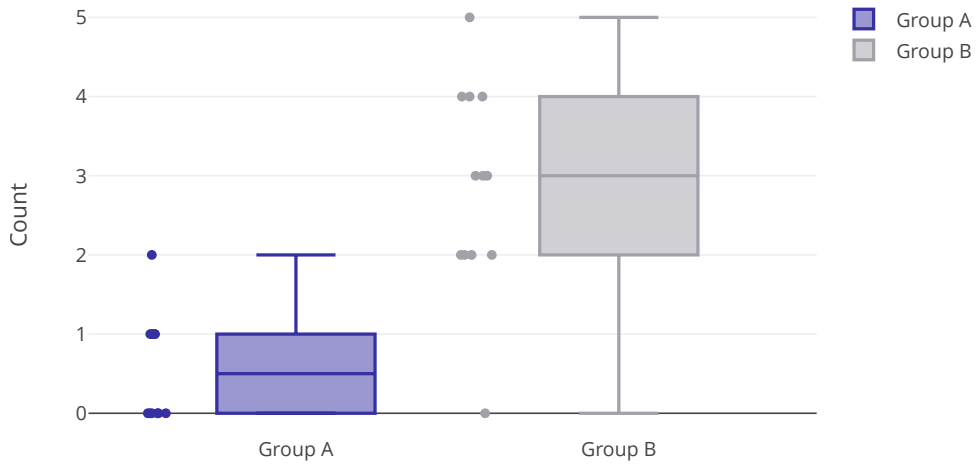


Figure 5.12: Number of times participants got confused.

5.3.2 Questionnaire

Collected responses had no coding errors, and internal reliability was acceptable at *Cronbach Alpha* $\rho_T = 0.708$ for Group A and $\rho_T = 0.895$ for Group B. The mean SUS score for Group A was 86.67 (97th percentile), while for Group B, it was 79.58 (86th percentile). While that represents a difference of about seven points, it did not reach the desired statistical significance ($\alpha = 0.1$, $p = 0.22$). Collected scores, statistical properties, and associated adjective and grade, can be seen in Figure 5.13. Placement on the percentile curve is captured in Figure 5.14.

The time it took the participant to prepare the tool and perform the assigned tasks is compared to the assigned SUS score in Figure 5.15. While there appears to be some correlation for both Group A and Group B, at $r = -0.28$ and $r = -0.23$, respectively, the combination of the small effect size and small sample size means the likelihood that the null hypothesis is high, with $p > 0.35$ for both groups.

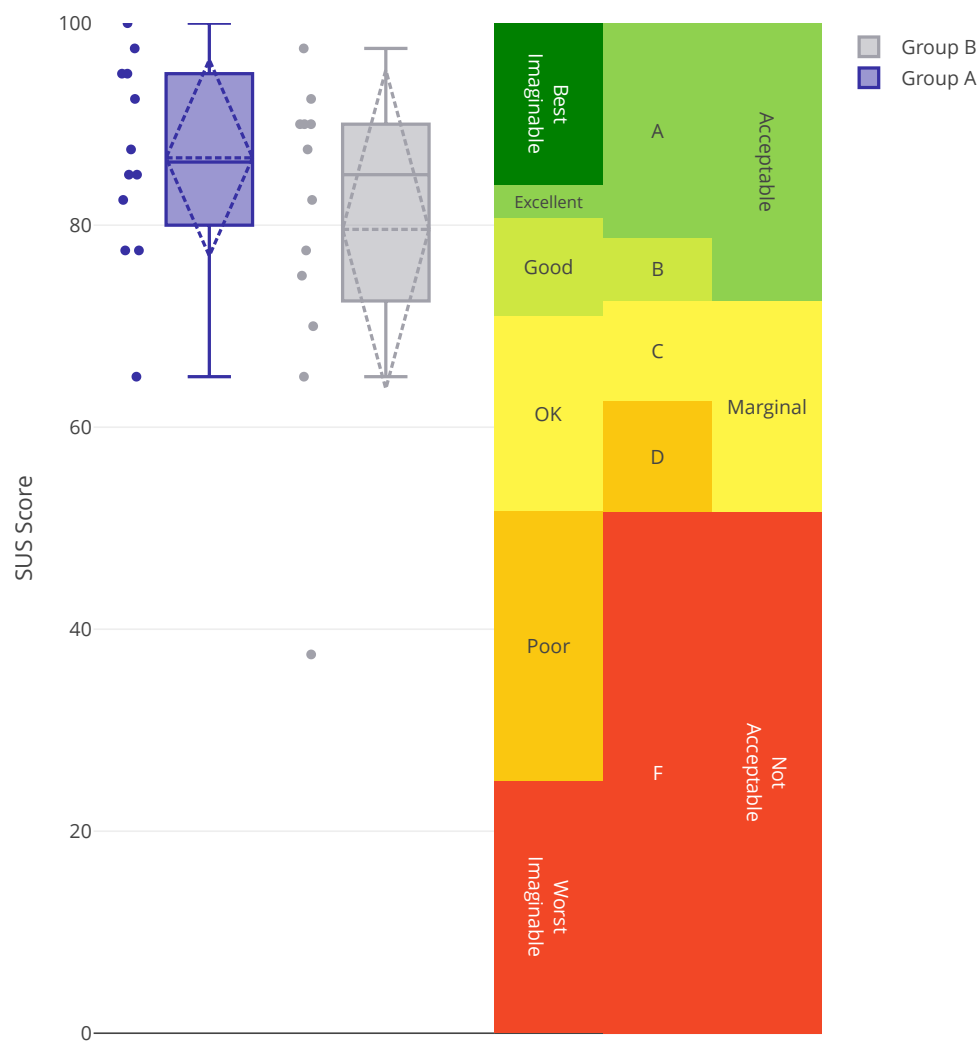


Figure 5.13: SUS score comparison.

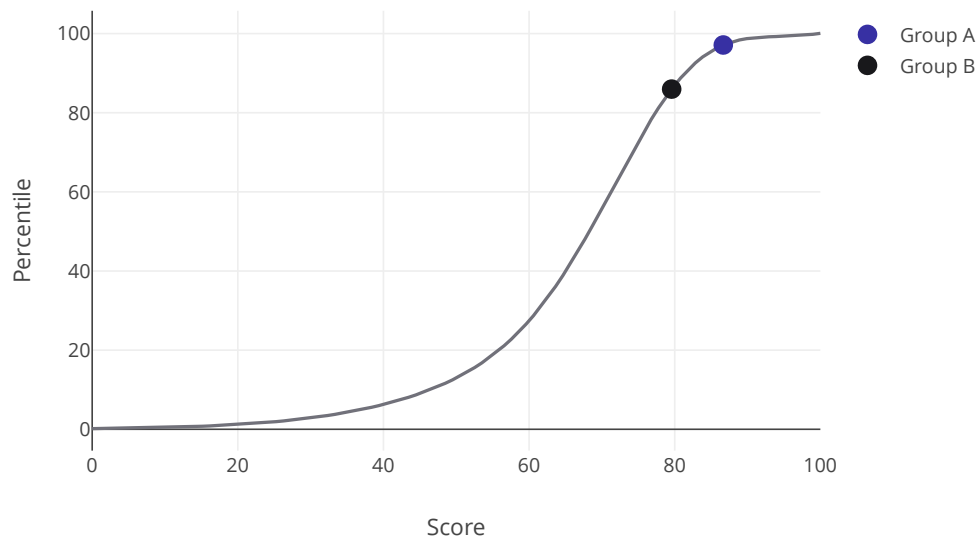


Figure 5.14: SUS score percentile.

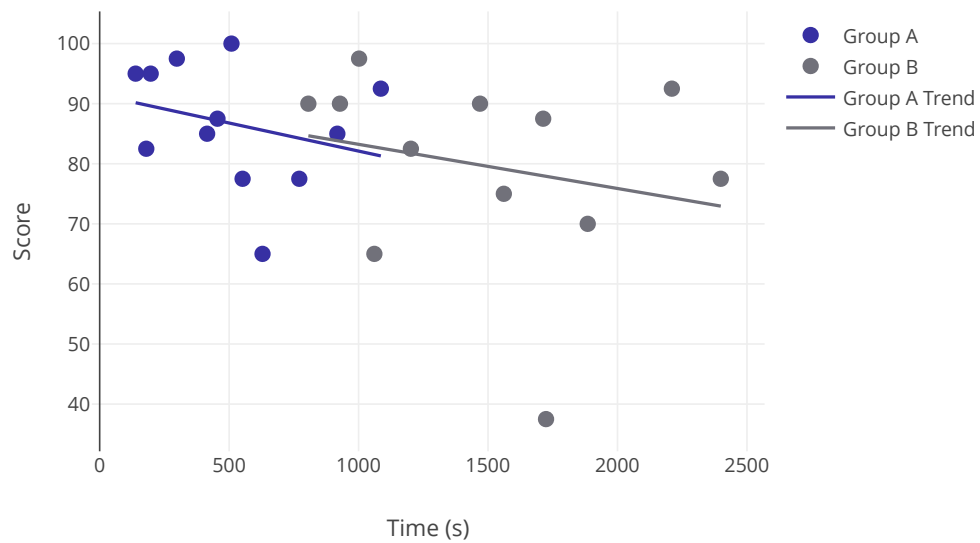


Figure 5.15: Relation between total time and SUS score.

5.3.3 Assignment Bias

Considering the large differences in some of the profile variables between the groups, this subsection shows correlations between the controlled vari-

ables with the greatest variance and dependent variables. Firstly, Figure 5.16 shows a moderate to strong correlation between the relevant education and time, $r(11) = -0.27, p = 0.39$ for Group A and $r(11) = -0.69, p = 0.01$ for Group B, but no correlation with the SUS score. Secondly, Figure 5.17 also shows a moderate correlation between the occupation and time, $r(11) = -0.55, p = 0.06$ for Group A and $r(11) = -0.43, p = 0.17$ for Group B, but there is likely no correlation with the SUS score as the data between the groups do not agree and $p > 0.1$ for both groups. Lastly, Figure 5.18 does not show a correlation between the age group and the performance or the SUS score with $r < 0.2, p > 0.1$.

