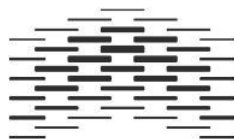


MASTER THESIS
in
Universal Design of ICT
May 2015

Building accessible Rich Internet Applications

Linn Steen-Hansen

Department of Computer Science
Faculty of Technology, Art and Design



OSLO AND AKERSHUS
UNIVERSITY COLLEGE
OF APPLIED SCIENCES

Preface

May 24th 2015

The purpose of this master thesis report is to document my 60p master project concerning how to create accessible *Rich Internet Applications* (RIA). This project came about when discussing possible master projects in cooperation with Enonic¹ concerning their new JavaScript based *Content Management System* (CMS), *Enonic Experience* (XP) with CTO Thomas Sigdestad and senior consultant Rune Forberg. It became clear that instead of doing a project directly linked to *one* system, it might be more fruitful to do a general project about RIAs, JavaScript and accessibility concerns, because so many systems are made with this technology. Interactive and dynamic applications built in client-side languages like JavaScript seems to be one of the major current challenges for accessibility. Although this is true, during the course of this project some important social issues and troubles with accessibility guidelines have also been uncovered. The result of this project is the second version of a set of guidelines aimed at meeting these challenges and smoothing the process of creating accessible RIAs.

Linn Steen-Hansen
Master student

¹ <https://enonic.com/>

Acknowledgements

May 24th 2015

There are several people I would like to thank whose help and support has made this project possible. First of all, I thank the five participants who gave so generously of their time, experience and knowledge when evaluating the first version of the guidelines. Secondly I thank my supervisor, Siri Fagernes, whose invaluable help has enabled me to present this large project in a way that lets others get an understanding of what it consists of and how it was conducted. I also want to thank Rune Forberg, firstly for being a part of sculpturing the starting point of this project, secondly for using his immense programming skills to help me find and create useful examples illustrating the guidelines, and thirdly for helping me set up the website where the guidelines are available, using *Enonic XP*. Lastly I would like to thank Thomas Sigdestad for his role as initiator for this project.

Linn Steen-Hansen
Master student

Summary

During this project a set of guidelines aimed at smoothing the process of making accessible *Rich Internet Applications* (RIA) has been created, evaluated and updated. This was done by studying literature about issues regarding RIA accessibility and suggestions for solutions to these problems. The solutions were processed and made into guidelines. It has become clear that not all issues can be solved with technical remedies. Two main themes evolved: *technology* and *process*. The *technology oriented* solutions address what technical features can be used to enhance accessibility. They address problems users face dealing with dynamic and interactive user interfaces. The *process oriented* solutions address what is important to think of during the process of web development and creation of accessible applications. They deal with issues like the confusions and uncertainties that exist within development teams when working with accessibility. One of the largest challenges of this project has been making the guidelines concrete enough to be helpful, and balancing the level of concreteness with usefulness across projects and technology.

The guidelines were evaluated by five participants working with web development and accessibility. Some attention was also given to what makes guidelines manageable to developers. It appears there *are* some definite steps one can take to make guidelines more usable. *These* guidelines were found to be useful, reliable and more manageable than existing guidelines. It was also believed they could contribute to build accessibility competence. In addition, many suggestions for changes for improvement were made. As a response to these findings several adjustments have been done, and a second version of the guidelines is presented as the result of this project. The second version of the guidelines is presented in *chapter 10*, but is also available at <http://accessibilityagent.no/guidelines>. Proposals for different ways to continue work with these guidelines in the future have been suggested. This projects' most important contribution to the field of web accessibility research are strong indications of a need for *process oriented* accessibility guidelines. A set of *process oriented* guidelines has been commenced and suggestions for continued work have been presented.

Content

Preface.....	2
Acknowledgements.....	3
Summary.....	4
Content.....	5
List of figures.....	10
List of tables.....	10
1. Introduction.....	11
1.1 Research questions.....	11
1.2 Goals and expected outcomes.....	12
1.3 Introduction to methodology.....	12
1.3.1 Studying accessibility issues and solutions.....	12
1.3.2 Creating guidelines.....	12
1.3.3 Evaluating and updating the guidelines.....	13
1.4 Outline of the master thesis.....	13
2. Background.....	15
2.1 The medical and social model of disability.....	15
2.2 The Gap model.....	15
2.3 What is web accessibility?.....	16
2.4 Assistive technology.....	16
2.5 Legislation.....	20
2.6 Standards and guidelines.....	20
2.6.1 WCAG 2.0.....	21
2.6.2 WAI-ARIA.....	21
3. The technologies of Rich Internet Applications.....	22
3.1 HTML.....	22
3.2 CSS.....	22
3.3 JavaScript.....	23
4. Methodology.....	24
4.1 Literature study.....	24
4.2 Creating guidelines.....	24
4.3 Evaluation of the guidelines.....	25
4.3.1 Interviews.....	26
4.3.2 Interview guide.....	27
4.3.3 Participants.....	28
4.4 Updating the guidelines.....	29
4.5 Limitations.....	29
5. Accessibility issues.....	30
5.1 Social issues.....	30
5.2 Tool issues.....	30
5.3 RIA issues.....	31
5.3.1 Assistive technology and Web 2.0.....	31
5.3.2 Updates not being detected and causing confusion.....	32
5.3.3 Non-existing semantics.....	32
5.3.4 Problems with keyboard navigation and access.....	32
5.3.5 Standard violations and errors.....	33
5.3.6 Pop-up windows.....	33
5.3.7 Over-engineered interfaces.....	34
6. Solutions.....	35
6.1 Have accessibility expertise on the team.....	35
6.2 Introduce accessibility from the beginning.....	35
6.3 Test accessibility at key stages.....	35

6.4 Follow existing design principles.....	37
6.5 Use WAI-ARIA mark-up	37
6.6 Follow and validate the HTML5 standard	38
6.7 Combine WAI-ARIA and HTML5 mark-up	39
6.8 Use Progressive enhancement.....	39
6.9 Use JavaScript unobtrusively.....	40
6.10 Make sure methods are device independent	41
6.11 Use accessible modal windows instead of pop-ups.....	41
6.12 Use technologies that facilitates accessibility.....	42
6.12.1 Toolkits and frameworks	42
6.12.2 Web components.....	42
6.13 Result of literature study	44
7. Findings: Evaluation of guidelines	45
7.1 Review of the answers to questions in the interview guide	45
7.1.1 Did the participants understand the guidelines?	46
7.1.2 Did the participants learn something?	46
7.1.3 Are the guidelines concrete enough?	46
7.1.4 Are the guidelines reliable?	47
7.1.5 Are the guidelines useful in the participants work?	47
7.1.6 Can participants use the guidelines in future projects?	47
7.1.7 How are the guidelines different from existing guidelines?.....	48
7.2 Positive feedback	49
7.2.1 Appreciation of the process oriented guidelines.....	49
7.2.2 Positive comments on the technical guidelines	49
7.3 Negative feedback and suggestions for changes	50
7.3.1 Introduce the guidelines and clarify scope.....	50
7.3.2 Adjust level of detail	50
7.3.3 Provide explanations of how the guidelines benefit accessibility	51
7.3.4 Link the guidelines to similar existing guidelines.....	51
7.3.5 Prioritize and estimate time use on the guidelines	51
7.3.6 Provide more examples	51
7.3.7 Change the main title.....	52
7.4 Suggestions for changes to process oriented guidelines	52
7.4.1 Point out that accessibility is interdisciplinary	52
7.4.2 Mention that accessibility should be a part of the specification.....	52
7.4.3 Elaborate on optimizing search functionality.....	53
7.4.4 Revise guideline on introducing accessibility from the start.....	53
7.4.5 Suggestions for changes to guideline about accessibility testing.....	53
7.4.6 Comments on simulation.....	54
7.4.7 Comments on testing with automatic tools	54
7.4.8 Suggestions for user testing.....	54
7.5 Suggestions for changes to technology oriented guidelines	54
7.5.1 Mention that WAI-ARIA is no substitute for good code	54
7.5.2 Highlight the exceptional qualities of WAI-ARIA	55
7.5.3 Comment problems with WAI-ARIA	55
7.5.4 Recommend following standards in general	55
7.5.5 Mention the most common HTML5 input types	55
7.5.6 Be critical towards <code><section></code> and <code><article></code> and drop recommending <code><canvas></code>	56
7.5.7 Mention issues with HTML5 element <code><nav></code> and WAI-ARIA role <code>navigation</code>	56
7.5.8 Explain what Progressive enhancement entails for accessibility	56
7.5.9 Clarify that HTML provides basic functionality in addition to content.....	56
7.5.10 Recommend using JavaScript to hide content	57
7.5.11 Progressive enhancement in CSS is somewhat outdated.....	57
7.5.12 Simplify language about Unobtrusive JavaScript.....	58

7.5.13 Clarify how Unobtrusive JavaScript benefits accessibility	58
7.5.14 Do not <i>not</i> expect JavaScript to be available.....	58
7.5.15 Do or do not to use pure JavaScript in coding examples?.....	59
7.5.16 Use a common example and reuse it throughout the guidelines	59
7.5.17 Remove tidy coding	59
7.5.18 Elaborate on device independent methods	60
7.5.19 Do not recommend frameworks that are basically not in use	60
7.5.20 Do not recommend specific frameworks at all.....	61
7.5.21 Clarify accessibility benefits of using Web components	61
7.6 Suggestions for additional guidelines	61
7.6.1 Communicate accessibility across the team.....	61
7.6.2 Where it is necessary to consider accessibility?.....	62
7.6.3 Log accessible script modules.....	62
7.6.4 Pay attention to placement of focus	62
7.6.5 Make data available in different ways.....	62
7.6.6 Merge Web components and framework proposals into one guideline	63
7.7 Suggestions on how to present the guidelines	63
7.7.1 Check-list vs. step by step guide	63
7.7.2 Presentation of testing tools	64
7.7.3 Tagging the guidelines	64
7.7.4 More examples and illustrations	64
7.7.5 Prioritised order of guidelines	64
7.7.6 Tool for using the guidelines actively within a project.....	64
8. Implications and discussion of findings	66
8.1 Usefulness of the guidelines	66
8.2 Reliability of guidelines	67
8.3 Comparison to existing guidelines	68
8.4 Explanations of how the guidelines benefit accessibility.....	69
8.5 Level of detail	70
8.6 Prioritising and estimating guidelines	71
8.7 Including others areas of web development	71
8.8 Are the guidelines relevant in any development process?.....	72
9. Alternation of guidelines	74
9.1 Changes related to the guidelines as a whole.....	74
9.2 Accessibility from the start.....	74
9.3 Accessibility testing	74
9.4 WAI-ARIA.....	75
9.5 HTML5	75
9.6 Progressive enhancement.....	75
9.7 Unobtrusive JavaScript.....	75
9.8 Device independent methods.....	75
9.9 Frameworks.....	75
9.10 Web components.....	76
9.11 Additional guidelines.....	76
9.12 Changes made to presentation	76
10. Results: Guidelines updated version	77
10.1 Introduction to guidelines.....	77
10.2 Have accessibility expertise on the team.....	78
10.3 Introduce accessibility from the beginning.....	78
10.3.1 Accessibility in the specification	78
10.3.2 Planning	78
10.4 Communicate accessibility within the team	78
10.5 Follow existing design principles.....	80
10.6 Test accessibility at key stages.....	81

10.6.1 Automatic tools.....	81
10.6.2 Expert based testing	82
10.6.3 Simulation	83
10.6.4 User testing.....	83
10.6.5 Log all accessible script modules	84
10.7 Use WAI-ARIA mark-up	84
10.7.1 WAI-ARIA roles.....	84
10.7.2 Document landmark roles	85
10.7.3 States and properties.....	86
10.7.4 Live regions	87
10.7.5 Tabindex.....	88
10.8 Follow and validate the HTML5 standard	90
10.9 Combine WAI-ARIA and HTML5 mark-up	91
10.10 Use Progressive enhancement.....	92
10.10.1 The core of Progressive enhancement	92
10.10.2 The principles of Progressive enhancement.....	92
10.10.3 The process of Progressive enhancement	93
10.11 Use Unobtrusive JavaScript.....	93
10.11.1 Do not make any assumptions.....	94
10.11.2 Find your hooks and relationships.....	94
10.11.3 Use CSS to traverse the DOM	94
10.11.4 Understand users and browsers.....	96
10.11.5 Understand events.....	96
10.11.6 Play well with others.....	99
10.11.7 Work for the next developer	104
10.12 Make sure methods are device independent	105
10.13 Use accessible modal windows instead of pop-ups.....	106
10.14 Use technology that facilitates accessibility	108
10.14.1 Libraries and frameworks	108
10.14.2 Web components.....	109
11. Conclusions.....	111
11.1 What problems exist with RIA accessibility?.....	111
11.2 What can be done to avoid RIA accessibility problems?	111
11.3 What makes guidelines manageable for developers?	112
12. Contribution to research and future work	114
12.1 What to do before the next round of interviews?.....	114
12.2 What to study before the next round of interviews?	114
12.3 What needs further studies?.....	114
12.3.1 Reliability of guidelines and prioritization	114
12.3.2 Further development of the process oriented guidelines.....	115
12.3.3 Guidelines related to other aspects of web development	115
12.3.4 Guidelines' relevance in different development processes	116
12.3.5 Time estimation for each guideline	116
12.3.6 Number of users with disabilities for accessibility testing	116
12.3.7 Further development of website where the guidelines are presented	116
12.3.8 Presenting the guidelines for active internal use	116
12.4 Most important contribution to web accessibility research.....	117
13. Reference list.....	118
Appendix 1: Guidelines 1 st version	125
1. Process oriented guidelines.....	125
1.1 Have accessibility expertise on the team.....	125
1.2 Implement accessibility from the beginning of a project	125
1.3 Test accessibility at key stages	126
1.3.1 Automatic tools.....	126

1.3.2 Expert based testing	127
1.3.3 Simulation	127
1.3.4 User testing	128
1.4 Follow existing design principles.....	128
2. Technology oriented guidelines	129
2.1 Follow and validate the HTML5 standard	129
2.2 Apply Progressive enhancement.....	129
2.2.1 Separate your stylesheets.....	130
2.3 Apply Unobtrusive JavaScript.....	131
2.3.1 Do not make any assumptions.....	131
2.3.2 Find your hooks and relationships.....	132
2.3.3 Leave traversing to the experts	132
2.3.4 Understand browsers and users	133
2.3.5 Understand Events.....	133
2.3.6 Play well with others.....	134
2.3.7 Work for the next developer	138
2.4 Apply tidy coding.....	138
2.5 Make sure methods are independent of input device.....	139
2.6 Apply WAI-ARIA mark-up	139
2.6.1 Use frameworks and libraries with built-in WAI-ARIA support	139
2.6.2 WAI-ARIA roles.....	139
2.6.3 Document landmark roles	139
2.6.4 States and properties.....	141
2.6.5 Live regions	141
2.6.6 Tabindex.....	143
2.7 Create or apply accessible Web components	143
Appendix 2: Screen shots - digital 1 st version of guidelines	145
Appendix 3: Screen shots - digital 2 nd version of guidelines	147

List of figures

Figure 1: The Gap model	16
Figure 2: Braille keyboard.....	17
Figure 3: Switch	18
Figure 4: Scanning software	18
Figure 5: Foot Mouse.....	19
Figure 6: Head mouse.....	19
Figure 7: Video of blind person using computer	79
Figure 8: Video of blind person using iPhone 4S.....	79
Figure 9: Video of scanning software	80
Figure 10: Video of person using computer with eye-scanning.....	80
Figure 11: Touch keyboard and input field with input type="number"	90
Figure 12: Combination of WAI-ARIA and HTML5 elements.....	91
Figure 13: W3C event model - Event capturing.....	98
Figure 14: W3C event model - Event bubbling.....	98
Figure 15: W3C event model - Combination of event capturing and event bubbling	98

List of tables

Table 1: Participant background.....	45
Table 2: Summary of findings from interview guide	46
Table 3: Automatic testing tools and their functionalities	82
Table 4: HTML5 elements and WAI-ARIA equivalents	92

1. Introduction

A *Rich Internet Application* (RIA) is a web application designed to deliver the same features and functions normally associated with desktop applications. A RIA normally runs inside a browser and usually does not require software installation on the client side to work (Braga, Jeferson Pezzuto Damaceno, Dotta and Torres Leme, 2012). This approach allows the client system to handle local activities, calculations, reformatting and so forth, thereby lowering the amount and frequency of client-server traffic. Most RIAs are made with standardised technologies; (X)HTML, CSS and JavaScript (Merayo Voces, 2011).

