

User Testing Tool

Towards a tool for crowdsource-enabled accessibility evaluation of
websites

Alexander Teinum

Supervisors

Janis Gailis (internal), Mikael Snaprud (external)

This Master's Thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

Preface

I want to take a moment to acknowledge and thank all who have been involved in some way or another during this research project.

First out, cand. scient. Janis Gailis—thank you for accepting the offer to supervise this project. I am especially grateful for your feedback regarding the choice and use of design research. Your enthusiasm for the open source philosophy has not gone unnoticed over the years at the university.

Dr. Mikael Snaprud, I should start by saying thanks for sharing the burden with Janis by being my supervisor. Your contribution during this project is simply beyond words. The last year in Tingtun has been a fantastic experience, and I would like to use this opportunity to give a nod to all my co-workers around the world.

This project would not be possible without the financial support received by The Research Council of Norway. I would like to express my sincere thanks to the partners of the UTT project for your invaluable feedback during the three rounds of evaluation.

I extend my gratitude to Birkir Gunnarsson and Jeroen Hulscher for engaging in email conversations about the implementation of UTT. Also, I would like to thank Thomas Holmstrøm Frandzen and Ruben Wangberg for providing insightful comments on the report.

I especially thank my family for providing love and support—and food—during this intense period of work. I also want to thank my friends for just being there, and for taking me on volleyball breaks every once in a while.

Thanks,

Alexander Teinum, 2013-06-07



Abstract

This thesis describes the first open source tool to combine user testing and automated testing to check accessibility of websites. *User Testing Tool* (UTT) integrates with an existing automated checker for testing websites against the WCAG 2.0 guidelines. UTT generates and presents questions that need human verification, such as whether an alternative text representation is appropriate for an image on a web page. In the future, collected data can be used to both improve the accessibility of the website, as well as making the automatic checker smarter.

How to enable more people to improve accessibility testing is the main question addressed. Based on our proposed solution the thesis also deals with how to integrate an automated checker with user testing, how user tests can enhance automated checker tests, how to design the user interface, and how to remove any obstacles preventing larger numbers of people to contribute for crowdsourcing the tests of web sites.

An open source prototype based on an iframe technique was built to demonstrate a viable path of development. The user interface integrates with the web page to be tested, and it satisfies several usability criteria. The solution covers design of an API for the automatic checker for receiving test results, a control flow mechanism, and a user interface iteratively refined involving evaluators.

The user interface is implemented using a SPA architecture, a fat client architecture for the web. This will reduce the demand for bandwidth and server capacity and should therefore be suitable for crowdsourcing. The user interface needs to take privacy into account by giving the user control about what data to collect. Currently this is not a problem since the prototype does not store any data. To initiate the development of a user testing tool, this thesis has focused on a solid architecture along with features that serve a demonstrative purpose. The proposed architecture and the open source approach is designed to facilitate further development.

Design Research was the chosen research method, where knowledge was derived from creation of the prototype and interaction with the users. The research problem is based on a practical need for better testing tools. User testing of the tool itself uncovered that efficient means of user entry is crucial, and that the tool needs to adjust for a variety of devices and assistive technologies.

Future research can include storage of collected data, explore approaches to deal with privacy and logging of data, data analysis for quality assurance, further investigation of the proxy-iframe technique, and integration with the eAccessibility Checker result view.

Contents

1	Introduction	1
1.1	Context	2
1.2	Research problem	2
1.3	Document overview	3
2	Literature review	4
2.1	Selection of literature	4
2.2	Design research	5
2.3	Human–computer interaction	10
2.4	Accessibility	12
2.5	Usability	14
2.6	User experience	16
2.7	Selected topics of web technologies	16
2.8	Crowdsourcing	22
2.9	Open source software	23
3	Existing tools	25
3.1	Selection process	25
3.2	The selection of tools	26
3.3	Loop11	28
3.4	Usabilla	32
3.5	Draft	33
3.6	Infomaki	34
3.7	ClickHeat	36
3.8	eAccessibility Checker	36
3.9	Achecker	37
3.10	Existing research projects	37
3.11	Recruiting services	38
3.12	Findings	39
3.13	Objectives and requirements	41
4	Method	44
4.1	Design research	44
4.2	Borrowing terminology	46
4.3	Outcomes	47
4.4	Research model	47
4.5	Collaboration	47

4.6 Other	52
5 Design	53
5.1 Revisiting requirements	53
5.2 Designing the user interface	54
5.3 Specific user interface functionality	56
5.4 The implementation	63
5.5 Implementing the frontend	65
5.6 Backend implementation and architecture	69
5.7 Integration with eAccessibility Checker	70
5.8 Architectural concerns	71
5.9 Deploying and monitoring the solution	72
5.10 Choice of technologies	73
6 Discussion	87
6.1 Reflecting on the method and prior art	87
6.2 Existing tools	88
6.3 Design	88
6.4 Limitations	91
6.5 Research questions	92
6.6 Suggested future research	95
6.7 Reflecting on the process	99
6.8 Concerns regarding privacy	100
7 Conclusion	103
8 Glossary	104
9 Appendix	105
9.1 Gantt chart	105
References	106

List of Figures

1	The Design Cycle model	7
2	The Design Science Research Cycles	8
3	The Learning Design Framework	9
4	Loop11 presenting a task during a test run	28
5	Loop11 user interface for managing a test run	30
6	Defining test page terms	31
7	Studying the proxy-iframe technique used by Loop11	31
8	Commenting on an element using Usabilla	33
9	Commenting on an image using Draft	34
10	Infomaki presenting a heat map	35
11	ClickHeat showing a heatmap	36
12	eAccessibility Checker checking uia.no	36
13	Achecker checking uia.no	37
14	Research model	47
15	Initial mockup drawn on paper	54
16	Higher fidelity mockup	55
17	Tests of same type shown in single view	55
18	The home page of UTT 0.3	56
19	The test page of UTT 0.3	57
20	The result page of UTT 0.3	58
21	UTT dressed in black	58
22	The pre-0.1 solution currently in use	61
23	The 0.1 solution proven to be ineffective	61
24	Tests of same type grouped into one page	62
25	UTT translated into Norwegian	63
26	Architecture overview	64
27	The directory structure in the git repository of UTT	71
28	CoffeeScript and JavaScript side-by-side	78
29	The result view of eAccessibility Checker	92
30	Gantt chart	105

List of Tables

1	The 7 principles of Universal Design	14
2	The attributes of usability (Nielsen-Hackos, 1993)	15
3	Proprietary tools—part 1	27

4	Proprietary tools—part 2	27
5	Open source tools	27
6	Automated checker tools	28
7	Stakeholders	48
8	The chosen technology stack	85
9	Glossary	104

1 Introduction

As the Internet is gradually changing our society with the increasing digitization of services, the need for accessible and usable websites becomes crucial. People with disabilities still find insurmountable barriers on websites and online services, preventing full participation. The need for accessible online services will continue to grow with increasing life expectancy (E. Commission, 2007).

The Web Content Accessibility Guidelines 2.0 (WCAG) from the World Wide Web Consortium (W3C) is a set of guidelines with defined success criteria on how the accessibility of a website should be designed (Caldwell, Cooper, Reid, & Vanderheiden, 2008). While it is possible to manually check a website against these guidelines, a more effective approach is to automate the testing. eAccessibility Checker¹ (Nietzio, Eibegger, Goodwin, & Snaprud, 2012) is one such tool that automatically checks a website against the success criteria defined in WCAG 2.0.

Some of the success criteria in WCAG 2.0 can be automatically tested, but there are also some whose result cannot be reliably determined by a computer. A checker can e.g. discover if the alternative text for an image is missing, or if it is a duplicate of another alternative text. Clearly, it is much harder to automatically determine to what extent an alternative text is indeed a good alternative source of information for those who cannot see the image. For this and similar cases, the eAccessibility Checker produces a result called “to-be-verified,” indicating that human verification is yet needed.

In the course of this master’s project, a software artifact will be designed that integrates with the eAccessibility Checker, to allow human testers to determine what cannot be determined automatically.

The aim of the thesis is to arrive at a prototype to show a viable way towards a crowdsourced user testing solution for websites. This implies that the tool should have the potential to reach a large audience, and to facilitate for this wide use, the tool is iteratively developed, and design decisions are informed by the feedback from evaluators. Given this background, design research is chosen as the research method.

¹<http://accessibility.egovmon.no/>

1.1 Context

The master's project is organized in the context of a pre-project supported by the Research Council of Norway, under the IT Funk program (*User Testing Tool forprosjekt – IT Funk*, 2012). The project is co-ordinated by Tingtun AS and is named *User Testing Tool* (UTT). The partners include Accessibility Foundation (the Netherlands), Evangelische Stiftung Volmarstein / Forschungsinstitut Technologie und Behinderung (Germany) and Seniornett (Norway). All partners have strong interest in website accessibility testing. Both to shape the UTT design and facilitate the wider take-up of the project results, the UTT project has established a strategic reference group. To obtain further user testing input, additional organizations have been invited to form a reference group, including The Delta Centre (The National Resource Centre for Participation and Accessibility), Agency for Public Management and eGovernment (Difi), The Norwegian Association of the Blind and Partially Sighted (NABP), Danish Agency for digitisation, Ministry of State for Administrative Development in Egypt, the government agencies Logius, and KING in the Netherlands and several individuals.

Dr. Mikael Snaprud, CEO of the Norwegian company Tingtun AS, is the initiator and co-ordinator of the UTT project. Tingtun AS has co-ordinated the eGovMon project² and earlier international research leading to the eAccessibility tools to be used for automatic evaluation in this thesis.

1.2 Research problem

The main research problem here is how to design a user testing tool that enable more people to improve accessibility testing of websites. Questions supporting this research problem are:

1. How to design a tool that integrates with an automated checker?
2. How can user tests enhance automated checker tests?
3. How to design a user interface to best support the user doing the user testing?

²<http://www.egovmon.no/> The eGovMon project was co-founded by The Research Council of Norway under the VERDIKT program and has delivered tools for benchmarking of eGovernment services.

4. How to best track the user behavior without any code injection on the website to be studied, or software installation on the client side?
5. What existing open source solutions can be helpful to solve the above questions?

1.3 Document overview

This introductory part (Section 1) has identified a problem and outlined requirements for a solution. The literature review gives an outlook to prior research and defines concepts used throughout the thesis. (Section 2) Subsequently a review of existing tools (Section 3) that UTT can potentially build upon is given. The review sections are followed by a discussion of research methods (Section 4), and an outline of how design research is applied.

The Design section (Section 5) details how UTT is implemented; its design choices and associated implications, and the choice of technologies. The Discussion and Conclusion sections (Section 6 and 7) discusses and sums up lessons learned, research contributions, and suggests directions for future research.

2 Literature review

This section introduces existing knowledge related to the research problem, and attempt to build a working model that will be used throughout the thesis. It starts out by describing the selection of literature, and then presenting the chosen subset of the literature that is thought to be relevant to this project. For instance, usability is one concept with many definitions/models. Several have emerged throughout the years, and there have been attempts of consolidating earlier definitions.

The author has chosen to structure the field into groups with each research field as a sub-section in this literature review.

2.0.1 A note on terms and writing conventions

In the literature related to this research there is more than one definition/model of one concept. Also, there are different views regarding how one concept is related to another, and even how fields of concepts relate to other fields. There also exists overlap between the entities mentioned.

We attempt to use a vocabulary that is unambiguous, and we try to stick to one definition/model per concept consistently throughout the thesis. While different views are presented in the literature review, a single definition/model will be used throughout the report. We have tried to choose definitions/models which seem to be well-established.

This report is filled with jargons. A choice regarding consistency had to be made, and concepts in this report are *not* written in title case. When a concept first is introduced, it is written in *italic*, and it is followed by a definition. Once a concept with an abbreviation has been introduced, the abbreviation is generally used, even if the concept has been introduced in an previous section. A glossary is provided for looking up abbreviations (Table 9).

2.1 Selection of literature

Literature review has been a continuous process throughout the project. Digital libraries such as ACM DL, Google Scholar, and Bibsys Ask2 have been used actively for searching the existing base of knowledge. This review also includes resources found on the web, such YouTube videos and blog posts.

Initially, the review was done in a broad, exploratory way—search phrases were strung together by combining keywords such as “hci,” “accessibility,” “usability,” and “web evaluation.” Later on, as we started to gain more insight about the research problem, the searches got more specific. The knowledge-building process has been stimulated by reading literature reviews, books, blog posts, reviewing existing software, and engaging in discussions both internally and with partners.

2.2 Design research

“Following a research through design approach, designers produce novel integrations of HCI research in an attempt to make the right thing: a product that transforms the world from its current state to a preferred state. This model allows interaction designers to make research contributions based on their strength in addressing under-constrained problems.” —Zimmerman, Forlizzi, and Evenson (2007)

Design research is the method used for carrying out the master’s project. The method consists of analytical techniques and perspectives for performing research in the field of information systems. Design research involves the analysis of the use and performance of designed *artifacts* to understand, explain and very frequently to improve on the behavior of aspects of information systems (Vaishnavi & Kuechler, 2004). The primary outcome of this study, the process experience, and the software used for evaluating websites, is an example of a such designed artifact.

Design research will in this section be viewed in a historic context, along with theory behind knowledge. Next, three models are presented to illustrate the nature of design research; two general models that describe its phases, and one viewing design research in an educational context.

2.2.1 The science of the artificial

Design science is also known as *the science of the artificial*. It is related to the term old school of learning that will be discussed in the upcoming section. Design science contrasts to the natural/behavioral science paradigm in how knowledge is built. In natural science, theories about objects or phenomenon in nature or society are developed and verified.

The theories describe and explain how the objects or phenomenon behave and interact with each other. On the other hand, the design science paradigm is about building new and innovative artifacts that set out to solve problems that are derived from practice and/or existing research (Vaishnavi & Kuechler, 2004; Hevner, March, Park, & Ram, 2004). Both paradigms are foundational to the IS discipline, and they are both concerned with humans, organizations, and technology.

In design research, knowledge is generated and accumulated through action (Owen, 1998). The process of creating involves analysis of existing designs, and a lot of experimentation.

2.2.2 Design research in a historic context

Within research communities, the value of design research has for a long time been met with skepticism. There has been uncertainty “whether there is such a thing as design knowledge that merits serious attention” (Buchanan, 2001). Buchanan addresses questions related to its validity in research in a paper that is based on a presentation at a conference about design research in the United Kingdom, and it brings to attention the struggle and conflict over the past centuries about the old school of learning that where design is a driving force versus the new school (natural sciences) of learning that is more theoretical.

There are two works that are considered the earliest examples of design research, and those are Galileo Galilei’s *Dialogues Concerning Two New Sciences* published in 1665, which is about Galilei’s 30 years of personal research on body movements that has resulted in his theory of motions, and Francis Bacon’s *Principia* (1686). Design was not one of the fields institutionalized in universities following the works of Galilei, Bacon, and others, except in general work of architecture and the fine arts. In this century, natural science drove out design from the professional school curricula in many professions (Vaishnavi & Kuechler, 2004).

Many researchers, Buchanan included, are optimistic, and believe that design research has its place. Daniel C. Edelson argues that it offers opportunities to learn unique lessons, yields practical lessons that can be directly applied, and that it engages researchers in the direct improvement of educational practice (Edelson, 2002). Despite that many want to see more design research project being carried out, it has been a struggle to integrate it in some academic institutions (Zimmerman et al., 2007).

2.2.3 A shift away from technological issues in information systems research

Early information systems research has focused on technological issues, but later there was a shift away from technological to managerial and organizational issues. A research commentary, drawing on a review of all articles published in the *Information Systems Research* journal ten years prior to its publish date, argues that the field has not deeply engaged its core subject matter—*the IT artifact*. Orlikowski & Iacono, 2001 argue that it tends to “disappear from view, be taken for granted, or is presumed to be unproblematic once it is built and installed.” According to Vaishnavi and Kuechler (2004), design disciplines have a history of building their knowledge base through making, which involves the construction of artifacts and the evaluation of artifact performance following construction.

2.2.4 Three models for design research

The concepts of design research is arguably easiest by grasp by studying models that explain its nature and the knowledge building that happens through creation in iterations.

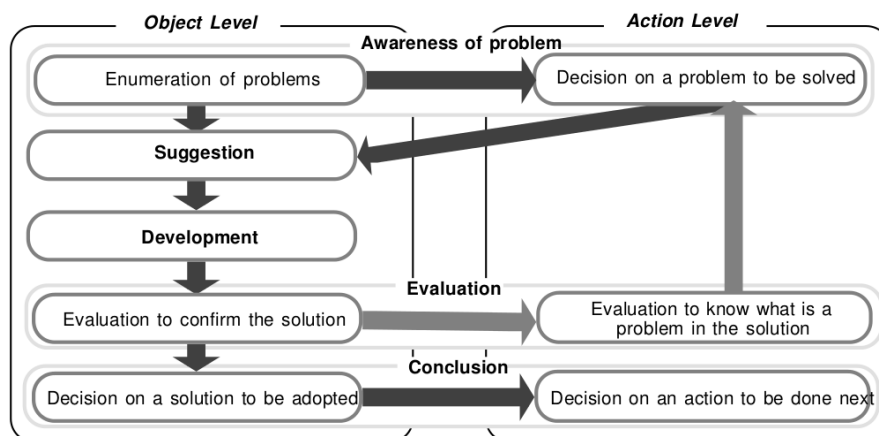


Figure 1: The Design Cycle model

Design Cycle model by Takeda et al. The *Design Cycle* model (Figure 1) by Takeda, Veerkamp, and Yoshikawa (1990) presents the design research process along with reasoning. It starts with the awareness of a set of problems. A problem is chosen to become solved along with a sugges-

tion for a solution. The suggested solution to the problem—or artifact—is designed, and finally evaluated. The process repeated iteratively, and it continues until a solution is found—or that one runs out of time. The work results in suggestions for future actions.

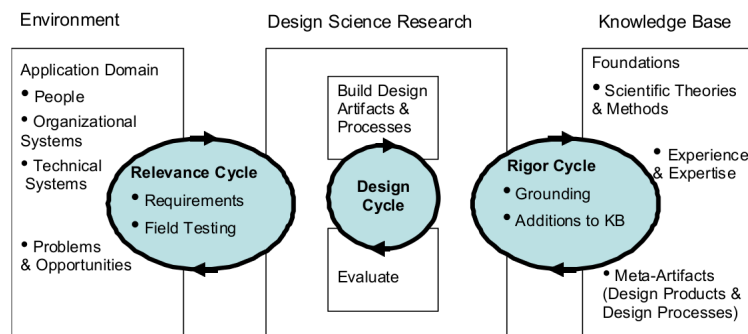


Figure 2: The Design Science Research Cycles

Design Science Research Cycles by Hevner The *Design Science Research Cycles* model (Figure 2) shows three continuous cycles in a design research project (Hevner, 2007). One cycle is about grounding the design to the existing base of knowledge. As the design evolves, it must be reevaluated that it is still addressing problems discovered by existing research. Part of the cycle is also providing new knowledge to the research base. Another cycle ensures that the artifact addresses requirements in the real-world environment. The artifact must be tested in a real environment. The final cycle is the design cycle, and it illustrates the iterative development of the artifact.

The Learning Design Framework There have been several studies about design research and its effects for learning, which is highly relevant since this project is carried out by a student and the project is not a full fledged research project intended for submission to either journals or conferences.

Bannan–Ritland proposes a *Integrative Learning Design Framework* (Figure 3) for design research in education, emphasizing stage sensitivity of research questions, data and methods, “and the need for researchers to design artifacts, processes, and analyses at earlier stages in their research that can then be profitably used (perhaps by different researchers) in later stages” (Bannan-Ritland, 2003). One of its objectives is to construct propositions about learning and teaching, and another is to engi-

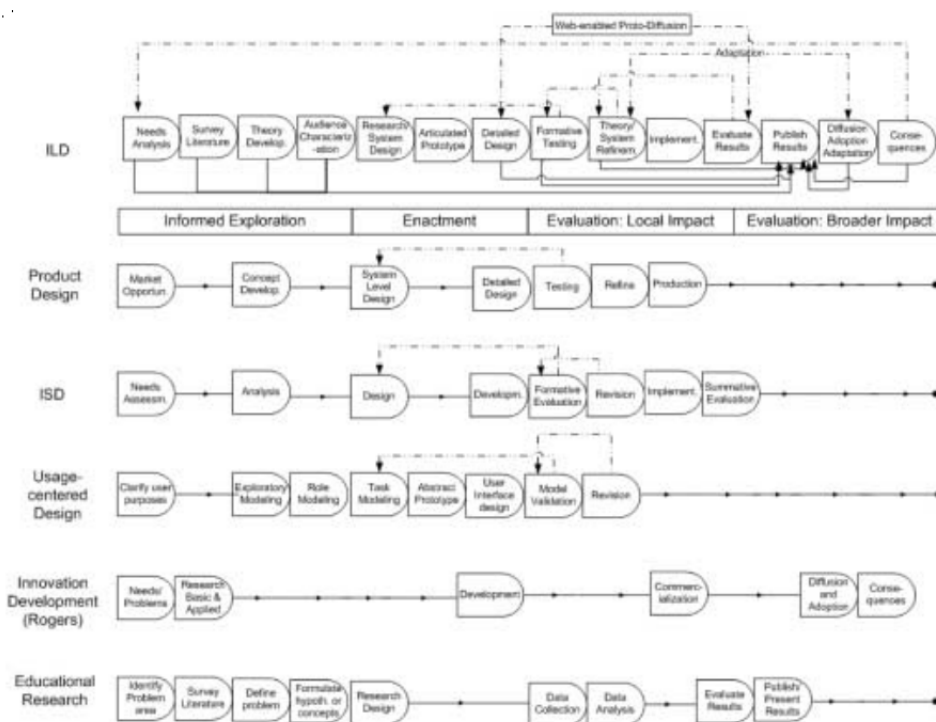


Figure 3: The Learning Design Framework

neer and construct effective learning environments using software and other artifacts to make the propositions actionable. It strives to “combine the creativity of design communities with appropriate adherence to standards of quantitative and qualitative methods in education.”

The framework has been used in LiteracyAccess Online (Bannan-Ritland & Baek, 2008), a project for fostering collaborative reading processes with children, and particularly those with disabilities.

2.2.5 Design research and HCI

Design research has gained a strong foothold in practice, but it has had less impact on the HCI community. Zimmerman et al. (2007) argue that there is no agreed upon standard of what research through design means, nor what a high quality contribution should be. A set of criteria is suggested for evaluating interaction design research: process, invention, relevance, and extensibility.

Also of relevance is *participatory* design that include “theories, practices, analyses, and actions, with the goal of working directly with users

(and other stakeholders)” in the design of software. Assumptions about technologies are questioned, such as if it is inevitable that technology is applied in ways that “constrain, deskill, and devalue human work,” or if software professionals “recognize and affirm the validity of perspectives other than their own, [...]” (Muller & Kuhn, 1993) Participatory design attempts to bridge the tacit knowledge developed and used by practitioners with “researchers’ more abstract, analytical knowledge.” Participatory design can be loosely defined as a methodology (Spinuzzi, 2005).

2.3 Human–computer interaction

Human–computer interaction (HCI) is a broad field concerned with the interaction between humans and computers. One definition of the field is “the ways that humans interacts with technologies for various purposes” (Zhang & Li, 2005). Other terms that refer to the the same field and/or include HCI and its fields are Nielsen and Hackos (1993):

- Computer–human interaction
- User–centered design
- Man–machine interface
- Human–machine interface
- Operator–machine interface
- User interface design
- Human factors
- Ergonomics

Human factors and ergonomics have a broader scope than human–computer interaction (Nielsen & Hackos, 1993). Ergonomics is defined as a “scientific discipline concerned with the understanding of interactions among human and other elements of a system, and the profession that applies theory, principles, and data (S. ISO, 2004). Both HCI and ergonomics broad, general fields, and both include three components; the user, interaction, and a system. Since HCI defines the system component as a computer system, we find it more relevant in the context of UTT.