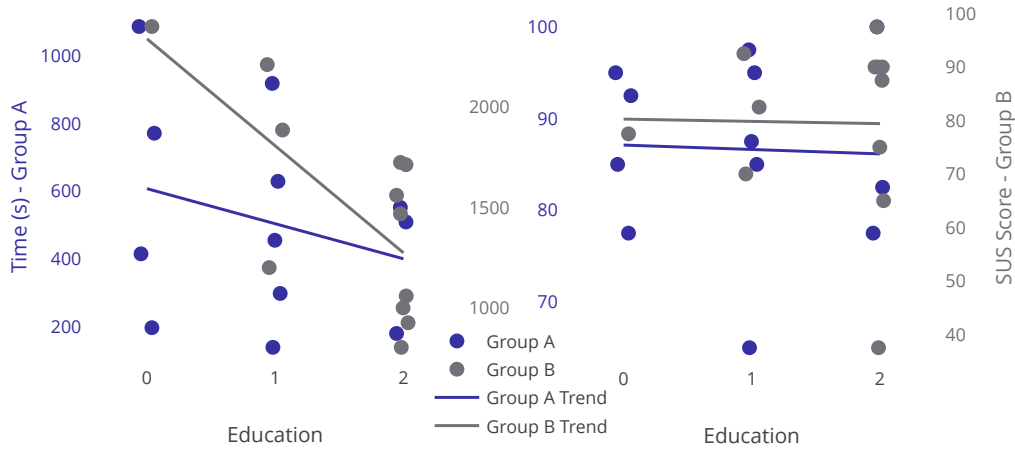


Figure 5.16: Relation between education and dependent variables.

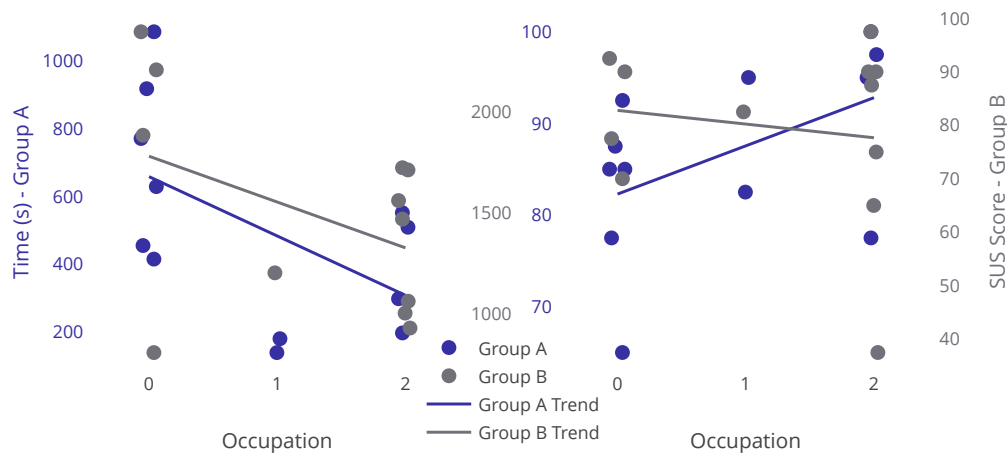


Figure 5.17: Relation between occupation and dependent variables.

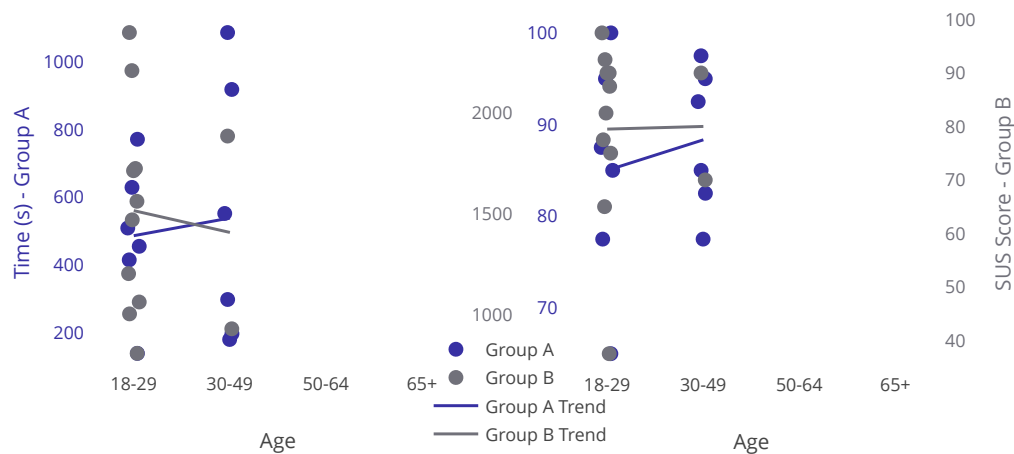


Figure 5.18: Relation between age and dependent variables.

5.4 Summary

In this chapter, we have first shown the scope, implementation, and results of the various tests. Proposed Software was checked for correctness, compatibility with Existing Software, and accessibility compliance. The second part of the evaluation has presented results from the comparative study - notably quantitative data on efficiency, SUS score, and selected correlations.

6 Discussion

This chapter first addresses most of the RQ1, specifically if it is feasible to efficiently reimplement the functionality of the desktop tool for learning computing principles by using web technologies. Then, it continues by addressing RQ2 on the usability and efficiency improvements before going back to device support improvements covered under RQ1. It finishes with the limitations and ideas for further research.

6.1 Feasibility

As can be seen in Chapter 4, it was not only possible to reimplement Existing Software using web technologies, but it was also possible to do so relatively efficiently while mitigating the design disadvantages of the web.

Although the implementation involved topics that one may not expect to be well supported on the web, e.g., parsing and code editing, it was shown that there are high-quality OSS libraries that can support an efficient web implementation. Notably, the library behind the most popular IDE is available as a web library¹ under a permissive OSS license and was made part of Proposed Software with relative ease. Similarly, it was possible to create two parsers—for the HDL and TST²—relatively quickly thanks to an easy-to-use Earley algorithm parser generator and the integrated lexer.³ The ability to perform the same selected functionality as Existing Software was confirmed

¹Monaco Editor, see Section 4.3.1, powering the popular IDE Visual Studio Code, see Section 2.4.2.

²See Appendix A, B, D, and E.

³Nearley and moo, respectively, see Section 4.3.2.

using a range of automated tests, including unit tests demonstrating that all built-in and Project 1 files from Existing Software can be parsed and interpreted.⁴ While it is impossible to say others (re)implementing learning tools on the web would find the ecosystem similarly mature, this work shows it could be worthwhile to evaluate that option.

A reasonable person might expect that web software, compared to desktop, requires an internet connection, and cannot be integrated into the OS the way desktop applications can. This thesis shows it is possible to use modern web APIs to enable use with limited or even no internet connection and install the application like a desktop one.⁵ Similarly, some might argue that a web application is dependent on servers and harder to reproduce. Proposed Software is a client-side web application and does not rely on the cloud, which makes it easier to reproduce and enables bundling as a fully functional offline desktop or mobile application.⁶ Moreover, we theorise that web-based learning tools could be easier to incorporate into various learning materials using embedding or linking. This thesis shows that common drawbacks associated with web applications can be solved or mitigated with architectural decisions and modern APIs.

It was not possible to implement all the functionality offered by Existing Software in the allocated time. This means there is a possibility some functionality could be harder to cover using the web. However, the author believes all observed features can be implemented on the web. Additionally, it should be feasible to do so without complicating the UI as it can adapt to the complexity of the solved problems by “growing” with the learner’s knowledge on the topic and using “sane defaults”.

⁴HDL and TST files, see Section 5.1.1.

⁵Using service workers and PWA technology, see Section 4.1.

⁶Using technology like Electron and Capacitor, see Section 4.1.

6.2 Improvements

The presented results show tangible improvements compared to Existing Software regarding usability (especially efficiency) and device support.

As was shown in Section 5.3.1, participants using Proposed Software embedded within the learning content were able to complete the assigned tasks in one-third of the time ($512 \pm 190s$ vs $1497 \pm 326s$, $\alpha = 0.05$, $p < 0.001$) it took participants using Existing Software. It is assumed this effect was partially caused by the overhead of using three different windows: a browser with the learning content, a text editor for the assignment code, and the actual learning tool. This is also explained by the mean number of times participants from each group got confused,⁷ which was 0.58 ± 0.42 for the group using Proposed Software and 2.83 ± 0.85 for the group using Existing Software ($\alpha = 0.05$, $p < 0.001$). The first set of commonly experienced problems was the inability to run the software due to a missing or incorrect version of Java or the launching of the application using a way unsuitable for their OS. The second set of commonly experienced problems was connected to the received errors: participants were often confused by them or could not see them at all since the UI would not scale with the window size. The third set of commonly experienced problems related to the actions available in the user interface—participants were unsure which button to use or how to proceed. Importantly, this was despite the participants seeing a video showing how to install and use the software. These problems were non-existent with Proposed Software; the only problem experienced was confusion around the syntax. This finding suggests it is possible to achieve substantial efficiency gains by reimagining existing desktop learning tools as embedded web learning tools focused on usability. We could theorise that less time needed to participate and fewer technical challenges would positively impact the MOOC dropout rates.⁸ However, further research is needed to confirm that efficiency improvement is reproducible with different learning tools from different fields of study. This thesis did not cover practical significance, e.g.,

⁷See Section 3.3.3 for the definition.

⁸Suggested by research, see Section 2.2.1.

to what extent this would impact the success rate or dropout rates.

The second metric used to gauge the improvement was the SUS score mentioned in Section 5.3.2. While there was a difference of 7 points in favour of Proposed Software (86.67 points [97th percentile] vs 79.58 points [86th percentile]), the result was not statistically significant ($\alpha = 0.1$, $p = 0.22$). This was likely due to the relatively small effect size and small sample size ($n = 12$ per group). The suggested sample size⁹ at this effect size to produce statistically significant data is $n = 45$ per group. The effect size grew considerably once the same participant was asked to try the other software—likely because of the anchoring. However, this data was not presented due to the likely bias, as many participants recognised which software was created by the author once they saw both. Still, it is possible a within-subject design would produce statistically significant data faster, provided the bias could be prevented—e.g., by using a blind data collector.