RIAs increasingly rely on client-side code execution. New content can be obtained using JavaScript/AJAX, without refreshing or loading a new page. User interactions can modify visible elements without requesting data from a server. Alternatively, an AJAX request to the server can fully modify the presented content. In both cases, a new version of a page will be available without changing the URI (Batista, Carriço, Costa, Duarte and Fernandes, 2013).

JavaScript is a lightweight scripting language that has become increasingly popular over the years. As of today, it is the most used front-end programming language on the Web and 94 % of the top 10,000 websites use it (BuiltWith, 2014). Nevertheless, research has shown that less than half of modern web applications are accessible to people using screen readers (Nederlof, Mesbah and van Deursen, 2014) and that use of JavaScript and client-side code execution can decrease accessibility for people with disabilities in general (Brown Jay, Chen and Harper, 2011; García-Izquierdo and Izquierdo, 2012; Moreno, Martínez, Iglesias and Ruiz-Mezcua, 2012; O Connor, 2012 pp. 67).

Several standards and guidelines exist for helping developers make their websites and applications more accessible. However, guidelines are often large and demand a great deal of time and effort to get familiar with. They can also be difficult and time-consuming to apply. There are different kinds of tools and it is not easy to choose the most appropriate ones for a project. Important factors are if the tool is easy to use and fast to learn. Research indicates lack of knowledge about guidelines or confusing guidelines as important accessibility issues (Lazar, Dudley-Sponaugle and Greenidge, 2004; Tanaka and Vieira da Rocha, 2011; Trewin et. al., 2010). The proposal of guidelines for accessible RIAs is still in very preliminary stages (Dell Anhol Almeida and Calani Baranauskas, 2012).

1.1 Research questions

This project aims at making it easier to create accessible RIAs using standardised technologies. To do that, it seems useful to examine what actual problems occur with these applications and what solutions can be implemented to avoid these issues. This research aims at answering the following questions:

RQ 1: What problems exist with RIA accessibility?

RQ 2: What can be done to avoid RIA accessibility problems?

However, it is not enough simply to know how to avoid RIA accessibility issues. This knowledge needs to be passed on to those working with web development, ideally without consuming too much time and resources. Therefore, this study also aims at answering a third question:

RQ 3: What makes accessibility guidelines manageable for developers?

1.2 Goals and expected outcomes

The goal for this project was to start the process of creating an understandable and easily applicable set of guidelines for developing accessible RIAs. This tool should be easy to understand and apply during a development process. It has a clear focus on issues concerning dynamic and interactive applications. In addition, the guidelines focus on how to bring accessibility into the development process. A developer knowing little about accessibility should be able to quickly get an overview and understand what is needed to secure accessibility within this scope. The expected outcome of this project was an in depth theoretical knowledge of how to create accessible RIAs using standardised technologies. The result is available at <http://accessibilityagent.no/guidelines>.

1.3 Introduction to methodology

The methodology is divided into five parts:

1. Literature survey studying RIA accessibility issues
2. Literature survey studying suggestions for solutions to these issues
3. Creation of guidelines aimed at smoothing the process of RIA accessibility
4. Evaluation of guidelines to examine what works and what does not work
5. Updating the guidelines

1.3.1 Studying accessibility issues and solutions

A literature survey was conducted to discover problems with RIA accessibility. A distinction was made between:

- Social issues concerning accessibility
- Tool issues concerning accessibility
- Issues directly linked to RIAs

Social issues include positions amongst web developers, managers, customers and other stakeholders when accessibility is concerned. Tool issues are related to the problems with automatic testing tools and guidelines. Issues directly linked to RIAs are concerned with the dynamic and interactive nature of these applications. During the process of uncovering issues, many suggested solutions also revealed themselves. A thorough study was made of these suggestions. Some solutions were directed towards the *technical* issues, others were more concerned with how to make accessibility a natural part of the development process.

1.3.2 Creating guidelines

The process of creating the guidelines started with a lot of material about what can be done to create RIAs that are accessible. This material had to be sorted and presented in a way that is helpful for development teams working with accessibility. This means answering the following questions:

1. *What* to do?
2. *How* to do it?

Several iterations were made to make the guidelines as concrete as possible. The material was first sorted into two main themes; *technology* and *process*. A division was also made between *what* is recommended, *how* to execute the recommendations, *why* is it recommended and *who* recommends it.

Some thought was put into whether or not to place explanations of *why* something is recommended within the guidelines. It was, however, desirable that understanding *what* to do should be presented as quickly as possible. It was therefore decided before the evaluation *not* to put explanations into the guidelines in an effort to make them as short as possible.

1.3.3 Evaluating and updating the guidelines

Due to time limitations this project focused on the usability of the guidelines and not their reliability when it comes to ensuring accessibility. The method was supposed to be semi-structured interviews. It was however, believed that allowing the participants to reflect on certain questions *while* reviewing the guidelines would create more in depth and thought out reflections which could be discussed during the interviews. This was confirmed and it resulted in a natural conversation about the guidelines and what reflections the participant had made in the beginning of each interview before moving on to the interview guide. These conversations touched upon many of the planned questions and in most cases the interview guide served as a way to summarize and wrap up the conversation. After the interviews a list of suggestions for changes was prepared. What changes to make when updating the guidelines was chosen from this list.

1.4 Outline of the master thesis

The following presents the outline of this thesis and gives a short description of what can be found within each chapter.

Chapter 1: Introduction

Introduction to theme and presentation of problem statement

Chapter 2: Background

Background information about what web accessibility entails, assistive technologies (AT) and standards and legislation

Chapter 3: The technologies of Rich Internet Applications

A brief introduction to HTML, CSS and JavaScript

Chapter 4: Methodology

A description of the methodologies used in this project

Chapter 5: Accessibility issues

A literature survey of existing issues with accessibility

Chapter 6: Solutions

A literature survey of proposed solutions to accessibility issues ending with a summary of the first version of the guidelines

Chapter 7: Findings: Evaluation of guidelines

A presentation of the findings from the evaluation of the guidelines

Chapter 8: Implications and discussion of findings

A discussion of the findings from the evaluation and the impact it has on the results

Chapter 9: Alternation of guidelines

A description of how the guidelines were altered after the evaluation

Chapter 10: Results: Guidelines updated version

A presentation of the updated version of the guidelines and the main result of this project

Chapter 11: Conclusions

A presentation of what conclusions that can be drawn from this study and the answers to the research questions

Chapter 12: Contribution to research and future work

A presentation of suggestions for how work can continue with this project in the future and what is the main contribution to the accessibility research field so far

2. Background

The nature of the Web has changed dramatically over the last years. From being similar to physical documents containing information, it has become increasingly interactive and dynamic (Brown et. al., 2011). While the Web is becoming more sophisticated, society is becoming increasingly technological and websites and systems more important (Kern, 2008). This is especially true in the western world. The Web simplifies our recreational activities and facilitates activities more associated with our duties as citizens. In addition, software programs have become indispensable in our work and study environments.

Software programs, websites and applications targeted at a broad audience should be usable for the whole intended user group, consisting of people of all ages, backgrounds as well as physical and cognitive abilities. Statistics from the *World Health Organization* (WHO) reveal that 11, 8 % of the population in high-income countries has some kind of disability (2011 pp. 27). This is a considerable sized user group, and not a homogenous one. It grows as the population grows older, and issues vary in severity. Additionally it is important to keep in mind that people with disabilities is not *one* marked segment. They exist in all segments and are just as diverse and different as able-bodied people.

2.1 *The medical and social model of disability*

The medical model of disability views disability as a “*problem*” that belongs to the disabled individual. It is not seen as an issue to concern anyone other than the individual affected. The social model of disability draws on the idea that it is society that disables people, through designing everything to meet the needs of the majority of people who are not disabled. There is a recognition within the social model that there is a great deal that society can do to reduce, and ultimately remove, some of these disabling barriers, and that this task is the responsibility of society, rather than the disabled person (Bingham, Clarke, Michielsens and Van de Meer, 2013.)

2.2 *The Gap model*

The medical model and the social model are often presented as dichotomous, but disability should be viewed neither as purely medical nor as purely social. Persons with disabilities can often experience problems arising from their health condition. A balanced approach is needed, giving appropriate weight to the different aspects of disability (WHO, 2011 pp. 4). The Gap model (figure 1) illustrates the gap that arises between an individuals’ condition and the demands of society. The Gap model uses both the views of the medical and the social model to fill in the gap. The medical model strengthens the individuals’ conditions and the social model changes and lowers the demands of society (Berget and Moseid, 2012). This project would be placed in the social model, trying to fill the gap by changing and lowering the demands for interaction with RIAs.



Figure 1: The Gap model
(Moseid and Berget 2012)

2.3 What is web accessibility?

The World Wide Web Consortium (W3C) defines accessibility in the following:

“Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the Web, and that they can contribute to the Web.” (2005)

Accessible websites is a natural by-product of good design. Good design comes from understanding some core matters:

- What you are designing for and what is the purpose of your site
- Your users’ needs
- What visitors really wish to do when using your site

Technically accessibility can be defined as a subset of usability. Accessibility is a continuum. It changes according to technology. However, for users of assistive technology (AT) there are some core issues for each user group that does not really change even if the technology does. For example, blind users need to be able to access equivalent content that describes what a particular image is about, and people with limited physical mobility will benefit from a minimal amount of links to tab through (O Connor, 2012 pp. 11-13).

2.4 Assistive technology

Numbers from WHO indicate that a significant group of people is likely to be or become dependent on AT to access the Web (2011 pp. 7). AT is an umbrella term that includes assistive, adaptive, and rehabilitative devices for people with disabilities. AT can be both software and hardware. This technology promotes greater independence by enabling people to perform tasks that they were formerly unable to, or had great difficulties to accomplish, by providing enhancements to, or changing methods of interacting with, the technology needed to accomplish such tasks (Assistive Technology, 2014). The most common ATs are:

- Screen-readers

- Screen magnification
- *Braille* keyboard
- Switch
- Scanning software
- Speech recognition
- Head- or foot mouse
- Mouth stick
- Eye tracking

A screen-reader is a text-to-speech tool that reads the content on the screen to the user when they navigate and give focus to items using the keyboard (pp. 33-34). There are many different screen-readers available. It is typically used by blind and visually impaired users or users with dyslexia. It can also be used by people who are deaf-blind by converting text into *Braille* characters on a *Braille* keyboard (figure 2) instead of speech. *Braille* keyboard also allows users to type and enter text instructions to the computer in *Braille*, a writings system for the blind and visually impaired. A good understanding of screen-reader technology is a valuable foundation for successful accessible design, regardless of the AT used.



Figure 2: Braille keyboard
(WebAim 2013)

Screen magnification software is used to magnify the screen, completely or in part and is often used by visually impaired who are not completely blind (O Connor, 2012 pp. 60).

Users with physical disabilities often use a device called *switch* (figure 3) to interact with their computer and access the Web. Switches vary in form and can be considered binary input or simple “on” “off” input. *Switches* are often large buttons designed so that the user can press it with minimal effort. They can also be controlled by other means of interaction. There is a variety of different interaction forms that meet the users’ abilities.



Figure 3: Switch
(Switch 2014)

Switches are generally used in conjunction with scanning software. Scanning software works by dividing the screen into a grid-type layout initially highlighting each row of content in the grid for a user defined period of time. When the time-period has lapsed, the next row is automatically highlighted and so on. When the row containing the letter or symbol the user wants, he or she activates the switch and the individual item in each row is highlighted in the same fashion (figure 4) (O Connor, 2012 pp. 61 – 63).

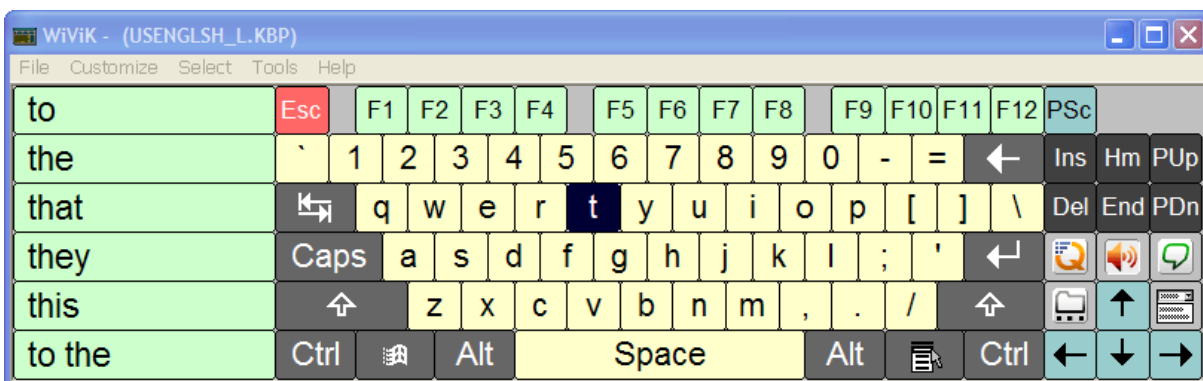


Figure 4: Scanning software
(AbleTech 2010)

Speech recognition is used for both dictation and commandoes and there are many existing on the market. This technology is very useful for someone with a motor impairment (Sandnes, 2011 pp. 168 – 170). It might also be useful as a writing tool for someone who is blind, or just in a hurry.

People who for some reason are not able to use a hand held mouse sometimes use a foot mouse instead. The foot mouse (figure 5) consists of two pedals where one controls the mouse click and the other controls the cursor. The pedals can be pressed back, front and sides through the ankle (pp. 171).