2.3.1 Dissecting the HCI term

One can dissect human–computer interaction, and view *humans* as the end users of a piece of software. There can be more than one user. In the context of evaluation software, one might rather than viewing the human component as a single user, instead consider it to be a community of users. (Computer Science 10 - Lecture 13, 2012) A *computer* might be a mobile client, a server in the cloud, or a desktop computer. The *interaction* is users telling the computer what they want done, and the computer gives result back. The idea of interaction describes the HCI core, but is often neglected in existing research (Agrawal, Boese, & Sarker, 2010). The interaction happens through a user interface.

2.3.2 User interface

A *user interface* is the part of an application that allow a dialog to happen (Computer Science 10 - Lecture 13, 2012). It might consist of elements such as dialog boxes, buttons, check boxes that are interacted with through the use of conventional input devices such as a keyboard, a mouse, and touch panels. The results of the interaction are shown on a computer display and/or through physical feedback such as vibration. An example of a user interface recently invented is the Reactable³—a software and hardware device shaped like a round table that let musicians manipulate sound by placing and moving physical objects on top of it. The objects affect each other, and some of the parameters related to each object are changed by using touch gestures on the display.

A web page can be considered a user interface. It provides information to the user, and the user can manipulate what information is shown by interacting with elements on the web page and navigating the website. The navigation is done by clicking links or searching through the content of the website.

2.3.3 Subfields and classification of fields

Classifying HCI and its related fields is a challenge, and HCI in IS has been described as a “fragmented adhocracy,” where IS researchers might be unsure if they should even consider themselves as HCI researchers (Agrawal et al., 2010). What is certain is that HCI has a multidisciplinary

³<http://www.reactable.com/>

nature, and it covers many fields and professions. A few of the many sub-fields of HCI that are especially relevant in the context of evaluation of websites are accessibility, usability, user experience, and visual design.

A study that is concerned with common agreed definitions of the terms ergonomics, usability, accessibility, and safety sets out to stimulate a consistent use of the terms (Wegge & Zimmermann, 2007). According to the study, all fields have a foundation in or relation to the area of ergonomics. In the context of the web, HCI is more specific, since the term says that the component that a user interacts with is in fact a computer. For this reason, and because most, if not all of the collected literature concerning web technologies use the HCI terminology, the author has chosen to stick with that. The working model for this report is that usability, accessibility, and user experience are three distinct concerns.

2.4 Accessibility

Accessibility is about making content and services available to individuals, regardless of any disabilities or environmental constraints they experience (Mankoff, Fait, & Tran, 2005). Other terms used to refer to this are universal usability, universal design, and design for all. While accessibility is a subfield of HCI, universal design is a broader term that implies that the society as a whole should be accessible for everyone. Terms such as Accessibility and universal design are often mixed up and replaced with synonyms (Wegge & Zimmermann, 2007).

Web accessibility encompass all disabilities including visual, auditory, physical, speech, cognitive, and neurological disabilities (*Introduction to Web Accessibility*, n.d.). There might be varying degrees of impairment, and there seems to be a well-established myth that accessibility is only about blind people (gotreehouse, 2012a). Universal design, while it includes accessible technologies, also covers the physical environment, i.e. persons using wheelchairs should be able to access buildings by way of properly designed ramps, lifts, etc.

There has been a shift in terminology in recent years, where the perspective has changed from focusing on problems to possibilities. *Human diversity* is an example of a phrase intended to be inclusive and to not stigmatize. “Only if and when human diversity becomes a natural starting point for architectural design and societal planning, the need for special terms will vanish” (Iwarsson & Ståhl, 2003). A paper about the ter-

minology suggests to rather use the term *functioning*, as it denotes positive aspects of interaction between an individual and that individual's contextual factors (Iwarsson & Ståhl, 2003). Possible problems with a change in terminology, is that they need broad acceptance in order to be useful, and additional new terms might in general cause confusion.

There are guidelines that state how a website should be designed to ensure accessibility. One example of this is WCAG from the WAI/W3C. Web accessibility is covered in depth in Section 2.7.8. There are several measures to make websites accessible for all. International legislation like the “Proposal for a Directive of the European Parliament and of the Council on the accessibility of public sector bodies’ website” (*Digital Agenda for Europe - European Commission*, n.d.), and the ratification of the UN Convention on the Rights of Persons with Disabilities (*Convention on the Rights of Persons with Disabilities*, n.d.) require that governments and society (including the private sector) take appropriate measures to ensure that persons with disabilities have access to information and communications technology. In Norway, a regulation to the Discrimination and accessibility law is expected to come into force in 2013 to enforce universal design of all public online content and a large share of the privately owned web sites (*LOV 2008-06-20 nr 42: Lov om forbud mot diskriminering på grunn av nedsatt funksjonsevne (diskriminerings- og tilgjengelighet-sloven)*, n.d.).

Finally, there is a distinction between what is considered accessible and what is usable. Accessibility is a prerequisite for all to be able to access the technology, but this does not ensure usability (Leventhal & Barnes, 2008). One business case argument for making a website more accessible, is that “28% of the WCAG Success Criterion are mapped to benefits for Senior users. Baby Boomers account for 47% of US families and have over \$2,000,000,000 in buying power.” So when a website performs better for older people, it is tied to ROI for the website owner (*Yes, actually, it may be you one day*, 2012). Another argument supporting making websites accessible is that it simply is the right thing to do, it leads to good practice, and it helps avoid legal concerns (gotreehouse, 2012b).

It should be mentioned that technology set out to help people with disabilities in many cases have also helped people without any disabilities. When supporting different font sizes and color schemes on a website, one might as well let the user customize the look to his or her liking. A another example is audiobooks which took off as a US government sup-

ported “Talking Books Program” in the 1930s aimed at persons with visual impairments, but which with the development of smaller and more mobile media has reached a much broader audience (Rubery, 2011).

Universal Design There are seven principles for universal design (*The Principles of Universal Design at Center for Universal Design*, n.d.), and they overlap somewhat with the usability definitions that follow.

Principle	Description
1. Equitable use	Usable and marketable for people with diverse abilities
2. Flexibility in use	Accommodates a wide range of individual preference and abilities
3. Simple and intuitive use	Easy to understand, regardless of experience, knowledge, language skills or current concentration levels
4. Perceptible information	Communicates necessary information effectively, regardless of ambient conditions or sensory abilities
5. Tolerance for errors	Minimizes hazards and adverse consequences of accidental or unintended actions
6. Low physical effort	Can be used efficiently and comfortably, with a minimum of fatigue
7. Size and space for approach and use	Appropriate size and space for approach, reach, manipulation, and use regardless of body size, posture, or mobility

Table 1: *The 7 principles of Universal Design*

2.5 Usability

During the past decades, there have been many definitions of the term *usability*. This undoubtedly has something to do with the shift in computer use during the past decades, although some of the definitions are general. In addition to several definitions, there have also been attempts at creating new definitions that consolidate previous definitions. For this report, I’ll stick to what seems to be two of the most cited definitions.

2.5.1 The Nielsen–Hackos definition (1993)

Usability as stated by Nielsen and Hackos (1993) is not a single, one-dimensional property of a user interface. It has multiple components, and is traditionally associated with these usability attributes:

Attribute	Description
Learnability	The system should be easy to learn so that the user can rapidly start getting some work done with the system.
Efficiency	The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible.
Memorability	The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.
Errors	The system should have a low error rate, so that users make few errors during the use of the system, and so if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.
Satisfaction	The system should be pleasant to use, so that users are subjectively satisfied when using it; they like it.

Table 2: The attributes of usability (Nielsen–Hackos, 1993)

2.5.2 ISO definitions related to usability

There are several ISO definitions related to usability, and two of them are ISO 9241-11 (W. ISO, 1998), and ISO/IEC 9126-1 (I. O. f. S. E. Commission, 2001). The former defines usability as “The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.” This standard is said to be process-oriented. The latter standard defines usability as “The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.” It is said to be product-oriented.

It has been proposed to turn these models into a consolidated one (Abran, Khelifi, Suryan, & Seffah, 2003) that takes both process and product into consideration. The study also takes the definition/model of Nielsen and Hackos (1993) and others into account. Because I have observed the model by Nielsen et al. being cited in a number of sources, I choose to use their

definition/model throughout the thesis.

2.5.3 User friendliness an inappropriate term

Some usability experts find the term *user friendliness* inappropriate (Nielsen & Hackos, 1993; Leventhal & Barnes, 2008). One of the reasons is that machines do not need to be friendly to users—they should solve problems without getting in the way of the user. A second reason being that the term implies that user’s needs can be described along a single dimension by systems that are more or less friendly. A system that is perceived as being friendly to one might not be perceived as being friendly to another user.

2.6 User experience

*User experience*⁴ is a term from the mid-1990s that has gained traction recently. As computer hardware continue to become more powerful, and computers are becoming an increasingly integrated part of peoples lives, user interfaces are not only expected to solve the problems in an efficient way, but to also involve the emotions of the user by having software that is attractive. Subtle use of animations, refined typography, and careful use of white space in designs, and social features are some attributes of the recent trends.

2.7 Selected topics of web technologies

Since a large portion of the thesis is devoted to implementing a tool for evaluating websites, it is crucial to have a solid understanding of the technologies that the web builds upon, its history, and also what the landscape looks like today.

2.7.1 Development of a website requires many skills

One indicator of its growth is that to maintain a modern web page requires more specialized expertise than previously. As roles tied to web development are growing, there are blog posts written about classifying roles such as frontend developer or backend developer (*Redefining Web*

⁴User experience is often abbreviated to UX, but we will write it out in this report.

Designers, Web Developers, and Web Hybrids for the modern market - Tristan Denyer, n.d.). In the early days of the web, it was not unusual that one person provided the code for a web page, designed it, and its content. Today, it is not uncommon for a larger website to have an information architect, one or more visual designers, one or more developers, and for larger websites there can be one person specializing in website performance. There are many services and technology that abstracts away/outsources tasks such as visual design. WordPress is one such example. Social networks abstracts away everything except the content.

2.7.2 HTML and CSS

It is important to understand the idea of markup versus presentation of markup. A web page has traditionally, and to some extent today, consisted of an *HTML file* (HyperText Markup Language) containing a tree structure of elements that define the content. One example of an element is a first-level heading, and its content might be “Sign up for our services.” W3C is the organization maintaining the HTML standards. (*Standards - W3C, n.d.*). The latest standard, which views the web not only as a page, but as a platform, is *HTML5*. HTML5 covers a broad range of technologies, such as local data storage, video, web camera, and location (*HTML5 Rocks - A resource for open web HTML5 developers, n.d.*).

From the very beginning when the web was invented by Tim Berners Lee and his team, it was intended to reframe the way information was used, and how people work together (*Tim Berners-Lee on the next Web / Video on TED.com, n.d.*). It was meant to be *semantic*, meaning that the content should be described, or tagged to make them easy to process by a computer (*The Semantic Web: Scientific American, n.d.*).

As the web evolved, and the the importance of visual design increased, the HTML files were filled with markup that not only said what content was, but also how it should be presented. There were several problems with this. The most fundamental being that content and presentation were mixed together, creating a cluttered markup. *Cascading Style Sheets*, or CSS, was created to solve this problem by separating content from presentation.

The problem of separating content from presentation is of course directly tied to web accessibility, because having them separated means that the content can be presented in alternative ways. One example of how se-

mantic HTML is useful for screen readers is the difference between tagging content as italic versus tagging content as having emphasis, or that the content is a citation. For a sighted user, it makes little or no difference if a word is tagged with `<i>` or ``. To a visually impaired user, the difference is significant, as the screen reader, at least in theory, can read that particular word in an emphasized way (*Bold and Italic Formatting / Accessibility*, n.d.). An other common example is the use of font size or bold typeface to indicate titles in a document. Such purely visual markup is much harder to interpret automatically than the appropriate markup using the HTML heading levels (*Heading Tags (H1, H2, H3, P) in HTML / Accessibility*, n.d.).

2.7.3 Frontend and backend

As websites became more complex, and grew into large stores (e.g. Amazon) and libraries (e.g. Wikipedia), there was a need to generate HTML pages instead of creating one and one “by hand.” A modern website is built on a *web framework*. A website is usually divided into a *frontend* and a *backend*. The backend receives e.g. an id of an article, and it delivers HTML files that are often generated by mixing templates with content from a database.

The web has emerged to become more dynamic, as there was with Web 2.0 a shift from a vendor providing content towards users building the content of a site. On a technical level, one very central component is the use of *JavaScript/ECMAScript* in the frontend, i.e. the part of the web page that is rendered/calculated on the client-side. JavaScript has been available in browsers since 1995 as the only language, and it is the only native scripting language available in web browsers. JavaScript has received a lot of attention by browser engine developers, and a lot of work has been put in making it perform fast.

In the early 2000s, the term *Dynamic HTML* was coined. It involved the use of HTML, CSS, and JavaScript and it involved changing a website in subtle ways, such as showing/hiding a menu on a website (*Web Style Sheets*, n.d.). In the mid 2000s, an interesting use of *XMLHttpRequest* emerged, a technology first introduced in Internet Explorer in 1999 (*Native XMLHttpRequest object - IEBlog - Site Home - MSDN Blogs*, n.d.). It was used to change the content dynamically on a page without reloading the page. Downloading the entire page from the server and re-rendering it

takes longer, and also causes the screen to flicker. Today entire frameworks are written in JavaScript that dynamically builds the website.

A common requirement for websites today is that they are expected to adapt to desktop, mobile, and tablet devices. A website that renders correctly on these devices is said to be *responsive* (*Responsive Web Design - An A List Apart Article*, n.d.).

2.7.4 Web application frameworks

The role of a *web framework* is to give an application structure, and to separate its responsibilities. An application needs some sort of structure, e.g. to separate how it looks (views) from how it is navigated (routes), and a way to handle the application data (model).

Some web frameworks dictate a directory structure. This has a number of advantages. First, it makes it easy to find the file that one's after, but perhaps more importantly, it means that as one jump from one project to the next, then it is consistent.

Some web frameworks provide a tool for initiating a new project, e.g. with a directory tree, an initial route, and a "Hello, World" view.

2.7.5 Single-page application architecture

Single-page application (SPA) is "a web application or web site that fits on a single web page with the goal of providing a more fluid user experience akin to a desktop application" (*Single-page application - Wikipedia, the free encyclopedia*, n.d.).

SPA has gained traction recently, and one notable example is SoundCloud, a website for artists sharing their music that rewrote its entire website to SPA in 2012 (*Building The Next SoundCloud - SoundCloud Backstage SoundCloud Backstage*, n.d.). The website now allows for playing music while navigating within the between pages, something that previously would have required opening an external popup window. Another example is the new web based Spotify client that is now in beta.

For a SPA, there is an initial HTML file requested by the user agent containing the application code. Any subsequent content than what is initially presented is either contained in the file, or it is transferred through a persistent WebSocket connection. A SPA is usually based on a library/framework.

Backbone.js⁵ and Ember.js⁶ are examples of SPA frameworks.

SPA can enable navigation similar websites built using the traditional architecture where each page is separately requested. One benefit of SPA however, is that it allows for more fine-grained definitions of what exactly a page is. But by doing so, it conceptually breaks the model of what a page is, and while logical pages can and should be defined, care must be taken when designing them.

The part of an URL that follow a HTML file⁷—the fragment identifier (*Fragment identifier - Wikipedia, the free encyclopedia*, n.d.)—is not sent to the server, regardless of architecture. The user agent recognizes a combination of a hash sign and identifier as a jump to a section defined by the identifier. However, since it is possible using a client-side script to pick up the fragment following the HTML, it can also be used for routing purposes.

One of the challenges with SPA is that a long-running web page might lead to memory leaks as elements are manipulated—i.e. elements are created and destroyed. Various mechanisms exists for addressing this problem (*How To: Detect Backbone Memory Leaks | Andrew Henderson*, n.d.).

2.7.6 State

State is a complex issue in software development. For a web shop, state might be the notion of a user that is logged in, and that the user should stay logged in when navigating between pages. Traditionally, clicking around on a web page means sending a HTTP request to the server. Since HTTP is a stateless protocol, state needs to be implemented in a layer above, and a session is usually implemented both by using cookies in the client, and by sharing the cookie data with the server.

2.7.7 Semantic versioning

The concept of versioning of software needs no introduction. The term *semantic versioning* refers to a formalization of a very common way of versioning scheme for software (*Semantic Versioning 2.0.0-rc.2*, n.d.), and

⁵<http://backbonejs.org/>

⁶<http://emberjs.com/>

⁷Usually one does not see the name of the HTML file, as web servers removes index.html by default from the URL.

it is especially useful for software components that are used in software products.

Semantic versioning follows an x.y.z scheme, where ‘x’ is a major version, ‘y’ is a minor version, and ‘z’ is a bug fix. While most software components follow this scheme, or a very similar one, the versioning of software products, such as operating systems, image manipulation applications, and web browsers are often tied to marketing. In a competitive market, a higher number might give the impression that it is better than a competing product with a lower number. Sometimes an internal version is used in addition to the “marketing version” to make it easier for developers to manage the various versions.

Recently, the version numbers for web browsers such as Chrome has reached a high number such 27, and it updates itself in the background. It is our observation that people pay less attention to version numbers, as there seems to be a trend towards incremental improvements—at least for software in this category.

2.7.8 Web technologies and accessibility

As stated in the beginning of the previous section, the web was initially designed to be semantic by wrapping content in the appropriate elements. Although this helps machine parsing, this alone is not a guarantee for accessibility. An image for instance, needs further to have its alt attribute filled out so that a screen reader can present the image to the user. The presence of an alt attribute is a requirement for a web page to become validated against the HTML standard.

Web Accessibility Initiative (WAI) is an effort by W3C launched in 1997 with the intent of improve the accessibility of the web (*WAI History*, n.d.). *Web Content Accessibility Guidelines* is a set of guidelines for making web content accessible.

As outlined in the previous section, the web is getting more complex, and there is need for additional markup beyond what is in the HTML standard to help users with disabilities. Recently, mechanisms has emerged for giving additional cues to assistive technologies. *WAI-ARIA* is one of them, and involves adding so-called landmarks to existing elements. An example is setting attribute role of a div-element to value navigation so that a screen reader quickly can move to this region of the page (*The Accessibility of WAI-ARIA*, n.d.).

Many of today's accessibility checking tools work by measuring the source code against implemented success criteria belonging to a guideline such as WCAG. There is one problem with this. Taking the source code as input is starting to get problematic as the web is transitioning into client side architecture that involves content being loaded into the document at run time. For UTT, the source code only contains the skeleton of the application. The fact that many accessibility tools rely on a source based DOM has been blogged about recently (*MOTHER EFFING TOOL CONFUSER*, n.d.).

An existing crowdsource approach to accessibility testing is FixTheWeb (*Addressing accessibility / Fix the Web*, n.d.), a website where one group of people report accessibility issues encountered on websites. Another group notifies owners of the websites about the reported issues.

2.8 Crowdsourcing

Crowdsourcing is defined as “the practice of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, and especially from an online community, rather than from traditional employees or suppliers” (*Crowdsourcing - Definition and More from the Free Merriam-Webster Dictionary*, n.d.). Here is a more throughout definition by Jeff Howe:

“Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals. The crucial prerequisite is the use of the open call format and the large network of potential laborers.” —Jeff Howe (*Crowdsourcing: Crowdsourcing: A Definition*, n.d.)

Wikipedia is perhaps *the* success story with its use of crowdsourcing, and Kittur, Chi, Pendleton, Suh, and Mytkowicz (2007) address the question whether it “is driven by a core group of ‘elite’ users who do the lion’s share of work,” or if there is a large number of “common” contributors. The study acknowledges that there has been a shift from elite users to

large crowd in recent years, but that the story is more complex than previous explanations. The results mirror the dynamics found in more traditional collectives.

Another example is Amazon’s Mechanical Turk⁸—a micro-task market where a large amount of workers solve tasks “for marginal costs within the timeframe of days or even minutes.” For this approach to be effective, special care is needed in formulating the tasks, and especially for measurements that are subjective or qualitative (Buhrmester, Kwang, & Gosling, 2011).

A further interesting approach is to solve large-scale computational problems through online games. The *ESP Game* was designed to assign labels to images and involves two players that are paired together, and the goal is to guess a label that the partner would give to an image. A number of suggestions can be typed, and the process is called “agreeing on an image.” A similar approach can be used for providing appropriate textual descriptions for images on the web (Von Ahn, 2006).

A recent, novel use of crowdsourcing is the Swedish DJ Avicii that for Ericsson produced X You, the world’s first crowdsourced hit (*Ericsson teams up with DJ and producer Avicii to try and crowdsource the world’s first hit song*, n.d.).

2.9 Open source software

The history of open source is well-known and will not be covered in depth here. It can be summed up by saying that it has been a history with strong personalities, hacker culture, politics, and open source is today implementing most of the infrastructure on the web and for many enterprises. There exists large ecosystems for common infrastructure components that runs banking systems, large e-commerce solutions, and mobile phones to mention some. Linux is perhaps the most known, and arguably the most successful open source technology.

There are several advantages with the open source model. Like scientific research, one builds on the shoulders of giants, and the model is transparent, as it allows one to inspect the code of the running software. One characteristic of open source technologies, is that a project can be forked when some people feel that a project should go into another direction. Similarly, two projects might merge if they should happen to share

⁸<https://www.mturk.com/>

the same vision.

There are many licenses for an open source project to choose from. [opensource.org](https://opensource.org/licenses/) maintains a list of all licenses approved by the Open Source Initiative. Care must be taken when combining components of various licenses into one product—this especially holds true for proprietary software. What often separates the various licenses is how restrictive they are when it comes to contributing back, and how they can be used in a proprietary product.

For this master's project, open source is important to assure the accountability of the resulting software, and to encourage further development of the results.

3 Existing tools

Existing tools and services are studied with several goals in mind; the first is to learn what is the state-of-the-art. Also, we need to find out if UTT can be built by extending an existing open source tool. A third goal is to select an automated accessibility checker to integrate with. Finally, a vocabulary is needed for further discussion of UTT. Terms will be defined based on analysis of existing tools.

What follows is a subsection describing the selection process. The selected tools are subsequently presented in tables, and a few tools are analyzed in detail. There is an additional section on recruiting services. This section closes with a list of findings, and the objectives and requirements for the new tool.

3.1 Selection process

The process of finding and selecting existing tools are described in this section. Tools covering any of the aspects accessibility, usability, user experience, and visual design are included in the review. Even though UTT currently only is intended for accessibility evaluation, the tool has the potential to cover additional aspects in the future, and nevertheless we believe there is a lot to learn about handling of user interaction from tools in the latter three categories.

The first goal is to include such variety of tools that the state-of-the-art is outlined. To accomplish this, we look at tools that use a traditional survey approach, as well as more novel approaches where e.g. a user gives feedback by clicking directly at an element on a web page. We also see how crowdsourcing is used by current tools.

Both proprietary and open source solutions are studied. One benefit of open source is the great potential to reuse and extend existing tools. For proprietary projects, one can borrow ideas for testing methods. Although the majority of tools found are non-academic, the findings includes research projects as well.

The process of searching for existing tools is similar to the literature review. Web search engines and journal search engines are searched using keywords, filtering is done based on a set of criteria, and then a number of tools are studied.

We have found lists of tools that have been of help, such as Ethnio's list

of usability and user experience tools⁹. There is a Wikipedia page with a list of GUI testing tools¹⁰ for “automating the testing process of software with graphical user interfaces,” but these belong to a different category of tools, as will be discussed in the upcoming paragraph.

A source of confusion *Automated UI testing* and *UI automation* are two related categories that have little to do with what we try to accomplish, but they need to be mentioned since they use several overlapping terms, and the implication of the term automation differ.

Automated UI testing—or regression automation—is about automating the testing process of UIs by simulating input device events in a UI, and in an automated way determining if a functionality works as intended. Selenium is a well-known project in this category of software. Similarly, UI scripting, or automation of UI tools to automate repetitive tasks. AutoHotKey and Automator are popular tools in the latter category.

As a sidenote, tools in these categories might be integrated into the development process of UTT in the future.

3.2 The selection of tools

Table 3 and Table 4 show the proprietary tools found, and Table 5 shows the open source tools. Table 6 shows the automated checker tools for accessibility testing that UTT potentially can be integrated with. No working demos were found of two tools from the literature, and consequently they are simply described.