The data suggest a relation between these two metrics—the time on task and the SUS score—at $r = -0.28$ and $r = -0.23$ for the group using Proposed Software and Existing Software, respectively. However, the combination of the small effect size and a small sample size means the likelihood that the null hypothesis is true is high, with $p > 0.35$ for both groups. More research is needed to confirm this relation in a larger sample size and different works.

Existing research suggests a pattern of increased reliance on devices that do not support running arbitrary desktop applications—particularly among the 18–29 age group.¹⁰ This pattern, combined with the increasing mobile device share on the internet traffic and a new generation of learners used to Chromebooks and tablets, suggests learners could be underserved with existing desktop learning tools. The problem of reliance on smart devices for learning is particularly apparent in developing countries where about 34% of households have no access to a desktop, laptop, or tablet device, and one-fourth of the population relies on smartphones as the only way to connect to the internet.¹¹ This problem was further emphasised during the COVID

⁹Using SUS Guide & Calculator Package, see Section 3.3.4.

¹⁰See Section 2.3.

¹¹See Section 2.3, Developing Countries.

pandemic when researchers called for the content to be available from a wide range of devices.¹² We can argue it is possible to considerably improve the inclusivity of the learning environment by designing for a wide range of devices, which is not the case with legacy desktop learning tools but can be achieved with new web learning tools. Proposed Software should be available to the vast majority of internet users (estimated 98%), while only 44% of the internet traffic is from the desktop, suggesting a large portion of the users would not be able to use the software how they consume web content. In developing countries, this could mean one-fourth of the population to one-third of the households would be able to get access to learning tools they currently find hard to access or simply cannot access. However, in developed countries, this may not have a very strong practical impact as the majority can access a compatible laptop or desktop.

Regarding qualitative data, learners who tried both voiced a slight or strong preference for Proposed Software. They praised the integration within the web page, the integration of the editor, and the automatic execution of tests. Many noted they felt more efficient, and some noted they would not be willing to install Java if it was not necessary—particularly those on Linux where it was necessary to change the configuration. Notably, three participants could not finish their session: one did not have the administrative privileges required to install Java, one did not feel comfortable installing software from the internet on the given device, and one could not see the error output as their screen was not large enough and Existing Software does not scale with the window size. However, it is unclear to what extent the opinions were biased as many of the learners were able to guess which software the author moderating the study wrote once they saw both, and it is assumed learners from developed countries would find a way to use the existing software if they were strongly motivated.

¹²See Section 2.2.1.

6.3 Limitations and Further Work

This thesis used the case of one specific desktop learning tool. As such, apart from the certain device support benefits, it is questionable to what extent the same results could be replicated on different learning tools from different fields of study. Also, it measured the results only on a subset of the functionality and during a session of only one hour. While the observed efficiency improvements were statistically significant and with a notable effect size, the practical significance will depend on the context. The results represent a situation where the learner must pick up a new learning tool. Essential metrics like the success rate or the dropout rate will likely depend heavily on the time limits—be it the time dedicated to the exercise in the classroom or the free time the learner is willing to spend at home before they give up or have to move on. This thesis did not cover performance improvements connected with a more comprehensive long-term use. While we could theorise that some of the presented concepts apply to long-term use as well, the effect size would likely get smaller over time.

Moreover, the chosen methodology had several limitations. Firstly, sampling bias, as the sample came primarily from the extended social circle of the author and only included participants who agreed to participate. Secondly, moderator bias, as the author was the moderator of the study and might have influenced the study despite the best efforts not to do so. Lastly, the group using Existing Software had to create the starting files themselves, which could have led to an unrealistic overhead. While we could argue that this is an expected part of the work with desktop tools, we could also argue that this could have been simplified, and the participants could have been able to download pre-created files. Using existing files was not possible in this case as the study featured the original bundle of Existing Software and the chosen simple assignments which were not part of the original bundle. A more extensive comparative study with participants available for longer could use the bundled set of assignments.

The suspected selection bias was shown to hold only for education and

occupation with a moderate to strong correlation with performance.¹³ Education data had a correlation coefficient $r(11) = -0.27, p = 0.39$ for Group A and $r(11) = -0.69, p = 0.01$ for Group B. Occupation data had a correlation coefficient $r(11) = -0.55, p = 0.06$ for Group A and $r(11) = -0.43, p = 0.17$ for Group B. Simply put, participants from Group B using Existing software had more relevant education and work experience, which seems to correlate positively with better performance. Therefore, it is possible performance was skewed in favour of Group B using Existing Software. That being said, if we assume possible moderator bias and the theoretical task overhead due to the file management, the skew was likely minimal and has no impact on the conclusion.

Considering the limitations and unexplored paths of this thesis, there are multiple exciting possibilities for further work:

- **Academic:**
 - Larger sample size—suggested $n = 45$ per group to achieve statistical significance on all data points.
 - Long-term study observing performance over several sessions.
 - Change of the methodology to within-subject with a blind study moderator.
 - Reproduction of the results with different learning tools from different fields of study.
- **Practical:** Finish implementating all functionality needed to take the whole Nand2Tetris course while using the web implementation.
- **For the Wider Public:** Extend the potential target group further by adapting or introducing content that would use Proposed Software but would be easier to consume by the public.¹⁴

¹³Higher performance here means lower time taken to prepare and perform tasks, hence the negative correlation coefficient.

¹⁴Akin to how <https://www.elementsofai.com> introduces the basics of AI.

7 Conclusions

This thesis showed that it is feasible to reimplement the functionality of the desktop tool for learning computing principles by using web technologies while achieving device support improvements (RQ1). It was possible to do so thanks to a mature ecosystem of OSS that allowed for an efficient implementation (RQ1). Modern web APIs—service worker and PWA—enabled the use with limited or even no internet connection and standalone OS integration, respectively. Thanks to the client-side architecture, Proposed Software does not rely on the cloud, which makes it easier to reproduce.

Participants using Proposed Software embedded within the learning content could complete the assigned tasks in just one-third of the time ($512 \pm 190s$ vs $1497 \pm 326s$, $\alpha = 0.05$, $p < 0.001$) it took participants using Existing Software. Additionally, the mean number of times participants using Proposed Software got confused was lower at 0.58 ± 0.42 compared to 2.83 ± 0.85 for the group using Existing Software ($\alpha = 0.05$, $p < 0.001$). This finding suggests it is possible to achieve noteworthy gains by reimagining existing desktop learning tools as embedded web learning tools focused on usability (RQ2).

While there was a difference of 7 points in favour of Proposed Software (86.67 points [97th percentile] vs 79.58 points [86th percentile]), the result was not statistically significant ($\alpha = 0.1$, $p = 0.22$) (RQ2). The data also suggest a relation between the time on task and the SUS score at $r = -0.28$ and $r = -0.23$ for the group using the new and existing software, respectively. However, the likelihood that the null hypothesis is true is high, with $p > 0.35$ for both groups.

Considering the pattern of increasing reliance on devices that do not sup-

port legacy desktop applications, particularly among the 18–29 age group, K12 students using Chromebooks and tablets, and within developing countries, practitioners could consider improving the inclusiveness of the learning environment by designing for a wide range of devices. Web-based learning tools should be usable from around 98% of devices and can be embedded within the learning content and reused in more learning materials. Combining inherent web advantages and a focus on usability could bring considerable efficiency improvements. Nonetheless, further research is needed to determine the practical significance and confirm that efficiency improvements are reproducible with different learning tools from different fields of study. Moreover, more work is needed to explore the statistically insignificant SUS score improvements as well as the correlation between the participant’s performance and the SUS score.

Bibliography

- Ali, W. (2020) “Online and Remote Learning in Higher Education Institutes: A Necessity in Light of COVID-19 Pandemic”. In: *Higher Education Studies* 10.3, pp. 16–25. ISSN: 1925-4741. DOI: 10.5539/hes.v10n3p16.
- Almazroi, A. A. (2021) “A Systematic Mapping Study of Software Usability Studies”. In: *International Journal of Advanced Computer Science and Applications* 12.9. DOI: 10.14569/ijacsa.2021.0120927. Available <https://doi.org/10.14569/ijacsa.2021.0120927>.
- Antinyan, V., Derehag, J., Sandberg, A. and Staron, M. (2018) “Mythical Unit Test Coverage”. In: *IEEE Software* 35.3, pp. 73–79. DOI: 10.1109/ms.2017.3281318. Available <https://doi.org/10.1109/ms.2017.3281318>.
- Baek, K. and Sugihara, K. (2015) *Discrete Mathematics II ICS 241 11.1 Introduction to Trees - Lecture notes*. Available <http://courses.ics.hawaii.edu/ReviewICS241/morea/trees/Trees-QA.pdf> [accessed 20th Aug. 2023]. Archived <https://web.archive.org/web/20221101000000/http://courses.ics.hawaii.edu/ReviewICS241/morea/trees/Trees-QA.pdf>.
- Bangor, A., Kortum, P. and Miller, J. (2009) “Determining what individual SUS scores mean: adding an adjective rating scale”. In: *Journal of Usability Studies* 4.3, pp. 114–123. Available <https://uxpajournal.org/determining-what-individual-sus-scores-mean-adding-an-adjective-rating-scale/>.
- Baramidze, V. (2013) “LaTeX for Technical Writing”. In: *Journal of Technical Science and Technologies*, pp. 45–48. DOI: 10.31578/.v2i2.63.