Figure 5: Foot Mouse
(Fentek Industries, 2015)

Another possibility is to use a *head mouse* (figure 6). Head movements are translated into cursor movements and clicks are recorded by holding the mouse pointer still a certain period or using a separate switch (pp.172).

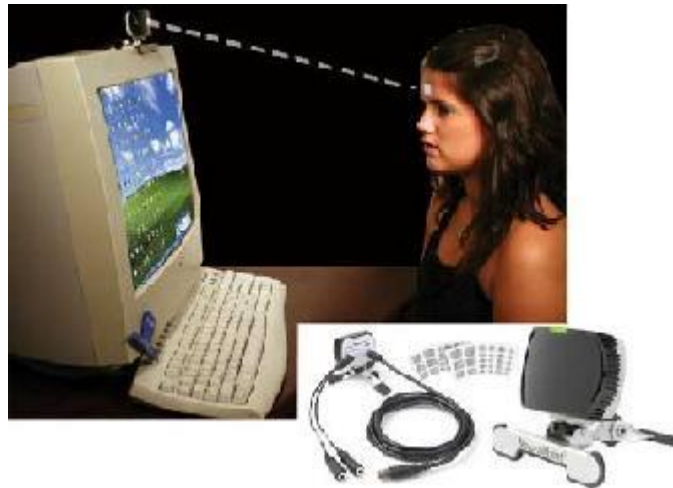


Figure 6: Head mouse
(RehabMart, 2012)

With *eye-tracking* the mouse cursor is controlled through eye movements independent of head movements. The user can freely manoeuvre the cursor and enter text via the virtual keyboard on the screen (pp. 172).

A *mouth-stick* is a more analogue possibility. It is a stick that is placed in the mouth. Due to its simplicity and low cost, the mouth stick is one of the most popular AT. Someone with no use of the hands could use a mouth stick to type and perhaps to manipulate a trackball mouse, depending on the amount of control that the person has with the mouth stick, and on the amount of patience that the person has if these movements are difficult. It is also possible to use *head wands*. Head wands are very similar in function to mouth sticks, except the stick is strapped to the head (WebAim, 2014).

2.5 Legislation

Making websites and applications accessible for people with disabilities is not only an ethical issue; it is also a legal one. Most western countries have some sort of enshrinement in the law that demands accessibility to some extent. On an international level, there is *The UN Convention on the Rights of Persons with Disabilities* (CRPD). CRPD states that products, environments, programmes and service should be universally designed i. e. to be usable by all people, to the greatest extent possible, without the need for adaptation or specialized design. However, universal design shall not exclude assistive devices for particular groups of persons with disabilities where this is needed.

Further CRPD Article 9 states that:

[...] States Parties shall take appropriate measures to ensure that persons with disabilities have access, on an equal basis as others [...] to information and communications, including information and communications technologies and systems

States Parties are to act appropriately to promote access for persons with disabilities to new information and communications technologies and systems, including the Internet. States Parties are also to promote the design, development, production and distribution of accessible information and communications technologies and systems at an early stage, so that these technologies and systems become accessible at minimum cost.

On the national level, there is the ADA - *Americans with Disabilities Act*² from 1990 and *the Rehabilitation Act* of 1973 – 1998: Section 508.³ In Australia, there is *the Disability Discrimination Act*⁴ from 1992 and the United Kingdom has *the Equality Act*⁵ from 2010, which used to be the DDA from 1995. In Sweden, there is *the Discrimination Act*⁶ from 2009 and Norway has *the Discrimination and Accessibility Act*⁷ from 2009.

2.6 Standards and guidelines

The *World Wide Web Consortium* (W3C) is an international consortium created in 1994, which it is responsible for establishing web standards. The W3C has dealt with accessibility since 1998 when the *W3C Web Accessibility Initiative* (WAI) was launched (*World Wide Web Consortium*, 2014). WAI works with organizations from all over the world to develop strategies, guidelines and resources aiming to make the Web accessible to people with disabilities. One of the things WAI is charged with is to develop guidelines and techniques that describe accessibility solutions for software and developers. From 2008, WAI started handling accessibility problems in RIAs by releasing the *Accessible Rich Internet Applications Suite* (WAI-ARIA) (*Web Accessibility Initiative*, 2014). Other than WAI-ARIA, WAI is behind guidelines such as *Web Content Accessibility*

² <http://www.ada.gov/>

³ <https://www.section508.gov/>

⁴ <http://www.comlaw.gov.au/Details/C2013C00022>

⁵ <http://www.legislation.gov.uk/ukpga/2010/15/contents>

⁶ <http://www.government.se/sb/d/3926/a/118187>

⁷ <http://lovdata.no/dokument/NL/lov/2013-06-21-61>

Guidelines (WCAG), Authoring Tool Accessibility Guidelines (ATAG) and User Agent Accessibility Guideline (UAAG). The most relevant guidelines for this project are WCAG and WAI-ARIA. A short description is given in the following.

2.6.1 WCAG 2.0

WCAG, now on version 2.0, covers a wide range of recommendations for making web content more accessible. Following these guidelines will make content accessible to a wider range of people with disabilities, including blindness and low vision, deafness and hearing loss, learning disabilities, cognitive limitations, limited movement, speech disabilities, photosensitivity and combinations of these. WCAG consists of four main goals. Content should be:

- *Perceivable* in the sense that any person must be capable of detecting the presented content.
- *Operable*, in the sense that any person is capable of interacting with the interactive elements on the page.
- *Understandable*, in the sense that any person is capable of understanding the meaning of the content.
- *Robust* in the sense that the content may be compatible with a greater variety of assistive user agents and technology (W3C, 2008).

2.6.2 WAI-ARIA

ARIA stands for *Accessible Rich Internet Applications*. WAI-ARIA became a standard as of March 2014 (Cooper, 2014). This standard ensures that RIAs are more accessible to people with disabilities. It especially helps with dynamic content and advanced user interface controls developed with Ajax, HTML, JavaScript, and related technologies. WAI-ARIA describes new navigation techniques to mark regions and common web structures such as *menus*, *primary content*, *secondary content*, *banner information*, and other types of web structures. With WAI-ARIA, developers can identify regions of pages and enable keyboard users to easily move among regions, rather than having to press the *tab* key many times (Web Accessibility Initiative, 2014). By enabling keyboard accessibility and making the *role* and *state* of controls explicit, WAI-ARIA provides a mechanism for noting *live regions*. These can have further attributes to indicate what type of updates to announce how important it is to announce the update, and how much of a region should be read when a part is updated (Moreno et. al., 2012).

WAI-ARIA is intended to augment semantics in supporting languages like HTML and SVG. It clarifies semantics to AT when authors create new types of objects, via style and script that are not yet directly supported by the language of the page. This is useful because the invention of new types of objects is faster than standardized support for them appears in web languages. WAI-ARIA is a positive, enabling technology. Rather than telling developers what they cannot do, WAI-ARIA allows developers to create RIAs (Web Accessibility Initiative, 2014).

3. The technologies of Rich Internet Applications

HTML, CSS and JavaScript are the cornerstone technologies used by most websites to create visually engaging web pages, user interfaces for web applications, and user interfaces for many mobile applications (Cascading Style Sheets, 2014). This chapter offers a brief introduction.

3.1 HTML

HyperText Mark-up Language (HTML) is the standard mark-up language used to create web pages. HTML is written in the form of HTML elements consisting of *tags* enclosed in angle brackets (*<html>*). HTML tags most commonly come in pairs like *<h1></h1>* and *<h2></h2>* although some tags represent *empty elements* and so are unpaired, for example **. The first tag in a pair is the *start tag* or *opening tag*, and the second tag is the *end tag* or *closing tag*.

A web browser can read HTML files and compose them into visible or audible web pages. The browser does not display the HTML tags, but uses them to interpret the content of the page. HTML describes the structure of a website semantically along with cues for presentation, making it a mark-up language rather than a programming language. However, it also provides basic functionalities which have been expanded in the latest version, HTML5.

HTML elements form the building blocks of all websites. It allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts written in languages such as JavaScript (HTML, 2014).

3.2 CSS

Web browsers can refer to *Cascading Style Sheets (CSS)* to define the look and layout of text and other material. CSS is designed primarily to enable the separation of document content from document presentation, including elements such as the layout, colours, and fonts. For each matching HTML element, it provides a list of formatting instructions. For example, a CSS rule might specify that "all heading 1 elements should be bold," leaving pure semantic HTML mark-up that asserts "this text is a level 1 heading" without formatting code such as a *<bold>* tag indicating how such text should be displayed. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate CSS file, and reduce complexity and repetition in the structural content.

This separation of formatting and content makes it possible to present the same mark-up page in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on *Braille*-based, *tactile* devices. It can also be used to display the web page differently depending on the screen size or device on which it is being viewed (Cascading Style Sheets, 2014).

3.3 JavaScript

JavaScript is used in combination with HTML and CSS and specifies the dynamic behaviour of a website and allows user to interact with the content. Its three main functions are to modify HTML through the *Document Object Model* (DOM) (the official W3C standard for accessing HTML elements), communicate with the server, and store data. It is used largely to determine how content will behave when activated or moved around, or how it responds when particular events fire. It is a client-side language, which means it operates entirely within the browser. It can be used in combination with languages that operate on the server side, such as PHP for more sophisticated server/client interaction (O Connor, 2012 pp. 68).

JavaScript is the most popular programming language in the world. 94 % of the 10,000 top websites use JavaScript (BuiltWith, 2014). An analysis of the 500 most popular websites showed that JavaScript was used in 93% of them (Brown et. al., 2011). According to W3C, all modern HTML pages are using JavaScript. It is the language for all computers, servers, laptops, tablets, phones, and much more. JavaScript code can be inserted into any HTML page, and all types of browsers can execute it (W3CSchools, 2014).

When JavaScript is used in combination with the *XMLHttpRequest* object to provide a method for exchanging data asynchronously between browser and server to avoid full page reloads, it is called AJAX, which is short for *asynchronous JavaScript and XML* (JavaScript, 2014).

4. Methodology

In this chapter, a description of the methodology for this project is presented. The process is divided into five parts:

1. Literature survey studying problems with RIA accessibility
2. Literature survey studying suggestions for solutions to these issues
3. Creation of guidelines aimed at smoothing the process of RIA accessibility
4. Evaluation of guidelines to examine what works and what does not work
5. Updating the guidelines

4.1 Literature study

To discover problems with RIA accessibility and suggestions for solutions to these problems a thorough literature study has been conducted. It soon became clear that variations of the same problems were discussed by several researchers. A distinction was made between three different themes:

- Social issues concerning accessibility
- Tool issues concerning accessibility
- Accessibility issues directly linked to RIAs

Social issues include positions amongst web developers, managers, customers and other stakeholders when accessibility is concerned. Tool issues are related to the problems developers face when working with accessibility guidelines and automatic testing tools and the fact that just following guidelines is not enough to ensure accessibility. These issues are not directly linked to RIAs. However, the complex nature of these applications may make these problems even more challenging and intricate. Issues that are directly linked to RIAs are concerned with the dynamic and interactive nature of these applications, like the fact that updates can be hard to detect and problems with navigation and keyboard access. These are technical issues that cause problems for the users.

Several researchers and developers had similar suggestions for solutions. A thorough study was made of these suggestions. It became clear that not all the issues could be solved with technical remedies, especially the social issues and problems with tools. Two main themes evolved: *technology* and *process*. The *process oriented* solutions address what is important to think of during the process of web development and creation of accessible applications. They deal with issues like the confusions and uncertainties that exist in web development teams when it comes to how they should go about working with accessibility. The *technology oriented* solutions address what technical features that can be used to enhance accessibility.

4.2 Creating guidelines

The process of creating the guidelines started with a lot of material about what can be done to make accessible RIAs. This material had to be sorted and presented in a way that is helpful for development teams working with accessibility. This means answering the following questions:

3. *What to do?*

4. How to do it?

It was a goal that a developer knowing little about accessibility would be able to quickly get an overview and understand what is needed to secure this. Therefore much effort was put into making the guidelines concrete and to the point. The material was first sorted into two main themes; *technology* and *process*. Then the process consisted of picking out the essence of the content, finding the concrete advice and good examples and presenting it as readable and short as possible. It became clear that it was easier to be specific and give concrete suggestions and examples when it came to the *technology oriented* guidelines. However, it has been attempted to be concrete in the *process oriented* part as well.

A division was made between *what* is recommended, *how* to execute the recommendations and *why* it is recommended and *who* recommends it. The guidelines only presented *what* to do and *how* to do it. Some thought was put into whether or not to place explanations of *why* something is recommended within the guidelines. However, it was desirable that understanding *what* to do should be presented as quickly as possible. Explanations of *why* would undoubtedly occupy space and make the guidelines longer and more time-consuming to get familiar with. Another point is that there is an entire chapter explaining the guidelines within the *master thesis (chapter 7: Solutions)*. It seemed this would cause unnecessary redundancies. On the other hand, it was also desirable that the guidelines should be usable outside the context of the thesis. They were also presented for evaluation outside this context. In the end, it was decided before the evaluation *not* to put explanations into the guidelines in an effort to make them as short as possible.

An attempt was made to address each individual guideline to one or more professions within a development team to make them more manageable and quick to get familiar with, because each team member can focus on the guidelines relevant for their profession. A prioritised list was intentionally not made. This is because the reliability of the guidelines has not yet been tested and their importance can therefore not be established.

4.3 Evaluation of the guidelines

To do a thorough evaluation of accessibility guidelines two things need to be established:

1. Are the guidelines usable and comprehensive to the target group, i.e. web developers?
2. Are the guidelines reliable, i. e. does applying them ensure accessibility?

Due to time limitations this evaluation will focus on the first question.

Methodologies used in previous evaluations of the guideline set WCAG (Tanaka and Vieira da Rocha, 2011; Brajnik, 2009; Trewin et. al., 2010) have served as inspiration when developing the methodology for this project. These studies have mainly approached the WCAG target group, web developers, and had them look at or try to use the guidelines. Afterwards the participants have been interviewed about their experiences. Questions were asked about how familiar the participants were with guidelines in general, their knowledge of accessibility, how they found using or reading WCAG, how time-consuming the tasks were, if the guidelines were understandable and usable and so on.

4.3.1 Interviews

The plan was to do semi-structured interviews. There were three main reasons for this. First of all semi-structured interviews start out with a set of questions, but one is free to let the conversation go where it may by asking follow-up questions or ask the participant to elaborate on certain comments. This way one can gather more insight and understanding than through a fully structured interview. The conversation still has a frame that makes it easier to conduct the interviews, especially for a novice interviewer. Secondly, this is the first evaluation of these guidelines. When first starting to examine something, a qualitative approach is recommended because this can supply more in depth information. It opens up for the possibility to explore a wide range of concerns and provide detailed responses. The respondents can reflect upon, and discuss the questions, which can provide data that would be hard to capture in a survey. This information can in turn be used to design fully structured interviews or quantitative surveys involving a much larger set of respondents in the future. When examining something there is little knowledge about, the danger of asking the wrong questions in a survey is much larger then when there is already some knowledge established on the subject. The third reason for this choice of method is that direct feedback from target group is fundamental to human-computer interaction (HCI) research. It is a good way to establish what a new tool should do, if the design does what it is supposed to and what changes should be made (Lazar, Feng and Hochheiser, 2010 pp. 178-189).