⁹<http://remoteresea.ch/tools/>

¹⁰http://en.wikipedia.org/wiki/List_of_GUI_testing_tools

	Loop11	Usabilla	Ethnio	Draft
<i>Website</i>	loop11.com	usabilla.com	ethnio.io	draftapp.com
<i>Test methods</i>	Sequence of tests, feedback-button that can be integrated	Feedback-button integrated on web page is clicked, user selects element on page, reports emotion, tags, shares comment, and rates speed of website	Survey-like screener shown to test visitors/recruits test participants for other services	Clicks a spot on a design and leaves a comment
<i>Test area/coverage</i>	Usability, user experience	Usability, user experience	User experience	Visual design
<i>Needs client-side installation</i>	No	No	No	No
<i>Needs modification of website</i>	No	Yes	Several options – One option is to use JavaScript if one wants to recruit visitors	N/A
<i>Needs to define test run</i>	Yes	Yes	N/A	N/A

Table 3: Proprietary tools—part 1

	UserTesting	UserZoom	Verify
<i>Website</i>	usertesting.com	userzoom.com	verifyapp.com
<i>Test methods</i>	Video feedback and written answers to questions	Task-based, card sorts, click tests, and more	Tool asks a question regarding e.g. preference, user performs action on web page, tool asks more questions about user
<i>Test area/coverage</i>	Usability, user experience	Accessibility, usability, visual design	User experience, visual design
<i>Needs client-side installation</i>	No	Only when capturing behavioral data such as heat maps	Not sure, probably not
<i>Needs modification of website</i>	Not sure	Not sure	Not sure
<i>Needs to define test run</i>	Yes	Yes	Probably

Table 4: Proprietary tools—part 2

	Infomaki	ClickHeat
<i>Website</i>	sourceforge.net/projects/infomaki	labsmedia.com/clickheat
<i>Test methods</i>	Ask questions regarding a screenshot about where to find something, capture mouse movements and click, generate heat map – also support surveys	Logs mouse activity on a web page which is used to generate heat maps
<i>Test area/coverage</i>	Usability, visual design	Unsure
<i>Needs client-side installation</i>	No	No
<i>Needs modification of website</i>	Yes	Yes
<i>Needs to define test run</i>	Yes	N/A

Table 5: Open source tools

	Achecker	Wave	eAccessibility Checker
Website	achecker.ca	wave.webaim.org	accessibility.egovmon.no
Test methods	Tests website against various guidelines/custom set of test criteria	Tests a web page for accessibility barriers	Automatically checks website against WCAG 2.0
Test area/coverage	Accessibility	Accessibility	Accessibility
Needs client-side installation	No	No, but it is a possibility to use the tool using a Firefox toolbar extension	No
Needs modification of website	No	No	No

Table 6: Automated checker tools

3.3 Loop11

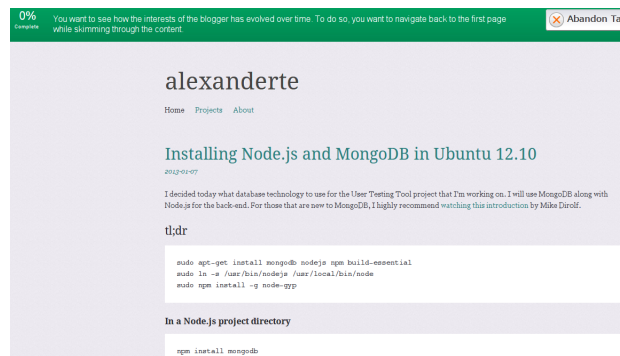


Figure 4: Loop11 presenting a task during a test run

Loop11 is a proprietary, web-based usability and user experience testing tool. It integrates with the tested website. The basic flow of the tool is that a person responsible for testing of a website sets up a *test run* from an administrative section. A test run is a set of *tests* that are presented in a sequence—i.e. one test per page. A test is either a *task* or a *set of questions*. Figure 4 shows a screenshot of a test page with a task. Once a test run is defined, it is shared with participants. Participants are either recruited using recruitment services, or they are providing by the user. Participants are identified by their email address.

Loop11 is free to use for a limited amount of time. The author chose to register using his personal email address, and to test his personal website using the tool. This was done to see how the testing process worked—both to get an idea of what the user interface was like, and to get an idea of its implementation.

3.3.1 Definition of a task

A Loop11 task has a name, a scenario, a start URL, and one to many success URLs. Loop11 provides an examples of a task name:

“Buy the music CD titled *The Essential Elvis Presley*.”

A task has also a *scenario*—an elaborated version of the task name, similar to a use case:

“In two weeks it is your father’s birthday and you need to buy him a present. He has been a big fan of Elvis Presley since he was a teenager and he has always talked about how the music CD titled *The Essential Elvis Presley* is the only one missing from his collection. Locate this CD and buy it for him.”

3.3.2 Definition of a question

A set of question might be of one of these types: multiple choice, rating scale, ranking question, or open ended. The various types have sub-types. It is possible to have an open ended answer that is restricted to one line. This suggests that one is after a short answer. Another possibility is to have a comments box. This invites the participator to write down his/her thoughts and feelings in detail.

What the variety ways of asking questions have in common is that they want the user’s opinion on something. Some ways of getting that is through multiple choices that are mutually exclusive, i.e. one can only choose one of them. Another where one can check zero to many. Other variants include rating scales (rate the usability of this website from 1–4), rating from (1–10) where you have a negative opinion on the left side, neutral in the middle, and a positive opinion on the right side.

The questions are not mandatory by default. One can make one mandatory by clicking a check box while creating the question. There are additional options for some types of questions. For most types of questions providing more than one answer, one can randomize the order. For the multiple choice questions, one can add “Other” as the final choice. For the rating scale with matrix and ranking question, one can add a “N/A” option.

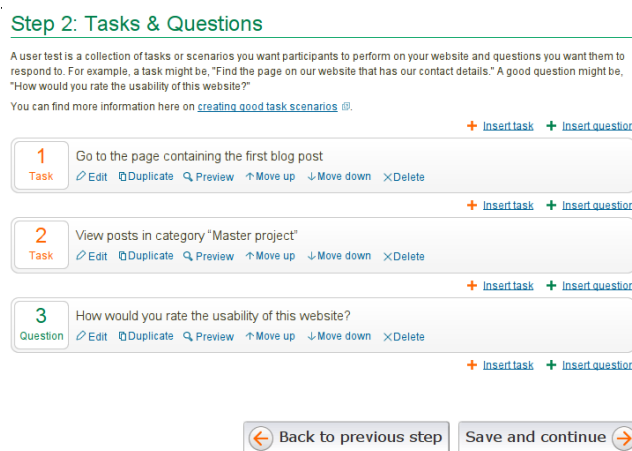


Figure 5: Loop11 user interface for managing a test run

3.3.3 Managing a test run

We found the interface for managing tasks and questions intuitive (Figure 5). It is possible to preview tasks and questions, duplicate them, preview them, delete them, and move them up and down by dragging and dropping with the mouse.

There are various features that a test administrator can use to customize a user test. One can limit the maximum number of participants, provide a custom thank-you text, or redirect the user to a page if they want to. It is also possible to append custom IDs to user test URL. This is useful to track participants that one should provide incentives to.

It is also possible to say if one should allow multiple responses per IP, and/or include/exclude IP ranges.

3.3.4 Test page

To enable further discussion of both the Loop11 and UTT user interface, we need to establish some general terms. Loop11 integrates questions with the web page that is being tested. In the tool—which itself is a website—there is a question on top, and the website is seamlessly presented underneath.

As shown in Figure 6, we define a *test page* as the web page showing tests, *test page header* refers to the upper part showing the question and navigational elements, and *test page body* to refer to the part showing the currently tested web page.

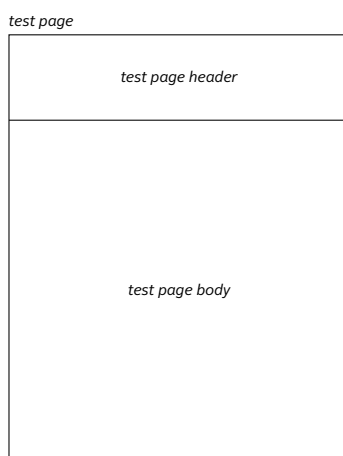


Figure 6: Defining test page terms

3.3.5 The proxy–iframe technique

Loop11 uses iframes to present a web page, and the web page is served through a proxy. The technique will from now on be referred to as the *proxy–iframe technique*, and it allows the tool to modify the tested website. Code is injected into the website, enabling the tool to track various data, such as navigation around a site and/or capturing events from input devices. Another use of this technique is highlighting of elements by injecting CSS.

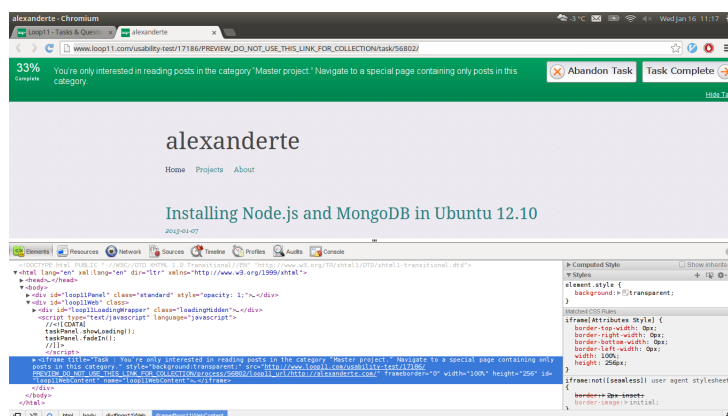


Figure 7: Studying the proxy–iframe technique used by Loop11

To study the implementation of this technique, we have used the Chromium web browser to inspect the test page (see Figure 7). Although one cannot always get a clear understanding of how the underlying system is imple-

mented this way, it does suggest a great deal about how the user interface is implemented. We observed that the value of the `iframe src`-attribute (URL of embedded web page) does not point directly the web page currently tested. Instead, the web page being tested is served through another server, one that is more than likely set up by Loop11 as a proxy.

One interesting note is that in the Loop11 tool, it is recommended to inject a JavaScript snippet into the site being tested for solving performance and rendering issues.

Using a proxy technique might be necessary for doing what we have mentioned, as an embedded website cannot be modified from the outer page.

iframes and accessibility The use of an `iframe` needs to be carefully considered for UTT. While frames are not inaccessible to modern screen readers, they can be disorienting (*WebAIM: Creating Accessible Frames*, n.d.). We consulted one of our partners, Birkir Gunnarsson, an Icelandic expert in web accessibility, and he suggested to design the test page body in such way that it presents no more than what is necessary for user of screen readers.

3.3.6 Inviting/recruiting participants

Loop11 has a variety of ways of inviting—and even recruiting—participants. One possibility is to generate a link that can be sent over email to a participant that one knows. Another way is to generate a popup invitation that needs to be inserted into the website. This is shown to visitors of the website. The last method is to recruit participants using a recruiting service. Loop11 integrates with three different ones. One of them, Ethnio, is covered in a later section.

3.4 Usabilla

Similar to Loop11 is Usabilla—a proprietary service used for improving the usability and user experience of websites. Usabilla is also used by large companies, and it appear high in search results when searching for tools. Unfortunately, we have not been able to try it out by registering. While it does have a 14 days free trial, one have to choose a plan with a monthly price. Since it is not my intention to pay for the product, We have learned as much as we can about the product by researching its website.

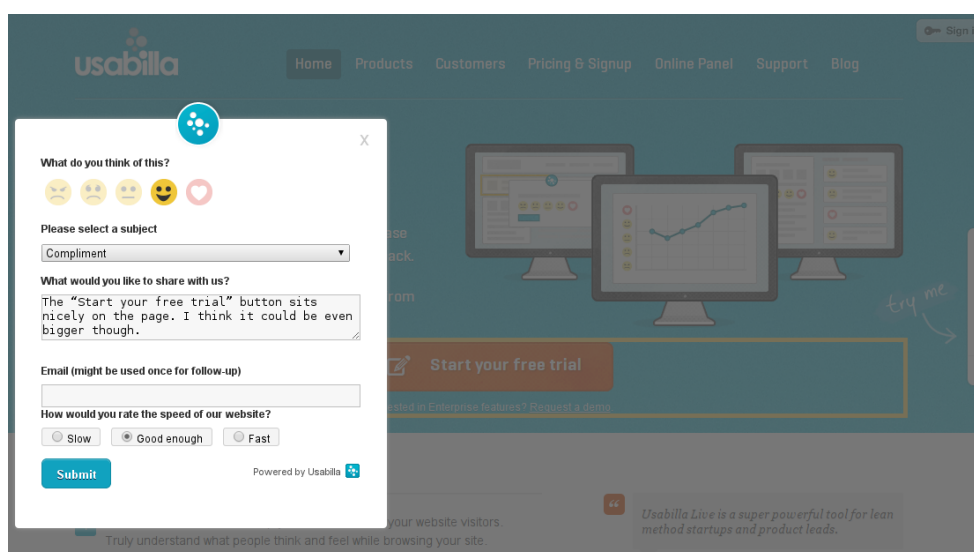


Figure 8: Commenting on an element using Usabilla

Usabilla works a bit different compared to Loop11. Instead of designing a test run that is used by participants, one inserts a feedback-button on the website where visitors selects a part of a website, selects an emotion (from hate to love), tags it with tags such as “attractive,” “interesting,” or “complex,” writes a comment, rates the website speed, and optionally enters ones email address (see Figure 8). The feedback box is shown on top of the website. While it is shown, the website is dimmed.

3.5 Draft

By following the startup community on HackerNews¹¹, one can discover new and interesting technologies made by small companies and/or individuals. Draft.¹² is a proprietary service for enabling easier collaboration between designers and clients.

The designer sends a design to a client to receive comments on. The image is annotated by the client, and they are able to communicate using multiple comment fields anchored to spots on the image. This somewhat is similar to the Usabilla idea, where feedback is given by selecting an element and commenting.

One detail regarding the implementation is that Draft uses HTML5 technologies to enable annotation, instead of using e.g. Flash or other

¹¹<http://news.ycombinator.com/>

¹²<https://www.draftapp.co/>

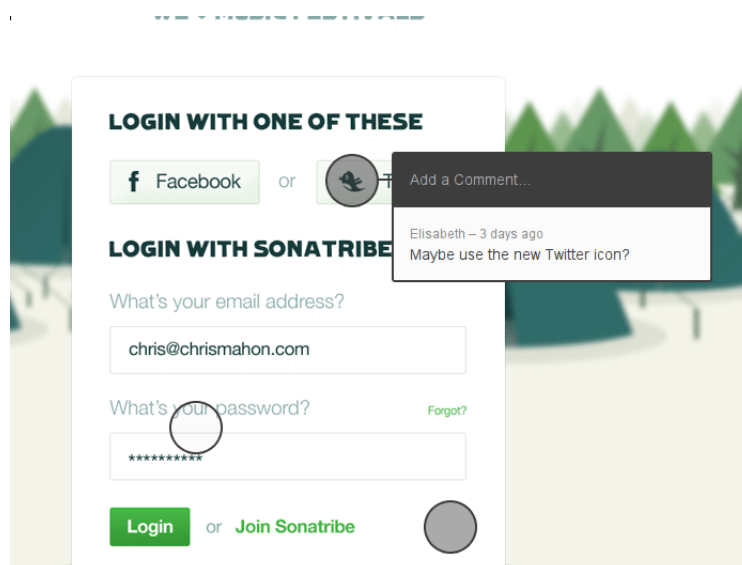


Figure 9: Commenting on an image using Draft

non-standard technologies.

Related tools to Draft are Frontify and Mocku.ps¹³. The former is for communicating between a frontend developer and a visual designer. The latter has approximately the same use case as Draft.

One possibility for UTT is to present comments from different users in the test page header. A possible problem with by implementing comments is that a commenter might be influenced by what has previously been written. This might have both positive and negative effects regarding the quality of the collected data. A solution might be to show other comments after the user has commented, but is might have a similar effect, only on a higher level—that is, the users learn from eachother how they are “supposed” to comment. The effects of having a comment field in the context of a user testing tool could be a topic for further investigation.

3.6 Infomaki

Two open source usability/user experience tools were found, and those are Infomaki and ClickHeat. These are candidates for extension, as one can either fork one of the tools and develop it further, or suggest for the developers permission to be a part of the project.

¹³<http://frontify.com/> and <http://mocku.ps/>.

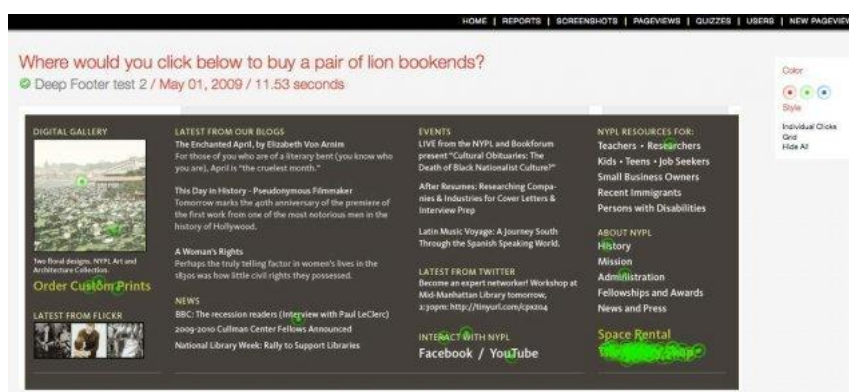


Figure 10: Infomaki presenting a heat map

Infomaki is a project initiated and developed by The New York Public Library's Digital Experience Group. It builds on the ideas of a popular proprietary service named Five Second Test, and is used to "evaluate new designs for the NYPL.org website and uncover insights about our patrons."¹⁴ It shows a popup to the user, asking the visitor to answer one question. The visitor is shown a question with a screenshot below, and is asked to locate a certain piece of information. Afterwards, the user can either answer another question or return to the website. Infomaki is developed on top of the Ruby on Rails web framework. The results from the user testing is a heat map showing where on the page the user is likely to click to find the information (Figure 10).

It is uncertain to the author whether InfoMaki is actively developed or not. On its SourceForge page, the latest commit is dated 2009-05-13, suggesting that it is not actively developed. On the summary page, there is a "last updated" date that says 2013-04-19 – less than a month relative to when this was written. The author is not sure what this date indicates.

On a superficial level, Infomaki might seem similar to Loop11. The user interface can be described using the test page model. However, the test page body shows a screenshot of a web page—and not a web page. Also, the test run capabilities of the tool seem limited. There is little to nothing that can be reused for UTT.

¹⁴<http://journal.code4lib.org/articles/2099>

3.7 ClickHeat

ClickHeat is the other open source tool that is a “visual heatmap of clicks on a HTML page”.¹⁵ It does not seem like a product as much as the previous tools, as it does not suggest any use cases for the heat map. Unfortunately, its demo site was down when planning to take screenshot of the tool, and the one found on the ClickHeat website was somewhat small (Figure 11).

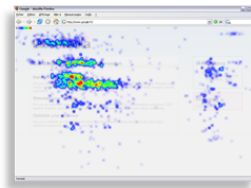


Figure 11: ClickHeat showing a heatmap

While the TT will not implement any tracking of mouse activities for the prototype, this might be interesting in the future.

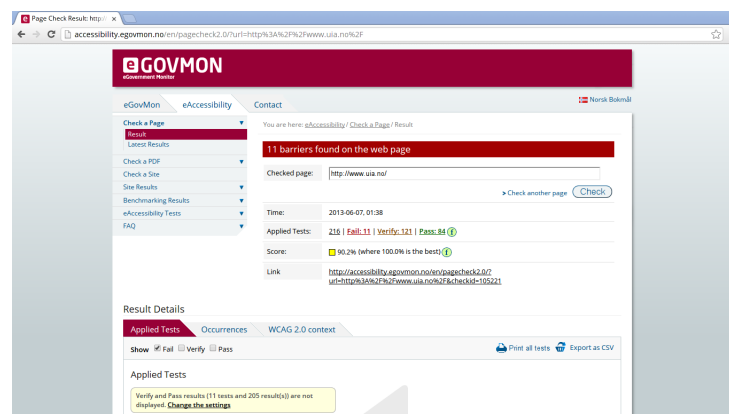


Figure 12: eAccessibility Checker checking uia.no

3.8 eAccessibility Checker

The eAccessibility Checker checks a web page, or a website against WCAG 2.0. In the user interface, the address of a web page to check is specified, and after clicking “Check,” the checker shows what tests have been applied, how many have passed, how many have failed, and how many that must be verified by a human. Checker results can be exported to CSV.

¹⁵<http://sourceforge.net/projects/clickheat/>

The eAccessibility Checker also has functionality to compare websites—or benchmark—that have been tested. This has been used by Norwegian municipalities for a while, but is now also used by the government in Qatar. The benchmark allows for monthly comparison of checked websites. The checker also has capabilities for checking PDF documents.

The author is already familiar with the project, as its development is coordinated by Tingtun.

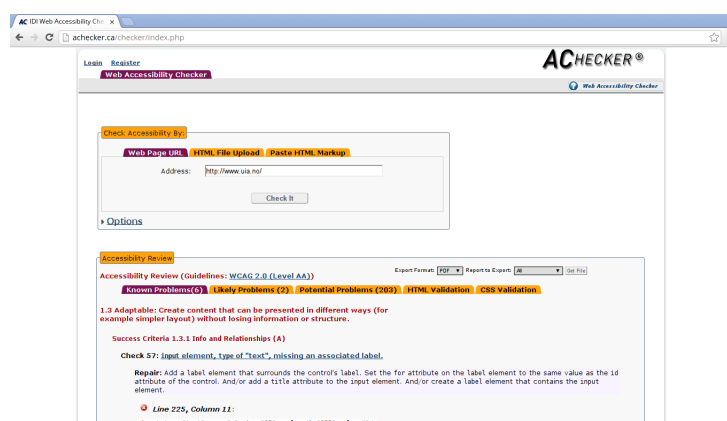


Figure 13: Achecker checking uia.no

3.9 Achecker

Achecker¹⁶ is very similar to eAccessibility Checker—it checks the accessibility of a website against accessibility guidelines using an automated approach. While the interface looks less polished to us than the eAccessibility Checker, it supports several more accessibility guidelines. Achecker provides a Web Service API¹⁷ returning the results in an XML format.

Achecker does not seem to have capabilities for carrying out large-scale testing of websites. On its website it says “This tool checks *single HTML pages* for conformance with accessibility standards to ensure the content can be accessed by everyone [emphasis added].”

3.10 Existing research projects

We have looked at existing research to see if we could find existing tools that have been developed for evaluating usability. We have chosen to write

¹⁶<http://achecker.ca/>

¹⁷http://atutor.ca/achecker/demo/documentation/web_service_api.php

about existing research projects under one topic, as we have only been able to find two that seem relevant—and they are developed a decade ago, which makes them not so relevant when looking at web evaluation tools.

3.10.1 AWUSA (2002)

AWUSA is a tool combining information architecture, automated usability, and web mining techniques (Tiedtke, Märtin, & Gerth, 2002). It is a prototype developed in Java that takes XML both as input and output. A SVG is used to represent the results. The tool builds on the assumption that future websites were developed in XHTML—in other words, that they were well-formed XML. While XHTML has gained some traction, many websites are written in HTML. HTML5 offers a XHTML5 variant.

3.10.2 WEBUSE (2003)

WEBUSE (WEBSITE USABILITY Evaluation Tool) is a research project that summarise website usability issues and groups the issues into a set of 24 usability guidelines (Chiew & Salim, 2003). It uses these guidelines to develop a survey-like tool that asks visitors to evaluate the website.

3.11 Recruiting services

Getting a large number of participants for testing a website can become a challenge. One way to do it is to use recruiting services such as Ethnio or Cint¹⁸ OpinionHUB. Cint claims to give access to “millions of verified panelists recruited from over 600 unique panels according to your specific criteria.” Another is Knowbility AccessWorks¹⁹ that enables recruiting participants with disabilities. The users perform tasks using their assistive devices, and give “a detailed analysis about accessibility pitfalls at the website.”