- Available <https://jtst.ibsu.edu.ge/jms/index.php/jtst/article/view/63> [accessed 1st Aug. 2023].
- Blattgerste, J., Behrends, J. and Pfeiffer, T. (2022) “A Web-Based Analysis Toolkit for the System Usability Scale”. In: *Proceedings of the 15th International Conference on Pervasive Technologies Related to Assistive Environments*. PETRA '22. Corfu, Greece: Association for Computing Machinery, pp. 237–246. ISBN: 9781450396318. DOI: 10.1145/3529190.3529216. Available <https://doi.org/10.1145/3529190.3529216>.
- Boreham, M. (2019) *Global K-12 Mobile PC Education Market Continues Growth Momentum*. Tech. rep. Hertfordshire, United Kingdom: Future-source Consulting. Available <https://www.futuresource-consulting.com/insights/new-k-12-mobile-pc-report-confirms-2018-growth-and-upbeat-future/> [accessed 10th Oct. 2021].
- Brewer, J. (2018) *Template for Accessibility Evaluation Reports*. Available <https://www.w3.org/WAI/test-evaluate/report-template/> [accessed 27th Dec. 2021]. Archived <https://web.archive.org/web/20210810181815/https://www.w3.org/WAI/test-evaluate/report-template/>.
- Brocke, J. v., Hevner, A. and Maedche, A. (2020) “Introduction to Design Science Research”. In: pp. 1–13. ISBN: 978-3-030-46780-7. DOI: 10.1007/978-3-030-46781-4_1.
- Brooke, J. (2013) “SUS: a retrospective”. In: *Journal of Usability Studies* 8.2, pp. 29–40. Available <https://uxpajournal.org/sus-a-retrospective/>.
- Brooke, j. (1996) “SUS: A ‘Quick and Dirty’ Usability Scale”. In: *Usability Evaluation In Industry*. CRC Press. ISBN: 9780429157011.
- Chen, B., deNoyelles, A., Brown, T. and Seilhamer, R. (2022) *Internet/Broadband Fact Sheet*. Available <https://er.educause.edu/articles/2023/1/the-evolving-landscape-of-students-mobile-learning-practices-in-higher-education>.
- Cohn, M. (2010) *Succeeding with agile: software development using Scrum*. The Addison-Wesley signature series. OCLC: ocn318420978. Upper Saddle River, NJ: Addison-Wesley. ISBN: 9780321579362.

- Court of Justice of the European Union (2012) *JUDGMENT OF THE COURT (Grand Chamber) on Case C-406/10*. Available <https://curia.europa.eu/juris/document/document.jsf?docid=122362&doclang=EN>. Archived: <https://web.archive.org/web/20230829142227/https://curia.europa.eu/juris/document/document.jsf?docid=122362&doclang=EN>.
- Darin, T., Coelho, B. and Borges, B. (2019) “Which Instrument Should I Use? Supporting Decision-Making About the Evaluation of User Experience”. In: *Design, User Experience, and Usability. Practice and Case Studies*. Ed. by A. Marcus and W. Wang. Cham: Springer International Publishing, pp. 49–67. ISBN: 978-3-030-23535-2. DOI: 10.1007/978-3-030-23535-2_4.
- Duras, J. (2020) *Replacing handwritten signatures with open electronic signature software*. Available <https://thesis.duras.me/draft.pdf> [accessed 29th Aug. 2023].
- Dziak, J. J., Dierker, L. C. and Abar, B. (2020) “The interpretation of statistical power after the data have been gathered”. In: *Current Psychology* 39.3, pp. 870–877. ISSN: 1936-4733. DOI: 10.1007/s12144-018-0018-1.
- Edyburn, D. L. (2010) “Would You Recognize Universal Design for Learning if You Saw it? Ten Propositions for New Directions for the Second Decade of UDL”. In: *Learning Disability Quarterly* 33.1, pp. 33–41. ISSN: 0731-9487. DOI: 10.1177/073194871003300103.
- Edyburn, D. L. (2021) “Universal Usability and Universal Design for Learning”. In: *Intervention in School and Clinic* 56.5, pp. 310–315. ISSN: 1053-4512. DOI: 10.1177/1053451220963082.
- European Commission (2021) *Web Accessibility*. Available <https://digital-strategy.ec.europa.eu/en/policies/web-accessibility> [accessed 4th Dec. 2021].
- European Commission (2023) *Covid-19 learning deficits in Europe: analysis and practical recommendations : analytical report*. Publications Office. Available <https://data.europa.eu/doi/10.2766/881143>.

- Fisher, W. W.-H. (2016) “Recalibrating originality”. In: *Hous. L. REv.* 54, p. 437. Available https://heinonline.org/hol-cgi-bin/get_pdf.cgi?handle=hein.journals/hulr54§ion=16.
- Fogel, K. (2022) *Producing Open Source Software: How to Run a Successful Free Software Project*. Second edition. O'Reilly Media. Available <http://www.producingoss.com/>.
- Frazão, T. and Duarte, C. (2020) “Comparing accessibility evaluation plugins”. In: *Proceedings of the 17th International Web for All Conference*. W4A '20. Association for Computing Machinery, pp. 1–11. ISBN: 9781450370561. DOI: 10.1145/3371300.3383346.
- García-Carballeira, F., Calderón-Mateos, A., Alonso-Monsalve, S. and Prieto-Cepeda, J. (2019) “Wepsim: an online interactive educational simulator integrating microdesign, microprogramming, and assembly language programming”. In: *IEEE Transactions on Learning Technologies* 13.1, pp. 211–218. DOI: 10.1109/TLT.2019.2903714.
- Global Industry Analysts (GIA) (2023) *Global E-Learning Industry*. Tech. rep. San Jose, California: Global Industry Analysts. Available <https://www.strategyr.com/market-report-e-learning-forecasts-global-industry-analysts-inc.asp>.
- Global Market Insights (GMI) (2023) *E-learning Market, 2023-2032*. Tech. rep. Selbyville, Delaware: Global Market Insights. Available <https://www.gminsights.com/industry-analysis/elearning-market-size>.
- Goopio, J. and Cheung, C. (2021) “The MOOC dropout phenomenon and retention strategies”. In: *Journal of Teaching in Travel & Tourism* 21.2, pp. 177–197. ISSN: 1531-3220, 1531-3239. DOI: 10.1080/15313220.2020.1809050. Available <https://www.tandfonline.com/doi/full/10.1080/15313220.2020.1809050>.
- Greenman, T. (2007) *Copyright, IP, Entertainment Law in Israel - Fair Use under Israel's New Copyright Act*. Israel. Available <https://www.tglaw.co.il/index.php?dir=site&page=articles&op=item&cs=10109&langpage=eng&language=eng>.

- Greenman, T. (2015) *Just Say No to More End-to-End Tests*. Available <https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html>.
- Hagedorn, G., Mietchen, D., Morris, R., Agosti, D., Penev, L., Berendsohn, W. and Hobern, D. (2011) “Creative Commons licenses and the non-commercial condition: Implications for the re-use of biodiversity information”. In: *ZooKeys* 150, pp. 127–149. DOI: 10.3897/zookeys.150.2189. Available <https://doi.org/10.3897/zookeys.150.2189>.
- Hassenzahl, M., Koller, F. and Burmester, M. (2008) “Der User Experience (UX) auf der Spur: Zum Einsatz von www.attrakdiff.de”. In: *Tagungsband UP08*.
- Henry, S. L. (2021a) *Evaluating Web Accessibility Overview*. Available <https://www.w3.org/WAI/test-evaluate/> [accessed 4th Dec. 2021]. Archived <https://web.archive.org/web/20211130145209/https://www.w3.org/WAI/test-evaluate/>.
- Henry, S. L. (2021b) *Introduction to Web Accessibility*. Available <https://www.w3.org/WAI/fundamentals/accessibility-intro/> [accessed 4th Dec. 2021]. Archived <https://web.archive.org/web/20211127115925/https://www.w3.org/WAI/fundamentals/accessibility-intro/>.
- Hevner, A. R., March, S. T., Park, J. and Ram, S. (2004) “Design Science in Information Systems Research”. In: *MIS Quarterly* 28.1, pp. 75–105. ISSN: 02767783. DOI: 10.2307/25148625.
- IDC (2021) *Chromebook and Tablet Growth Continued in the Second Quarter Despite On-Going Supply Concerns*. Tech. rep. Needham, Massachusetts: International Data Corporation. Available <https://www.idc.com/getdoc.jsp?containerId=prUS48120621> [accessed 10th Oct. 2021]. Archived <https://web.archive.org/web/20230126230623/https://www.idc.com/getdoc.jsp?containerId=prUS48120621>.
- IDC (2022) *Tablet and Chromebook Shipments Continued to Decline in Q3 Amidst Ongoing Market Headwinds, According to IDC Tracker*. Tech. rep. Needham, Massachusetts: International Data Corporation. Available <https://www.idc.com/getdoc.jsp?containerId=prUS49812222> [ac-

- cessed 10th July 2023]. Archived <https://web.archive.org/web/20230330020313/https://www.idc.com/getdoc.jsp?containerId=prUS49812222>.
- Indigo, J. and Smart, D. (2022) *Page Weight*. Available <https://almanac.httparchive.org/en/2022/page-weight>.
- International Standards Organisation (2012) *ISO/IEC 40500:2012 Information technology — W3C Web Content Accessibility Guidelines (WCAG) 2.0*. ISO/IEC 40500:2012. Available <https://www.iso.org/standard/58625.html>.
- International Standards Organisation (2018) *ISO 9241-11:2018 Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. ISO 9241-11:2018. Available <https://www.iso.org/standard/63500.html>.
- International Standards Organisation (2020) *ISO/IEC 10779:2020 Information technology — Office equipment — Accessibility guidelines for older persons and persons with disabilities*. ISO/IEC 10779:2020. Available <https://www.iso.org/obp/ui/fr/#iso:std:iso-iec:10779:dis:ed-2:v1:en>.
- Ivankovic, M., Petrovic, G., Just, R. and Fraser, G. (2019) “Code coverage at Google”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 955–963.
- Kelly, D., Glavin, F. G. and Barrett, E. (2021) “Denial of wallet—Defining a looming threat to serverless computing”. In: *Journal of Information Security and Applications* 60, p. 102843. DOI: 10.1016/j.jisa.2021.102843. Available <https://doi.org/10.1016/j.jisa.2021.102843>.
- Khademi, M., Haghshenas, M. and Kabir, H. (2011) “A Review On Authoring Tools”. In: *5th International Conference on Distance Learning and Education*. IACSIT Press. Available <https://www.semanticscholar.org/paper/A-Review-On-Authoring-Tools-Khademi-Haghshenas/fac59b388f822adac8bf3338fbb98b4a0690629b> [accessed 21st Aug. 2023].