When designing the interview guide, presented below, many of the questions asked in previous evaluations where used. In some cases they were slightly modified to fit this context. Questions number 1, 2, 3, 4, 6 and 10 are inspired from previous studies. They have been used because their value was evident in the previous studies and they are typical questions that need to be answered when one wants to establish the usability of guidelines. The other questions are based on what knowledge the author wishes to gain about the guidelines in order to improve them.

Before the interviews the participants where asked to review the guidelines and consider the following:

- Is the guideline understandable to me?
- Did I learn something new from this guideline?
- Does the guideline seem reliable? Do I believe it will improve accessibility?
- Is the guideline concrete enough to really guide me in what to do to ensure accessibility?
- Could I use these guidelines in my next project?

It was decided to ask these questions beforehand because they are directly linked to the guidelines' value. They are inspired from some of the questions in the interview guide. It was believed that allowing the participants to reflect on certain questions *while* reviewing the guidelines would create in depth and thought out reflections which could be discussed during the interviews. This was confirmed. Each participant had made quite a lot of reflections. This resulted in a natural conversation about the guidelines and what reflections the participant had made in the beginning of each interview before moving on to the interview guide. These dialogs touched upon many of the planned questions and in most cases the interview guide served as a way to

summarize and wrap up the conversation. The interviews took approximately one to two hours. All participants accepted that audio be recorded during that time.

4.3.2 Interview guide

The interview guide is presented in the following.

1: What profession are you? What role do you normally have on the team?

- Team leader
- Programmer
- Interaction designer
- Graphic designer
- Writer
- Other (what?)

2: What level of expertise in accessibility do you consider yourself to have?

3: Are you familiar with any resources for information about accessibility requirements and solutions?

4: Have you ever used guidelines or tools for accessibility on previous projects? / Software or service that helps you to find, understand and fix accessibility issues in your product?
If yes, which ones?

4a: In what way do you find that these guidelines differ from guidelines you have used in the past? Do you feel they are better/worse/the same? Why?

5: Did you spend a long time getting familiar with the guidelines? Can you give an estimation of how much time?

6: Did you understand all of the guidelines you have been asked to evaluate?
If no, which ones were confusing? Why?

7: Did you learn something new about how to work for accessibility from the guidelines?
If yes, what?

8: Do you think the guidelines are useful to you in your work for accessibility?
If yes, in what way?
If no, why not?

9: Did you find the guidelines concrete enough to really guide you in the work for accessibility?

10: Do the guidelines seem reliable? Do you believe that if you follow them the accessibility level of the product you are creating will be improved?

11: Was there something you found missing in the guidelines?

What?

12: Was there something you found unnecessary in the guidelines?

What?

13: Do you think you could use these guidelines in your next project?

If no, why not?

4.3.3 Participants

Five participants were interviewed. To recruit, emails were sent to consultant firms and IT firms to ask for interested parties. Large firms and firms who seemed intending on accessibility were approached.

Participant 1 is an interaction designer who also does some front-end coding. He is working on how to integrate accessibility in the processes of his place of work. His knowledge about accessibility is relatively high, but he has not been working with it for more than a few years. However, he believes himself to know more than most developers. He knows about most existing resources for information about accessibility requirements and solutions.

Participant 2 is lead interaction designer of his firm. He has a master in informatics and has done some programming in the past. He works mainly with public websites and has worked a great deal with universal design and accessibility. He has held some *breakfast seminars* and several courses on the subject. He does a lot of work promoting accessibility internally and is the most knowledgeable on the subject at his company. He has also worked with *Standard Norge*⁸, developing standards for *universal design*. He has knowledge of various resources for information about accessibility requirements and solutions.

Participant 3 is a programmer and technical manager at his firm. He does not work with accessibility on a daily basis. He has some knowledge about the subject and attended a course about ten years ago. However, he is not up to date. Nevertheless, he understands the importance of accessibility and has some knowledge about WCAG and WAI-ARIA.

Participant 4 is an accessibility consultant working for a company specialising in accessibility, offering consulting services to other firms. His background is interaction design, and he is now an expert on universal design in digital interfaces. The company has projects on an EU-level. He has participated in the design of policies that are going to be law in all EU countries and is one of the top accessibility experts in the country.

⁸ <https://www.standard.no/toppvalg/om-oss/standard-norge/>

Participant 5 is a senior IT-consultant within the programming field. He has somewhat knowledge about accessibility, but does work with this on a regular basis. He has an overall knowledge of what is needed, but not necessarily how to implement it. He does, however believe he has the knowledge to *figure out* how to implement it. He knows the importance of accessibility and the reasons for working with this and has used parts of WCAG in previous projects.

The participants' different backgrounds and levels of accessibility knowledge give them each unique perspectives that are valuable to the evaluation and further development of the guidelines. Where their opinions differ, it is mostly due to differences in background and knowledge. This is important to notice when developing guidelines for all types of web developers.

4.4 Updating the guidelines

After the interviews a list of suggestions for changes was prepared. What changes to make when updating the guidelines was chosen from this list. The most critical weaknesses and the changes that were feasible within the time limits of this project were prioritized. The first version of the guidelines, the version that was evaluated, is placed in *appendix 1*. The changes that were made to this version are presented in *chapter 9*. The second version of the guidelines is presented in *chapter 10*.

4.5 Limitations

The most important limitation in the methodology for evaluating the guidelines is that interviewing developers can only establish if the guidelines are useable to the target group, and say nothing about their reliability. However, some of the participants have some accessibility expertise. Asking questions about how reliable they believe the guidelines to be might give some indications. On the other hand, the fact that the participants have so much knowledge about accessibility might also be a limitation. It means that the evaluation cannot really establish how usable the guidelines would be to an accessibility novice, which is an important part of the target group. Nevertheless, there are different levels of knowledge, so some indications of the usability for this target group can be made.

One important limitation is that it is the creator of the guidelines performing the interviews for evaluating them. This may cause participants not to say many negative things about the guidelines or give them higher praise than they actually believe they deserve because they do not want to be rude or insult the person who has made the guidelines. It does not however seem to be a large problem, as many negative comments and suggestions for changes have come up during the evaluation.

Another important limitation is that there was only time for *one* iteration and no time to re-evaluate the updated version of the guidelines. However, creating guidelines is a vast and complicated process, and this project only aims at beginning this process by establishing RIA accessibility problems and solutions to these issues and beginning to present this in a guideline format.

5. Accessibility issues

This chapter presents issues related to accessibility. *Social* issues and problems concerning *tools* is presented first. These issues are not only applicable to *Rich Internet Applications* (RIA). However, the complex nature of these applications may make them even more prominent and challenging because it makes accessibility that much more complex as well. Lastly, issues directly linked to RIAs will be presented.

5.1 Social issues

Convincing colleagues, clients and management of the importance of accessibility is difficult. This results in a lack of time and funding to address accessibility (Lazar et. al., 2003). This often results in many trying to add accessibility towards the end of the development process. Anyone who knows something about accessibility would know that this is not a good idea (Hoffman, 2014).

Accessibility is complicated and time-consuming. Developers often view accessibility as difficult, expensive and “in the way” and people working with accessibility as the “party poopers” of web development (Groves, lecture April 5th 2014). This attitude leaves a lot up to the client and some web development companies may be waiting for an accessibility-minded client to catalyze the learning process.

Accessibility is a massive area and competence and knowledge about it is low. There is a need for expertise and for training web developers. Accessibility is interdisciplinary and relates to everything from good coding and design practises to user testing with people with disabilities and using tools when testing for accessibility. This might overwhelm some developers and leave them puzzled as to where to start (Hoffman, 2014).

Some people are still questioning the need for accessibility (Lazar et. al., 2003). They consider it unnecessary and inappropriate. Some even feels it is as an intrusion into their graphical design sensibilities. Work with enlightening developers, managers and even clients on what accessibility is and why it is important is still vital.

5.2 Tool issues

It has been argued for the importance of using guidelines and other tools for automatic and manual accessibility testing. Many developers and designers do not have the academic foundation that gives them a deep understanding of accessibility and how to identify and fix accessibility problems. Therefore they may have their first contact with accessibility through tools. The guideline review is one of the most widely used methods to evaluate web accessibility (Tanaka and Vieira da Rocha, 2011). Developers who use tools when they work with accessibility prefer automated testing tools with error explanations (Trewin et. al., 2010). However, using tools is not without its problems.

Tools are time-consuming and hard to get familiar with. Existing tools and guidelines can be hard to learn (Tanaka and Vieira da Rocha, 2011). Using them reliably and effectively requires

accessibility expertise because of their complexity. New standards, incomplete browser implementations and inconsistencies in AT have to be taken into account (Trewin et. al., 2010).

Tools and guidelines are difficult to understand. They can be unclear and confusing. Although developers find tools helpful to discover accessibility problems, the explanations provided by are not always enough to help understand the accessibility issues and to fix them. Some checkpoints WCAG 1.0 were found to be difficult to understand and even hard to relate to accessibility or imagine which groups of users may be affected by them (Tanaka and Vieira da Rocha, 2011).

Tools are time-consuming and difficult to apply (Trewin et. al., 2010). They can be very difficult to manage (Tanaka and Vieira da Rocha, 2011). Some believe that one of the reasons for the general lack of accessibility testing is due to the tedious and time-consuming tool support for such checks (Rosson, Ballin, Rode, and Toward, 2005). Using test tools and finding technology workarounds can be equally hard and time-consuming as designing for accessibility in the first place (Trewin et. al., 2010). Developers request better software tools and less confusing guidelines (Lazar et. al., 2003).

Tools are not always 100 % reliable or valid. They are often incomplete with respect to the standards that must be met. Automatic testing cannot check the conformance of all WCAG 1.0 checkpoints. An expert analysis will always be required (Tanaka and Vieira da Rocha, 2011). WCAG checkpoints in general fare very low in terms of reliability, and that from this perspective WCAG 2.0 is not an improvement over WCAG 1.0. There are large differences in effectiveness for the different checkpoints, and between guidelines sets. None of the guidelines sets have checkpoints whose reliability is definitely higher than 80% (Brajnik, 2009).

5.3 RIA issues

Some of the most important accessibility issues directly linked to RIAs are:

- Assistive technologies (AT) have trouble keeping up with Web 2.0.
- Updates not being detected
- Un-existing semantics
- Standard violations and errors
- Navigation problems
- Problems with keyboard access
- Pop-up windows
- Over-engineered user interfaces

5.3.1 Assistive technology and Web 2.0

AT have trouble keeping up with the Web 2.0 changes. This has been pinned this down to be the main issue with accessibility (Brown et. al., 2011). Some believe that RIAs represent a shift in an interaction paradigm because people that access Web using AT are tied to a linear navigation approach, while content updates in RIAs move focus from one area of the application to another without following a rigid sequence. Enriching an application may therefore seriously prevent the

use of AT. They are not ready to cope with the dynamic content (Batista et. al., 2013; Dell Anhol Almeida and Calani Baranauskas, 2012; García-Izquierdo and Izquierdo, 2012; Lemon, 2008).

5.3.2 Updates not being detected and causing confusion

Continual updates can be confusing for users. AT often have trouble presenting updates and the natures of them. RIAs update parts of information presented asynchronously on a web page. Screen readers might not read messages generated by AJAX technology because everything loads on the client side before the actual interaction occurs. Therefore, screen readers may not be able to detect the updates if not implemented properly. Some screen readers give no indication of updates, while the more sophisticated ones announce updates triggered by clicking, but not those that occur automatically (Braga et. al. ,2012; Moreno et. al., 2012; García-Izquierdo and Izquierdo, 2012; Dell Anhol Almeida and Calani Baranauskas, 2012; Lemon, 2008).

The fact that websites have become dynamic and interactive changes the entire model of a page dramatically. Users now need to understand whether a page has changed, which parts have changed, and whether the changes are of interest. This complexity may cause users who access the Web through AT to feel lost while surfing the Web (Brown et. al., 2011). To emphasize the extent of this issue it should be mentioned that out of 4000 web applications examined in a study, 90% performed DOM manipulations after they are loaded (Nederlof et. al., 2014).

5.3.3 Non-existing semantics

Semantic mark-up supplies AT with information about the element so users can perceive and interact with them. By for example allowing a screen reader to point out the role of a certain DOM element, users are able to speed up their browsing experience by letting the screen reader only read out the sections they are interested in (Merayo Voces, 2011). Some researchers are concerned about the fact that the role of the widget, what it does, along with its states and properties is not always available to AT. If semantics are not there or not used correctly, this affects the usability of a web application for an AT user (Lemon, 2008).

One study highlights the prevalence of non-existing semantics. It uncovered that 60% of the sites examined did not provide screen readers with any information about the structure of the page. A mere 6% of the sites adhered to the specification. Just 30% made use of semantic tags for interactive elements such as `<dialog>`, `<menu>` or `<menuitem>` and there was a tendency to either equip all images and figures with `longdesc` attributes or to do it with none at all. Of the web applications with a table in any of their states, almost none had heading rows or columns and even less had a summary or caption. More than a third of the sites using input elements did not declare an appropriate label for an input ID. The sites that did use labels for input elements only applied it on average to 20% of them. Only five out of the 3292 sites that have input elements applied the technique to all input elements (Nederlof et. al., 2014)

5.3.4 Problems with keyboard navigation and access

It is common to find interfaces that are only interactive with the mouse (Merayo Voces, 2011). Many are built with device dependent event handlers such as `onMouseOver` and `onMouseOut` (O

Connor, 2012 pp. 75). RIA toolkits introduce complex user-interface components and dynamically changing content (Brown et. al., 2011; Moreno et. al., 2011). Users are not just unable to access widgets, but are also being blocked out of hidden content that needs mouse input to become visible (O Connor, 2012 pp. 72; Braga et. al., 2012). With Dynamic HTML (DHTML), CSS and JavaScript is used to display, hide, or move information based upon input from the user or pre-programmed commands. Most drop-down or fly-out menus or other types of rich interactions involve scripting. Because most of these elements are modified based upon mouse input, they are inaccessible to keyboard and AT users (WebAim, 2013).

Client-side programming and server requests in the background and are rarely keyboard accessible (Lemon, 2008). If dynamic content is constantly changing or changes while the user is reading it or has set focus to it, this can interfere with navigation. For example, if an element that has keyboard focus is significantly changed, hidden, or removed from the page, keyboard focus may revert to the very beginning of the page (WebAim, 2013).

5.3.5 Standard violations and errors

A substantial number of web applications contain a diversity of structural errors that can potentially break the application. This is also a problem for accessibility because standards are the basis for getting websites, browsers and AT to communicate well with each other. Studies have shown severe W3C standard violations and errors in the websites. Half of the observed websites contained ambiguous IDs, potentially leading to misplaced DOM manipulations. Almost a fifth of the applications did not define or have an invalid DOCTYPE, causing the browser to enter an unpredictable render mode. A significant amount of websites also contain broken HTML, exhibited errors concerning attributes, and have misplaced elements (Nederlof et. al., 2014).