What these services have in common is that they find real people that can accomplish the user testing. Ethnio uses Twitter to find participants, and the participants are paid with Amazon Gift Certificates. One possible problem is that these people might not be representative for the average of population.

¹⁸<http://cint.com/>

¹⁹<http://www.knowbility.org/>

Ethnio, in addition to being a recruiting service, also has a survey-like screener that one can add to the website. It is added by adding JavaScript to web pages one wants to intercept on.

One weakness of these recruiting services is that the testing subjects might not be representative for the average of the population. For instance, Ethnio uses Twitter to recruit users. This means that all participants are likely to be Twitter users. One can with most of these service choose demographics, such as country, gender, age, and/or educational level. The Knowability AccessWorks service is unique in that the recruited participants perform tasks using assistive devices.

3.12 Findings

This section goes through what has been learned by looking at the existing tools, both by placing them in a matrix by comparing their capabilities, and by looking closely at a few of them.

3.12.1 Interactive usability/user experience tools

For all the usability and user experience tools that are looked at, they seem to have a variety of ways in which testing is accomplished, and how testers are recruited. Some showed a survey to visitors, others provided a feedback button where a user can report feedback regarding one specific element. Others were more complex, and had a test run with various tests. What all the usability/user experience tools have in common, is that they are proprietary, and all of them are provided as services with various pricing plans. All data are stored at the domain and business that hosts them, and the tools provide reports and export to formats such as Excel, SPSS, or CSV. All the tools were using a manual approach for gathering the opinion of the user — i.e. no automated checker to figure out if a website is usable or not. There were two tools found that were related to usability and user experience and that were open source, but the only functionality that they had was tracking mouse activity.

3.12.2 Accessibility tools

For accessibility tools, it was quite different. The tools were automated testing tools that checked the website against implementation of accessibility guidelines. Two out of three tools looked at were open source, and

the last one, while not open, is supported by W3C which generally has open tools.

3.12.3 There is no existing tool to build upon

Two tools that were open source were found; Infomaki and Labsmedia ClickHeat. While they seem to be good tools, their goals differ so much from UTT and its requirements, that it is not practical to build on top of them. Both of them tracks mouse activity, and it is not unlikely to include this in UTT at some point.

While UTT will not be based on an existing tools, it will be built using open source technologies. Ideas can be borrowed from all existing tools, and ideas regarding implementation can be borrowed later on from the open source tools mentioned.

3.12.4 Conclusion

According to what we know so far, no tool exists which can combine automated accessibility testing of websites with user testing. Because we intend to support checking of websites, the tool will integrate with the open source tools developed by the eGovMon project and the user testing experience among the partners and in the reference group. The tool will be released under an open source license to facilitate wider use, extensions, and research to build on the master's project results. Also, choosing an open source approach is essential to enable external review of the implementation to uncover potential bugs and establish a reliable implementation of the tool.

The following list sums up our findings:

1. Existing usability and user experience tools tests manually, and they are proprietary services with paid subscription
2. Most accessibility tools tests in an automated way against implementation of accessibility guidelines, most tools are open source, and they have a public instance available
3. There are few existing usability and user experience tools that are open source, and those who are has limited functionality
4. There are few, if any, academic projects concerned with usability and user experience testing that are both recent and active

5. There is no tool found that combines automated checking with manual checking

3.13 Objectives and requirements

Based on our findings, we have identified four objectives with respective requirements; to support tests whose results cannot be determined by an automated checker, to build a tool that supports crowdsourcing, to have a tool that is considered accountable, and finally that the tool should not require any installation by test users, nor any modifications of the websites to become tested.

3.13.1 Objective 1: Support users to verify tests that are not automated

About 20% of the perceivable tests for accessibility can currently be automatically determined. Extensive user testing is needed to get a more complete overview of the barriers of a given website. The main objective of this thesis is to explore and demonstrate ways in which this user testing can be supported.

Requirement 1.1: Integrate with automated checker To integrate with the automated checker, the automated checker needs an *Application Programming Interface* (API) that provides tests whose result needs to be determined by humans.

3.13.2 Objective 2: Support crowdsourcing

A core concept is the use of crowdsourcing to collect large amounts of testing data. Several benefits are expected from this approach including:

- Better access to more user testing data to enable quality assurance of the collected data.
- Wider coverage of client applications and their configurations including browsers, operating systems, devices and assistive technologies.
- More effective user engagement, allowing disability groups to run their own targeted initiatives e.g. to cover a group of central services for their needs.

A remark on privacy Privacy is a critical issue for UTT since the approach is planned to use data about users capabilities, their installed tools and settings, and the user behaviour. It needs to be carefully considered in the future what data is to become collected, how to inform users about this, and how to conform to the laws regarding data collection, which varies among countries. The prototype will not store any of the collect data and we therefore leave the detailed solution of these issues for a later stage (see Section 6.8).

Requirement 2.1: Intuitive to use The tools needs to communicate in a plain language that is easy for all people to understand, and it should have as few visual elements as possible to prevent the user from becoming distracted. The usability and user experience should be carefully considered when designing the user interface.

Requirement 2.2: Works on any device Some existing solutions for user testing require expensive equipment such as eye tracking input devices, and the testing may be conducted in a testing lab. This limits the applicability for crowdsourcing and makes user testing too expensive for many businesses. A goal of this research is to create a tool to enable more people to carry out better tests of websites. Desktop, tablet, and mobile devices should be supported.

Requirement 2.3: Supports assistive technologies The tool must be usable by people with disabilities. It needs to work with assistive technologies such as screen readers.

Requirement 2.4: No need to define a test run There is no need for a user role for setting up a test run associated with a website. One should be able to test any website without having to e.g. define a sequence of tests.

3.13.3 Objective 3: Assure accountability

Several accountability aspects needs to be built into the tool. This is especially important for certification of website accessibility.

Requirement 3.1: Enable review of test specification In addition to the code of tests being available, there should be a textual description describing the intent of the tests, and what they do.

Requirement 3.2: Enable review of test implementation The source code of tests needs to be open, in a way that enables people to review how they are implemented.

Requirement 3.3: Enable reproduction of test results It should be possible to review how the testing software has arrived at a certain result.

3.13.4 Objective 4: Requires no installation or modifications of websites

Requirement 4.1: No need to modify website The tool should be able to test a website without modifying it.

Requirement 4.2: No need to install any additional software UTT will not require installation of any additional client-side software besides a web browser.

Requirement 4.3: Support for different techniques The tool should be able to test content implemented using different techniques, such as HTML or PDF. Also digital TV formats should ideally be possible to test.

4 Method

This section starts out by arguing why design research was chosen over similar methods for this research project. It proceeds by detailing how the research project was carried out, and then the research model that show how the artifact relates to the context is presented. The third section describes the collaborative process, and the people involved. Finally, the process of both developing the artifact and writing the thesis is reflected upon.

4.1 Design research

It will in this section be argued why design research is chosen over similar methods for carrying out this research project. The section closes by mentioning how we choose to borrow terminology from software development methods to be able to discuss the content of an iteration.

There are three methods that will be discussed, and those are design research (DR), Action Research (AR), and a recent Action design research (ADR). For the sake of brevity, we will refer to the technology artifact as artifact, and organizational concerns as context.

4.1.1 Design research and action research

DR is concerned with the construction and evaluation of artifacts to meet organizational needs as well as the development of their associated theories” AR is a change-oriented approach in which the central assumption is that complex social processes can best be studied by introducing change into these processes and observing their effects (Cole, Purao, Rossi, & Sein, 2005).

It has been argued that in how knowledge is built, it makes little difference in practice which approach is used, despite the fact that their perspectives differ, but that there are nuances of differences (Papas, O'Keefe, & Seltsikas, 2011). Others acknowledge their similarities, while suggesting to combine the two methods into a single method, as the research approaches are said to be compatible, and that they can inform each other (Cole et al., 2005).

4.1.2 Action design research

In a research essay that proposes a new research method, ADR, it is argued that DR pay little attention to its shaping by the organizational context, that DR focus on building the artifact and relegate evaluation to a subsequent and separate phase, that technological rigor is valued at the cost of organizational relevance, and that it fails to recognize that the artifact emerges from interaction with the organizational context even when its initial design is guided by the researcher's intent (Cole et al., 2005). ADR is designed to address this problem, and it reflects "the premise that IT artifacts are ensembles shaped by the organizational context during development and use."

What seems to separate the three methods involving design is to what degree they weight the different concerns. For AR, context is a primary concern, and artifact a secondary one (an artifacts effect on the context is studied). For DR, artifact is a primary concern and context is a secondary one (an artifact is built in a context). For ADR, it seems like two are concerns are weighed equal (an artifact is built in a context, and its effect on its context is studied) ADR is fairly recent (2011), and it will be interesting to see if it gains traction in the future.

4.1.3 Choice of method

All methods discussed are in one way or another related to building knowledge through design. While there seem to be far less research projects involving design compared to natural science, it is not the intent to choose the design paradigm for experimental reasons, but rather because it is a prerequisite to solve the research problem. Given the small size of the project, we believe that the nuances of differences in the various methods are not going to affect the outcome in a drastic way. We have suggested that AR, DS, and ADR weight concerns differently. This research project is primarily concerned with designing an artifact, and its effect on a context cannot be fully understood until the artifact has reached a certain stage of maturity. We do however receive feedback during the development, and for this reason, the context is certainly not ignored—we let the feedback actively influence the development of the artifact.

Because of the limited amount of documented practice about ADR, and because we have chosen a design method out of necessity rather than for experimental reasons, we have chosen to go with a method that is well-

established. This is consistent with the choice of technologies, where we favor those that have already gained traction over those that are still at an experimental stage, despite the fact that recent one might have attractive qualities.

4.1.4 Intended use of design research

In this section, it will be stated how we intend to apply DR to guide us carrying out the research project. The Discussion section (Section 6) revisits this discussion, and reflects upon how DR was used.

The research project stem from both practical problems that needed to become solved, as well as unexplored areas of research as indicated by the existing knowledge. More specifically, this project builds further on the existing work that has been done on the eAccessibility Checker through the eGovMon project.

By using a method centered around design, it is the belief that we gain new knowledge through creation, and creation is an iterative process. The iterative nature is both emphasized by the Design Cycle model by Takenda et. al, and the Design Science Research Cycles by Hevner. It is acceptable, and even a part of the method to go back and forth between phases (Vaishnavi & Kuechler, 2004).

The development of the artifact has been split into three milestones. Evaluation has been carried out for each iteration by involving evaluators, and have them comment on the artifact.

4.2 Borrowing terminology

Design research does not include a vocabulary for discussing what is accomplished during an iteration, as it might be used for more than just software development. Software development methods such as Scrum defines one such vocabulary, where the iteration itself is named a sprint. The complete set of unsolved tasks associated with a project is named a backlog. A set of tasks to become solved is assigned to one sprint.

There are many services on the Internet for hosting the source code of open source projects, and these services usually offer an issue tracker, a wiki—and some of them even have capabilities reminiscent of social networks. GitHub is chosen for UTT, and the reasoning will follow. GitHub has an issue tracker where one can create a milestone with associated issues. A milestone can, at least for this project, translate to a sprint/iteration,

and an issue can be viewed upon as a task. On GitHub, the backlog is all open issues belonging to a project. We will try to use the GitHub terminology throughout this thesis, but Scrum is mentioned as a point of reference since we assume that the reader is already somewhat familiar with it.

4.3 Outcomes

There are two primary outcomes of this research project; the UTT artifact, and the knowledge and experience documented in this thesis.

There is one secondary outcome, and that is the findings about existing tools (See Section 3.12).

4.4 Research model

The following research model illustrates the research problem and the context that it lives in.

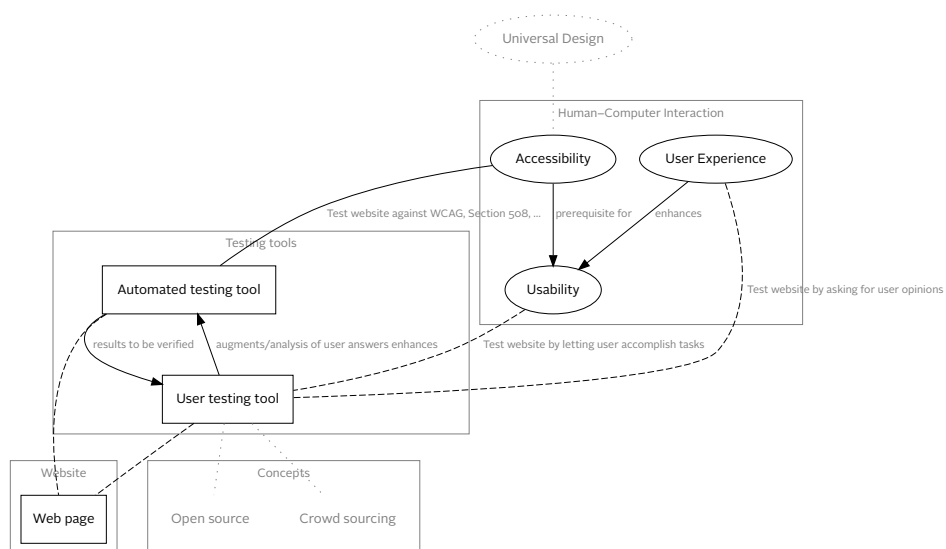


Figure 14: Research model

4.5 Collaboration

Many people in one way or another have been involved in this research project. In the introduction, the context has been described, and it will not be reiterated here. Below is a table of all people involved.

Organization	Country	Role
Agency for Digitisation	Denmark	Reference group member
Agency for Public Management and eGovernment (Difi)	Norway	Reference group member
Die Evangelische Stiftung Volmarstein	Germany	Comment on specifications and testing
Logius	The Netherlands	Reference group member
Ministry of State for Administrative Development	Egypt	Reference group member
Seniornett Norge	Norway	Testing of demonstrator
Stichting Accessibility	The Netherlands	Test, comment on specifications and method
The National Resource Centre for Participation and Accessibility	Norway	Reference group member
The Norwegian Association of the Blind and Partially Sighted	Norway	Reference group member
Tingtun AS	Norway	Coordination and development

Table 7: Stakeholders

What can be considered the core project group is Alexander, Janis, and Mikael. Alexander is the student, Janis the internal supervisor of this project. Mikael has several roles in this project. Beside being the initiator of the project and coordinator for the larger project, he is the external supervisor, responsible for keeping in touch with partners, and project owner.

All three are familiar with each other, and the atmosphere has been informal within the project group. Because of having working together for a while already, a way of working has already been established, such as how to contact each other. During the collaboration, there has been a short feedback loop, which has been beneficial for the progress. All three are deeply into the philosophy open source, and we believe in the saying “release early, release often.”

4.5.1 Meetings with supervisors

As indicated by the Gantt chart mentioned in the previous section, there have been several meetings during the project, and these were organized and led by Alexander. The planned frequency was every second week, but meetings were held less often.

An email was sent by Alexander to the supervisors beforehand containing URL the last meeting report, and the agenda. The meeting was held over Skype, and action points and decisions were the outcome of each meeting.

4.5.2 Physical location

Alexander has during the project worked at different physical locations depending on what has been working on, but most of the development and writing has been done at the Tingtun headquarters. Drafting the text was done at home, the university cafeteria, in bars and restaurants.

4.5.3 Building of the artifact

This sections will look at the development process in regards to the collaboration—how to get feedback from the evaluators.

All contact with partners, such as sending invitations for feedback, have been done by Mikael by email. For some of the emails, Alexander wrote a draft, and they were refined by Mikael before sending them out. Each round of feedback has had a duration of approximately 1–2 weeks. We expected that not all of the Norwegian evaluators could participate before the 0.3 milestone because of lack of Norwegian translations, but it did not make much difference in any direction.

When asking users for feedback, rather than sending a survey or asking open questions, we simply asked the participants to provide their comments. The lack of a predefined form for answers has resulted in a great variety of responses, and we think it worked out well. What makes this a sensible approach is both the fact that Mikael has a long history of collaboration with the evaluators, and also that the evaluators were paid for giving feedback. One example of feedback that we received was a three page long document with experiences from a number of users. Another example is a short paragraph written by one person.

The feedback that we got was from people with different backgrounds.

Birkir Gunnarson, an Icelandic expert of web accessibility that is visually impaired, gave valuable feedback of what needed to be done to improve the tool for visually impaired users. We had detailed discussions with him over email regarding implementation of the tool, such as how to present the test page body in a way that would be effective for a person using a screen reader.

A topic that needs to be touched upon is who has the final say regarding artifact design decisions when opinions differ. The answer for UTT is that the project owner has the authority to make decisions if there are disagreements. There have been discussions, such as how to receive input in the user interface. The way that we have worked is that we have discussed an idea, we have come up with a few possible solutions, and then we have decided upon what is the best idea. If there is no consensus, then one either discusses possible solutions further, or the project owner makes a decision for what solution to implement for the upcoming iteration. Not before trying a solution on real users do we know if an idea is good or not. Sometimes one have to go many rounds with ideas before a solution can be reached. When looking beyond the next iterations, there is really no “final say”—working with accessibility on the web has a never-finished nature, and continuous user feedback is the only way to measure whether a solution is effective or not.

As the technologies on the web advances, web accessibility must keep up with the changes. This way of putting it implies that new technology does not take accessibility into account, and this is in our experience often true. So as the technology are constantly changing, we need to keep up to not leave out people with disabilities.

4.5.4 Milestones and associated issues

As mentioned, GitHub is chosen for handling milestones and issues. It is used to host the source, and it has good integration between source code and issues. It was chosen because we are already familiar with it—and we like it, but there are many similar alternatives such as CodePlex, Bitbucket, Gitorious and Google Code that provide similar functionality.

Starting from UTT 0.2, for each iteration has been defined as a milestone on GitHub. A milestone has a description, and an optional date. We have used the version number as the name of a milestone, and we have set the date to a deadline that has been agreed upon. For UTT, issues stem

from discussions about objectives and requirements, feedback by users, and bugs that are discovered. For each milestone, a number of issues were selected.

An issue is on GitHub said to be open until either the creator of the issue, or one with administrator privileges chooses to close it. One can label an issue with “wontfix” if there is some reason why the issue should not be fixed; this might be because it is a feature not in line with the project’s goals, that the issue is not really an issue (as the saying goes; it’s not a bug, it’s a feature), or that the issue is a duplicate of a previous issue.

For now, only Alexander has created issues, but anyone registered as a user on GitHub is free to open an issue. During this project, all feedback has been reported by email. An issue can be labeled, and two labels that are provided by default are “bug” and “enhancement.” GitHub does not have built-in functionality to estimate the length of an issue, so we have set up custom labels for how many hours we believe an issue takes to solve. A Fibonacci sequence is used, since estimation becomes less precise as the task complexity increases.

Estimation has proven to be a challenge, and it was not done at all—at least not at a task-level—before planning the 0.3 iteration. Only the dates of the previous iterations were planned. Some of the estimation errors was caused by an underlying assumption that the developer is productive all the time. Also the tasks were not broken down to smaller tasks, which could have made the estimation more precise if we did.

One of the goals of the eGovMon project is transparency. Having the issue tracker open enables people to read the reasoning of design decisions. At the start of the project, the issues was stored as a private to-do list. Later on, it was decided to use the GitHub issue tracker, and a links to the issue tracker was put in UTT, and sent in emails to partners. It should be mentioned that we do not force evaluators to use the issue tracker to give feedback, as most of them are not software developers, and they are more used to email.

There are cases where public issues needs to be considered. In 0.2 there was a related bug where typing “http://http://tingtun.no” (or another invalid protocol) in UTT would crash the backend. We chose to fix it without making the issue public, but we then published the issue for reference. The worst case scenario in this case is that the backend goes down, and evaluators will not be able to test the tool.

A note on versioning Software components used for UTT use semantic versioning, and for UTT, we start the first iteration at 0.1. The second is 0.2, and the third is 0.3. 1.0 communicates that software is production ready. For that reason, ending the UTT development at 0.3 signals that it is an early prototype that need. This semantic versioning scheme was also used for documents belonging to master’s project—even for meeting reports—but was later dropped in favor of postfixing filenames with dates.

When it is referred to version 0.2 in the report, it really means 0.2.x—or, any version of 0.2.

4.6 Other

It will here briefly be mentioned some methods considered for developing UTT.

4.6.1 Code review

In a professional software development environment, one common procedure is to submit newly written code to be reviewed before it is included in the code base. For UTT, there is only one software developer involved, and for such a small project there is no resources for reviewing the code. Code review is likely to be more interesting if the project grows, and the number of developers increases.

4.6.2 Blogging

Alexander decided early to blog about the progress of the UTT development on his blog. The advantages was that it would function as a diary of progress, and that other people could read. Initially, some blog posts were written, but he lost interest after a while. Blogging might work better in the future, but for this project, direct communication such as email and Skype was effective.

5 Design

“View it, code it, jam – unlock it” —Technologic, Daft Punk

In this section, the design activities of building a tool that matches the project objectives and derived requirements is described in detail. We have found it most sensible to start by presenting the development of the user interface, then proceed by describing the architecture of UTT, and detailing how UTT is implemented. Finally we will describe how choices were made.

In many ways, what has been designed has stretched far beyond the scope of a prototype. A number of concerns have been taken into consideration, such as scalability and longevity, that usually are not considered when building a prototype. Because this project is a pre-project for a larger project, it was decided to carefully design the architecture, so that the artifact can be further extended in the future.

Describing the design process is in many ways a challenge, as it is a creative process where hundreds, if not thousands of small decisions are made unconsciously. There is a lot of trial-and-error involved, decisions affecting other decisions, and we have often relied on past experience and intuition rather than planning everything step out in detail. There have been frequent changes of the artifact based on experimentation and feedback from evaluators.

5.1 Revisiting requirements

The tool needs to be accessible. Since the tool should be used by many people, it should be easy to learn. And since we want the users to answer questions, the interface needs to look attractive, and it should have a fast response time after having answered a question. The next question needs to show up instantly to not tire the user. It is acceptable that the application takes a bit longer to load if it means that the user can quickly navigate between/answer tests.

The user interface also needs to be *rock stable*. Clicking a button twice by mistake while something is loading should not confuse the state of the application. Unfortunately, these issues are very common, and often require one to restart an application. Choice of technology is discussed later on, but we need to use technology that can make all of this happen, while being assured that it will stay relevant for a long time.

5.2 Designing the user interface

In this section we start by looking mockups preceding the implementation of the tool, and then present the user interface of the current version of UTT.

5.2.1 Preliminary visual design mockups

Mockups were designed during the Fall of 2012 to guide the discussion of the project. The first mockup was drawn on paper (Figure 15). The final two (Figure 16; Figure 17) were presented at a meeting at The Agency for Digitisation in Copenhagen for partners and members of the reference group about UTT and related projects. While the details have changed somewhat along the way, the general idea remain unchanged.

Grouping tests of the same type Figure 16 suggests how to present a test. Figure 17 shows how more tests of the same type can be grouped together on a test page. The current version of UTT supports only the former, but evaluators have suggested to group tests together.

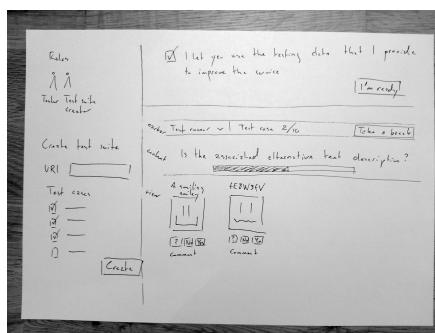


Figure 15: Initial mockup drawn on paper

5.2.2 The current user interface

The user interface of UTT provide three pages; a page shown initially for entering URL of a web page to test, a test page where the user answer questions, and a result page that both shows the answers provided in addition to automated test results.

All three pages share a navigation bar—or *navbar*—that provide buttons for navigating through the tool pages. A label is provided for the test and result page. The home page is reached by clicking “UTT.”