- Kjolstad, F. (2023) *Introduction to Parsing CS143 Lecture 5 - Lecture notes*. Available <https://web.stanford.edu/class/cs143/lectures/lecture05.pdf> [accessed 10th Aug. 2023]. Archived <https://web.archive.org/web/20230327182646/https://web.stanford.edu/class/cs143/lectures/lecture05.pdf>.
- Knauff, M. and Nejasmic, J. (2014) “An Efficiency Comparison of Document Preparation Systems Used in Academic Research and Development”. In: *PLOS ONE* 9.12, e115069. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0115069. Available <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0115069> [accessed 1st Aug. 2023].
- Leo, J. M. I. M. (1991) “A general context-free parsing algorithm running in linear time on every LR(k) grammar without using lookahead”. In: *Theoretical Computer Science* 82.1, pp. 165–176. ISSN: 0304-3975. DOI: 10.1016/0304-3975(91)90180-A. Available <https://www.sciencedirect.com/science/article/pii/030439759190180A> [accessed 21st Aug. 2023].
- Lewis, J. R. (2014) “Usability: Lessons Learned ... and Yet to Be Learned”. In: *International Journal of Human-Computer Interaction* 30.9, pp. 663–684. ISSN: 1044-7318. DOI: 10.1080/10447318.2014.930311.
- Lewis, J. R. (1995) “IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use”. In: *International Journal of Human-Computer Interaction* 7.1, pp. 57–78. ISSN: 1044-7318, 1532-7590. DOI: 10.1080/10447319509526110.
- Lewis, J. R. (2002) “Psychometric Evaluation of the PSSUQ Using Data from Five Years of Usability Studies”. In: *International Journal of Human-Computer Interaction* 14.3–4, pp. 463–488. ISSN: 1044-7318, 1532-7590. DOI: 10.1080/10447318.2002.9669130. Available <http://www.tandfonline.com/doi/abs/10.1080/10447318.2002.9669130>.
- Lewis, J. R. (2018) “The System Usability Scale: Past, Present, and Future”. In: *International Journal of Human-Computer Interaction* 34.7, pp. 577–590. ISSN: 1044-7318, 1532-7590. DOI: 10.1080/10447318.2018.1455307.

- Macefield, R. (2009) “How to specify the participant group size for usability studies: a practitioner’s guide”. In: *Journal of Usability Studies* 5.1, pp. 34–45. DOI: 10.5555/2835425.2835429.
- McCloskey, M. (2014) *Task Scenarios for Usability Testing*. Available <https://www.nngroup.com/articles/task-scenarios-usability-testing/>.
- Moran, K. (2019) *Usability Testing 101*. Available <https://www.nngroup.com/articles/usability-testing-101/>.
- Nielsen, J. (1993) “Chapter 2 - What Is Usability?” In: *Usability Engineering*. Morgan Kaufmann, pp. 23–48. ISBN: 9780125184069. DOI: 10.1016/B978-0-08-052029-2.50005-X. Available <https://www.sciencedirect.com/science/article/pii/B978008052029250005X>.
- Nielsen, J. (2008) *Usability ROI Declining, But Still Strong*. Available <https://www.nngroup.com/articles/usability-roi-declining-but-still-strong/> [accessed 21st Dec. 2021].
- Nisan, N. and Schocken, S. (2021) *The elements of computing systems: building a modern computer from first principles*. Second edition. Cambridge, Massachusetts: The MIT Press. ISBN: 9780262539807.
- Onah, D. F., Sinclair, J. and Boyatt, R. (2014) “Dropout rates of massive open online courses: behavioural patterns”. In: *EDULEARN14 proceedings* 1, pp. 5825–5834. Available <http://wrap.warwick.ac.uk/65543/>.
- Owoseni, A. and Akanji, S. A. (2016) “Survey on Adverse Effect of Sophisticated Integrated Development Environments on Beginning Programmers’ Skillfulness”. In: *International Journal of Modern Education and Computer Science* 8.9, pp. 28–34. ISSN: 20750161, 2075017X. DOI: 10.5815/ijmecs.2016.09.04. Available <http://www.mecs-press.org/ijmecs/ijmecs-v8-n9/v8n9-4.html> [accessed 1st Aug. 2023].
- Pew Research Center (2021) *Internet/Broadband Fact Sheet*. Tech. rep. Washington, D.C.: Pew Research Center. Available <https://about.coursera.org/press/wp-content/uploads/2021/11/2021-Coursera-Impact-Report.pdf>.
- Rodriguez, A. M. and Woodhouse, T. (2022) *Mobile data costs have increased, making internet connectivity unaffordable for many*. Available <https://>

- webfoundation.org/2022/03/mobile-data-costs-have-increased-making-internet-connectivity-unaffordable-for-many/.
- Rose, N. and Barton, P. (2012) *Software development: Europe's top court rules on software copyright*. United Kingdom. Available <https://www.fieldfisher.com/en/insights/software-development-europes-top-court-rules-on-software-copyright>.
- Sauer, J., Sonderegger, A. and Schmutz, S. (2020) "Usability, user experience and accessibility: towards an integrative model". In: *Ergonomics* 63.10. PMID: 32450782, pp. 1207–1220. DOI: 10.1080/00140139.2020.1774080.
- Sauro, J. (2011) *A practical guide to the system usability scale: background, benchmarks & best practices*. Denver, CO: Measuring Usability LLC. ISBN: 9781461062707.
- Sauro, J. and Lewis, J. R. (2016a) "Chapter 3 - How precise are our estimates? Confidence intervals". In: *Quantifying the User Experience (Second Edition)*. Boston: Morgan Kaufmann, pp. 19–38. ISBN: 9780128023082. DOI: 10.1016/B978-0-12-802308-2.00003-5. Available <https://www.sciencedirect.com/science/article/pii/B9780128023082000035> [accessed 15th Aug. 2023].
- Sauro, J. and Lewis, J. R. (2016b) "Chapter 5 - Is There a Statistical Difference between Designs". In: *Quantifying the User Experience (Second Edition)*. Boston: Morgan Kaufmann, pp. 61–102. ISBN: 9780128023082. DOI: 10.1016/B978-0-12-802308-2.00003-5. Available <https://www.sciencedirect.com/science/article/pii/B9780128023082000035> [accessed 15th Aug. 2023].
- Schmutz, S., Sonderegger, A. and Sauer, J. (2016) "Implementing Recommendations From Web Accessibility Guidelines: Would They Also Provide Benefits to Nondisabled Users". In: *Human Factors* 58.4, pp. 611–629. ISSN: 0018-7208. DOI: 10.1177/0018720816640962.
- Schmutz, S., Sonderegger, A. and Sauer, J. (2017) "Implementing Recommendations From Web Accessibility Guidelines: A Comparative Study of Nondisabled Users and Users With Visual Impairments". In: *Hu-*

- man Factors* 59.6, pp. 956–972. ISSN: 0018-7208. DOI: 10 . 1177 / 0018720817708397.
- Schmutz, S., Sonderegger, A. and Sauer, J. (2018) “Effects of accessible web-site design on nondisabled users: age and device as moderating factors”. In: *Ergonomics* 61.5, pp. 697–709. ISSN: 0014-0139. DOI: 10 . 1080 / 00140139.2017.1405080.
- Schmutz, S., Sonderegger, A. and Sauer, J. (2019) “Easy-to-read language in disability-friendly web sites: Effects on nondisabled users”. In: *Applied Ergonomics* 74, pp. 97–106. ISSN: 0003-6870. DOI: 10.1016/j.apergo.2018.08.013.
- Schrepp, M., Kollmorgen, J. and Thomaschewski, J. (2023) “A Comparison of SUS, UMUX-LITE, and UEQ-S”. In: *Journal of User Experience* 18.2, pp. 86–104. ISSN: 1931-3357.
- Shieber, S. (2014) “Why scholars should write in Markdown”. In: available <https://www.semanticscholar.org/paper/Why-scholars-should-write-in-Markdown-Shieber/08cfff45a757343c9fc3fe42cef253b11c30a727> [accessed 1st Aug. 2023].
- Shocken, S. and Nisan, N. (2017) *From Nand to Tetris - Building a Modern Computer From First Principles*. Available <https://www.nand2tetris.org/> [accessed 24th Aug. 2021].
- Silver, L., Smith, A., Johnson, C., Jiang, J., Anderson, M. and Rainie, L. (2019) *Mobile Connectivity in Emerging Economies*. Tech. rep. Washington, D.C.: Pew Research Center. Available <https://www.pewresearch.org/internet/2019/03/07/mobile-connectivity-in-emerging-economies/>.
- Sommerville, I. (2019) *Engineering software products*. First edition. Pearson. ISBN: 9780135210642.
- Stack Overflow (2023) *Stack Overflow Developer Survey 2023*. Available <https://survey.stackoverflow.co/2023> [accessed 1st Aug. 2023].
- StatCounter (2023a) *Desktop vs Mobile Market Share Europe*. Available <https://gs.statcounter.com/platform-market-share/desktop->

- mobile-tablet/europe/#monthly-201208-202307 [accessed 1st Aug. 2023].
- StatCounter (2023b) *Desktop vs Mobile Market Share North America*. Available <https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/north-america#monthly-201208-202307> [accessed 1st Aug. 2023].
- StatCounter (2023c) *Desktop vs Mobile Market Share North America*. Available <https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/africa#monthly-201208-202307> [accessed 1st Aug. 2023].
- StatCounter (2023d) *Desktop vs Mobile Market Share North America*. Available <https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/asia#monthly-201208-202307> [accessed 1st Aug. 2023].
- StatCounter (2023e) *Desktop vs Mobile Market Share Worldwide*. Available <https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-200907-202307> [accessed 1st Aug. 2023].
- StatCounter (2023f) *Operating System Market Share Worldwide*. Available <https://gs.statcounter.com/os-market-share#monthly-200907-202307> [accessed 1st Aug. 2023].
- StatCounter (2023g) *Screen Resolution Stats Worldwide*. Available <https://gs.statcounter.com/screen-resolution-stats#monthly-200907-202307> [accessed 1st Aug. 2023].
- Stikkolorum, D. R., Demuth, B., Zaytsev, V., Boulanger, F. and Gray, J. (2014) “The MOOC Hype: Can We Ignore It? Reflections on the Current Use of Massive Open Online Courses in Software Modeling Education.” In: *EduSymp 2014 : MODELS Educators Symposium 2014: proceedings of the MODELS Educators Symposium, co-located with the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014)*. Valencia, Spain, pp. 75–86. Available [https://dare.uva.nl/personal/pure/en/publications/the-mooc-hype-can-we-ignore-it\(b0bff0f2-7e02-446e-afd3-33b18ad155cf\).html](https://dare.uva.nl/personal/pure/en/publications/the-mooc-hype-can-we-ignore-it(b0bff0f2-7e02-446e-afd3-33b18ad155cf).html).