5.3.6 Pop-up windows

Pop-up windows have been described as a unique accessibility concern. When the page the browser is viewing suddenly changes or refreshes, the user may become disoriented or confused, especially if that person is using an AT. For a visual user, it may be difficult to notice and navigate to the new window or tab. For someone who is using AT, the new window may be annoying and confusing because the default behaviour for the link has been modified. For someone who is blind, there is typically an indication that a new window has opened, but it may be burdensome to return to the original page. When the screen reader user attempts to return to the previous page by selecting the back button, it may be confusing to find that this does not work. JavaScript implementations may make the new window difficult or impossible to resize or scale for someone using a screen enlarger. Someone with a motor disability may rely upon large tool bars to accurately control the browser, so removing or modifying them may introduce difficulties for this user.

When using JavaScript to open new windows, it is possible to modify the size and position of the new window. Adding or removing functionality of the window, such as the ability to resize, display scroll bars, show tool bars etc. is an option. It is, however recommended to be careful when

changing the default behaviour of the browser window. Avoiding use of pop-up windows except in extreme cases is good practice (WebAim, 2013).

5.3.7 Over-engineered interfaces

Some interfaces are simply over-engineered. Functions, animations and dynamic behaviours have been added for their own sake and not because they have a good and thought through reason for being there. This can often be unintuitive and confusing, especially for elderly people or people with some sort of cognitive disability (O Connor, 2012 pp. 73). An overload of notifications is an issue for all users, but even more so for people using AT (Dell Anhol Almeida and Calani Baranauskas, 2012).

6. Solutions

In this section, the first step in the process of developing the guidelines is described. The chapter presents a literature study over suggestions for solutions to RIA accessibility issues.

6.1 Have accessibility expertise on the team

To use tools and guidelines reliably and effectively, accessibility expertise is required. Experience is essential to successfully evaluate or develop accessible websites and applications. This has to do with the complexity of accessibility issues (Brajnik, 2009; Mankoff, Faith and Tran, 2005; Trewin et. al., 2010; Yeasilada, Brajnik and Harper, 2009).

Having someone on the team who is familiar with testing tools for accessibility, different kinds of AT and how they work, good coding practices to ensure things like keyboard access and understandable navigation for a screen reader, simulation exercises and user testing with users with disabilities will save time and give a more accessible result. It is recommended that this expertise is a part of the team from the beginning and possibly throughout the process (W3C, 2002).

6.2 Introduce accessibility from the beginning

Testing and finding workarounds for accessibility can be more time-consuming than designing for accessibility from the beginning (Trewin et. al., 2010). It is important to note that if accessibility is an integral aspect of the design process and not an add-on or separate activity the result will be better, i. e. the applications will be more accessible and usable to everyone (Foley and Regan, 2002). It will also make implementing accessibility easier for the development team (Tollefsen, 2011). It is therefore recommended to introduce accessibility from the beginning of a project and make it a priority throughout the development process. Incorporate accessibility from the first draft and then verify accessibility at key stages. When deciding upon technology and software, accessibility should be factored in. It is important to choose technology that facilitates and supports the creation of accessible websites and applications (W3C, 2011).

6.3 Test accessibility at key stages

Testing is fundamental in any development process. It is important to know that the end product actually works. It is good practice to test for accessibility on an equal basis as usability or user experience. It is essential to start testing early in the process (Trewin et. al., 2010). Testing with *one* user early is better than testing 50 near the end (Krug, 2006 pp. 134). It is recommended to continue verifying accessibility at key stages and testing throughout the process (W3C, 2011). The more often testing is done, the more certain developers can be of the quality of their product. For example, evaluating an application during the development process can be a good way to make sure JavaScript is accessible (O Connor, 2012 pp. 295).

Using automated testing tools for checking conformity to accessibility guidelines is one method for testing. However, not all automatic tools are well functioning. There is a need for new algorithms

that reduce false positive results and lightweight visualizations that support novices in performing manual checks. More automation of manual checks would also be welcome. Accessibility tools should promote knowledge transfer, i. e. understanding of accessibility is broadened by using the tool (Trewin et. al., 2010).

Present tools for evaluating how well a web page conforms to WAI-ARIA are either still in trials, or permit only a limited inspection. Therefore, a complete accessibility evaluation requires manually validating specifications and good practices. Even so, it is worth trying some of the automatic tools that provide WAI-ARIA testing features, such as the *Web Accessibility Toolbar (WAT)*⁹ for IE and Opera, *Firefox Accessibility Extension*¹⁰ and *Firefox Juicy Studio Accessibility Toolbar*¹¹ (Moreno et. al., 2011)

Tools should be used in combination with expert testing. Expert testing should be done before user testing. *The heuristic evaluation* checks the interface against a set of guidelines and figures out what works and what does not work. *The consistency inspection* checks consistency of colours, fonts, icons and so on in the interface. *The cognitive walk-through* is an important test method where an expert goes through tasks in the system. They might simulate typical users with the personas method (Lazar et. al., 2010 pp. 256 - 258).

Research show that developers feel that experiencing a site as someone with a particular disability would, would be valuable to them in their work for accessibility (Trewin et. al., 2010). Simulation is a way to do this. For example testing a site with a screen-reader, using only keyboard to navigate or turning of JavaScript in the browser, will to some extent imitate the experience and can be used during a *cognitive walk-through* or even a *heuristic evaluation*. Using simulation exercises in combination with a personas method is also recommended. All AT do not work exactly the same. Therefore testing with these technologies as much as possible is beneficial (O Connor, 2012 pp. 314).

Screen-reader users have different browsing strategies as well as levels of digital literacy, same as none screen-reader users. Therefore, simulation, although valuable in its own right, is no substitute for testing with real users. The psychology is also different. The tester can always turn the screen reader of and turn the display on or plug the mouse pack in if the experience is frustrating. This is why user testing is always required in addition to other forms of testing (O Connor, 2012 pp. 316).

User testing with people with disabilities gives a good idea of what it is like to use an interface if one has some sort of impairment. However, it is a bit different from user testing with able-bodied users. In some cases, users with disabilities require AT that needs to work with the website or application. In other cases the user does not depend on AT, but still might have other types of input or other browsing habits than a user without a disability. Bring in users during the entire

⁹ <http://www.paciellogroup.com/resources/wat/>

¹⁰ <https://addons.mozilla.org/en-us/firefox/addon/accessibility-evaluation-toolb/>

¹¹ <https://addons.mozilla.org/En-uS/firefox/addon/juicy-studio-accessibility-too/>

development and design process. This gives a fuller picture of the user experience and enables effective design choices early in the development (Lazar et. al., 2010 pp. 260; O Connor, 2012 pp. 285 - 286).

6.4 Follow existing design principles

Creating accessible interfaces really boils down to good design practice, well-formed code, solid information architecture, accessible form validation and error recovery. This will ground the designs in best practices and therefore bring extra benefits such as more future-proof, interoperable applications and websites (O Connor, 2012 pp. 20).

Instead of arguing that an accessible website is better for everybody, the argument can be reversed. Making a website usable for everyone is one of the most efficient ways of making it more usable for someone with a disability. If something confuses most people, it is almost certain that it will confuse users with accessibility issues. In addition, they might have a harder time recovering from their confusion. If good design principles are not followed, it does not matter how accessibility compliant a website is. It is still not going to be usable for someone with a disability (Krug, 2006 pp. 174-175).

6.5 Use WAI-ARIA mark-up

WAI-ARIA is an accessibility-focused language which can bridge the semantic accessibility gap in projects (O Connor, 2012 pp. 10). Studies have shown that WAI-ARIA enhances the accessibility of *Web components* (Braga et. al., 2012), and several researchers advocate using it when working with RIA accessibility. In order to make a function explicit to the user of an AT, the function must be determined from the code. With WAI-ARIA, semantics developers should be able to adapt interfaces to meet specific needs. It supplies an application with the semantic metadata it needs to communicate well with AT, enabling user agents and browsers to understand the semantic mark-up (Brown et. al., 2011; Dell Anhol Almeida and Calani Baranauskas, 2012; Moreno et. al., 2011).

An important aspect of WAI-ARIA is that it contributes with semantics that does not exist in other languages. One example is the properties connected to *live regions*, with ensures updates being detected by AT. Notification or presentation based on a couple of simple rules can save users considerable confusion because WAI-ARIA can alert the user if AJAX has updated any part of the page. Users can turn off the updates if they wish, but it is also possible to set updates in a polite state or an assertive state, making them easier to deal with. Consistent and full support for these standards across browsers and AT is important (Trewin et. al., 2010). User agents have to support WAI-ARIA mark-up so that they can pass information about what has changed to the AT, which in turn can decide *what* to present and *how* to present it. Full support for the *Document Object Model* (DOM), including events, is needed to supply the AT with all the information about what has changed and when. With access to a live DOM, detection is not difficult (Brown et. al., 2011). WAI-ARIA now has support in most browsers and AT (O Connor, 2012 pp. 102; Lemon, 2008).

6.6 Follow and validate the HTML5 standard

Standards are the basis for getting websites, browsers and AT to communicate well with each other. Avoid old standards as they have weaker support for universal design than newer standards. Correct code gives a stable presentation across browsers and platforms, simplifies maintenance of the site and is good in terms of search engine optimization (Difi, 2015).

HTML5 is the next generation of the web standards that W3C is standardizing and is specially designed to deliver rich web content without the need for additional plug-ins. HTML5 has new semantic, graphics, and multimedia elements, new form elements and new API's to make it easier to build web applications. It is cross-platform and designed to work on different types of hardware (HTML5, 2014).

A number of researchers and developers recommend using HTML5 for accessibility because its rich semantics creates better structure. It also offers more advanced input types and better integration with scripts (Dell Anhol Almeida and Calani Baranauskas, 2012). Using HTML5 has some of the same benefits as adding WAI-ARIA (O Connor, 2012 pp. 6-9). A study of the accessibility of websites built in HTML5 focusing on the top 100 most visited websites uncovered that accessibility of websites built with HTML5 is somewhat higher than in websites built with earlier specifications (Park, Lim and Lim, 2014).

One example of the benefits of richer semantics is the `<nav>` element. To mark up landmarks on a page in earlier versions of HTML the only available element was the `<div>`, which has no semantics of its own. While screen readers had useful access allowing users to navigate around HTML *headings, lists, blockquotes* and *tables*, there was no alternative for this with navigation blocks. There was no unique element for the screen reader to hone in on. With the `<nav>` element the user can jump between navigation blocks and quicker get an overview of the website (Lawson and Faulkner, 2011).

Other HTML5 elements for applying semantics to structural areas of a page include `<header>`, `<footer>`, `<section>`, `<article>`, `<aside>`, and `<figure>`. These elements supplement the `<div>`. HTML5 also has various kinds of input elements that allow browsers to validate input forms. The `<input>` element has a *type* attribute which have many new values such as *search, email, url, number* and *time*. The new properties can be combined with the `<form>` element making it possible to perceive which data is put into which area. Lastly, there is a strong support for graphics with `<canvas>` and `<svg>` and for multimedia with `<video>` and `<audio>` (Park, Lim and Lim, 2014; Lawson and Faulkner, 2011; O Connor, 2012 pp. 6-9; Dell Anhol Almeida and Calani Baranauskas, 2012).

As previously described, use of semantic mark-up is one of the primary ways to convey information about content to users of AT. There are some benefits of using HTML5 as opposed to WAI-ARIA. The problem with WAI-ARIA is that it is added on an already functioning code, which can be of various qualities. HTML5 on the other hand, is the actual semantic page mark-up

(Lawson, 2009). In other words, HTML5 has built-in semantics. WAI-ARIA could in theory be «bolted on» some low quality code just to satisfy conformance checkers.

HTML5 elements should be used even if there is uncertainty about the support for the element or whether it will work 100 %. If a browser or an AT does not understand the element it will generally be treated like text and not break anything. Besides, more support is likely to come very soon (O Connor, 2012 pp. 1 - 10).

6.7 Combine WAI-ARIA and HTML5 mark-up

Many WAI-ARIA roles are similar to HTML5 elements. Using these in conjunction is a way of making applications backwards compatible with legacy versions of AT. HTML5 is not yet supported in all browsers. In addition, even if the newest version of a browser supports something, the older versions will not be updated to support it, and some people will continue to use older browsers. WAI-ARIA has been around for a while and most new browsers and AT supports it. When a browser comes across a combination of HTML5 controls and WAI-ARIA, the WAI-ARIA control will generally trump the HTML5 (O Connor, 2012 pp. 22; Peterson, 2012).

WAI-ARIA support is a part of the HTML5 specification and will therefore validate without problems. Previous HTML versions does not validate with WAI-ARIA. There are also certain correspondences between the two specifications in the HTML5 spec and some elements have built-in WAI-ARIA roles (Lawson and Faulkner, 2011).

6.8 Use Progressive enhancement

Progressive enhancement is a strategy for web design that emphasizes accessibility, semantic HTML mark-up, and external style sheet and scripting technologies (Buckler, 2009). It is a widely used method for creating web applications that works on a wide variety of devices, user agents and connection speeds. *Progressive enhancement* uses web technologies in a layered fashion and separates them as such:

- HTML (Content and basic functionalities)
- CSS (Presentation)
- JavaScript (Behaviour)

The content is marked up by well-structured HTML, correctly labelled forms, suitable alternative text and so on. Then a separate CSS file is added for presentation. Lastly, JavaScript is added to aspects of the site to enhance functionality to user agents that can handle it correctly.

Progressive enhancement benefits users of AT because the basic content will always be available and independent of the scripting to work, while also providing an enhanced version of the page to those with more advanced browser software or greater bandwidth. Using *Progressive enhancement* ensures that the code does not break the user agents that cannot understand JavaScript. It also ensures keyboard accessibility as long as device independent JavaScript methods are used (O Connor, 2012 pp. 70 – 71).

Another benefit to *Progressive enhancement* is that it ensures backward compatibility with older browsers and AT. This can solve some of the problems with use of old versions of browsers or equipment (pp. 10). In addition to future proofing *Progressive enhancement* is useful when it comes to scalability, performance and maintenance (Wright, 2012 pp. 17).

Accessible web applications might use a combination of *Progressive enhancement* and another method called *graceful degradation* (Buckler, 2009). *Graceful degradation* is the practice of building a website or application that provides a good level of user experience in modern browsers, but will degrade *gracefully* in older browsers. The system may not be as pleasant or as pretty, but the basic functionality will work on older systems. Developers adopting *graceful degradation* often specify their browser support level, e.g. level 1 browsers (best experience) and level 2 browsers (degraded experience).

Some believe that *graceful degradation* is approaching the problem from the wrong direction and is less fair-minded than *Progressive enhancement* (Buckler, 2009). It is a matter of focus. *Graceful degradation* focuses on building the website for the most advanced/capable browsers. Testing in browsers deemed “older” or less capable usually takes place during the last quarter of the development cycle and is often restricted to the previous release of the major browsers (IE, Mozilla, etc.). Under this paradigm, older browsers are expected to have a poor, but passable experience. Small fixes may be made to accommodate a particular browser. Because they are not the focus, little attention is paid beyond fixing the most serious errors.