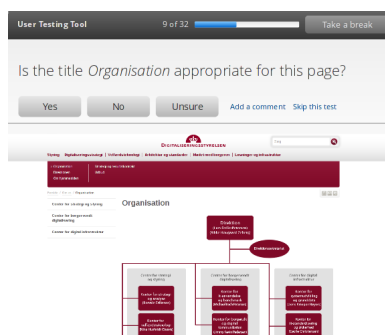


Figure 16: Higher fidelity mockup

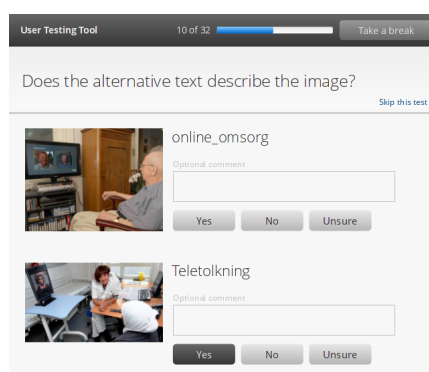


Figure 17: Tests of same type shown in single view

Home page The home page (Figure 18) is the landing page for the tool. Other than presenting general information about the tool, it lets the user enter a web page to test.

Test page The test page (Figure 19) is where the user answers questions. The page has a test page header consisting of a progress bar, a question, answer buttons, a link for going to the previous test, and a link for skipping to the next test. The test page body is implemented as an iframe.

Result page The result page (Figure 20) shows all test results returned from the automated checker, and the user tests are shown first. The non-user tests are hidden by default. Colors are used in the table to emphasize if a test has passed (green), failed (red), or if it needs to become verified (orange).

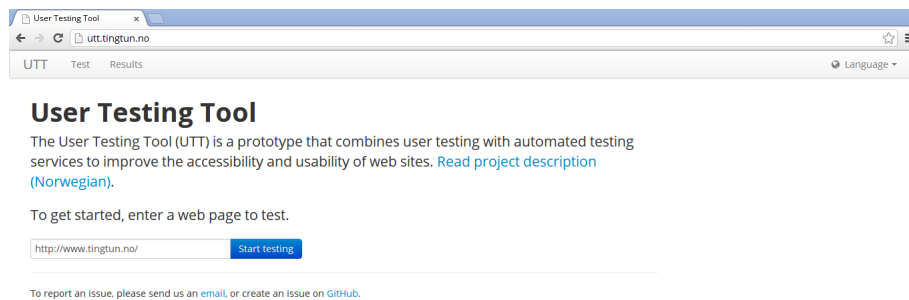


Figure 18: The home page of UTT 0.3

5.3 Specific user interface functionality

This section deals with user interface functionality where we have received a lot of feedback. Also, there are components that we have made, and some stuff that we did not have time to implement.

5.3.1 The flow of a test run

Of the results to become verified, UTT selects maximum 10 tests. Maximum two tests of the same type are presented, and tests of the same type are grouped so that they are displayed in succession. It is allowed to skip questions, and it is possible to navigate back and forth between tests.

5.3.2 Highlighting element being tested

One of the original goals we had for the 0.3 milestone was to highlight the element currently being tested on a website. As mentioned, the test page body is an embedded web page (iframe), and what seems like a good solution is to manipulate the content of the embedded web page from its parent web page.

One problem is that one cannot access the content iframe directly from the parent website. It is limited what one can do to an iframe beyond setting its URL. So to detect for instance the mouse coordinates, one needs to inject code into the web page itself. We do not want to require the website

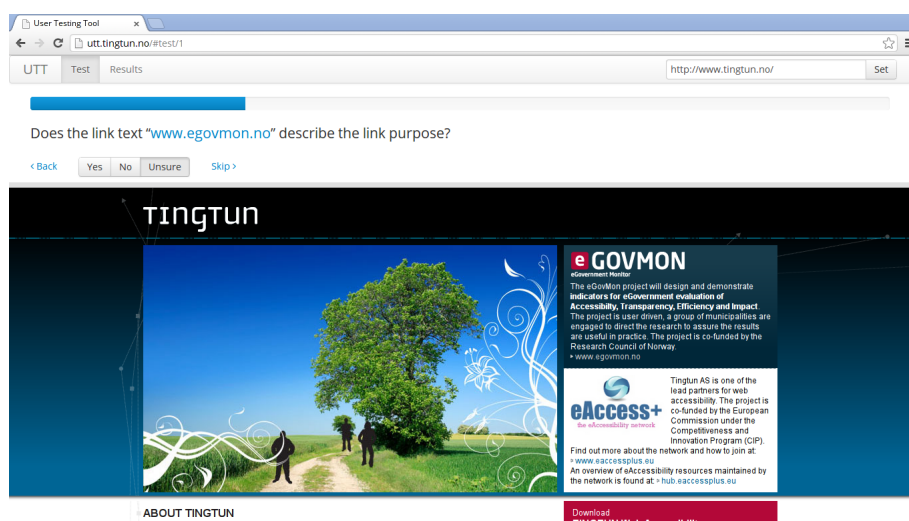


Figure 19: The test page of UTT 0.3

owner to modify it, so a solution is to run the web page through a proxy while injecting the needed code. This seems to be the approach taken by Loop11.

We did not have time to implement highlighting of elements, but we would like to use a technique similar to Loop11 if this should be done in the future.

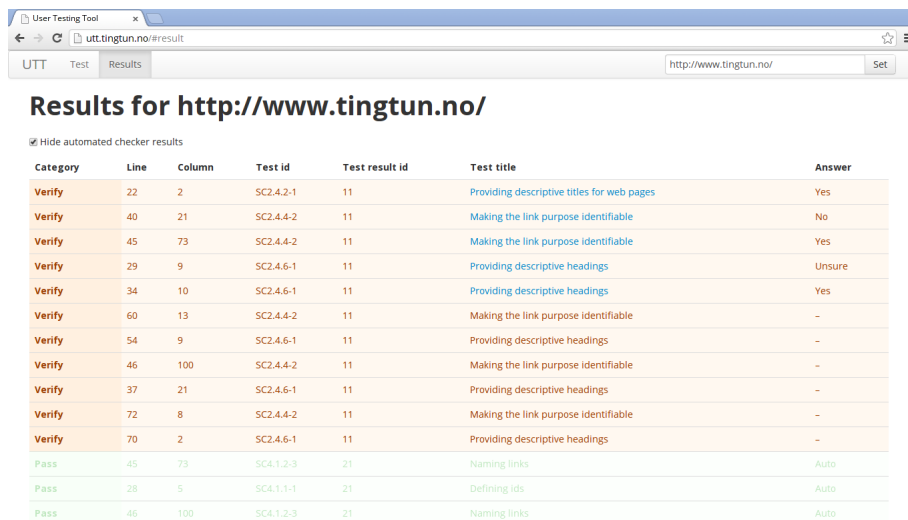
5.3.3 Change of colors

Bootstrap allows for swapping out the default theme with others that can either be downloaded for free, or bought on a website such as WrapBootstrap.²⁰ We tried a few of the free designs, and found one with dark background and light text aesthetically pleasing (See Figure 21²¹). As can be seen in the figure, it can be hard to distinguish between the header and body of a test page. This topic will be covered in a minute. It was reported by users that the choice of colors were not optimal both in terms of contrast and readability. The default Bootstrap theme was chosen instead, which has dark text on light background.

One of the feedback we got the last round was that the blue link color was too light, and that it should be underlined. What we have observed a trend the past years that the use a dark grey text color instead of black

²⁰<https://wrapbootstrap.com/>

²¹The screenshot is shown in a small size to be as print-friendly as possible.



Category	Line	Column	Test id	Test result id	Test title	Answer
Verify	22	2	SC2.4.2-1	11	Providing descriptive titles for web pages	Yes
Verify	40	21	SC2.4.4-2	11	Making the link purpose identifiable	No
Verify	45	73	SC2.4.4-2	11	Making the link purpose identifiable	Yes
Verify	29	9	SC2.4.6-1	11	Providing descriptive headings	Unsure
Verify	34	10	SC2.4.6-1	11	Providing descriptive headings	Yes
Verify	60	13	SC2.4.4-2	11	Making the link purpose identifiable	-
Verify	54	9	SC2.4.6-1	11	Providing descriptive headings	-
Verify	46	100	SC2.4.4-2	11	Making the link purpose identifiable	-
Verify	37	21	SC2.4.6-1	11	Providing descriptive headings	-
Verify	72	8	SC2.4.4-2	11	Making the link purpose identifiable	-
Verify	70	2	SC2.4.6-1	11	Providing descriptive headings	-
Pass	45	73	SC4.1.2-3	21	Naming links	Auto
Pass	28	5	SC4.1.1-1	21	Defining ids	Auto
Pass	46	100	SC4.1.2-3	21	Naming links	Auto

Figure 20: The result page of UTT 0.3

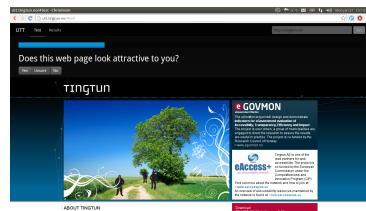


Figure 21: UTT dressed in black

is common²², and link colors of Bootstrap are somewhat lighter than the browser default link color. While these colors might be more aesthetically pleasing, they might be a hindrance for people with disabilities.

The power of a framework such as Bootstrap is that its default are used by several thousands of websites. Accessibility improvements in the default configuration will benefit many websites, and even more users. While it is nice to have a framework that looks good, it needs to be carefully considered by its developers and community what default values it should have. The changelog for 2.3²³ and a number of accessibility related issues opened by the lead developer²⁴ indicate that accessibility is actively considered.

²²A quick inspection at Google's design reveals the darkest text color in use is #222, which is *almost* black. Another example is The Next Web, where article excerpts both have a light font, in addition to having a medium gray color.

²³<https://github.com/twitter/bootstrap/pull/6346>

²⁴<https://github.com/twitter/bootstrap/search?p=2&q=accessibility&ref=cmdform&type=Issues>

5.3.4 Hard to distinguish between test page header and test page body

One reported issue for the 0.1 iteration, was that there were no obvious way to see what part of the user interface was test page header, and what part of the user interface was the test page body. This problem can be more prominent in cases where UTT tool and the website happen to share the same colors.

We can think of three solutions that might be combined to indicate that these are separate parts. We have chosen to implement two of them. One solution is that one can detect what background color the website to become tested has, and use a different color for the test page header. Another solution is to have a thick line separating the two. A third is to animate the test page header so that it slides down, while the test page body is shown. We have chosen to implement the latter two.

The test page header is only animated for the first test. A slight delay is added before the slide animation, so that the user can comprehend the test page body first. A small part of the test page header is shown initially so that the user can sense that there is something above, before it slides down.

5.3.5 Handling keyboard input

We received feedback about the lack of keyboard support for UTT. Having tools that are keyboard friendly are likely to prevent injuries such as repetitive strain injury. Having good support for keyboard is important for tasks that are repeated often, and a common use case for the keyboard in a UTT test page is to trigger one of the answer buttons. One currently needs to press the Tab-key a number of times before arriving at the buttons.

Although we did not have time to implement a solution, several solutions were discussed. One is to focus the first answer button. Then the user can press until the desired answer is highlighted, and finally press enter. The downside of this solution, and especially for a test user that gets paid for the amount of tests answered, is that he or she might choose the first answer because it is convenient. A slightly better solution might be to focus on a neutral answer such as “Unsure.” However, if that button is placed in the middle of “Yes” and “No,” then one risk that the same thing happens, as it is more work to press Shift-Tab to focus the left but-

ton compared to focusing on the next button, which requires one to press Tab. Also, not all users know about Shift-Tab to focus on the previous element, so they might cycle through the list before arriving at the first.

A better solution for ensuring that has less risk of reducing the integrity of the collected data is to map a letter on the keyboard to each button; ‘y’ for “Yes,” ‘n’ for “No,” and ‘u’ for “Unsure.” An alternative/additional solution is to map numbers to the answer buttons. It is also a possibility to enable the user to choose answer with arrow keys—this solves the Shift-Tab problem, but people are might be more used to tab over arrows when navigating through user interface elements.

Whatever solution is chosen, it should be communicated to the user in some way. We have not discussed how this can be done, but one possibility is a notification hint in the upper right corner that disappears after a few seconds. A cookie might be used for ensuring that the hint is shown three times or so. Another solution is to have a help section within the tool.

While we did not arrive at a solution for making UTT accessible by using the keyboard, we did one small keyboard focus detail that we find essential; after UTT has been loaded, the keyboard focus is set on the element accepting URL. After receiving feedback from a user, we implemented the URL input does not require one to specify the protocol.

5.3.6 Handling mouse input

Currently the user interface is optimized for answering questions using the mouse. For the 0.1 iteration, answers were implemented using a set of radio buttons—one radio button for each answer, and a separate button for proceeding to the next test as shown in Figure 23. This proved to be ineffective. The user had to click an answer, move the mouse pointer to the right to click “Next,” and then move the pointer back to the previous location to select an answer for the next test. This back-and-forth movement was reported to put strain on the hand after only a few tests. One user coined the phrase *mouse kilometers* to describe how he felt about the mouse movements.

In a pre-0.1 solution (never used by evaluators), we implemented answers as buttons that would go straight to the next test when clicked, as shown in Figure 22 (the buttons were implemented as a Bootstrap button group²⁵). However, we decided to go for the solution using radio buttons

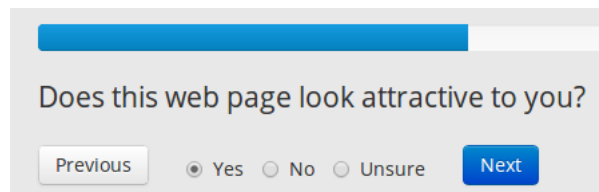
²⁵<http://twitter.github.com/bootstrap/components.html#buttonGroups>



Does the link text “www.egovmon.no” describe...

< Back Yes No Unsure Skip >

Figure 22: The pre-0.1 solution currently in use



Does this web page look attractive to you?

Previous Yes No Unsure Next

Figure 23: The 0.1 solution proven to be ineffective

right before the release of 0.1. This was done because we assumed testers would be more familiar with forms for the purpose of evaluating websites. Secondly, the fact that a form must be submitted makes it convenient for a user to correct mistakes, and because it allow the user to click various answers, and think through them before committing to one²⁶.

Based on user feedback following the 0.1 release, we reverted to the pre-0.1 solution for the 0.2 release, and consequently we chose to not view the user interface as a form. The use of a button group can be justified, as people are used to toolbars and similar user interface components for manipulating objects. In addition, toolbar buttons often lead to an action. Small navigational links was included for the user to either go back to correct an answer, or to skip a test.

Still, there are two concerns that speak in favor of radio buttons. For a group of tests within one test page (Figure 24), it might be more appropriate to have a form with a submit button at the end. The alternative is to use the current button groups, which can act as radio buttons (one of them can be pressed down, similar to a button group on a vintage stereo system), and proceed to the next test automatically when the all tests are answered.

ELMER²⁷ is a set of guidelines for helping making simpler and more effective web forms. There is no guideline stating that a form choice with

²⁶The latter is a possibly unintended use of radio buttons, but still interesting observation of how functionality is used. A similar phenomena is the use of a mouse pointer as an alternative to the index finger for reading text on a desktop device.

²⁷<http://www.brreg.no/elmer/>

Does the alternative text convey the intent of the image?

"European Commission logo"



Yes No Unsure

Comment:

"José Manuel Barroso"



Yes No Unsure

Comment:

"Irish presidency logo ©eu2013.ie"



Yes No Unsure

Comment:

Figure 24: Tests of same type grouped into one page

a submit button. We sent an email to Thomas Bjørkan, the maintainer of ELMER, and we asked if we had overlooked something. We also sent a link to UTT that shows the choice of using toolbar-like buttons. Thomas confirmed that there was no such guideline, but that he would consider including it in ELMER 3. He agreed with the solution of using buttons rather than radio buttons.

5.3.7 Other refinements

A lot of thought went into what information to present as well as what information to not present in the user interface. It was carefully considered how much emphasis to place on the various text elements. User interface test adjustments was mostly feedback from Mikael. Also formatting issues such as line breaks in the introductory text that shows up when first opening the tool were dealt with.

5.3.8 Internationalization

For an application to be able to reach to a wide audience, it is a prerequisite that it is not only available in English, but also other languages as well. For the UTT development, having the tool support other languages means that more people can give feedback. Until UTT 0.3, there was no support for languages other than English, and this might have excluded some from participating.

Adding internationalization capabilities took a bit longer than estimated, as it involved changes to the user interface, and the code base, and a throughout research on what JavaScript libraries were available for



Figure 25: UTT translated into Norwegian

offering this. Some concerns that makes internationalization complex are pluralization rules specific to various languages, and different ways of structuring a sentence.

We have prior experience with translating the eAccessibility Checker, which might have guided us to go for a solution that is more complex than what is required in a prototype; a simpler solution for translation could be realized with a two-level associative array where the first level key is the language, the level key being the string id, and the value is the translated string. As mentioned already, we have made a few decisions which makes this implementation more of an architectural base than simply just a prototype, and this is a good example of that.

5.4 The implementation

“Load it, check it, quick – rewrite it” —Technologic, Daft Punk

In previous sections, the UTT has been presented from the perspective of the user. This section dives deep into the internals, and details how it has been implemented.

UTT consists of two parts—the frontend and the backend. The frontend is the implementation of the user interface. The backend provides data such as checker results to the frontend. This is shown in Figure The frontend and the backend is implemented using CoffeeScript—a scripting language known for its high readability that compiles to JavaScript.

The frontend is structured around Backbone.js, a JavaScript library for developing SPAs, and the user interface components is based on Bootstrap, which is a collection of technologies suited towards building web applications. The backend is built using Node.js, a server side software system designed for writing scalable Internet applications.

The communication between the frontend and the backend is done through a persistent WebSocket connection. A request is sent to the backend with an URL to be user tested, and the backend issues an HTTP GET request to the eAccessibility Checker. The checker sends a JSON data structure in return containing test results. These are processed by the backend, and sent to the frontend through WebSocket connection. The general idea in current architecture is that one layer prepares data for the next layer. There is processing at each layer that formats it appropriately.

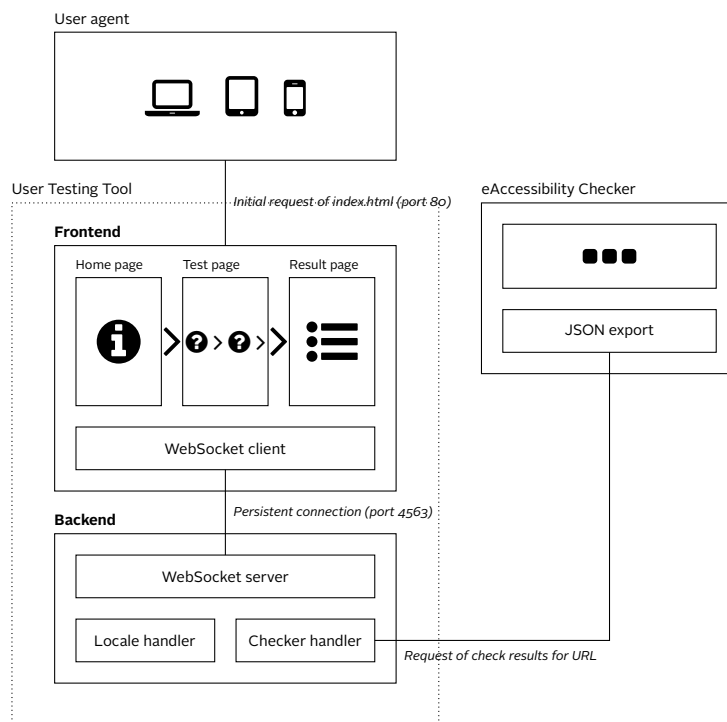


Figure 26: Architecture overview

The frontend and the backend reside in the same Git repository, and they share much of the same technologies. They are even hosted on the same server—albeit in different ways (see Section 5.9 about deployment). The remainder of this section refers to source code where functionality

is implemented. Sometimes a commit—or change—is referred to, which might include modifications of several files, and/or files that are added/removed from the source code.

5.5 Implementing the frontend

The user interface components are handled by Backbone.js *views*, the data of an application are handled by Backbone.js *models* and collections, and an application router handles navigation between pages. These elements can be configured in a myriad of ways depending on the requirements of the application.

The HTML file contains elements for each major user interface component, such as the navigation bar on top, and it also contains *templates* for each component and their various states. A template is a snippet of HTML that can contain JavaScript code to display *template variables*. To give an example of how templates are used, the test page header has a container element. To this element it is rendered a loading template when UTT is waiting for the backend to deliver checker results. Another template is rendered inside the test page header element when the results have arrived. The question and answers are among the template variables.

The remainder of this section covers specific aspects regarding the implementation. First views and router. Then state and handling of dependences.

5.5.1 Backbone.js views

Backbone.js gives the developer great freedom in how to implement views, and even define what a view is for a particular application. Technically, a view is a JavaScript class²⁸ provided by Backbone.js that is sub-classed by the application using the framework. The class provides a render function that is implemented in the sub-class, and a HTML element, usually a div element, is specified that acts as a container for the view. When render is called, the render function is supposed to update the content of the HTML element. It is up to the developer to how and when render function should be triggered. More than one view can be displayed at the same

²⁸Strictly speaking, the current version of ECMAScript does not have the notion of a class, although this is planned to appear in future versions of the language. JavaScript libraries and frameworks provide alternatives that work around this limitation.

time in a user interface, and a view can consist of other views.

UTT has user interface components such as the navigation bar, the iframe, and the. The navigational bar is shared by all views. The remaining components are shown or hidden depending on which view is currently active.

One general question in programming is what level of granularity is appropriate for one layer of abstraction, and in this context it has been a trial-and-error process of getting the granularity of Backbone.js views just right. For UTT, it made sense to have one view for the navigation bar, one for the home screen, one for the test page, one for the iframe, and one for the result view. These views are contained in a composite view.

Another issue about views are their lifecycle. Right now all views are created when the application is loaded, but the home view, for instance, is hidden instead of destroyed when navigating to the test page. Backbone allows for both mechanisms, but the risk of destroying a view is that memory leaks may occur. There are several blog posts dealing with this issue, and consequently there is a variety of suggested solutions. For UTT, we decided to reuse a view by showing and hiding it. Our main argument for this solution is that there are very few views in UTT, they do not take a long time to load, nor do they occupy a lot of memory. If there were substantial amount of views, then we would consider destroying them and creating them.

Performance was mentioned as a goal, and especially when navigating from one test to the next in a test run. Using a non-SPA/non-AJAX architecture, one would request a new page when navigating, and this would cause the iframe, and the page surrounding the question to be reloaded. This both causes the interface to blink, and it takes more time. We only want to change what actually changes, and this what is currently done. With the current SPA architecture, components render independent of each other, and they all individually respond to events. This is done instead of having a full page reload for every navigational step. The downside of the current solution is that there is a risk of accumulating view-related show/hide-bugs through a session. This has not happened with UTT so far.

5.5.2 The Backbone.js application router

This section will describe the overall flow of the user interface; how a user can navigate between pages, and how the navigation between tests are implemented. In addition, we address how the browser back button works in a SPA.

With the current architecture, it is possible to navigate around in the UTT user interface while a page is loading. Currently there is one case that is not supported, and that is changing website while a website is loading. The fields for entering website are disabled while a website is being loaded.

A user feedback that was received after the initial iteration was whether it matters in which order one answer the questions. Although a sequence is set up, it does not matter. The user can skip tests, go and back to a test to correct the answer.

Important that the user can navigate the test view using the back button and forward button. This is done by defining a parameterized route for the test page. The URL contains the id of the current test. In the result view, there are links for quickly jumping to a test for correcting the result.

5.5.3 State is in the client

For UTT, the application state is not persisted, nor shared—it lives in a tab in the web browser, and it is not stored locally in any cookie, nor on a server. The advantage of this solution, is that one can have multiple instances of UTT running, and it allows having a different web page in each tab. The drawback is that one cannot duplicate the view of one particular instance.

One use case was discovered, and this was an evaluator not being able to open new tab for comparing answers on two tests. To enable the user to do this, we would need to store the state of the instance in a cookie. This however, will as mentioned prevent the user from running multiple instances of UTT. An alternative solution for solving this issue is to provide sufficient amount of information so that the need for another instance vanishes.