- Suslina, I. and Tarasova, V. (2018) “Approaches to Legal Protection of Software Made by Foreign Authors in the State of Israel”. en. In: *Procedia Computer Science* 145, pp. 551–554. ISSN: 18770509. DOI: 10.1016/j.procs.2018.11.120. Available <https://linkinghub.elsevier.com/retrieve/pii/S1877050918324086>.
- Tadesse, S. and Muluye, W. (2020) “The Impact of COVID-19 Pandemic on Education System in Developing Countries: A Review”. In: *Open Journal of Social Sciences* 08.10, pp. 159–170. ISSN: 2327-5952, 2327-5960. DOI: 10.4236/jss.2020.810011. Available <https://www.scirp.org/journal/doi.aspx?doi=10.4236/jss.2020.810011>.
- Tan, C. S. S., Schöning, J., Luyten, K. and Coninx, K. (2013) “Informing intelligent user interfaces by inferring affective states from body postures in ubiquitous computing environments”. In: *Proceedings of the 2013 international conference on Intelligent user interfaces - IUI '13*. ACM Press, p. 235. ISBN: 9781450319652. DOI: 10.1145/2449396.2449427. Available <http://dl.acm.org/citation.cfm?doid=2449396.2449427>.
- The Free Software Foundation (2023) *What is Free Software?* Available <https://www.gnu.org/philosophy/free-sw.en.html> [accessed 29th Aug. 2023].
- Thüring, M. and Mahlke, S. (2007) “Usability, aesthetics and emotions in human–technology interaction”. In: *International Journal of Psychology* 42.4, pp. 253–264. ISSN: 1464-066X. DOI: 10.1080/00207590701396674.
- Tomassetti, G. (2017) *A Guide To Parsing: Algorithms And Terminology*. Available <https://tomassetti.me/guide-parsing-algorithms-terminology/>.
- Tullis, T. and Stetson, J. N. (2004) “A Comparison of Questionnaires for Assessing Website Usability”. In: available <https://www.semanticscholar.org/paper/A-Comparison-of-Questionnaires-for-Assessing-Tullis-Stetson/9d621a71a9c9e24f689f5263ed5b74e53535374a>.
- U.S. Department of Education (2021) *Digest of Education Statistics*. Washington, D.C.: National Center for Education Statistics.

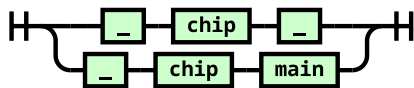
- UNESCO (2022) *Education: From disruption to recovery*. Available <https://en.unesco.org/covid19/educationresponse>. Archived <https://web.archive.org/web/20220107074613/https://en.unesco.org/covid19/educationresponse>.
- Valez, M., Yen, M., Le, M., Su, Z. and Alipour, M. A. (2020) “Student Adoption and Perceptions of a Web Integrated Development Environment: An Experience Report”. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, pp. 1172–1178. ISBN: 9781450367936. DOI: 10.1145/3328778.3366949. Available <https://dl.acm.org/doi/10.1145/3328778.3366949> [accessed 1st Aug. 2023].
- Vanderheiden, G. (2000) “Fundamental principles and priority setting for universal usability”. In: *Proceedings on the 2000 conference on Universal Usability*. CUU '00. Association for Computing Machinery, pp. 32–37. ISBN: 9781581133141. DOI: 10.1145/355460.355469.
- Vihavainen, A., Helminen, J. and Ihanola, P. (2014) “How novices tackle their first lines of code in an IDE: analysis of programming session traces”. In: *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*. ACM, pp. 109–116. ISBN: 9781450330657. DOI: 10.1145/2674683.2674692. Available <https://dl.acm.org/doi/10.1145/2674683.2674692>.
- Voegler, J., Bornschein, J. and Weber, G. (2014) “Markdown – A Simple Syntax for Transcription of Accessible Study Materials”. en. In: *Computers Helping People with Special Needs*. Ed. by K. Miesenberger, D. Fels, D. Archambault, P. Peñáz and W. Zagler. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 545–548. ISBN: 9783319085968. DOI: 10.1007/978-3-319-08596-8_85.
- Web Accessibility Initiative (2014) *Involving Users in Evaluating Web Accessibility*. Available <https://www.w3.org/TR/2014/NOTE-WCAG-EM-20140710/> [accessed 11th May 2023]. Archived <http://web.archive.org/web/20230511002641/https://www.w3.org/TR/2014/NOTE-WCAG-EM-20140710/>.

- Web Accessibility Initiative (2016) *Accessibility, Usability, and Inclusion*. Available <https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/> [accessed 4th Dec. 2021]. Archived <https://web.archive.org/web/20211204150932/https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/>.
- Web Accessibility Initiative (2017) *Selecting Web Accessibility Evaluation Tools*. Available <https://www.w3.org/WAI/test-evaluate/tools/selecting/> [accessed 4th Dec. 2021]. Archived <https://web.archive.org/web/20211120123630/https://www.w3.org/WAI/test-evaluate/tools/selecting/>.
- Web Accessibility Initiative (2018) *Web Accessibility Laws & Policies*. Available <https://www.w3.org/WAI/policies/> [accessed 4th Dec. 2021]. Archived <https://web.archive.org/web/20211201202600/https://www.w3.org/WAI/policies/>.
- Web Accessibility Initiative (2020) *Involving Users in Evaluating Web Accessibility*. Available <https://www.w3.org/WAI/test-evaluate/involving-users/> [accessed 4th Dec. 2021]. Archived <https://web.archive.org/web/20210513020157/https://w3c.github.io/wai-website/test-evaluate/involving-users/>.
- Web Accessibility Initiative (2021a) *Involving Users in Evaluating Web Accessibility*. Available <https://www.w3.org/WAI/test-evaluate/conformance/wcag-em/> [accessed 4th Dec. 2021]. Archived <https://web.archive.org/web/20210101134807/https://w3c.github.io/wai-website/test-evaluate/conformance/wcag-em/>.
- Web Accessibility Initiative (2021b) *Web Content Accessibility Guidelines (WCAG) Overview*. Available <https://www.w3.org/WAI/standards-guidelines/wcag/> [accessed 27th Dec. 2021]. Archived <https://web.archive.org/web/20211227183535/https://www.w3.org/WAI/standards-guidelines/wcag/>.
- Wegge, K. P. and Zimmermann, D. (2007) “Accessibility, Usability, Safety, Ergonomics: Concepts, Models, and Differences”. In: ed. by C. Stephanidis. Lecture Notes in Computer Science. Springer, pp. 294–301. ISBN: 9783540732792. DOI: 10.1007/978-3-540-73279-2_33.

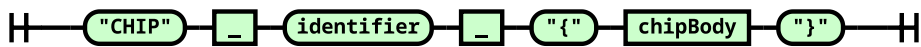
- Wilson, C. (2009) *User Experience Re-Mastered: Your Guide to Getting the Right Design*. Morgan Kaufmann Publishers Inc. ISBN: 9780123751140. DOI: 10.1016/C2009-0-20682-9.
- Wilson, R. J. (2009) *Introduction to graph theory*. 4. ed., [Nachdr.] Harlow Munich: Prentice Hall. ISBN: 9780582249936.
- Yu, X., Karsten, E., Kenney, D., Sinha, A. and Brunamonti, C. (2021) *2021 Coursera Impact Report*. Tech. rep. Mountain View, California: Coursera. Available <https://about.coursera.org/press/wp-content/uploads/2021/11/2021-Coursera-Impact-Report.pdf>.