Progressive enhancement is more focused on the content than the browser. Content is the reason for creating websites to begin with. In addition, *Progressive enhancement* does not require selecting supported browsers or revert to table-based layouts. Instead, a level of technology is chosen, e.g., the browser must support HTML 4.01 and standard page request/responses. Therefore, *Progressive enhancement* may be a more appropriate approach.

It is also possible to use *Progressive enhancement* within the CSS. Other than making the *stylesheets* easier to maintain, separation can help obtain greater consistency of presentation across the browsers and media types targeted. Web standards-aware developers should always start by testing designs in the most standards-compliant browsers available, and then provide fixes for those browsers that just need a little nudge to get them on the right track (Gustafson, 2008).

6.9 Use JavaScript unobtrusively

JavaScript can do almost anything. It is easy to go overboard and use it too much (Wright, 2012 pp. 52). Using JavaScript *unobtrusively* means only using it when absolutely necessary and not making it a requirement for a functional application. It is a script that is not forced on users or stand in their way but rather tests whether it can be applied and does so if possible (Fergusson and Heilmann, 2013). JavaScript is only added to enhance the user experience, make the application more responsive to user needs and provide quick access to information. It is removed from the

HTML and tightens the separation between structure, presentation and behaviour (Wright, 2012 pp. 53).

Unobtrusive JavaScript also means avoiding unnecessary movements on a web page, unintuitive widget functionality and unfamiliar controls. It aims at giving users a more seamless experience. Good technology should be invisible. Users will notice unwanted behaviours and especially if something does not work (O Connor, 2012 pp. 71).

Although *Unobtrusive JavaScript* is good practice and should be implemented, it should be noted that it is not necessarily *accessible JavaScript*. *Unobtrusive JavaScript* is often mentioned in connection with catering for people whose browser lack JavaScript support, or if JavaScript fails. However, this is no guarantee for keyboard navigation or screen reader support (Johansson, 2010).

6.10 Make sure methods are device independent

When used correctly to add behaviour to an object, *device independent JavaScript methods* should ensure accessible interaction with any input device from *eye-tracking* and *scanning software* to *keyboard*, *mouse* and *touch* (WebAim 2015). Use of *device independent methods* is a simple and powerful example of a more accessible way to add JavaScript to content because it allows for making some events keyboard accessible through JavaScript alone, not having to add WAI-ARIA elements (O Connor, 2012 pp. 73).

6.11 Use accessible modal windows instead of pop-ups

A modal window or dialog window a sub window of an application that asks a question, requires some input, or allows for setting options etc. There are some unique properties that distinguish them from other windows. The primary one is that it typically cannot be moved out of the main application window. Another is that usually, when the modal window is open, the application around it becomes inert, i. e. not possible to use before the modal window is closed (Zehe, 2015). Modal windows have replaces pop-up windows on the Web. They are in many ways better and more usable. They open inside the webpage instead of in a new, window, which saves the user the trouble of dealing with multiple windows. They cannot hide behind the browser windows and be missed by the users. They grab the users' attention and sets focus to the tasks in the modal window. They match the website and feel connected to the site and look less like pop-up ads. This feels safer and more secure for the user. Modal windows also darken the background to cut the background noise. This allows users to focus on the content in the window. This is effective because users become focused on reading the message and can make the correct decision and finishing their task before they go back to the browser window.

One should use modal windows only when the purpose is to completely take the user's focus and attention off the browser window to the new window, e. g. for an important dialog box or action like saving something in a text editing program before closing it (UX Movement, 2011). Creating a modal dialog that is fully accessible with WAI-ARIA requires some work, but it is absolutely possible (Zehe, 2015).

6.12 Use technologies that facilitates accessibility

Technology that facilitates accessibility should in theory make it easier to quickly ensure accessibility without applying time-consuming tools and guidelines doing extensive accessibility testing or even knowing much about accessibility. However, technology facilitating accessibility is far from perfect at this point. It is undoubtedly the right path to take, but there is still a way to go.

6.12.1 Toolkits and frameworks

Toolkits and *frameworks* are technologies that facilitate RIA development. *Toolkits* assist in abstracting the differences of browsers and on providing basic functions to handle events. *Frameworks* assist on standardizing sites development (Braga et. al., 2012). In order to facilitate accessibility, researchers are exploring *toolkits* and *frameworks* that enable faster and easier Web 2.0 application development and broaden web accessibility. The efficiencies of accessible widget libraries and platforms cannot be underestimated, mainly because they often have already conquered issues between platforms and browsers (Trewin et. al., 2010). Some believe the immediate challenge related to accessibility and RIA is building Web 2.0 development *frameworks* and technologies with a greater degree of WAI-ARIA support so that developers do not have to be experts in what constitutes accessibility.

Several *framework*, *toolkit* and *library* solutions are available. *Dojo* and *Bindows* are two solutions that have been highlighted for their support of WAI-ARIA and tagging that permits access for AT. Both solutions include components that offer keyboard support and provide an accessibility statement and have received special recognition from the W3C for their compliance with the WAI-ARIA specification (Moreno et al., 2011). Other possibilities are *jQuery UI* and *Fluid Infusion* (O Connor, 2012 pp. 84).

Although using these frameworks can be a great benefit, it is important to accurately wire and set the properties of accessible UI widgets. It is also important to do research on what the *toolkits* and *frameworks* actually facilitate when it comes to accessibility (Trewin et. al., 2010). Another concern is to what degree these technologies are in fact used in the web development business.

6.12.2 Web components

Web components are a collection of standards, which are working their way through the W3C. They enable truly encapsulated and reusable components for the Web. The component model for the Web consists of four pieces designed to be used together. These pieces are:

- Templates
- Decorators
- Custom elements
- *Shadow DOM* (W3C, 2013)

Web components facilitate accessibility because they enable web application authors to define widgets with a level of visual richness and interactivity not possible with CSS alone. It also offers an ease of composition and reuse not possible with script libraries today. *Web components*

present a great opportunity for improving accessibility on the Web largely because of their ability to create custom elements and extend existing HTML elements. This helps accessibility because HTML elements have built-in behaviours. It is possible to extend e.g. the `<button>` element and add all kinds of extra `<div>`s for hanging styles off and so on, but without having to reinvent the base element. JavaScript is needed to define and register the *fancy-button* element, but the basic mark-up would be `<button is="fancy-button">`. As such, user agents that do not support JavaScript would fallback to showing a `<button>`. With this form of abstraction, developers do not need to add anything to the button element to get these behaviours.

Web components are quite well supported in AT. Although the *Shadow DOM* can hide things from the real DOM of the page, the browser mashes them all together to render them, and AT sit on top of browsers. The fact that components are encapsulated in the *Shadow DOM*, do not make it better or worse than if they were in the page in the traditional way (Lawson, 2014).

Screen readers can also access content in *Shadow DOM* without issue. The *Shadow DOM* can in fact be traversed and any node within it has a *shadowRoot* property, which points to its shadow document. Most AT hook directly into the browsers rendering tree, so they just see the fully composed tree (Faulkner, 2012; Sutton, 2014).

The *Shadow DOM* is navigable by keyboard. The *Shadow DOM* drop-down and button widgets were fully keyboard accessible and responded to the same input as their regular DOM counterparts. New standards are developed and there are problems and bugs that must be addressed over time. *Shadow DOM* may have had issues at one time, but are now accessed by everyone (Sutton, 2014).

It is important to add WAI-ARIA information and *tabindex* to the custom elements. Invention of new types of objects is faster than standardized support for them appears in web languages. WAI-ARIA supplements semantics in supporting languages to clarify semantics to AT when authors create new types of objects that are not yet directly supported by the language of the page. Given that the point of *Web components* is to add new types of objects, via style and scripts that are not yet directly supported by the language of the page, applying WAI-ARIA seem very appropriate (Lawson 2014).

Inclusion of WAI-ARIA roles, states and properties in content wholly inside the *Shadow DOM* works fine. The accessibility information is exposed correctly via the accessibility API. Section 7.5 of the *Shadow DOM* spec actually states that:

“User agents with AT traverse the document tree as rendered, and thus enable full use of WAI-ARIA semantics in the shadow trees.” (Faulkner, 2012)

One issue with *Web components* is that it is not fully supported in browsers at this point.^{12 13} However, *Polyfills* may be a solution to this. This concept is gaining traction with browser

¹² [http://caniuse.com/#search=Web components](http://caniuse.com/#search=Web%20components)

¹³ [http://caniuse.com/#search=custom elements](http://caniuse.com/#search=custom%20elements)

developers. Essentially, a *Polyfill* is code that provides support for a web specification that the browser is expected to support natively in the future. *Polyfills* have already been used to enable support for HTML5 features in older browsers. Now Google is working on a *Polyfill* that will make at least some features of *Web components* work on existing rendering engines. In fact, one way to look at *jQuery*, and similar toolkits is that they were early *Polyfills* for some of what *Web components* aim to do. One might think of them as a *to-do list* for the browser vendors. If *Web components* become a standard, browsers can cross that of the list (McAllister, 2014).

It is important to remember that *Web components* will only be as accessible as they are made to be. It is essential when using a *Web Component* that the developers establishes its accessibility, and makes changes if needed. Because anyone can make a *Web component*, it is likely that the quality will be low on many of them. It is the hope however, that over time a considerable amount of accessible *Web components* will be built and reused to efficiently ensure accessibility of that component.

6.13 Result of literature study

Studying possible solutions for RIA accessibility issues resulted in the following list of guidelines:

Process oriented guidelines

- Have accessibility experience on the team
- Implement accessibility from the beginning of a project
- Test accessibility at key stages
- Follow existing principles for good design and user experience

Technology oriented guidelines

- Follow and validate the HTML5 standard
- Apply progressive enhancement
- Apply unobtrusive JavaScript
- Apply tidy coding
- Make sure methods are independent of input device
- Apply WAI-ARIA mark-up
- Create or apply accessible *Web components*

For a detailed presentation of the first version of the guidelines see *appendix 1*.

7. Findings: Evaluation of guidelines

In this chapter the findings from the evaluation of the guidelines will be presented. Firstly, there will be a review of the answers to the questions in the interview guide. Then positive feedback and negative feedback will be presented along with suggestions for changes and suggestions for additional guidelines. Lastly, suggestions for how to present the guidelines are introduced.

7.1 Review of the answers to questions in the interview guide

Table 1 shows the participants' background. Table 2 gives an initial overview of the findings from the interviews. The following chapters will elaborate the findings presented in Table 2. Finally, there will be a presentation of positive comments and suggestions for changes.

Participant	Profession	Level of expertise	Familiarity and use of tools	Time spent on guidelines
1	Front-end developer and interaction designer	Knows more than most	Highly familiar Uses somewhat	2 hours
2	Interaction designer	Knows more than most	Highly familiar Uses somewhat	15 minutes
3	Programmer	Knows a little bit	Somewhat familiar Does not use much	2 hours
4	Accessibility consultant	Among top five in Norway	Highly familiar Uses a lot	2 hours
5	Programmer and consultant	Knows more than many others	Somewhat familiar Has used to some degree	1 st round 40 minutes 2 nd round (in depth) 5 hours

Table 1: Participant background

Participant	Understood guidelines	Learned something	Found guidelines useful	Found guidelines concrete	Are guidelines reliable	Use guidelines in future projects
1	Not all	Some on WAI-ARIA	Inspiration for own guidelines	Yes and no	No, they are not exhaustive	Part of them
2	Mostly	Some technical issues	Content yes (not presentation)	Yes and no	To some extent	Part of them
3	Not all	Yes	Yes	Yes and no	Yes	Part of them
4	Mostly	New perspective	Indirectly	Yes	Yes, but needs updating	Indirectly

Table 2: Summary of findings from interview guide

7.1.1 Did the participants understand the guidelines?

Most of the guidelines were understood, both what they entailed and why they were relevant for accessibility. It was mentioned that although some guidelines were difficult to comprehend immediately, they were presented in a way that made it easy to find further information.

However, the guideline *Apply tidy coding* (appendix 1, guideline 2.4) was thought unnecessary because what it stated was basically covered in the *Unobtrusive JavaScript* guideline (appendix 1, guideline 2.3). The part about separating *stylesheets* within the *Progressive enhancement* (appendix 1, guideline 2.2) was also criticised. The participants failed see how this would facilitate accessibility and found it confusing and unnecessary. The guideline concerning *Unobtrusive JavaScript* also puzzled some of the participants. They failed to see how this benefits accessibility. It was discussed that this should be more clearly stated within the guideline.

7.1.2 Did the participants learn something?

All the participants learned something about accessibility when evaluating the guidelines. They mentioned learning about WAI-ARIA, *Web components*, how to bring accessibility in to the development process and also getting many excellent advice on testing for accessibility. It was said that the guidelines give a good overview of the discipline. One participant mentioned feeling a lot more equipped and competent to be responsible for accessibility in a project after reading the guidelines than before. It had given him a better understanding of the interdisciplinary nature of accessibility. It also gave some new perspectives on how to think about accessibility as a part of usability and user experience, and what developers need to focus on.

7.1.3 Are the guidelines concrete enough?

It was commented that the level of concreteness was not even throughout the guidelines. For example the guidelines about WAI-ARIA or *Unobtrusive JavaScript* (appendix 1 guidelines 2.3 and

2.6) were quite extensive and concrete, whereas for example *Implement accessibility from the beginning of a project* (appendix 1 guideline 1.2) was more diffuse and less detailed. In general, the *process oriented* guidelines were deemed less concrete than the technical ones.

However, the level of concreteness was mostly satisfactory, and it was mentioned as a particularly strong suit.

7.1.4 Are the guidelines reliable?

The guidelines were deemed reliable within the reach of the scope. It was believed that they could contribute in a project process and help ensure accessibility and that using them would create a higher ground level of accessibility in general and overcome typical recurrent problems like *keyboard navigation, contrast and form handling*. One participant had tried to apply them in a process at his place of work with some success. It was suggested that if more people on the team read through the guidelines they would understand the importance and also understand their own role in ensuring accessibility. This was believed to be the most important because it can give motivation to go into the concrete examples and learn more.

However, there are some challenges with keeping the guidelines updated and reliable in the future. This has to do with the level of technical concreteness. Though this can be a good thing, it will also cause the guidelines to become outdated faster than if they were less concrete.

7.1.5 Are the guidelines useful in the participants work?

The content of the guidelines was mostly found useful. It was said it gave a good overview of the discipline and contributed to highlighting *why* and *how* one should work with accessibility. It also was thought they could serve as an inspiration for how to bring accessibility into the web development process.

The accessibility expert said that if the guidelines became commonly used it would free up his time to focus on the special cases that might be individual to each project. This means he would get to use his competence to a greater extent.