5.5.4 Managing dependencies

As will be discussed in the section about choice of technology, the version of JavaScript that currently exists does not have support for defining and loading modules. For backend technologies, the Node.js platform has one way to do it, and for frontend technologies, it is usually solved by using the Require.js JavaScript library.

One issue with UTT is that the user interface takes a while to load. At version 0.2 there was a bug that was quickly corrected²⁹ that resulted in the user interface to sometimes not load at all³⁰. This was caused by a race condition bug where one module that depends on another did not load in a specified order.

Since the frontend uses a number of libraries, and since it consists of a number of views that spans over multiple files, it would be beneficial if all JavaScript files were concatenated into one file. That would reduce the number of HTTP requests for the tool and speed up loading. Also, it is possible to minify the JavaScript and CSS files. Minifying files involves using a tool that shortens function and variable names, and strips the files for whitespace. This makes the files much smaller, and as a result they load faster. Another technique that can be used in combination with the former two is to run the source code through a tool that removes functions that are not in use.

We currently only have limited experience with these tools, based on the estimated effort, we decided to not implement the abovementioned techniques in the current prototype. To make loading of libraries a bit faster, we chose CDN/hosted versions of libraries that were available. Vendors such as Google host jQuery and libraries that are used often on a fixed URL that anyone can reference. An advantage of doing so is that these files—since they are referenced by so many—are often cached at the Internet service providers. A disadvantage is that some environments accessing UTT might only have Intranet access.

As of UTT 0.3, loading can take a few seconds, and it could be helpful to display a loading screen. We hope that applying the mentioned minimification steps can help, so in the future, that should be attempted first. We created an issue on GitHub suggesting to add a loading indicator³¹, but we later closed with the comment “This should not be necessary if us-

²⁹<https://github.com/alexanderte/utt/commit/a91184517e87af7ba976578adb998b92aad97302>

³⁰<https://github.com/alexanderte/utt/issues/10>

³¹<https://github.com/alexanderte/utt/issues/25>

ing CDN, minified JavaScript and CSS, ..." This is an example of the issue tracker to document known issues and suggesting solutions.

5.6 Backend implementation and architecture

While the frontend is the implementation of the user interface, the backend is what provides data and stores data for the frontend. The data that the backend provides is data from the automated checker. It also provides translation strings to the frontend.

The backend starts a WebSocket server that the frontend initiates connections to. The backend sets up a WebSocket server listening to port 4563, a port which is unused according Internet Assigned Numbers Authority³² If the project should continue after this pre-project, then it would make sense to request that port 4563, or any other available port, gets assigned to UTT.

The life of a connection spans for the period that a client instance of UTT is open. When the user opens UTT, the frontend connects to the backend. Once connected, the frontend requests the backend to send translation strings for English, which is the initial language³³. When the translation strings are received, the frontend asks for test results for the initial website, which is tingtun.no. From that point on, translations and results are requested as they are changed in the user interface.

The backend asynchronously asks the checker for check results, and delivers them to the frontend when they arrive. Because this is done in a non-blocking manner, it means that the backend can process additional requests from the same source, or for other instances, while waiting for the checker. The current backend architecture allows for a sequence like the one below:

1. Frontend₁ requests backend for foo.com checker results
2. Frontend₁ requests backend for translation
3. Frontend₂ requests backend for bar.com checker results
4. Backend reponds Frontend₁ with translation
5. Backend reponds Frontend₂ with bar.com checker results
6. Backend reponds Frontend₁ with foo.com checker results

³²<http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml>

³³It might become possible to configure this later e.g. for businesses to run a private installation of UTT.

As mentioned, each layer in the architecture prepares data for the next layer until the template is reached that finally renders the data for the user. Once the backend has received checker results, it prepares results for the frontend. Possible answers to results that are to-be-verified are added, and translation parameter strings are prepared. Note that both to-be-verified results and pass/fail results are received from the checker and sent to the frontend—results of all types are shown in the results view.

5.7 Integration with eAccessibility Checker

The to-be-verified results that the user will provide answer to with UTT comes from the automatic eAccessibility Checker, as indicated in the literature review. There are two ways of getting the data from the eAccessibility Checker; one is to use its export functionality, and the second involves using its component to get the results, and connect to a database that parses the result. The former was chosen, as it does both steps.

First we attempted to use eAccessibility Checker's CSV export functionality. This is accessed by sending a HTTP GET request to the eAccessibility Checker with "csv" and an encoded URL as arguments. The checker runs a check on the web page associated with the URL, and returns a CSV file with the barrier data. A module was first written that would make sense of the results and build result objects that could be used³⁴. However, the data returned from the checker was among other things lacking line numbers and columns to locate the position of a barrier in the code behind the page.

Since we have access to the eAccessibility Checker, we have the option to modify its CSV export, but this can break others that depend on it. JSON is a data type that is more native to JavaScript as it is syntactically almost identical (it is in some cases) and thus require less processing by the UTT backend, and we chose to implement JSON export functionality right within the automated checker.

One limitation that we stumbled upon when implementing the JSON export functionality, is that the data that a template associated with a export type has access to is formatted for being presented in eAccessibility Checker user interface. Some of the data include custom markup such as [at] [/at]. This was needed to work around, and the JSON exporter does

³⁴<https://github.com/alexanderte/utt/blob/5af73db797a124b7b0429b93de037f31812f5540/js/egovmon-checker/egovmon-checker.coffee>

some search–replace processing of the data before sending it to the host that sent the HTTP request.

5.8 Architectural concerns

5.8.1 Directory structure

The UTT directory structure is pretty straightforward; there is a backend and a frontend directory at the highest level of the Git repository as shown in Figure 27 (alternatively, the Git repository can be browsed on GitHub³⁵). At the root level, there is a `.gitignore` file to prevent certain files to be checked in, a shell script used for making deployment easier, and a `readme` Markdown document that in few words describe what UTT is and how to set it up.

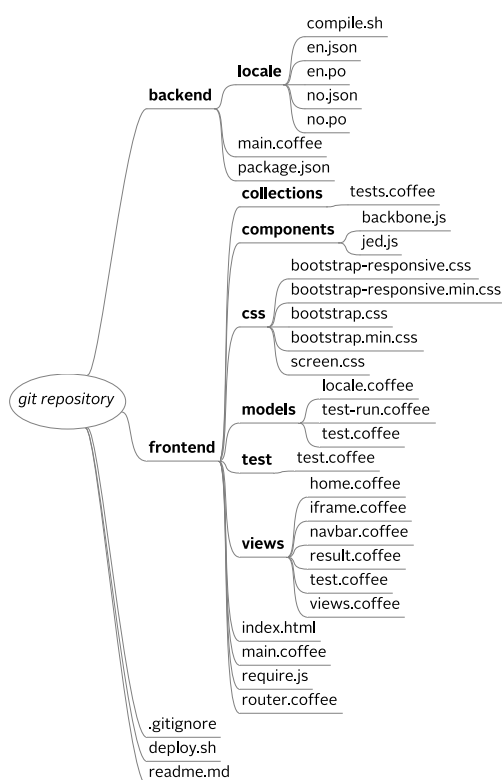


Figure 27: The directory structure in the git repository of UTT

³⁵<https://github.com/alexanderte/utt/tree/0.3.0>

The frontend directory is where `index.html` resides. The directory contains one sub-directory for each type of Backbone component; views, models, and collections. There is usually only one router for an application, so the router code is put right inside the frontend directory.

There is a directory named `components` that refer to third party components. Because many of the UTT dependencies are hosted by content delivery networks (CDN), there are currently only two files in this directory; `backbone.js` and `jed.js`. The directory name `components`, sometimes named `vendor` or `lib`, was chosen because it is used by a popular package manager for web applications named `bower`.³⁶ `bower` is currently not used for managing external components, but it is likely to be used in the future. A package manager such as `bower` streamlines among other things the way external components are retrieved, and it is more convenient to type “`bower install backbone`” on the command line rather than hunting on the web for the `.js` file.

Early on, for the sake of tidiness the CoffeeScript files were stored separately from the compiled JavaScript files. This resulted in a somewhat deep directory structure, which made it cumbersome to navigate to the files from the editor. Later on the directory structure was flattened out³⁷, and the uncompiled and compiled files now live side by side—this has no technical implications.

Both the frontend and the backend has a `main.coffee` file, and these are the main entry points of each part of the system. For the backend, the `main.coffee` can be started directly on the command line—it is compiled on the fly. The `main.coffee` for the frontend needs to be compiled, and `main.js` is referenced in `index.html`. The `main.coffee` for the frontend also references `require.js` which is what actually loads `main.js`.

The directory structure evolved gradually—the development process has in general alternated between chaos and tidiness. Splitting the frontend and backend into separate directories³⁸ was done as late as a month after the development started.

5.9 Deploying and monitoring the solution

It will briefly be discussed how UTT is deployed into a production environment. Today, it is common to use cloud services for hosting web appli-

³⁶<https://github.com/bower/bower>

³⁷<https://github.com/alexanderte/utt/commit/c07b5b7e3bd45853215cf649c349d051317237fc>

³⁸<https://github.com/alexanderte/utt/tree/fea9f608599f86c32b8ed48449e443fd2115fbd3>

cation. One variety is platform as a service. Since the backend builds on the Node.js platform, it is possible to use a service like Heroku³⁹ that provides an environment where Node.js is set up, and the service abstracts the work of setting up an operating system with a working Node.js environment. In addition, a service like Heroku provides command line tools that makes deployment even easier.

For UTT, we have chosen a infrastructure as a service solution, where an instance of the Debian operating system is set up in a cloud environment. We need to manually set up Node.js with the required dependencies. These are documented in the `readme.md` in the UTT repository.⁴⁰ The frontend directory consists of static files, and it is simply served by Apache—it could be hosted by any web server.

The backend is started by running `backend/main.coffee`. This process is somewhat involved, and it is simplified by running a `deploy` script. More about this in a second. The `main.coffee` script prints debug information to standard output, and this can be useful to log. `deploy.sh` is a script that fetches the latest revision of UTT from the remote git repository, and it starts `main.coffee` by issuing the following:

```
nohup coffee backend/main.coffee&
```

nohup(1) is a UNIX command that is useful when logged in to a SSH session. It makes sure that the program following it still runs when the SSH connection is closed, and it also logs the standard output from the started program to a file named `nohup.out`.

The `deploy` script is intended to be run both the first time the UTT backend is run, and also when it should be updated to the latest revision. One challenge we faced was how to kill the running instance of the UTT backend, before doing the update followed by spawning a new process. It currently works by writing the UNIX process id (PID) of the UTT backend to `/var/run/utt.pid` once it has started. If the `.pid` file exists when launching the `deploy` script, then it is read and evaluated into the first parameter to `kill`.

5.10 Choice of technologies

This section describes the process of choosing a technology stack. First, a number of concerns are discussed, then a number of ecosystems with

³⁹<https://www.heroku.com/>

⁴⁰<https://github.com/alexanderte/utt/blob/0.3.0/readme.md>

associated programming languages are looked at, then there is a review of components such as web frameworks, databases, and libraries. Finally the stack of technologies is listed.

We have some requirements that influences the choice, but much of it admittedly boils down to what the developer enjoys working with. There are many choices that can lead to a working prototype, and it should be possible to get something working almost regardless of choice.

5.10.1 Concerns for choosing technologies

Several concerns were considered when choosing the technologies to implement the tool with. We will present 11 concerns for choosing technologies. These come from our prior experience with software development:

1. Good fit for the problem
2. Easy to integrate with existing technologies in use
3. Scalability
4. Popularity/longevity
5. Healthy community
6. License
7. Accessible
8. Past experience and/or what network of people knows
9. Layers of abstraction
10. Linux and web-centric
11. Productivity

At the most basic level, the chosen technology needs to be a good fit for the problem. If not, then one need to work around its limitations. For instance, a web framework will need to strike a balance between ease-of-use to address predefined problems and flexibility to support other problems.

The technology needs to integrate well with the existing technologies in use. For UTT, we need to interface with the eAccessibility Checker. This involves issuing an HTTP request, and a JSON data structure is returned. This is not complex. What is slightly more complex is the future

plan of integrating the results view of the eAccessibility Checker with the UTT results view. If this is to be done in a seamless manner (e.g. not embedding the results component using an iframe), then we need to ensure that its dependencies can be satisfied by UTT.

Scalability is a common concern, and in this case the crowd sourcing requirement. While a prototype usually does not need to scale, we have put some effort in the architecture, and allowing it to scale in the future is important. Scalability is not a fixed attribute tied to a technology. What scaling means is very specific to the problem to solve. For instance, for UTT and its current requirement, its an advantage to have the SPA architecture, as one does not need a HTTP request when navigating back and forth between tests. This drastically reduces the amount of bandwidth needed.

When we build software today, we are tied to the technology choices for a very long time. Changing the underlying technologies means that the software needs to become rewritten. This is true especially for programming languages and web frameworks. Specific components might be costly to change, but it is possible, and sometimes needed. The popularity of a technology is an indicator of its longevity—how long we can expect it to be supported and maintained. Choosing a technology that is established in the market is a requirement.

Related to the popularity and longevity of a technology is the the community surrounding it. One will regardless of choice encounter many minor and larger issues, and for that reason it is vital that the community is healthy with friendly and helpful people. It certainly also helps that its a vibrant community that love what they do, and do interesting things with it both professionally and personally.

It is a requirement that license of the software is compatible with what we're trying to do—to build an open source tool. This is not an issue with the many programming languages and web frameworks out there. Most are licensed using a very liberal license such as MIT or BSD.

Although the technologies mentioned in this section are somewhat general, we need to keep in mind that they make it possible to have a solution that is accessible.

Time is a limitation in any project, and for that reason our past experience has been weighed, as we need to get up to speed as fast as possible. What network of people the person has might influence the choice. Then it is easier to ask for help, and it is nice to have someone discussing it with.

At the same time, this needs to be balanced with the other factors.

In the world of web, there are abstraction layers for everything. They are there for browser incompatibilities and for simplifying APIs. One possible downside is when one does not understand what is happening below the layers. Another downside is added cruft. It's a fine balance when to use an abstraction layer and when not to.

The technologies mentioned are all open source. Most of the technologies are Linux-centric, meaning that they are designed around the Linux directory structure, and configuration is done in text files. Most of the technologies are available from a Linux distribution's package manager, and they should be easy to install on OS X. On Windows however, they are not first-class citizens. For many popular technologies, a "one click" Windows installer is provided on their respective websites. Many of them work by installing Cygwin or similar.

Productivity is the final concern. There are some factors that make a technology choice productive. One is that the developer can solve a problem with as little code as possible. Another is that the feedback loop between writing the code and running the code is short. Third, the author must find the technology enjoyable to spend a lot of time with it.

5.10.2 Choice of ecosystem

Choosing a programming language for doing web development is not about programming language as much as it is about choosing an ecosystem consisting of components, websites for browsing available packages, command line tools, a community of people and key personalities. Often there is a culture that shares a philosophy, such as Python's "There should be one—and preferably only one—obvious way to do it." (*PEP 20 - The Zen of Python*, n.d.)

Since JavaScript is the only language supported by web browsers, it is a good idea to learn that in addition to the chosen language. While there are many languages that compile to JavaScript, it is necessary to know JavaScript in order to debug running code in the browser. In many ways, the following categorization makes little sense, since technologies are mixed and matched in a myriad of ways. Despite of this fact, this is an attempt at looking at the ecosystems with one language being the dominant one.

This is not a systematic review of languages; we found the subject too

complex to present in e.g. a matrix. Rather it is a review taking into account the before-mentioned concerns, and seeing how they fit with our objectives and requirements. All the ecosystems covered are large, highly successful, and they are used to power some of the largest websites on the Internet.

Haskell We originally planned to write the tool using Haskell, as Haskell has several characteristics and qualities that makes it desirable for technology where correctness is important. The author had no former experience with Haskell, or functional programming, and the learning curve seemed too steep to deal with in the limited time-span of the project.

While Haskell has a steep learning curve, it seems to be valued highly by those who use it, and it is common that even an experienced programmer needs to make many attempts at learning it before being able to grasp its fundamentals. One possible drawback of choosing Haskell is that the number of potential contributors might be lower than if using ecosystems that are used by more people. On the other hand, it is likely that the quality of contributions is very high, since Haskell programmers usually are very competent.

JavaScript JavaScript is the language for the web. All modern browsers supports it. In fact, they support no other language, and it is required to have JavaScript enabled for many modern websites to function. Although it has Java in its name, it does not have anything to do with Java beside superficial, syntactic similarities. At the specification level, the name of JavaScript is ECMAScript. The language is dynamic, weakly typed, has first class-functions, and it supports object-oriented, imperative, and functional paradigms.

JavaScript is not particularly known for being a robust language, and it has been subject to a lot of jokes in the community of software development. One striking example of what JavaScript is less than good at is comparing one entity with another (*JavaScript Garden*, n.d.; Pierre Kircher, 2012). JavaScript mahatma⁴¹ Douglas Crockford has written a short book named JavaScript – The Good Parts (Crockford, 2008), where he describes what subset of the language that can be safely used. There

⁴¹“I do not think of myself as a guru. I’m more of a mahatma.” —Douglas Crockford <http://www.oreillynet.com/pub/au/3252>

are abstraction languages on top of JavaScript that prevents the user of the language to use features beyond “The Good Parts.”

The latest iteration of ECMAScript lacks features that are common in most other languages, such as the ability to define and load modules. This is planned to become solved in future revisions of the language. Although JavaScript is capable of programming in an object oriented-style, it does not have a class keyword. Currently, what JavaScript lacks is solved by third-party modules, and languages such as CoffeeScript.

While privacy and security continues to be a concern, modern web applications are starting to resemble native desktop applications both in terms of features and performance. JavaScript was previously known for having poor performance, but a lot of innovation have happened during recent years, and it is safe to say that this is no longer the case. We are starting to see first-person shooter games being ported to JavaScript.

Despite its shortcomings, JavaScript is becoming increasingly more relevant. The fact that it is available in every web browser makes it attractive, and unless using an abstracted language, it does not require one to compile. Instant compilation time gives a short feedback loop between working on a piece of code and testing it in the browser, which for many makes programming feel fun and productive. And what JavaScript lacks is currently duct taped on top using external libraries.

CoffeeScript There have recently been a number of languages that abstracts the complexity and somewhat verbose syntax of JavaScript. Some of them are CoffeeScript and Dart. There is also a Java web framework such as Google AWT that generate JavaScript. Figure 28 is an example from coffeescript.org showing what CoffeeScript looks like when compiled into JavaScript.

```

grade = (student) ->
  if student.excellentWork
    "A+"
  else if student.okayStuff
    if student.triedHard then "B" else "B-"
  else
    "C"

eldest = if 24 > 21 then "Liz" else "Ike"

```

load

```

var eldest, grade;

grade = function(student) {
  if (student.excellentWork) {
    return "A+";
  } else if (student.okayStuff) {
    if (student.triedHard) {
      return "B";
    } else {
      return "B-";
    }
  } else {
    return "C";
  }
};

eldest = 24 > 21 ? "Liz" : "Ike";

```

Figure 28: CoffeeScript and JavaScript side-by-side

As can be seen, CoffeeScript attempts to make JavaScript easier to read by dropping much of its verbose syntax. Also it compiles into the subset of JavaScript from “The Good Parts.” The simplified might not be for everyone, especially as many developers are familiar with the mustache brackets and semicolons found in C-like languages. But code written in CoffeeScript is most often significantly shorter.

The downside of using CoffeeScript, or any other abstraction layer over JavaScript, is that there is yet another technology to learn, there is an additional compilation step, and it can make debugging more complicated, as there is limited tool support for debugging the original code that was written. The web development community on HackerNews seem to be divided because of the reasons mentioned. Also, the syntax is said to be a bit magic, and in some cases ambiguous when optional parentheses are left out.

For UTT, we decided to use CoffeeScript. It was introduced early, as it would be no need to port the whole thing later on.

Node.js Node.js is an asynchronous, evented I/O framework making it possible to create server applications written in JavaScript. It has gained much traction recent years, and it has a large and vibrant community. Node.js exposes a POSIX-like API for reading and writing files, managing operating system processes and threads, and provides a non-blocking programming model especially suited for real-time applications.

Node.js was chosen for the backend. As with the frontend code, one can write scripts for Node with CoffeeScript. Both the frontend and the backend of UTT are written using CoffeeScript. One of the advantages with Node, is that modules can be shared between the frontend and the backend.

Ruby Ruby is known for being one of the most readable programming languages. It is known to perform slightly slower than some of its alternatives, but in most situations, it being easy to read and the fast development time outweigh the execution speed. Ruby is used a lot on the web, and Ruby on Rails web framework powers large websites like GitHub and Twitter.

Rails is an opinionated framework that dictate how to structure the application, but there are more lightweight web frameworks available for those that desire more choice .

Recently there have been written about some serious security flaws. While recent patches has addressed these, some say that this is an indicator of a deeper problem (*What The Rails Security Issue Means For Your Startup* / *Kalzumeus Software*, n.d.).

Ruby is a still a good candidate for UTT.

Python Python is an all-purpose language designed by Guido van Rossum, and it widely used for web development. There are many different web frameworks written in Python. Some frameworks such as web.py, by the late Aaron Swartz, provide the bare minimum functionality while there are web frameworks such as Django⁴² that is complex similar to Rails. Django is one of the most widely used frameworks, and it is used by some of the worlds most visited websites. The list includes Instagram and Mozilla.

Python has a readable syntax, it is a great language, and it would be suitable for UTT.

PHP PHP is a widely adopted language known for its great performance. Much of Facebook is written in PHP. Its standard library is known for being transparent regarding the API, as they are very similar to the C libraries underneath. Because these C APIs vary a great deal, and because the PHP API seems to not be carefully designed, it is often criticized. One software developer goes as far as saying that “PHP is an embarrassment, a blight upon my craft. It’s so broken, but so lauded by every empowered amateur who’s yet to learn anything else, as to be maddening. It has paltry few redeeming qualities and I would prefer to forget it exists at all” (*PHP: a fractal of bad design - fuzzy notepad*, n.d.).

We have been using PHP along with the CakePHP web framework earlier, but found it less than pleasant.

Java Java is a widely used technology, and perhaps more so in large enterprise environments than in smaller environments. Compared to some of the other languages mentioned in this section, Java is somewhat more verbose, it takes longer to compile, but it might be more desirable than interpreted languages for large code bases. Compile-time type checking catches errors that otherwise would could show up when the application is production. Also, type checking means that the code can be refactored/shaped in radical ways using refactoring tools.

⁴²<https://www.djangoproject.com/>

Java has a large number of web frameworks. Some of these are Apache Click, Apache Tapestry, and Apache Wicket. Another well-known web framework is Google AWT.⁴³

One Java web framework that should be mentioned is Grails. The code is written in Groovy, which is a dynamic programming language for the Java platform.

Java is arguably more suited towards enterprise applications rather than prototypes, and it is not considered an option for UTT.

Others There are a large number of languages that could be reviewed, such as Scala, Lua, Go, Lisp, Erlang, Perl, or even C—yes, there are web frameworks for C, and they might be suitable in embedded environments with performance and memory constraints. Still we think the most used technologies are covered.

5.10.3 Choice of web framework

The choice of web frameworks is limited to SPA frameworks. During the past few years a lot of SPA frameworks have appeared. One of major aspect in which they differ is how opinionated they are. Frameworks such as Ember.js and Angular.js shares some similarities to Ruby on Rails, while Backbone.js in particular leaves it up to developer on how to combine its various features.

Of all frameworks (and even all projects on GitHub), Backbone.js is most popular⁴⁴, indicating that it is used on more project than the other frameworks hosted on GitHub. Because Backbone.js leaves so many decisions up to the developer, there are frameworks built on top of Backbone.js. But its simplicity might be its strength. The following is a comment from a developer on StackOverflow:

“Then I remembered why I originally chose backbone.js: simplicity. All those frameworks are adding what backbone has left out by design” (*What are the real-world strengths and weaknesses of the many frameworks based on backbone.js?* - Stack Overflow, n.d.).