A HDL Railroad Diagrams

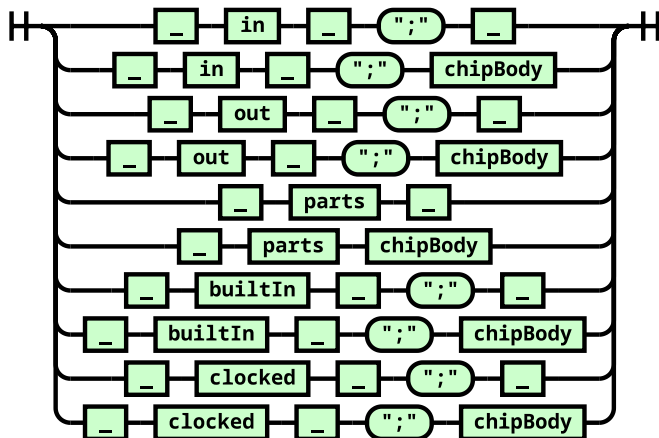
main



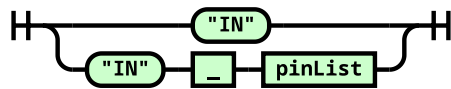
chip



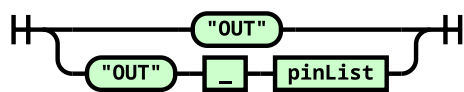
chipBody



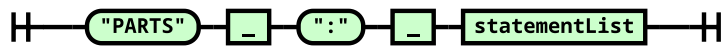
in



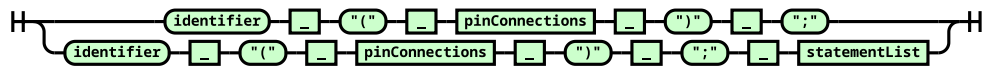
out



parts



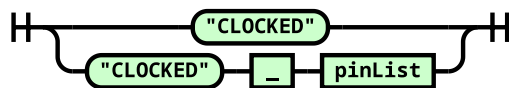
statementList



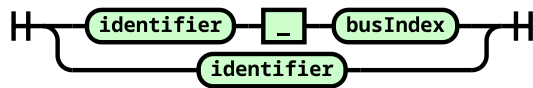
builtIn



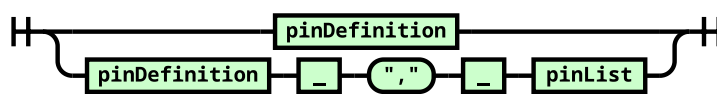
clocked



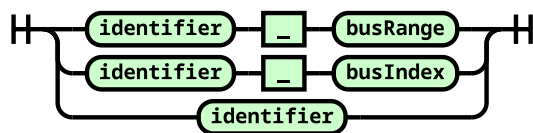
pinDefinition



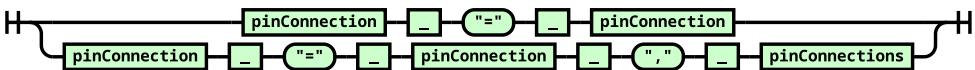
pinList



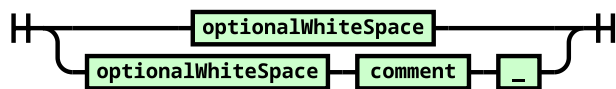
pinConnection



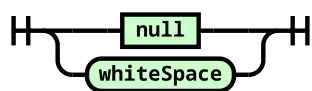
pinConnections



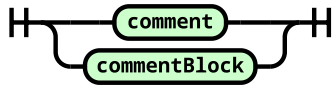
—



optionalWhiteSpace



comment



B HDL Token, Grammar Definition

This appendix is also available online with syntax highlighting & easier to browse at github.com/durasj/chipsandcodeblob/4576a0b/src/lib/editor/hdl.

```
@{%
import moo from 'moo';
const lexer = moo.compile({
  leftBrace:    '{',
  rightBrace:   '}',
  leftParen:    '(',
  rightParen:   ')',
  semicolon:    ';',
  colon:        ':',
  comma:        ',',
  equals:       '=',
  keyword:      ['CHIP', 'IN', 'OUT', 'PARTS', 'BUILTIN', 'CLOCKED'],
  identifier:   /[a-zA-Z_][a-zA-Z0-9_\.]*/,
  busIndex:     { // specification for a bus like [1] [8]
    match: /\[\d+\]/, value: s => +s.slice(1, -1)
  },
  busRange:     { // bus lane range selection [0..7]
    match: /\[\d+\.\.\d+\]/,
    value: s => s.slice(1, -1).split('..').map(Number)
  },
  number:       /\d+/,
  whiteSpace:    { match: /\s+/, lineBreaks: true },
  comment:      { match: /\\/\[^\n]*\/, value: c => c.slice(2) },
  commentBlock: {
    match: /\\/\*[\s\S]*?\*\/,
    lineBreaks: true,
    value: c => c.slice(2, -2),
  }
});
```

Appendix B. HDL Token, Grammar Definition

```
    }
  });

  const getIdentifier = ({ value, col, line, lineBreaks, offset }) =>
    ({ value, col, line, lineBreaks, offset });
}%}

@lexer lexer

main ->
  _ chip _      {% ([, stm]) => [stm] %}
  | _ chip main {% ([, stm, acc]) => ([stm, ...acc]) %}

chip ->
  "CHIP" _ %identifier _ "{" chipBody "}" {% ([, , name, , , body]) => ({
    ↪ type: 'chip', name: getIdentifier(name), body }) %}
chipBody ->
  _ in _ ";" _      {% ([, id]) => [id] %}
  | _ in _ ";" chipBody {% ([, id, , , acc]) => [id, ...acc] %}
  | _ out _ ";" _     {% ([, id]) => [id] %}
  | _ out _ ";" chipBody {% ([, id, , , acc]) => [id, ...acc] %}
  | _ parts _        {% ([, id]) => [id] %}
  | _ parts chipBody  {% ([, id, acc]) => [id, ...acc] %}
  | _ builtIn _ ";" _  {% ([, id]) => [id] %}
  | _ builtIn _ ";" chipBody {% ([, id, , , acc]) => [id, ...acc] %}
  | _ clocked _ ";" _  {% ([, id]) => [id] %}
  | _ clocked _ ";" chipBody {% ([, id, , , acc]) => [id, ...acc] %}

in ->
  "IN"           {% () => ({ type: 'input', pins: [] }) %}
  | "IN" _ pinList {% ([, , pins]) => ({ type: 'input', pins }) %}

out ->
  "OUT"           {% () => ({ type: 'output', pins: [] }) %}
  | "OUT" _ pinList {% ([, , pins]) => ({ type: 'output', pins }) %}

parts -> "PARTS" _ ":" _ statementList {% ([, , , , id]) => ({ type:
  ↪ 'parts', statements: id }) %}
statementList ->
```

Appendix B. HDL Token, Grammar Definition

```
%identifier _ "(" _ pinConnections _ ")" _ ";"                                {%
  ↳ ([chip, , , , connections]) => [{ type: 'statement', chip:
  ↳ getIdentifier(chip), connections }] %}
| %identifier _ "(" _ pinConnections _ ")" _ ";" _ statementList            {%
  ↳ ([chip, , , , connections, , , , , acc]) => [{ type: 'statement',
  ↳ chip: getIdentifier(chip), connections }, ...acc] %}

builtIn -> "BUILTIN" _ %identifier {% ([, , id]) => ({ type: 'builtin',
  ↳ template: getIdentifier(id) }) %}

clocked ->
  "CLOCKED"                                {% () => ({ type: 'clocked', pins: [] }) %}
| "CLOCKED" _ pinList                      {% ([, , pins]) => ({ type: 'clocked', pins })
  ↳ %}

pinDefinition ->
  %identifier _ %busIndex                   {% ([pin, , spec]) => ({
  ↳ ...getIdentifier(pin), width: spec.value }) %}
| %identifier                             {% ([pin]) => getIdentifier(pin) %}

pinList ->
  pinDefinition                            {% ([pin]) => [pin] %}
| pinDefinition _ "," _ pinList            {% ([pin, , , , acc]) => ([pin,
  ↳ ...acc]) %}

pinConnection ->
  %identifier _ %busRange                   {% ([pin, , spec]) => ({
  ↳ ...getIdentifier(pin), selection: spec.value }) %}
| %identifier _ %busIndex                   {% ([pin, , spec]) => ({
  ↳ ...getIdentifier(pin), selection: spec.value }) %}
| %identifier                             {% ([pin]) => getIdentifier(pin) %}

pinConnections ->
  pinConnection _ "=" _ pinConnection      {%
  ↳ ([left, , , , right]) => [{ type: 'assignment', left, right }] %}
| pinConnection _ "=" _ pinConnection _ "," _ pinConnections              {%
  ↳ ([left, , , , right, , , , acc]) => [{ type: 'assignment', left,
  ↳ right }, ...acc] %}

# Space and comments
_ ->
```

Appendix B. HDL Token, Grammar Definition

```
optionalWhiteSpace      {% id %}
| optionalWhiteSpace comment _ {% ([, comment]) => comment %}
optionalWhiteSpace ->
  null {% id %}
| %whiteSpace    {% () => null %}
comment ->
  %comment      {% ([content]) => ({ type: 'comment', content }) %}
| %commentBlock {% ([content]) => ({ type: 'comment', content }) %}
```


C HDL Example

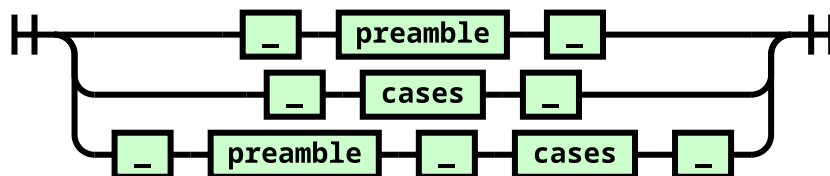
This example uses the language introduced by Nisan and Schocken (2021).

```
/**
 * XOR - Exclusive OR gate
 *
 * Outputs 1 only if both inputs differ
 * Else outputs 0
 */
CHIP Xor {
    IN a, b;
    OUT out;

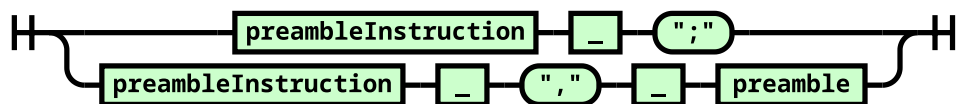
    PARTS:
    Not(in=a, out=nota);
    Not(in=b, out=notb);
    And(a=a, b=notb, out=w1);
    And(a=nota, b=b, out=w2);
    Or(a=w1, b=w2, out=out);
}
```