One participant was however somewhat sceptic. He thought the guidelines represented an ideal process and believed that the complex situation in the industry and the many different companies involved in a project would make them difficult to use. He explained that sometimes it was the clients who chose the technology, and in some firms they only used one specific system because that was what all the employees are familiar with. In his opinion this leaves too little control over the project to be able to follow the guidelines.

7.1.6 Can participants use the guidelines in future projects?

All participants believed they could use at least parts of the guidelines in their future projects and one had, as mentioned, already used parts of them successfully. It was believed that if the guidelines are completed and some changes are made to the presentation they will be very useful and become something that could be shared between web developers. It was generally believed, however, that the guidelines could not be used exclusively, but as a supplement to point at certain

things. They might be used as a reference works to learn more or check if one has forgotten something.

The accessibility expert pointed out that he would not use the guidelines directly. However, he could use them as a resource for developers when he his is working with development of accessibility competence and holding courses. He believed they would be useful as such.

7.1.7 How are the guidelines different from existing guidelines?

The participants were all familiar with WCAG. Other guidelines mentioned were the *Difi quality criteria for digital services* and *UU-skolen for nettsider*. WCAG was the guideline the participants mostly compared with when presented with this question.

It was believed that these guidelines are more applicable than WCAG because they are not so comprehensive and difficult to read and one gets faster to some concrete advice on what to do. The downside is that the consequences of the different guidelines are not explained. It was mentioned how these guidelines are more specific in relation to the theme, directly linked to JavaScript, *front-end* and RIAs. They do not cover all that WCAG does, but it was viewed as beneficial to have a clear focus. It makes the guidelines more manageable and less time-consuming to get familiar with.

It was believed that these guidelines links the requirement to the description of what to do in a clearer way than WCAG does. This was thought to be both because WCAG covers so much material, but also because these guidelines have a more pedagogical presentation form. To understand what to do in WCAG one has to go to the descriptions of what techniques to use. These guidelines focus directly on the techniques from the beginning.

The guidelines were believed to have more in common with the *Difi quality criteria for digital services*, which are quite concrete and mentions specific tools to work with. The *Difi quality criteria for digital services* also go beyond WCAG when something has been noticed to raise the user experience in such a degree that it should be a criterion. These guidelines do something similar. They are partly more specific expression of what WCAG says and partly go beyond WCAG. The point about *Progressive enhancement* was especially mentioned. This is not well founded in WCAG. A lot of the thought behind *Progressive enhancement* is not described in WCAG whereas this is much clearer in these guidelines.

One of the participants gave a rather compelling metaphor. He compared WCAG to *The Complete Works of William Shakespeare* which he every once in a while picked up to read a specific sonnet to learn by heart. This could be done with the WCAG points that are relevant for each project. *These* guidelines however, he described as a *short story* designed to read all at once. They give an overview of what is needed to ensure accessibility within the *front-end* scope and at the same time tell the user where to look to find more details.

Lastly, the *process oriented* guidelines were pointed out to be quite unique. There are several guidelines concerned with how to technically ensure accessibility, but none addressing how to bring accessibility into the development process. The need for this was much appreciated. The participants also appreciated that each guideline was linked to one or more professions within the web development team because it facilitates quick access to the guidelines relevant to their own profession.

7.2 Positive feedback

There was thus an overall positive attitude towards the guidelines. The shortness and concreteness was, as mentioned viewed as a strong suit. The content was seen as informative and educational. The links offered provide more in depth information were also appreciated. Not everyone is very good at finding information on their own, and being lead to more useful information can therefore be very valuable. This also makes the guidelines less comprehensive while still providing enough information when working with a specific guideline. It was encouraged to provide even more links and explanations. It was also commented positively on the fact that each guideline is directed to the different roles within the team. It was believed this makes them easier to relate to and more efficient to apply.

7.2.1 Appreciation of the process oriented guidelines

There was a general appreciation of the *process oriented* guidelines. A need for guidelines focusing on *how* to incorporate accessibility into the development process that are transferable to different processes was recognized. The participants had not seen something like this before. It was also appreciated that some ways of working with accessibility results in more accessible applications than others. These guidelines were viewed as an excellent project methodology. Things like introducing accessibility from the beginning, testing for accessibility and having expertise on the team were all deemed good practises. The concrete advice on testing was found useful, e.g. about users who might need special equipment or to do the testing in their home or workplace or what tools to use for different tests. It was suggested hiring accessibility expertise when necessary or educating team members on accessibility. It might however be more efficient having *one* accessibility expert rather than everyone on the team trying to figure out their responsibility. This way the expert could educate the others. This is an interesting point that deserves further research. Expertise was believed to be especially valuable when testing with users with disabilities. This person would know for example how to interpret if something was an actual problem or if it was just that the user had an individual approach, or if the testing team was over-interpreting a standard use of a screen reader. It was also thought that experience with testing, tools, screen readers and other methods would both save time and raise quality of the testing. Whether or not this expert is a permanent part of the team or is hired to do the job when necessary might be of lesser importance. However, the latter demand some knowledge on the project managers' part of when it is necessary to hire an expert.

7.2.2 Positive comments on the technical guidelines

Many of the technical guidelines also received praise. The guidelines about WAI-ARIA, *Progressive enhancement* and *device independent methods* were highlighted as especially informative. It was

commented that the detailed focus on WAI-ARIA is logical because of the focus on *Rich Internet Applications* (RIA). The accessibility expert especially highlighted the part about *tabindex* and said it was very much in line with what they believed at his consultant company. It was also mentioned that it was good that there was a warning about not using *tabindex* as a replacement for good structure. It was further believed that using *Progressive enhancement* is a very important aspect of working with accessibility and is the basis for working with script and dynamic, interactive applications. *Device independent methods* ideally make an application usable no matter what input device is used to navigate or interact with it.

Other guidelines that were positively commented were the ones about HTML5, how to use frameworks and tools for accessibility and *Unobtrusive JavaScript*. There was agreement that following the HTML5 standard seemed reasonable and some of the participants had thought about the benefits HTML5 semantics would have for accessibility. It was pointed out that mentioning qualities about frameworks that facilitates accessibility was very useful. This could be used as one of the filters when searching for a suitable framework. It was believed that many developers make the mistake of choosing frameworks based on what “everyone” uses or what they themselves prefer to learn. One of the participants said the guideline about *Unobtrusive JavaScript* was very educational and useful. It offered some practical concepts and reminders, e.g. about how to think about users and browser when developing. The importance of thinking of users and browsers when working with different JavaScript frameworks was highlighted because frameworks sometimes change or ruin conventional navigation methods.

7.3 Negative feedback and suggestions for changes

There was some negative feedback and several suggestions for changes to the guidelines. Some of the feedback was general relating to the guidelines as a whole, and some were directly linked to each guideline. This chapter will focus on general feedback. The following two chapters will present feedback directly linked to each guideline, first the *process oriented* ones before the *technology oriented* ones.

7.3.1 Introduce the guidelines and clarify scope

There was a strong request for an introduction to the guidelines. This introduction should explain the *front-end* and *process* focus clearly and highlight why there is not much focus on interaction design or visual accessibility. Some of the participants thought interaction design and visual accessibility should be included in the guidelines, but understood the need for a scope within the project. It was however mentioned that the scope causes the guidelines to not be exhaustive and will only ensure accessibility within the scope. This should be highlighted in the introduction. The introduction should also address how to use the guidelines in a development process and why the level of detail is somewhat uneven.

7.3.2 Adjust level of detail

Although it was appreciated that the scope allowed for going into more detail on some guidelines and not on others, it was suggested to adjust the level of detail. It was pointed out that when some guidelines are much larger and more detailed than others, they might seem more important.

There might be a danger of ignoring an important guideline just because it is small. The guideline about *device independent methods* was brought up as an example of a small, but very important guideline which might benefit from being expanded.

7.3.3 Provide explanations of how the guidelines benefit accessibility

Explanations within the guidelines about what their uses are *why* they are important and *how* they benefit accessibility was much sought after. It was commented that there is always time pressure in projects, and being able to explain *why* one is spending time on something is essential. Another important factor is that it is more difficult to *not* do something if there is an awareness of why it is important. This issue was carefully considered before presenting the guidelines for evaluation. This finding indicates that it was a mistake not providing explanations of how the guidelines benefit accessibility within the guidelines. A different solution to this question will have to be considered.

7.3.4 Link the guidelines to similar existing guidelines

It was suggested to link the guidelines to guidelines covering some of the same issues, e.g. WCAG or the *Difi quality criteria for digital services*. This might be a good idea as it could make the guidelines more recognisable to some users and add value as a way of providing further information and different examples. It also gives a basis for background information which adds weight to the guidelines. It was also mentioned that developers will have to follow WCAG as well as these guidelines because WCAG deals with other things that are outside the scope of these guidelines like *contrast*, *font-size*, *alt-text* and so on. This should be clearly stated in the guidelines.

7.3.5 Prioritize and estimate time use on the guidelines

It was proposed to estimate the guidelines reliability and place value on them. This could result in a prioritised list where the most important guidelines are presented first. This will help the team to know what they *must* do as opposed to what they *can* do to ensure accessibility. It might also help the team decide when to hire an accessibility expert, what that person should be hired to do, and how much time he or she would need. However, for this project a prioritization cannot be conducted because the reliability of the guidelines would have to be established first. It is also uncertain whether these guidelines are fit for prioritization the same way for example WCAG is.

It was further suggested estimating how much time it would take to work with each guideline. Although this may be valuable for project managers and a definite improvement to the guidelines, it is hard to do without putting them to use in projects and making further studies of this.

7.3.6 Provide more examples

A suggestion was made of providing more examples throughout the guidelines, not just coding examples. It was thought this would make them more usable and easier to understand. Providing more examples overall may also ensure a more equal level of detail. One participant pointed out that he had to do some research on his own to find good examples and it would be beneficial to have access to them directly from the guidelines. While this may be true, finding good examples for the different guidelines that are concrete enough to be helpful and at the same time abstract

enough to be transferable will demand a great deal of work and perhaps some further studies as well.

7.3.7 Changing the main title

It was recommended to change the title from *Building accessible RIAs using JavaScript* (appendix 1) to *Building accessible Rich Internet Applications* because this would make the title more readable and easier to pinpoint to a reader not familiar with the acronym RIA. It was also believed that having JavaScript in the title gave a focus to JavaScript that was not reflected in the guidelines.

7.4 Suggestions for changes to process oriented guidelines

There was a request for more in depth explanation within the *process oriented* part of the guidelines. The participants wanted to know more about how to bring accessibility in to all parts of the process and suggested clarifying this even more by giving more details and examples. It should be noted that it was the participants working with bringing accessibility into their own processes that requested this. They have first hand knowledge of how difficult this can be. It is therefore a safe assumption that the *process oriented* part in fact *would* benefit from further development.

It was pointed out that this is a difficult issue and less straight forward than the *technology oriented* guidelines. There have not been done many previous studies in this area. For further development, the *process oriented* guidelines would have to be brought into different processes to see what works and what does not work. A general analysis of development processes would also be valuable to uncover problem areas and how to address these issues. There might not be definitive *right* and *wrong* answers because it depends on the individual processes. Nevertheless, in the following some concrete suggestions for improvements are presented.

7.4.1 Point out that accessibility is interdisciplinary

It was mentioned that it should be pointed out that accessibility is interdisciplinary and therefore the different team members need to be aware that they have different responsibilities when it comes to accessibility. One participant referred to seeing project leaders give a design bureau the entire responsibility for accessibility, not realizing it is interdisciplinary, and the design bureau saying *ok*, not realizing they would have to work within programming and content as well as design and process design. This approach may result in accessibility only being taken into account within the design, but ignored during the implementation of the web application.

7.4.2 Mention that accessibility should be a part of the specification

It was suggested mentioning that accessibility needs to be a part of the specification. The participants had seen many obscure references to accessibility in specifications on different projects. Simply stating that accessibility is a requirement or that WCAG should be followed is not enough. It must be specific enough for developers to be able to measure fulfilment. If not, accessibility requirements will most likely be very randomly implemented. One has to look at the interface and the planned functionalities and determine what criteria need to be fulfilled. Only

then is it possible to decide what WCAG points are relevant to that specific project and get detailed requirements, e.g. forms that are navigable by keyboard.

It was highlighted that the specification is not just the responsibility of the project manager. Having a clear specification is essentially the responsibility of the procurer or project owner. The problem is that owners often have no idea how to create that which they are ordering. Therefore, one can often see that when it comes to accessibility it fails already during this phase of the project.

7.4.3 Elaborate on optimizing search functionality

It was suggested saying something about more about optimising search functionality, e.g. mentioning few hits rather than long lists and giving the user the opportunity to view more hits of a certain type or that search phrases should be highlighted. This is however, for now outside the scope. It is the area of an information architect, which will know very well how to do this. The important thing here is stating that optimising search functionality is beneficial to accessibility because it facilitates the information seeking process.

7.4.4 Revise guideline on introducing accessibility from the start

It was mentioned that the guideline about *introducing accessibility from the beginning of a project* (appendix 1 guideline 1.2) might be a bit too comprehensive and shallow. It covered everything from planning to specific frameworks without going into depth on any of it. It was suggested to provide some more details and remove the part about frameworks, as it did not seem to belong with this guideline. It was believed this would make this guideline easier to apply and relate to.

There was also a suggestion to highlight how important it is to introduce accessibility from the beginning because when the project is handed the programmers, most of the concept is already thought through. If the design and interaction design is not accessible, then it does not matter how accessible the code is, using standards, adding WAI-ARIA and so on.

7.4.5 Suggestions for changes to guideline about accessibility testing

Many detailed suggestions were made to the testing guideline. They were mostly excellent suggestions and not too complex or time-consuming to implement. It was mentioned that it is often the front-end developers that do the testing, not always dedicated testers. Therefore, the testing guideline should be aimed at front-end developers as well as testers. It should however not be aimed at project managers because they would be more concerned with how much time to spend on testing and the possibility to hire an expert to do it rather than the actual testing and what tools to use.

It was further mentioned that although it is a good idea to test in *iterations*, this is a general advice and not directly linked to accessibility. There are however other points in the guidelines that benefit other things in addition to accessibility. One example is the advice on focusing attention on one aspect at the time and test a limited part of the system. This advice was appreciated. It was however mentioned that it applies to testing in general and not only to *simulation*.

7.4.6 Comments on simulation

Lastly, it was brought up that *simulation* is a part of *expert based* testing and should be placed beneath that. In connection with *simulation*, it was also suggested mentioning that it might be difficult to turn of JavaScript in new browsers and that on *Macs* the keyboard is by default not enabled in browsers.

7.4.7 Comments on testing with automatic tools

There were some concrete suggestions for testing tools. One was *Selenium*, which automates browsers and checks how an application works in different browsers and devices. Another was *Sim daltonism* for Mac which quickly tests the colours in relation to colour blindness. It was also suggested presenting the tools in a table explaining what uses they had and what areas of accessibility they could analyse.

7.4.8 Suggestions for user testing

It was suggested highlighting the importance of tasks being the same when testing with users with disabilities as with user testing in general. However one might communicate a bit differently, for example say *choose the link* instead of *click on the link*, talk about content, rather than colour or structural placement rather than visual placement. It is important that the test manager keeps in mind that the user might use different methods of interaction.