I think one could choose any SPA framework, and accomplish just about the same. We chose Backbone because of its popularity, simplicity, and also because it was said to be faster to learn than the others. The user in-

⁴³These web frameworks are listed on http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks#.

⁴⁴<https://github.com/popular/starred>

terface of UTT is highly specialized, which is a good reason to go for a framework that is not opinionated. We learned the basics of Backbone.js by experimenting and reading resources on the web.

5.10.4 Choice of additional components

This is a section about components whose choice is of less significance, and it is for that reason short. These components can be replaced without requiring a rewrite of the code. Some, such as Require.js, do require a bit more. jQuery is not covered here, as it is a requirement for nearly all web development.

As mentioned, JavaScript does not have a concept of modules. Require.js by far the most popular framework for handling dependencies in the browser. Node.js has its own way of loading modules. browserify is an alternative loader for the browser that makes loading of modules similar to Node.js⁴⁵ It is beyond the scope of the thesis to go further into detail about module loading.

WebSocket are used for communicating between the frontend and the backend. WebSocket are not supported by all browsers, and for this reason the Socket.IO library provide a client-side module with that can fall back using other means of transport. caniuse.com is a site for checking which browser supports e.g. WebSocket or other recent browser technologies. Internet Explorer 9 and below does not support WebSocket, and in this case Socket.IO uses a Flash implementation.

Backbone.js is dependent on a helper library named Underscore.js. Underscore.js provides utility functions for advanced list manipulation, and many of these are similar to map, head, tail functions in Haskell. In addition, Underscore.js provides functions that compensate for what is error-prone in native JavaScript—such as checking if a list is empty. Lo-Dash is a competing project⁴⁶ that seem to be actively developed⁴⁷ The developer of Lo-Dash, John-David Dalton, states in a post on StackOverflow that Lo-Dash has a more consistent API, more thorough documentation, and tests. He has also had push access to Undercore.js, and he had con-

⁴⁵<http://browserify.org/>

⁴⁶Lo-Dash is a pun reference to underscore. This is common for open source libraries. CoffeeScript has a build system named Cake, and a “third party” build system that simplifies Cake is named Applecake, both as a reference to Cake, but possibly also Apple’s philosophy of making simple user interfaces.

⁴⁷<http://stackoverflow.com/questions/13789618/differences-between-lodash-and-underscore>

tributed with bug fixes to Underscore.js. This is a form of competition that is unheard of in proprietary software projects, and it is likely that Underscore will be replaced with Lo-Dash as the default utility library in the future. With that said, we do not find the choice of utility library critical for UTT, and we went for Underscore.js because it is suggested on the Backbone.js website, it seems to be widely used, and it is developed by Jeremy Ashkenas, the developer of Backbone.js and CoffeeScript. The fact that it is widely used means that any possible problems are easy to google⁴⁸. In other words, technical merit is a less significant criteria in this case, as we feel sticking to what is suggested by default leads to the least number of surprises. We assume that we share this pattern of thinking with many others.

Although there are currently no automated unit tests for the source code, we initially did a review of unit testing frameworks. Mocha by TJ Holowaychuk seems like a widely used project, and it has an appealing syntax. This could be used for further development.

Bootstrap, formerly known as Twitter Bootstrap, was initially developed at Twitter to streamline the user interface of the service, but it has since “grown out of” the organization. Bootstrap makes it simple to create good looking user interface components such as progress bars and navigation buttons. It also provides a grid-based layout. Grids are useful within visual design to establish solid visual and structural balance of websites (*Designing With Grid-Based Approach / Smashing Magazine*, n.d.). Version 3 of Bootstrap is in development as this is written, and the difference between the current version and the next is that the latter is responsive by default, and mobile-first.

Supporting multiple languages can be complex, as different languages have different pluralization rules, and the structure of sentences differ widely. Also, there are software specifications such as gettext and MessageFormat that attempt to solve this. There are a JavaScript libraries for making a program internationalized, and I have chosen `jed`, which is a GNU gettext solution. GNU gettext is also used by eAccessibility Checker.

⁴⁸We find it appropriate right here to use the google as a verb—besides, it is a verb (*Google - Definition and More from the Free Merriam-Webster Dictionary*, n.d.).

5.10.5 Choice of database technology

Using database technology is the common way to store data associated with a web service. Unfortunately, we did not have time to implement any data storage, but this section could be useful for future development.

There are what seem to be two directions for database technology for the web. One is NoSQL and the other is traditional, relational databases. The term NoSQL has appeared recent years, and differs from relational databases in that one does not model the relationship between data. There are no schemas, but rather collections of data structures. The data structures are most often encoded as JSON. Since JSON is close to being JavaScript data structures, it means that for JavaScript applications, the data can travel through whole stack nearly unmodified—the whole way from the database, and up to the view template. Nearly all languages have functions built in to serialize and deserialize a native data structure to and from JSON.

Two popular NoSQL database technologies are MongoDB and CouchDB. We have observed that there have been reported stability issues with MongoDB, but we are uncertain whether these are major flaws of the product. CouchDB is a cloud storage service. PostgreSQL, and MySQL are examples of popular relational databases. For smaller applications and/or test environments of a larger application, there is SQLite which is a relational database, but the database is contained in a single file that can easily be moved around.

Another variety that is neither NoSQL or relational is Redis, which is an in-memory key-value data store. It is fast and simple, and especially useful for caching.

For future development of UTT, we suggest to use PostgreSQL, as it is already used by the eAccessibility Checker, and also because the domain of testing websites seem to fit well with the schema approach.

5.10.6 Building a software stack

A software stack is not only related to web technologies, but generally a set of technologies intended to work as a whole. Things changes fast in the world of web development, but we hope that the larger components will stay relevant and supported in at least five years to come.

The following table sums up the stack of technologies that was chosen for implementing UTT.

Technology	Description
Node.js	Event-driven I/O server-side JavaScript environment
CoffeeScript	Programming language that compiles into JavaScript
Backbone.js	JavaScript frontend framework
Underscore.js	JavaScript utility library
Require.js	JavaScript file and module loader
Socket.IO	Cross-browser WebSocket framework
Mocha	JavaScript test framework
jed	JavaScript gettext translation library
Bootstrap	Frontend user interface framework
PostgreSQL	Relational database

Table 8: The chosen technology stack

5.10.7 License

UTT is released under an open source license. There are many reasons for this. When the software is free—both as in price and freedom—the threshold for others starting using is lowered considerably. Also, having an open source license is an invitation for developers to contribute with additional tests, or refine existing tests. A possibility that lie further ahead is having a team of data analysts study large amounts of collected test data.

An objective of the Integrative Learning Design Framework (page 8) is to design artifacts that can be profitably used in later stages, and it should be possible for other researchers to do this. An open source license facilitate for letting others continue the development. This way, most is gotten out of the time developing the software artifact.

UTT is licensed under the GPLv2 license⁴⁹. Licenses can be hard to understand if one is not lawyer within the field of software. What follows is an easy-to-understand summary (*GNU General Public License v2 (GPL-2) Explained in Plain English*, n.d.):

You may copy, distribute and modify the software but you must
relicense any changes and your entire project under GPLv2 and

⁴⁹<http://www.gnu.org/licenses/gpl-2.0.html>

disclose all the source code. You must also track changes and dates in the source code. The GPLv2 is complex and may not be compatible with common commercial licensing models. Be careful and view [Full Text](#) for details.

6 Discussion

This section starts out with a section reflecting on how design research was used to carry out the study, and follows with the main findings of the study of existing tools. The section is followed up by a discussion about the implementation with choice of technologies, and the implementation of the backend and the frontend. The section that follows goes through known limitations of the research results. Then the research questions asked initially in the introduction will be addressed. The remainder of this section suggests future directions that projects can go into that builds on this research, there will be a section for reflecting of the process of developing the artifact and writing of the thesis, and finally there is a short section about concerns regarding privacy of the users of the tool.

The discussions also aims to describe contribution to research—additions to the knowledge base, and verification of existing knowledge—and lessons learned during the process. Lessons learned include what we would do different if we started today, and what should have been, but was not delivered. This is summed up in the conclusion.

6.1 Reflecting on the method and prior art

It will be reflected on how design research was used to carry out the research, and also how the study of the existing tools has shaped the development of the tool.

6.1.1 Design research

The Design Science Research Cycles model by Hevner was used to guide the research process. In addition to the research method, we used a Scrum-like process of handling tasks and sprints. It was necessary to use a subset of the software development method in addition to the research method, since design research is not strictly tied to software development. As GitHub was chosen as a tool for supporting the development, its terminology was used during the project.

The Design Cycle is the heart of a design research project (Hevner, 2007), and it was given a great deal of attention. With the exception of letting formal evaluators shape the project, the relevance between practice and theory was kept in mind.

Design research seemed to be a good fit for this project. In this project, knowledge was derived from creation and user interaction. It is a possibility that future projects building on this research might use other methods, and one suggestion is to use natural science methods to either confirm or invalidate findings from this study. Much like automated checking tools and manual checking tools can complement each other, design research and natural science methods can as well, something that is suggested by Hevner in an interview (Winter, 2009).

Because this was a project with the intent to build an artifact, we still feel that design research was more appropriate than action research—the artifact is still too experimental to be used in a real-world context, and the feedback received has been from experts within the accessibility field.

6.2 Existing tools

One of the outcomes of this study beside this thesis and the artifact, is an overview of a selection of related tools given in Section 3.12. The main finding is that there is no tool that combines automated checking with user testing—neither open source nor proprietary, and not within any of the many fields of human–computer interaction included in this study.

Also, another finding was that most existing usability and user experience tools are interactive in some way or another, and they are proprietary services with paid subscription preventing a wider use in crowdsourcing. Most accessibility tools tests in an automated way against implementation of accessibility guidelines, and most of these tools are open source. All the surveyed open source tools have an public instance that can be accessed publicly.

Although we could not find any suitable tools to base the project upon, there was open source components to build upon. This will be discussed in the upcoming section about implementation.

6.3 Design

This section reflects on the Design activities—the choice of technologies, the design of the artifact, and the feedback process with the evaluators.

6.3.1 Choice of technologies

Most of the chosen components are widely used on the web today. When there was a problem, a quick search on the web helped us solve the issue. No custom components needed to be developed during the project. For each component, there were often several alternatives to choose from, which is an indication that the chosen ecosystem was suitable for the problem.

Bootstrap, the framework that the user interface is built upon, is used on a large number of websites today. As reported by one evaluator, links were not styled in a way that would be clear to everyone. A continuation of the UTT project is an excellent opportunity to help shape the major building blocks on the web by making them more accessible. This has the potential to make a difference for thousands of websites, and millions of users.

6.3.2 Designing the artifact

This section sums up what was learned from a technical point of view when designing the artifact.

The initial goal was to build a prototype. During the project the focus was shifted more to an architectural design base. This can be helpful for further work as there is a solid foundation to build upon. The shift of focus to architecture has caused a reduction of the number of features implemented.

Fidelity is a word often used to describe the distance between a prototype or mock-up to the final product. A lower fidelity prototype might be designed using pen and paper, and a slightly higher fidelity might mean that it is designed using a prototyping software tool that simulates interaction. For UTT, there is no difference between what was designed and what would be a real implementation.

Because the task involved picking technologies to build upon, it was most natural to build a high fidelity prototype using these technologies. We could have taken shortcuts by e.g. doing less refactoring, but this would have resulted in a less solid architecture.

Single-page application One idea that was attempted for the frontend was to use SPA architecture. This architecture seems to be effective for

designing this particular user interface—when answering a large number of tests, it is important that the user interface is responding fast when moving from one test to another, so that the user does not lose focus. Also, this architecture seems to be a good fit for crowdsourcing and scaling, as there is no need for the client to request a page for every test case. The web application is sent initially, and from that point the minimum amount of data is sent from the server.

Backbone.js seems to be a good solution for user interfaces with unique requirements. Using a non-opinionated framework like Backbone.js might be less than ideal for solving known problems such as building a web shop.

iframe Using iframe to embed a web page into UTT worked in many cases for integrating the two, but there are still cases where this technique is problematic. It was tried to merge the source code of two web pages into one, but this was less successful than using an iframe, as it was very difficult to create test page markup that would not be affected by the styling and scripts belonging to the page to become tested. It will be further discussed in future research about possible alternatives and/or workarounds.

As an alternative to using iframes, it was discussed if the test page header and the web page body should be split into two windows. An advantage with this solution is that one view cannot possibly interfere with the other. A couple of downsides is that it is very uncommon for a web application to be split into different windows, and the user needs to manage the windows themselves. This can be a challenge for many users, and especially on non-desktop devices.

Performance At this point, UTT takes between 1–5 seconds to load initially. Minifying JavaScript dependencies can help reduce the loading time, and this should be considered.

The backend Implementing the backend started after the user interface was in place. The responsibility of the backend is to provide the frontend with locale data such as translations, and it communicates with the eAccessibility Checker. The backend crashed a few times because of initial bugs, and it will be a good idea to have better monitoring in the future.

Feedback from users We received a lot of feedback from users, and especially near the end of the project. This section sums up what we learned.

When integrating a web page with the test page, it must be easy to distinguish between the two. This is currently done by animating the test page, and having a border between the two parts of the view, but in the future one should also consider checking that the color of UTT test page is different from the website.

Having well-designed input functionality is a determinable factor when asking a user to do repeating tasks. For keyboard support, there are many ways it can be done, and one solution is to let the user press ‘y,’ ‘n,’ and ‘u.’ Additional solutions were discussed, but none were implemented because of time constraints. For mouse support, it is crucial that the user do not have to move more than absolutely necessary. It was the assumption at first that the test page header was a form to become submitted. This required the user to choose an option, and then confirm it. It was later turned into something similar to toolbar with buttons, and this reduced the mouse movements significantly.

A comment from a evaluator is grouping similar questions within the same view makes it possible to compare the answers before submitting, as well as predicting new answers based on previous. More work is needed to see how this can be solved.

Color combination is a critical factor. At first, a light-on-dark color scheme was used, but later the default Bootstrap color scheme was chosen after requests from several evaluators. With the default Bootstrap color scheme, it was reported that links were hard to identify because they are too light, and also because they are not underlined.

Internationalization is important to reach a wide audience, but lack of it did not have too much impact for the feedback of this project.

6.4 Limitations

In this section some of the not yet implemented functionality is outlined.

6.4.1 Not collecting data

One important limitation of the current version of UTT, is that it does not store the collected data. For the prototype, it was prioritized to implement functionality that both would provide a base for the architecture while having a demonstrative effect.

A JSON data structure is already built during a test run. To reach the goal of storing the collected data, the data structure needs to become persisted. PostgreSQL is the suggested technology for data storage.

6.4.2 Supporting old browsers

Another limitation with the current version, is that it uses newer web technologies, and while many of the components in use have fallback mechanisms for functionality not supported by older browsers, we did not test UTT in older browsers. We suspect that IE 8 and IE 7 might be problematic. It is not planned to support Internet Explorer 6.

6.4.3 Using the result view of eAccessibility Checker

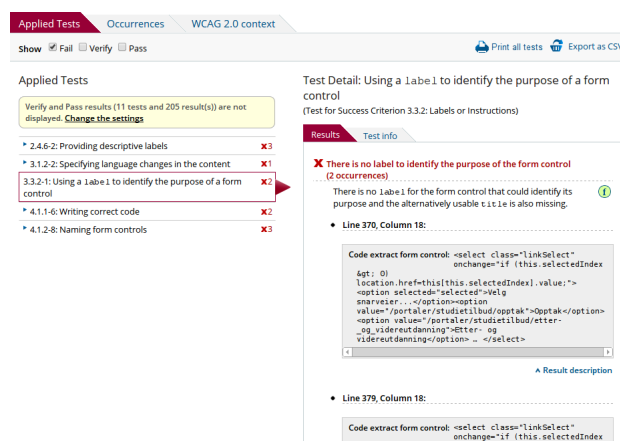


Figure 29: The result view of eAccessibility Checker

For the 0.3 release, it was planned to swap out the table on the result page with the result view of eAccessibility Checker (see Figure 29), as this has been developed iteratively over several years with feedback from Norwegian municipalities. The suggestion to use it came after choosing the technology stack for UTT. Because the frontend eAccessibility Checker is based on a different stack of technologies, and since it is integrated into the user interface, this might require some work.

6.5 Research questions

The project has been guided by several research questions as outlined in the introduction. The main question was:

How to design a user testing tool to enable more people to improve accessibility testing of websites?

This question is attempted to become answered not directly, but rather through the five questions that supports it.

6.5.1 Research question 1: How to design a tool that integrates with automated checker?

The naïve answer to this question is to create a tool that connects to an automated checker, passes it an URL, and receives test results. For UTT, we chose to integrate with eAccessibility Checker because of prior working experience with it, it belongs to an existing research project, and we have the possibility to extend it.

Technically, to solve this problem we need to design an export Application Programming Interface (API) at the automated checker that takes an URL as input, checks it, and gives a data structure back containing test results for tests that passes, fails, and tests that needs to become verified. UTT generates questions for the latter group.

There is already a CSV export, but it does not contain all data that we need. In addition, CSV is not as convenient to parse in a JavaScript application as JSON. So we have designed a JSON export functionality that is received and further processed by the UTT backend. The JSON export API can be used by others, but is currently not documented.

6.5.2 Research question 2: How can user tests enhance automated checker tests?

Clearly having a user verifying the results termed “to be verified” can give us a more complete check result. The automatic checker returns results for a given web page in three categories; tests that have passed, tests that have failed, and tests that needs to become verified by a human. The automated checker cannot determine if an alternative text is appropriate for an image⁵⁰. A goal is to continuously push the limit for what can be determined automatically, and user testing can contribute to improve an automated checker by e.g. reporting to it that a text looks like a placeholder text (inserted by a CMS). The idea is to contribute improving the

⁵⁰As a side note, the eAccessibility Checker currently has not implemented this test, but it is planned for inclusion in the near future.

data that heuristic tests are based on. Similarly, one might use existing data to predict answers to user tests.

6.5.3 Research question 3: How to design a user interface to best support the user doing the user testing?

We think that the tool must integrate the web page to become tested with the testing tool. It needs to be easy to learn for a wide group of people. Moreover, it has to be effective at letting the user accomplish repeating tasks. This is both done by having it perform fast when going between tests, and requiring as little input as possible.

The tool might need to adapt to various assistive technologies. For users of screen readers, it needs to show the element being tested in a context that is limited, and not the whole web page.

Respecting the privacy of the user is critical, and for the user interface it means giving the control to the user about what data to collect. The user interface should whenever possible store data regarding assistive technologies in use in the client, as this is sensitive data.

6.5.4 Research question 4: How to best track the user behavior without any code injection on the website to be studied, or software on the client side?

We have not implemented features for this in the prototype. However, both for tracking the user behavior, and for highlighting the element currently being tested, seem to require use of a proxy that serves a web page that is modified to contain a tracking script and style information that highlight the element. While this does inject code into a temporary copy of the page, the original page remains unchanged.

On the client side, the use of the proxy-iframe technique means that one does not require any additional browser extensions or browser technologies beyond HTML5.

6.5.5 Research question 5: What existing open source solutions can be helpful to solve the above questions?

The main finding from doing a review of existing tools is that there is no existing open source products to extend to satisfy the objectives of UTT. There are however open source components that are well-suited for UTT.

Since UTT is to a large extent tied to the browser, JavaScript technologies are used throughout its stack. The backend uses the Node.js platform, and the frontend uses the Bootstrap framework for the user interface, and the Backbone.js framework for handling interactivity.

In the future, a proxy-iframe solution might be implemented with either PhantomJS or CasperJS.

6.6 Suggested future research

This section suggests directions for future research that extends the UTT idea outlined during this research project. First, some possibilities are presented that could be implemented for upcoming research projects, and then long-term possibilities that lie further ahead are presented. All suggested possibilities are derived from unfinished issues, the objectives and requirements, ideas that have appeared along the way, and from user feedback.

6.6.1 Short-term

Some of the short-term items are issues that we did not have time to finish. Others are ideas that should not require large amount of work to implement.

Storing data A data structure is already built during a test run that store the answers of the user, and this is shown in the result view along with the automated tested results. This data needs to be stored in a database for further processing. We suggest to use PostgreSQL.

Implement additional to-be-verified tests Not all test results from the automated checker that needs to become verified by a human is currently implemented. What is needed is to formulate questions and answers for the remaining test results. This is a small task.

Integrate with eAccessibility Checker results view This is a slightly larger task if the goal is seamless integration, both because it is based on some other technologies, and because the result view integrated into frontend of the automated tool, i.e. it is not a component. Alternatively, the results view can be integrated using an iframe, but this is less elegant.

Covering more aspects of HCI Currently only accessibility tests are supported by UTT. Usability and user experience aspects should be covered in addition. For instance, one can measure how long it takes to accomplish a task on a website.

There are also completely different use cases that UTT used for, such as improving OCR detection algorithms by using a machine learning approach, or similarly, by parsing free text calendar dates.

Support websites—not only web pages Currently, UTT can check only a web page—or, one URL at the time. It should be able to check a website, either by allowing the user to enter a set of URLs, or by having a crawler or similar technology take a web page as input, and then automatically discover related pages.

Work needed to support websites without modifications It was reported from users, that the iframe technique did not work in all cases; it seems to have problems with web pages that redirects to other pages, and/or web pages that have a lot of interaction. Another issue that was not reported, but that can be problematic, is that the iframe has its own scrollbar that is separate from the UTT scrollbar. This might confuse some users. A third issue, is that the iframe presents the whole web page, which is inconvenient when using screen readers.

Highlight element currently being tested In the test page body, it should be highlighted what element is currently tested. This can be done if using a tool like Casper.JS, as it can inject CSS into the page. It can with little effort produce a screenshot. However, that means that it will not be presented for people with screen readers. What needs further work, is to figure out how to implement a proxy like Loop11 that downloads a page, modifies it, and serves it.

Better handling of many tests of same type Right now there is one test per view. When there are more occurrence of the same test, then it might be an idea to group them together. This allows for a side-by-side comparisons of answers, so that the user can check for consistency among his/her answers.

It was suggested in a feedback to have a pattern-detecting mechanism

for answers, suggesting answers for upcoming questions based on previous answers.

Other We received a lot of valuable feedback, and there are functionality that we feel belong into a last category of features that would be nice to have implemented. One such feature is having a export for test results. Another feature is linking tests up against guidelines for further reading.

A few tweaks is needed to make UTT load faster. UTT is currently not designed for mobile or tablet devices, but it would not require much to make it responsive.

Also, we received feedback related to the tests. Sometimes, there is a test for an element that is not visible. One specific example is “jump to content” links. These should be handled in a special way.

Crowdsourcing Once the tool has matured, it might be a possibility to use a recruiting service to get a large amount of users evaluate it.

6.6.2 Long-term

This is a list of possibilities of where the project can go in the future. Making sense of the data

Once data such as test answers or mouse activity has been collected from a large number of users, it needs to be made sense of to be of any value. Determining the reliability of collected data is both a science and art of its own, and there are possible interesting to to combine results provided by the users. If UTT is used to certify that a website is accessible, then one need to take into account aspects such as demographics of the participants, and that there is a variety of devices used for testing a web page.

One concern that needs to be addressed is test users gaming the system. One has to implement preventive mechanisms similar to web advertisement vendors refers to as false clicks.

Use browser DOM instead of source based DOM Today’s automated accessibility checkers bases their checks on the HTML source code. Since the Web 2.0 phenomenon, websites are becoming more dependent on browser technologies such as JavaScript and WebSocket for building its structure, and filling that structure with content. UTT is itself an example of a step

is this direction, as the meat of the application is not contained in the HTML file that is initially requested by the user agent of the person visiting the website, but rather loaded dynamically.

Since a HTML file does not always represent the content that is finally rendered on the web page, we cannot use the HTML as a source when checking if it is accessible. More work needs to be done in this area, and especially for automated checker tools.

User preferences It should be possible for users of UTT to change the text size, color scheme, and have it adjust for assistive technologies. When doing so, one need to keep privacy in mind, and prefer storing the user preferences in the client.

One use case that we became aware of during the project is a business that is interested in using UTT internally in a software development team/department. The business needs to restrict who are able to access the tool, and to only log anonymous usage by the software developers.