D TST Railroad Diagrams

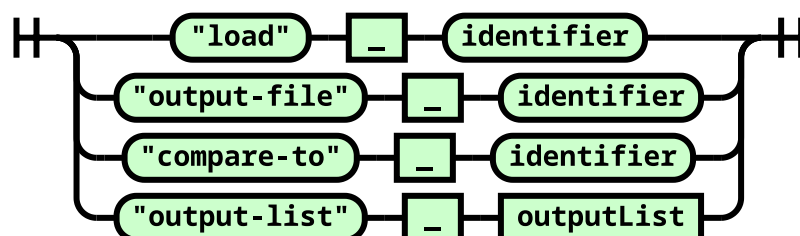
main



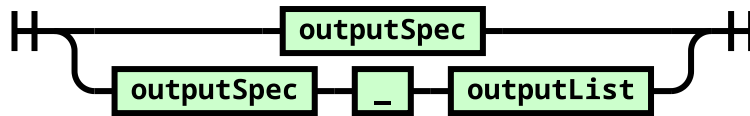
preamble



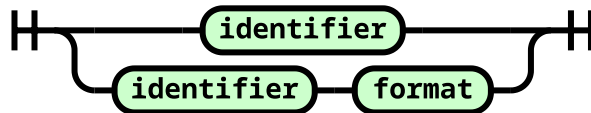
preambleInstruction



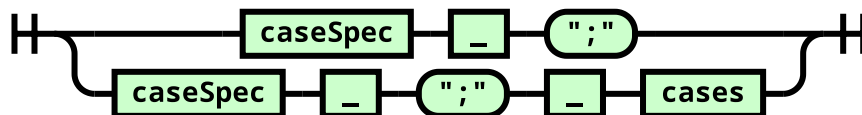
outputList



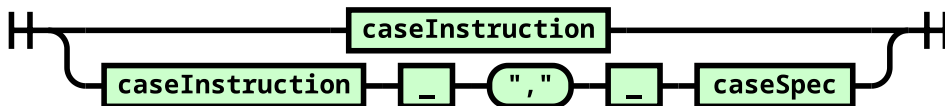
outputSpec



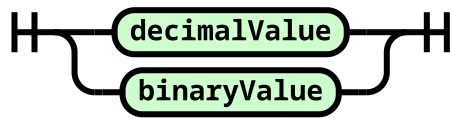
cases



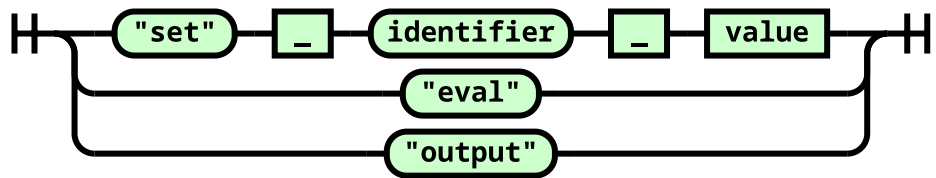
caseSpec



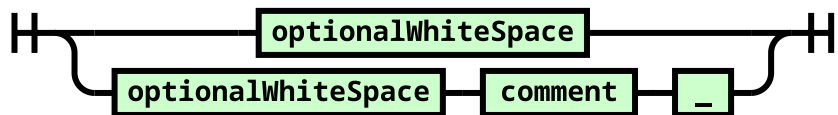
value



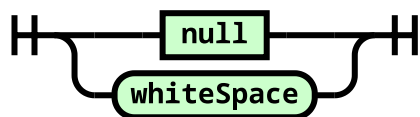
caseInstruction



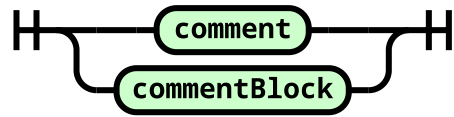
—



optionalWhiteSpace



comment



E TST Token, Grammar Definition

This appendix is also available online with syntax highlighting & easier to browse at github.com/durasj/chipsandcodeblob/4576a0b/src/lib/editor/tst.

```
@{%
import moo from 'moo';
const lexer = moo.compile({
  semicolon:      ';',
  colon:          ':',
  comma:          ',',
  equals:         '=',
  format: {
    match: /%[BXDS] [0-9]+\.[1-9] [0-9]*\.[0-9]+/,
    value: f => [f[1], ...f.substr(2).split('.').map(Number)],
  },
  identifier: {
    match: /[a-zA-Z_][a-zA-Z0-9-_\.\.]*/,
    type: moo.keywords({
      keyword: ['load', 'output-file', 'compare-to', 'output-list',
        ↪ 'set', 'eval', 'output'],
    }),
  },
  decimalValue: { match: /\d+/, value: (v) => +v },
  // TODO: Add support for X, D, S values
  binaryValue: {
    match: /%B[01]+/,
    value: f => ({ type: 'value', format: 'binary', value:
      ↪ f.substr(2).split('').map(Boolean)}),
  },
  whiteSpace:     { match: /\s+/, lineBreaks: true },
  comment:        { match: /\s*\/\s*\/[^\n]*/, value: c => c.slice(2) },
```

Appendix E. TST Token, Grammar Definition

```
commentBlock:  { match: /\/*[\s\S]*?\*\/, lineBreaks: true, value:
  ↪ c => c.slice(2, -2) }
});

const getIdentifier = ({ value, col, line, lineBreaks, offset }) =>
  ({ value, col, line, lineBreaks, offset });
%}

@lexer lexer

main ->
  _ preamble _      {% ([, preamble]) => ({ type: 'script',
  ↪ preamble, cases: [] }) %}
| _ cases _         {% ([, cases]) => ({ type: 'script', preamble:
  ↪ [], cases }) %}
| _ preamble _ cases _ {% ([, preamble, , cases]) => ({ type: 'script',
  ↪ preamble, cases }) %}

preamble ->
  preambleInstruction _ ";"          {% ([instruction]) =>
  ↪ [instruction] %}
| preambleInstruction _ "," _ preamble {% ([instruction, , , acc]) =>
  ↪ ([instruction, ...acc]) %}

preambleInstruction ->
  "load" _ %identifier      {% ([, , file]) => ({ type: 'load',
  ↪ file: getIdentifier(file) }) %}
| "output-file" _ %identifier {% ([, , file]) => ({ type: 'output',
  ↪ file: getIdentifier(file) }) %}
| "compare-to" _ %identifier  {% ([, , file]) => ({ type: 'compare',
  ↪ file: getIdentifier(file) }) %}
| "output-list" _ outputList  {% ([, , outputs]) => ({ type:
  ↪ 'outputList', outputs }) %}

outputList ->
  outputSpec          {% ([spec]) => [spec] %}
| outputSpec _ outputList {% ([spec, , acc]) => ([spec, ...acc]) %}

outputSpec ->
```

Appendix E. TST Token, Grammar Definition

```
%identifier          {% ([name]) => ({ type: 'outputSpec', name:
  ↳ getIdentifier(name) }) %}
| %identifier %format  {% ([name, f]) => ({ type: 'outputSpec',
  ↳ name: getIdentifier(name), format: f.value[0], length: f.value[2],
  ↳ padLeft: f.value[1], padRight: f.value[3] }) %}

cases ->
  caseSpec _ ";"      {% ([instructions]) => [instructions] %}
| caseSpec _ ";" _ cases {% ([instructions, , , , acc]) =>
  ↳ ([instructions, ...acc]) %}

caseSpec ->
  caseInstruction      {% ([instruction]) => [instruction]
  ↳ %}
| caseInstruction _ "," _ caseSpec {% ([instruction, , , , acc]) =>
  ↳ ([instruction, ...acc]) %}

value ->
  %decimalValue {% id %}
| %binaryValue  {% id %}

caseInstruction ->
  "set" _ %identifier _ value {% ([kw, , name, , value]) => ({ type:
  ↳ 'set', name: getIdentifier(name), value: value.value, col: kw.col,
  ↳ line: kw.line }) %}
| "eval"          {% ([kw]) => ({ type: 'eval', col:
  ↳ kw.col, line: kw.line }) %}
| "output"        {% ([kw]) => ({ type: 'output', col:
  ↳ kw.col, line: kw.line }) %}

# Space and comments
_ ->
  optionalWhiteSpace  {% id %}
| optionalWhiteSpace comment _ {% ([, comment]) => comment %}
optionalWhiteSpace ->
  null {% id %}
| %whiteSpace  {% () => null %}
comment ->
  %comment          {% ([content]) => ({ type: 'comment', content }) %}
```


Appendix E. TST Token, Grammar Definition

```
| %commentBlock {% ([content]) => ({ type: 'comment', content }) %}
```

F TST Example

This example uses the language introduced by Nisan and Schocken (2021).

```
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;
```

```
set a 0,  
set b 0,  
eval,  
output;
```

```
set a 0,  
set b 1,  
eval,  
output;
```

```
set a 1,  
set b 0,  
eval,  
output;
```

```
set a 1,  
set b 1,  
eval,  
output;
```

G Study SUS Scores

The data presented in this appendix are also alternatively available online at https://docs.google.com/spreadsheets/d/1CGNs1pCKvGJ4QuRDCEXnc_3SZ4pgX3U1TXSKJU8hpa8 within a Google Spreadsheet.

Group A

Person	1. Frequently	2. Complex	3. Easy to Use	4. Technical support	5. Well Integrated	6. Inconsistency	7. Learn Quickly	8. Awkward	9. Felt Confident	10. Learn Lot	Score
M	4	2	5	3	5	2	5	2	3	2	77.5
LU	5	1	5	4	5	1	3	1	5	2	85
Y	5	1	5	2	5	1	5	1	5	3	92.5
MI	3	1	5	1	4	2	5	1	4	2	85
MX	1	1	5	1	5	1	3	2	5	1	82.5
R	4	1	5	1	4	1	3	1	4	1	87.5
MK	5	1	5	1	5	1	4	1	5	1	97.5
L	5	1	5	1	5	1	5	1	5	1	100
MA	4	3	4	4	4	2	4	2	3	2	65
KY	5	1	5	2	4	1	5	1	5	1	95
KA	4	2	5	2	4	2	5	2	4	3	77.5
E	4	1	5	1	5	1	4	1	5	1	95

Group B

Person	1. Frequently	2. Complex	3. Easy to Use	4. Technical support	5. Well Integrated	6. Inconsistency	7. Learn Quickly	8. Awkward	9. Felt Confident	10. Learn Lot	Score
P	3	2	4	2	4	1	5	2	5	1	82.5
S	3	2	4	3	4	3	5	2	4	2	70
MM	4	2	4	1	5	1	5	2	5	1	90
SA	3	4	2	2	2	4	3	4	2	3	37.5
JO	4	1	3	3	5	1	4	1	3	2	77.5
PE	3	1	4	1	5	1	5	1	5	1	92.5
MS	5	2	5	1	5	1	5	1	5	1	97.5
B	4	1	5	1	4	1	5	3	5	2	87.5
V	4	4	4	2	2	1	4	4	5	2	65
MH	5	2	4	1	4	1	5	1	4	1	90
LB	4	2	4	2	4	1	3	1	4	3	75
MT	4	2	4	1	5	1	5	1	5	2	90