To do this effectively, an instance of UTT can become installed within the boundaries of the department, and this will offer the business full control over the collected data. They should be able to delete the collected data if they wish to.

Analytics user interface for studying data In the future, UTT might include a dedicated tool for making sense of the collected data, similar to the Google Analytics interface. There might be a way for them to suggest/report a specific pattern, and perhaps classify it as being a CMS-related issue. Some of this feedback functionality might become a part of the testing user interface.

Using collected data to improve automated checker The analysis of data collected by UTT can might lead to the automated checker becoming smarter. Once it is able to recognized a new pattern, it might make a test fail instantly that previously had to be verified by a human. One example of this is placeholders for alternative texts inserted by tools and/or CMS systems. Improving the automated checker might be done from a combination of machine learning, and from having data analysts study the data and recognize patterns.

6.7 Reflecting on the process

We will in this section reflect on the process of carrying out the research project. It is split into two parts—the development of the artifact, and the writing of the thesis.

6.7.1 Development of the artifact

There are many aspects that influence the process of a software development project. Some important ones are the problem itself, the people working on solving the problem—their past experience and their preferences. Sometimes a method is followed, other times not. Most projects are constrained by time. Implementing a large number of features in a short amount of time is sometimes possible, although it might affect quality negatively. Also, there are differences between open source projects and proprietary ones beside the license.

From a years of doing software development both for leisure and professionally, we have noticed people work in ways that differ greatly. Some choose to follow a method, and for those who do it usually varies how strictly the method is followed. Some developers start with a design on paper, while other like trying things out first, and then learning afterwards why it either worked or did not work as expected.

This is a greenfield project, which means that it does not build on an existing code base—and this influences design decisions quite a bit. Working as a single developer on a greenfield project has given a great deal of freedom, much like a painter starting with an empty canvas. Use of frameworks such as Bootstrap and Backbone.js for implementing the frontend provided some grid lines that simplified the decision making process somewhat. When implementing the API for eAccessibility Checker, the design decisions were more constrained the existing code base, but this was a small part of this project.

When having worked on projects in larger teams before, the author has used a slightly more structured approach, both with the method, and the surrounding tools. The design research method principles, along with a small subset of Scrum was used to guide the development. The basic tools consisted of Git as the version control system, and the GitHub issue tracker to keep track of milestones and issues. There was no heavyweight development environments such as Continuous Integration, and there was no specific development approach such as Test-Driven Development.

We have found the latter useful for more backend centric projects in the past, but less so for frontend centric projects where one need to interact with the application to see that it behaves correctly.

For the coding activity, the personal preference of the author is to let the fingers do the work, ask questions along the way, get new insight that might lead to new directions. The code has been refactored along the way to make it maintainable. It has been a process of trusting prior experience, following the intuition, and question assumptions when things have not been working. We do not think there is any right or wrong when doing a creative process—people are wired differently. Also, no two projects are alike. If one has a clear idea of what to create, then typing it right might be the way. Other times, one has to try a number of different things, which might lead to unexpected, but positive surprises.

For much of UTT, many of the architecture drawings came after the implementation. Also, the code started out unstructured, and became increasingly structured. The requirements changed along the way. We believe refactoring makes this possible. Very often, and especially for proprietary software projects, the code gets less maintainable over time. The solution is to refactor, and this is true both for writing code and writing prose—writing is rewriting.

6.7.2 Notes on writing of the thesis

“Write it, cut it, paste it, save it” —Technologic, Daft Punk

Writing of a thesis is a huge undertaking, and the experience has included feelings of joy, pain, and everything in between.

The process of producing the text evolved along the way, but one sequence of steps that seemed to work fine for the author was to draft in a paper notebook. This was done in various locations such as coffee shops and the university cafeteria. The drafts were transferred to a LibreOffice document, where it was further structured, and sent for review. The document was finalized in \TeX .

It was especially helpful to receive feedback from supervisors often.

6.8 Concerns regarding privacy

Privacy is a concern in every web application collecting data. While we do not currently log any data, we intend to collect data beyond answers

to tests. This needs to be done in a way that both respects the privacy of the user, and is according to laws in the countries where the tool is in use. While it is likely that the IP address will be logged, in addition to the user agent string, we do not intend to collect name or email of the user.

To be able to provide a user interface optimized for assistive technologies, we need to ask users what devices they are using and/or what disabilities that they have. One advantage of the SPA architecture used, is that it has the potential to adapt the user interface without ever sending this information to a server. This is one advantage of fat client architecture that is rarely mentioned.

Storing of mouse activities such as movements and clicks has been discussed earlier, and a couple of tools have been looked at. This is sensitive data, because the it might reveal that the user has motor impairments and/or that the user is an inexperienced computer user (e.g. double clicks where it is more appropriate to single click.) The solution is to clearly inform the user about what we collect, and it might be necessary for the user to decide what we should be able to collect.

To be able to serve the currently tested web page through a proxy, we need to store its URL. It is possible for the user to some extent surf the web within the iframe—and thus, we are logging sites that are visited. We find it unlikely that the user would want surf the web through the test page, but it is still a possibility that the user loses focus, and starts clicking around in the iframe.

One of the privacy concerns of logging of keystrokes is if a user by accident enters his/her username/password. Entering usernames and passwords is sometimes done by external tools that generate keystrokes⁵¹, and it is a possible that keystrokes can be logged if the tool is started accidentally in a non-password field. It should be noted that we only log keystrokes inside the iframe of the website being tested. It is not possible with the technique used, or any other in-browser techniques that we are aware of, to log keystrokes on an operating system level. What needs to be considered is how to handle testing of websites where the user needs to log in.

A future possibility is to use a web camera to detect and log eye movements. Face-detection is a feature that might be used creatively. Marko Dugonjić has with the use of face-detection created an experimental website using a new HTML5 API that uses the web camera and an algorithm

⁵¹Alexander has developed one such tool named mlt—or, minimal login typer—for simplifying logins. <https://github.com/alexanderte/mlt>

to determine how far away the user is from the computer display. The distance affects the size of the text,⁵² resulting in what appears for the user to be a constant size. The web browser (Chromium on Linux) did ask for user permission before the web page could access the web camera content.

To sum up, once UTT starts collecting data in the future, it needs to ask the user for permission about what is allowed to collect. Since the prototype does not collect anything, it has no such options.

⁵²<http://webdesign.maratz.com/lab/responsivetytography/>

7 Conclusion

The master's project represents the first steps towards enabling accessibility testing of websites using a combination of automated testing and user testing with a view to a crowdsourcing approach. This section concludes the research.

How to enable more people to improve accessibility testing is the main question addressed. Based on our proposed solution the thesis also deals with how to integrate an automated checker with user testing, how user tests can enhance automated checker tests, how to design the user interface, and how remove any obstacles preventing larger numbers of people to contribute for crowdsourcing the tests of web sites.

An open source prototype based on an iframe technique was built to demonstrate a viable path of development. The user interface integrates with the web page to be tested, and it satisfies several usability criteria. The solution covers design of an API for the automatic checker for receiving test results, a control flow mechanism, and a user interface iteratively refined involving evaluators.

The user interface is implemented using a SPA architecture, a fat client architecture for the web. This will reduce the demand for bandwidth and server capacity and should therefore be suitable for crowdsourcing. The user interface needs to take privacy into account by giving the user control about what data to collect. Currently this is not a problem since the prototype does not store any data. To initiate the development of a user testing tool, this thesis has focused on a solid architecture along with features that serve a demonstrative purpose. The proposed architecture and the open source approach is designed to facilitate further development.

Design Research was the chosen research method, where knowledge was derived from creation of the prototype and interaction with the users. The research problem is based on a practical need for better testing tools. User testing of the tool itself uncovered that efficient means of user entry is crucial, and that the tool needs to adjust for a variety of devices and assistive technologies.

Future research can include storage of collected data, explore approaches to deal with privacy and logging of data, data analysis for quality assurance, further investigation of the proxy-iframe technique, and integration with the eAccessibility Checker result view.

8 Glossary

Abbreviation	Definition
CSS	Cascading Style Sheets
DOM	Document Object Model
HCI	Human-computer interaction
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
SPA	Single-page application
UI	User interface
URL	Uniform Resource Locator
UTT	User Testing Tool
W ₃ C	The World Wide Web Consortium
WAI-ARIA	Web Accessibility Initiative—Accessible Rich Internet Applications
WCAG	Web Content Accessibility Guidelines
XHR	XMLHttpRequest

Table 9: Glossary

References

- Abran, A., Khelifi, A., Suryan, W., & Seffah, A. (2003). Consolidating the ISO usability models. In *Proceedings of 11th international software quality management conference* (p. 23–25).
- The Accessibility of WAI-ARIA*. (n.d.). <http://alistapart.com/article/the-accessibility-of-wai-aria>. Retrieved 2013-06-02, from <http://alistapart.com/article/the-accessibility-of-wai-aria>
- Addressing accessibility | fix the web*. (n.d.). Retrieved 2013-06-07, from <http://www.fixtheweb.net/>
- Agrawal, A., Boese, M., & Sarker, S. (2010, August). A review of the HCI literature in IS: the missing links of computer-mediated communication, culture, and interaction. *AMCIS 2010 Proceedings*. Retrieved from <http://aisel.aisnet.org/amcis2010/523>
- Bannan-Ritland, B. (2003). The role of design in research: The integrative learning design framework. *Educational Researcher*, 32(1), 21–24. Retrieved 2013-06-02, from <http://edr.sagepub.com/content/32/1/21.short>
- Bannan-Ritland, B., & Baek, J. Y. (2008). Investigating the act of design in design research: The road taken. *Handbook of design research methods in education: Innovations in science, technology, mathematics and engineering*. Mahway, NJ: Taylor & Francis.
- Bold and italic formatting | AccessAbility*. (n.d.). Retrieved 2013-06-07, from <http://accessibility.psu.edu/boldface>
- Buchanan, R. (2001). Design research and the new learning. *Design issues*, 17(4), 3–23.
- Buhrmester, M., Kwang, T., & Gosling, S. D. (2011). Amazon's mechanical turk a new source of inexpensive, yet high-quality, data? *Perspectives on Psychological Science*, 6(1), 3–5.
- Building the next SoundCloud - SoundCloud backstage SoundCloud backstage*. (n.d.). Retrieved 2013-06-07, from <http://backstage.soundcloud.com/2012/06/building-the-next-soundcloud/>
- Caldwell, B., Cooper, M., Reid, L. G., & Vanderheiden, G. (2008). Web Content Accessibility Guidelines (WCAG) 2.0. *World Wide Web Consortium (W3C)*.
- Chiew, T. K., & Salim, S. S. (2003). Webuse: Website usability evaluation

- tool. *Malaysian Journal of Computer Science*, 16(1), 47–57.
- Cole, R., Puro, S., Rossi, M., & Sein, M. K. (2005). Being proactive: where action research meets design research. In *Proceedings of the twenty-sixth international conference on information systems* (p. 325–336). Retrieved 2013-06-02, from http://www.researchgate.net/publication/221599527_Being_Proactive_Where_Action
- Commission, E. (2007). *The social situation in the european union 2005-2006: The balance between generations in an ageing europe*. Office for Official Publications of the European Communities.
- Commission, I. O. f. S. E. (2001). ISO/IEC 9126-1 standard, software engineering, product quality, part 1: Quality model. *Author, Geneva*.
- Convention on the rights of persons with disabilities*. (n.d.). <http://www.un.org/disabilities/default.asp?id=150>. Retrieved 2013-06-02, from <http://www.un.org/disabilities/default.asp?id=150>
- Crockford, D. (2008). *JavaScript: the good parts*. Yahoo Press. Retrieved 2013-06-02, from <http://www.google.com/books?hl=no&lr=&id=PXa2bby0oQ0C&oi=fnd&pg=PR7&dq=javascript>
- Crowdsourcing: Crowdsourcing: A definition*. (n.d.). http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html. Retrieved 2013-06-02, from http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html
- Crowdsourcing - definition and more from the free merriam-webster dictionary*. (n.d.). <http://www.merriam-webster.com/dictionary/crowdsourcing>. Retrieved 2013-06-03, from <http://www.merriam-webster.com/dictionary/crowdsourcing>
- Designing with grid-based approach | smashing magazine*. (n.d.). <http://www.smashingmagazine.com/2007/04/14/designing-with-grid-based-approach/>. Retrieved 2013-06-02, from <http://www.smashingmagazine.com/2007/04/14/designing-with-grid-based-approach/>
- Digital agenda for europe - european commission*. (n.d.). Retrieved 2013-06-07, from <http://ec.europa.eu/digital-agenda/en/news/proposal-directive-european-parliament>

- Edelson, D. C. (2002). Design research: What we learn when we engage in design. *The Journal of the Learning sciences*, 11(1), 105–121.
- Ericsson teams up with dj and producer avicii to try and crowdsource the world's first hit song.* (n.d.). <http://thenextweb.com/media/2013/01/08/ericsson-teams-up-with-dj-and-producer-avicii-to-try-and-crowdsource-the-worlds-first-hit-song/>. Retrieved 2013-06-02, from <http://thenextweb.com/media/2013/01/08/ericsson-teams-up-with-dj-and-producer-a>
- Fragment identifier - wikipedia, the free encyclopedia.* (n.d.). Retrieved 2013-06-07, from http://en.wikipedia.org/wiki/Fragment_identifier
- GNU general public license v2 (GPL-2) explained in plain english.* (n.d.). Retrieved 2013-06-07, from [http://www.tldrlegal.com/license/gnu-general-public-license-v2-\(gpl-2\)](http://www.tldrlegal.com/license/gnu-general-public-license-v2-(gpl-2))
- Google - definition and more from the free merriam-webster dictionary.* (n.d.). <http://www.merriam-webster.com/dictionary/google>. Retrieved 2013-06-04, from <http://www.merriam-webster.com/dictionary/google>
- gotreehouse. (2012a, March). *Web accessibility tutorial - vision pt. 1.* Retrieved 2013-06-02, from <http://www.youtube.com/watch?v=DvkKN-sXleE>
- gotreehouse. (2012b, March). *Why learn web accessibility?* Retrieved 2013-06-02, from <http://www.youtube.com/watch?v=GNRYSdselkQ>
- Heading tags (h1, h2, h3, p) in HTML / AccessAbility.* (n.d.). Retrieved 2013-06-07, from <http://accessibility.psu.edu/headingshtml>
- Hevner, A. R. (2007). The three cycle view of design science research. *Scandinavian Journal of Information Systems*, 19(2), 87. Retrieved 2013-02-17, from <http://community.mis.temple.edu/seminars/files/2009/10/Hevner-SJIS.pdf>
- Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 28(1), 75–105. Retrieved 2013-06-02, from <http://dl.acm.org/citation.cfm?id=2017217>
- How to: Detect backbone memory leaks | andrew henderson.* (n.d.). Retrieved 2013-06-07, from

<http://andrewhenderson.me/tutorial/how-to-detect-backbone-memory-leaks/>

- HTML5 rocks - a resource for open web HTML5 developers.* (n.d.). Retrieved 2013-06-07, from <http://www.html5rocks.com/en/>
- Introduction to web accessibility.* (n.d.). Retrieved 2013-06-07, from <http://www.w3.org/WAI/intro/accessibility.php>
- ISO, S. (2004). 6385. 2004. *Ergonomic principles in the design of work systems (ISO 6385: 2004)*, 1.
- ISO, W. (1998). 9241-11. ergonomic requirements for office work with visual display terminals (VDTs). *Guidance on usability.*
- Iwarsson, S., & Ståhl, A. (2003). Accessibility, usability and universal design-positioning and definition of concepts describing person-environment relationships. *Disability & Rehabilitation*, 25(2), 57-66. Retrieved 2013-02-17, from <http://informahealthcare.com/doi/abs/10.1080/dre.25.2.57.66>
- JavaScript garden.* (n.d.). <http://bonsaiden.github.io/JavaScript-Garden/#types.equality>. Retrieved 2013-06-02, from <http://bonsaiden.github.io/JavaScript-Garden/#types.equality>
- Kittur, A., Chi, E., Pendleton, B. A., Suh, B., & Mytkowicz, T. (2007). Power of the few vs. wisdom of the crowd: Wikipedia and the rise of the bourgeoisie. *World Wide Web*, 1(2), 19.
- Leventhal, L. M., & Barnes, J. A. (2008). *Usability engineering: process, products, and examples.* Pearson/Prentice Hall. Retrieved 2013-02-17, from <http://www.lavoisier.fr/livre/notice.asp?id=OR2WRAAL00SOWZ>
- LOV 2008-06-20 nr 42: Lov om forbud mot diskriminering på grunn av nedsatt funksjonsevne (diskriminerings- og tilgjengelighetsloven).* (n.d.). <http://www.lovdatab.no/all/hl-20080620-042.html>. Retrieved 2013-06-02, from <http://www.lovdatab.no/all/hl-20080620-042.html>
- Mankoff, J., Fait, H., & Tran, T. (2005). Is your web page accessible?: a comparative study of methods for assessing web page accessibility for the blind. In *Proceedings of the SIGCHI conference on human factors in computing systems* (p. 41-50). Retrieved 2013-02-17, from <http://dl.acm.org/citation.cfm?id=1054979>

- MOTHER EFFING TOOL CONFUSER*. (n.d.). Retrieved 2013-06-07, from <http://mothereffingtoolconfuser.com/>
- Muller, M. J., & Kuhn, S. (1993). Participatory design. *Communications of the ACM*, 36(6), 24--28.
- Native XMLHttpRequest object - IEBlog - site home - MSDN blogs*. (n.d.). Retrieved 2013-06-07, from <http://blogs.msdn.com/b/ie/archive/2006/01/23/516393.aspx>
- Nielsen, J., & Hackos, J. T. (1993). *Usability engineering* (Vol. 125184069). Academic press San Diego.
- Nietzio, A., Eibegger, M., Goodwin, M., & Snaprud, M. (2012). Following the WCAG 2.0 techniques: experiences from designing a WCAG 2.0 checking tool. *Computers Helping People with Special Needs*, 417-424. Retrieved 2013-02-17, from <http://www.springerlink.com/index/1V262422218WNW14.pdf>
- Orlikowski, W. J., & Iacono, C. S. (2001). Research commentary: Desperately seeking the "it" in it research—a call to theorizing the it artifact. *Information systems research*, 12(2), 121-134. Retrieved 2013-02-17, from <http://infosys.highwire.org/content/12/2/121.short>
- Owen, C. L. (1998). Design research: building the knowledge base. *Design Studies*, 19(1), 9-20.
- Papas, N., O'Keefe, R. M., & Seltsikas, P. (2011). The action research vs design science debate: reflections from an intervention in eGovernment. *European Journal of Information Systems*, 21(2), 147-159. Retrieved 2013-06-02, from <http://www.palgrave-journals.com/ejis/journal/v21/n2/abs/ejis201150a.html>
- PEP 20 - the zen of python*. (n.d.). <http://www.python.org/dev/peps/pep-0020/>. Retrieved 2013-06-02, from <http://www.python.org/dev/peps/pep-0020/>
- PHP: a fractal of bad design - fuzzy notepad*. (n.d.). <http://me.veekun.com/blog/2012/04/09/php-a-fractal-of-bad-design/>. Retrieved 2013-06-03, from <http://me.veekun.com/blog/2012/04/09/php-a-fractal-of-bad-design/>
- Pierre Kircher. (2012, April). *Gary bernhardt WAT*. Retrieved 2013-06-03, from <http://www.youtube.com/watch?v=kXEgk1Hdze0>
- The principles of universal design at center for universal de-*

- sign.* (n.d.). <http://www.ncsu.edu/project/design-projects/udi/center-for-universal-design/the-principles-of-universal-design/>. Retrieved 2013-06-02, from <http://www.ncsu.edu/project/design-projects/udi/center-for-universal-design/the-principles-of-universal-design/>.
- Redefining web designers, web developers, and web hybrids for the modern market - tristan denyer.* (n.d.). Retrieved 2013-06-07, from <http://tristandenyer.com/redefining-web-designers-web-developers-and-web-hybrids/>.
- Responsive web design - an a list apart article.* (n.d.). Retrieved 2013-06-07, from <http://alistapart.com/article/responsive-web-design>
- Rubery, M. (2011). *Audiobooks, literature, and sound studies*. Taylor & Francis.
- Semantic versioning 2.0.0-rc.2.* (n.d.). <http://semver.org/>. Retrieved 2013-06-02, from <http://semver.org/>
- The semantic web: Scientific american.* (n.d.). Retrieved 2013-06-07, from <http://www.scientificamerican.com/article.cfm?id=the-semantic-web>
- Single-page application - wikipedia, the free encyclopedia.* (n.d.). Retrieved 2013-06-07, from http://en.wikipedia.org/wiki/Single-page_application
- Spinuzzi, C. (2005). The methodology of participatory design. *Technical Communication*, 52(2), 163--174.
- Standards - W3C.* (n.d.). Retrieved 2013-06-07, from <http://www.w3.org/standards/>
- Takeda, H., Veerkamp, P., & Yoshikawa, H. (1990). Modeling design process. *AI magazine*, 11(4), 37. Retrieved 2013-06-02, from <http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/855>
- Tiedtke, T., Märtin, C., & Gerth, N. (2002). AWUSA—A tool for automated website usability analysis. In *PreProceedings of the 9th int. workshop DSV-IS* (p. 251-266).
- Tim berners-lee on the next web | video on TED.com.* (n.d.). Retrieved 2013-06-07, from http://www.ted.com/talks/tim_berniers_lee_on_the_next_web.html

- User testing tool forprosjekt - it funk.* (2012).
http://www.itfunk.org/docs/prosjekter/User_Testing_Tool.htm.
 Retrieved 2013-06-02, from http://www.itfunk.org/docs/prosjekter/User_Testing_Tool
- Vaishnavi, V., & Kuechler, B. (2004). *Design research in information systems.* <http://desrist.org/design-research-in-information-systems/>. Retrieved 2013-06-02, from <http://desrist.org/design-research-in-information-systems/>
- Von Ahn, L. (2006). Games with a purpose. *Computer*, 39(6), 92--94.
- WAI history.* (n.d.). Retrieved 2013-06-07, from <http://www.w3.org/WAI/history>
- WebAIM: creating accessible frames.* (n.d.). Retrieved 2013-06-07, from <http://webaim.org/techniques/frames/>
- Web style sheets.* (n.d.). Retrieved 2013-06-07, from <http://www.w3.org/Style/#dynamic>
- Wegge, K., & Zimmermann, D. (2007). Accessibility, usability, safety, ergonomics: concepts, models, and differences. *Universal Access in Human Computer Interaction. Coping with Diversity*, 294-301. Retrieved 2013-02-17, from <http://www.springerlink.com/index/82jlr1j18j655g83.pdf>
- What are the real-world strengths and weaknesses of the many frameworks based on backbone.js? - stack overflow.* (n.d.). <http://stackoverflow.com/questions/10847852/what-are-the-real-world-strengths-and-weaknesses-of-the-many-frameworks-based-on>. Retrieved 2013-06-02, from <http://stackoverflow.com/questions/10847852/what-are-the-real-world-strengths-a>
- What the rails security issue means for your startup | kalzumeus software.* (n.d.). <http://www.kalzumeus.com/2013/01/31/what-the-rails-security-issue-means-for-your-startup/>. Retrieved 2013-06-02, from <http://www.kalzumeus.com/2013/01/31/what-the-rails-security-issue-means-for-you>
- Winter, R. (2009). Interview with Alan R. Hevner on "design science". *Business & Information Systems Engineering*, 1(1), 126-129.
- Yes, actually, it may be you one day.* (2012). Retrieved 2013-06-02, from <http://www.karlgroves.com/2012/11/16/yes-actually-it-may-be-you-one-day/>

- Zhang, P., & Li, N. (2005). The intellectual development of Human-Computer interaction research: A critical assessment of the MIS literature (1990–2002). *Journal of the Association for Information Systems*, 6(11), 227–292.
- Zimmerman, J., Forlizzi, J., & Evenson, S. (2007). Research through design as a method for interaction design research in HCI. In *Proceedings of the SIGCHI conference on human factors in computing systems* (p. 493–502).