It was also pointed out that many people with disabilities do not have difficulties coming to a testing facility. They might not use special equipment, or it is small enough to bring along. Therefore, it is not necessarily essential to do the testing at their home or place of work.

It was further suggested saying something about how many users to test against when performing user testing. The literature states that ca. five persons will be able to find 80% of the most important errors (Lazar et. al., 2010 pp. 263). However, how many persons to test with when testing against users with disabilities have not been established. This should be further studied. One participant had some experience cooperating with a firm dedicated to testing with persons with disabilities. They usually had five participants; one blind, one visually impaired, one with dyslexia, one old person and one with motor difficulties. It would be possible to mention something about this in the guidelines. However, this does not take into account that people with disabilities may have different levels of IT literacy and different Internet habits.

7.5 Suggestions for changes to technology oriented guidelines

There were many suggestions for changes to the *technology oriented* guidelines. Many of them are quite detailed, but at the same time important for increasing the quality of the guidelines. They are presented in the following.

7.5.1 Mention that WAI-ARIA is no substitute for good code

It was mentioned that it should be highlighted that WAI-ARIA should be not used as a substitute for good mark-up. Adding WAI-ARIA to poor code may satisfy conformance checks, but it does not

under any circumstances deliver an accessible application. This should absolutely be given focus in the guidelines, as it is one of the pitfalls of adding WAI-ARIA.

7.5.2 Highlight the exceptional qualities of WAI-ARIA

There is some functionality that WAI-ARIA is the only language providing at the moment, e.g. handling updates and notifications with *live regions*, and extension of *tabindex*. It was suggested that this should be highlighted. *Tabindex -1* is relevant every time one wants to set focus to something, not just when using script. In addition, WAI-ARIA is not just relevant for scripting because it also can be used for conveying HTML5 functionality to assistive technology (AT). Examples are *sliders* or *extend/collapse* functionality that exists in HTML5, but without the possibility of communicating this to a screen reader. These suggestions represent a good opportunity to underline why using WAI-ARIA is beneficial, how to use it and when it is relevant.

7.5.3 Comment problems with WAI-ARIA

There was a request for mentioning some issues with WAI-ARIA, for example that it is not supported in all browsers and screen readers. However, the literature states that WAI-ARIA is now mostly supported, and it should not be a problem using it. One solution might be to link to some articles about this and give a warning that it might be an issue. If there is a problem that WAI-ARIA is not compatible with AT, this might be an even greater problem with HTML5, which only recently became a standard. Even so, literature on accessibility highly recommends using HTML5 and WAI-ARIA in combination.

7.5.4 Recommend following standards in general

It was suggested mentioning something about following standards in general, as a good way to ensure accessibility. This is because development of AT is done according to standards. This is a valid point and there might reason to clarify this further. There could be a separate point called *Follow standards* explaining that development of AT is done according to standards and that following them allows for better compatibility with different AT.

It was also proposed to clarify further the importance of being true to the HTML standard. Although there might be ways to clarify this even more, the guidelines already state that developers should follow and *validate* the HTML5 standard (appendix 1, guideline 2.1). To validate the standard needs to be followed.

7.5.5 Mention the most common HTML5 input types

Questions were raised as to why not *all* new HTML5 input-types were mentioned in this guideline. It was suggested that *<search>* and *<email>* are probably more commonly used than *<range>*. This is a valid point. It is a good idea to address the most commonly used input-types in this guideline. It is both more recognisable and more helpful for developers. Addressing all input-types might be too much, which highlights the importance of mentioning the most frequently used. It was also suggested illustrating the importance of correct input types with an example, e.g. that if using the right input types a different keyboard pops up for *numbers*, *email* and *search* on touch devices. Another good example is how this can be used for automatic validation.

7.5.6 Be critical towards <section> and <article> and drop recommending <canvas>

It was mentioned that the use of the elements <article> and <section> should be critically examined. The reason for creating these elements in the first place, was to give an extra way to structure information as an aid to screen readers. Originally, it was meant to be one *h1* inside the document and then start with *h1* again within each <section> or <article>. The screen reader was meant to understand that when reaching a <section> or <article> it should interpret the heading levels within this element separate from the rest of the document. The problem is that not all screen readers implement and interpret this correctly. For some users it looks like *h1*, *h2*, *h3*, but for others it became *h1*, *h1*, *h1*. The recommendation now is to use heading levels as if one was not using <section> and <article>. That means the functionality of these elements disappears. In addition, the screen reader tells the user every time it reaches a <section> or an <article>, which causes unnecessary reading and may cause annoyance. It was believed that the guidelines should either not mention <section> or <article> specifically, or link to a closer, critical look at the issue. These should be further examined. The literature study has not revealed their issues, but the participant suggesting this has years of experience working with accessibility in practice, and his opinion should be taken into account. Therefore, there should probably be placed a warning within the guidelines.

Several of the participants questioned how accessible <canvas> is. They recommended examining this further before promoting it. Some further studying showed this was not necessarily recommended by all accessibility experts (O Connor, 2012 pp. 218).

7.5.7 Mention issues with HTML5 element <nav> and WAI-ARIA role *navigation*

It was mentioned that there is an issue with the HTML5 element <nav> because it does not work well with the WAI-ARIA role *navigation* when using *Windows Eyes* version 8 with Internet Explorer (IE). It was recommended only using <nav> and not *navigation*, since this HTML5 element has achieved a wide approval and is well handled. Although this is quite detailed, it might be advisable to put in the guidelines.

7.5.8 Explain what Progressive enhancement entails for accessibility

Progressive enhancement was believed to be a technique that can be relevant in many situations. It was suggested to clarify and give examples of what the different aspects of what *Progressive enhancement* entails for accessibility. The examples should be typical challenges, for example using *onClick* with a *div* instead of with a *button* or making sure a form is usable even if JavaScript crashes. This is a good idea, which will add quality to this guideline, and make sure *Progressive enhancement* is properly explained, not just how it benefits accessibility, but also how to use it accurately. This will give this guideline depth and also make it more user-friendly and helpful for the developer.

7.5.9 Clarify that HTML provides basic functionality in addition to content

It was suggested to adjust the text beneath *Progressive enhancement* to clarify that not all functionality is handled by JavaScript. HTML is not just content, but also basic functionality. It is

possible to provide acceptable fallback solutions to a greater extent than before. When HTML can handle functionality, this is what should be used. Even though users might have JavaScript, it could still be failing from time to time. JavaScript should just be *enhancement* of functionality. One example is *search*. The *search* should work with HTML, but *suggestions for search*, which improves the user experience, needs JavaScript. It is very important to get this right.

7.5.10 Recommend using JavaScript to hide content

It was mentioned looking at the possibility of presenting all content to begin with and then use JavaScript to *hide* it. One example is *extend/collapse* functionality. This way everything would be loaded as extended, but as soon as JavaScript kicks in it collapses and the user needs to interact to extend it. That means the user would have access to all the content even if JavaScript was disabled. Whether or not this is a good solution would depend on how much data is presented. This needs further study before recommendation. In any case it would have to be underlined in the guidelines that before using this technique a study would have to be made of how much data and what kind of data the application hold. There might also be other solutions that work better for this kind of functionality, like perhaps partly server rendering the site. This is a question that would have to be answered from project to project.

7.5.11 Progressive enhancement in CSS is somewhat outdated

There was a lot of scepticism towards *Progressive enhancement* in CSS. The participants failed to understand the importance and benefits of this. It was suggested adding something about what the different *stylesheets* should contain and explain why the CSS should be divided as described (appendix 1, guideline 2.2). One participant wondered if the fact that the developer has a clear overview of his code has to do with accessibility. Although this will help with maintenance, the main point with this guideline is creating something that works across browsers and devices. This could be more clearly expressed in the guidelines. However, the participants mentioned using frameworks like *SASS*¹⁴ or *LESS*¹⁵ to ensure cross browser functionality. Division technique was used when working with these frameworks because it is easier for the developer to work with shorter code documents. Afterwards however, everything is compiled and sent to the browser in *one* CSS file. Therefore, this section in the guidelines this might be somewhat outdated and should be updated.

Questions were raised as to why *CSS structure* was relevant within the scope of these guidelines. CSS is generally related to presentation, and it had been decided to keep this aspect outside the scope. There was no mention of for example *font-size* and *contrast*. This however has to do with responsive design that is usable on different devices and across browsers which is highly relevant for accessibility. Nevertheless, it can still be argued that it is outside the scope. For this reason and the fact that this needs extensive further studies and update to be useful, the part about *Progressive enhancement in CSS* had been removed from the guidelines for now to avoid confusions.

¹⁴ <http://sass-lang.com/>

¹⁵ <http://lesscss.org/>

7.5.12 Simplify language about Unobtrusive JavaScript

The general opinion was that the points concerning *Unobtrusive JavaScript* should be made clearer, more detailed and put into simpler words. Some also wanted more explanations and examples on conventions for interaction like *pinch/zoom* or *swipe*. It was suggested to link to a list of shortcuts that work in all browsers and explaining what the requirements are for this to work and also adding an illustration of an event order, and perhaps also explaining the concepts of *event bubbling* and *event capturing* as well as the combination of these.

7.5.13 Clarify how Unobtrusive JavaScript benefits accessibility

It was commented that it should be clarified why *Unobtrusive JavaScript* (appendix 1, guideline 2.3) is important for accessibility. It was understood that the thought behind is that clean and clear code also makes it easier to create accessible code. This principle is relevant no matter if *pure JavaScript* or a framework like *Angular*¹⁶ or *React*¹⁷ is used. Other ways *Unobtrusive JavaScript* facilitates accessibility are clean and clear code makes it easier to create accessible code, not making JavaScript a requirement for a functional application, only using it when absolutely necessary and only add it to enhance the user experience, not to provide basic functionality. This should be highlighted in the guidelines. It should also be mentioned that accessibility is only *one* of the benefits to *Unobtrusive JavaScript* and that *Unobtrusive JavaScript* is not automatically accessible.

It was also commented on the level of detail in this guideline and that it might be a bit too detailed compared to the rest of the guidelines. This is nonetheless an important point with many examples. To some extent it is the code examples that make this guideline so detailed and comprehensive. These examples do serve their purpose. They might however, be shortened somewhat down. It might also be a good idea to link to more examples and not keep all of them within the guidelines. It can however be argued that it is logical that the level of detail is more excessive within this guideline considering the scope of this project.

7.5.14 Do not *not* expect JavaScript to be available

It was expressed that not expecting JavaScript to be available was too excessive and not rooted in reality. Everyone uses JavaScript at this point. It was suggested rather using partly *server rendering* as a solution to render have fancy RIAs without JavaScript. This should however not be used instead of *Unobtrusive JavaScript*. Although most users have JavaScript available at this point, it may still fail. Not expecting JavaScript to be available is merely a way of thinking when creating something so as to not make everything dependent on JavaScript. Perhaps this should be re-phrased to be expressed better. However, this point us about assumptions, and that they should not be made

¹⁶ <https://angularjs.org/>

¹⁷ <https://facebook.github.io/react/>

7.5.15 Do or do not to use pure JavaScript in coding examples?

One participant disliked providing code examples in *pure* JavaScript (appendix 1, guideline 2.3). There are many ways of doing the same thing, and if that particular example is not relevant, the developer might not be able to do anything with that guideline. It was thought *pure* JavaScript examples were irrelevant when using JavaScript frameworks or libraries. It was therefore suggested explaining that this is just one of many ways of doing something and adding value by explaining *why* this is important for accessibility and link to other examples. Another solution may be interactive examples where users can try out for themselves and see results immediately. This was one of the suggestions from the participants. This way the users can try for themselves and see results immediately. It might also be possible to use different libraries and frameworks for these examples. This is a very interesting thought, inspired by for example W3Schools¹⁸ and *JSFiddle*.¹⁹ It might however need too much work to be feasible in this process. For now, it might be enough to link to *JSFiddle* and let the user try it out there.

It is important to keep in mind that it is the principle behind the examples that are important when it comes to *Unobtrusive JavaScript*. The principle might be constant, even if the technology changes. A front-end developer should be able to transfer the principle into the code he is working on. This was appreciated by another participant who thought use of *pure* JavaScript made the examples more transferable to other situations than using specific frameworks. It may however be necessary to explain this more clearly within the guideline.

7.5.16 Use a common example and reuse it throughout the guidelines

It was suggested having one or two examples explaining different aspects throughout the guidelines. This would make the examples recognizable immediately and would illustrate better the differences in the coding examples and what the difference entails. It was also proposed to make the examples smaller and less comprehensive using for example a link or a menu as these are common features in all websites.

7.5.17 Remove tidy coding

The participants wondered about the guideline concerning *tidy coding* (appendix 1, guideline 2.4). They thought it was very short and did not understand the point as they failed to see how it was different from *Unobtrusive JavaScript* and *Progressive enhancement*. Although it might seem similar, *tidy coding* is a concrete advice on how to make code clear and orderly and easier to work with. It is understandable however, that it is confusing that this is a separate point. One participant suggested placing this guideline within the guideline about *Unobtrusive JavaScript*, but most thought it best to simply remove this guideline. This might be for the best as it was not appreciated by any of the participants, and others, more comprehensive guidelines give some of the same advice.

¹⁸ <http://www.w3schools.com/>

¹⁹ <http://jsfiddle.net/>

7.5.18 Elaborate on device independent methods

There was a strong request for more elaboration about *device independent methods*. It was suggested mentioning that all functionality should be usable with different input devices, mainly *mouse*, *keyboard* and *on touch*. This will clarify this point and make the users quickly see the benefits of this. Another suggestion was giving a short description of when it might be smart to use the different methods. One example is how *onBlur* is good for *live validation* in forms because the validation will happen as the user skips from one input field to the next. This will add value and give the users a deeper understanding of the importance of *device independent methods*. It might however, need some further studies to be complete and accurate.

There was also a call for mentioning more methods. Methods like *onSelect* and touch methods like *touchstart*, *touchmove*, *touchend* and *touchcancel* were suggested. These should absolutely be mentioned in the guidelines. It was mentioned that *onClick* works with *touch* because mobile browsers are designed to cope with the large amount of existing websites that were not developed specifically for *touch*. There is however a delay when activating *onClick* with touch. The guidelines should offers developers information about this.

It was suggested mentioning what pitfalls to avoid, for example using *onMouse* methods for everything and make the application completely inaccessible to keyboard users or not ensuring that *hover-effects* work when *focus* is placed on the element. It was further proposed to place a warning in the guideline explaining that the methods *onFocus* and *onChange* go against WCAG (3.2.1. and 3.2.2). An example is *onChange* with a drop-down menu that is usually navigated with the arrow keys. The problem occurs when using the arrow keys directly and there is an *onChange*, the site might reload immediately when using the arrow key to reach an element and focus is moved to the beginning of the page. The same goes for *onFocus* if there is a change immediately when focus is placed. These are good suggestions, which should be included in the guidelines.

7.5.19 Do not recommend frameworks that are basically not in use

Some of the participants had researched the frameworks suggested in the guidelines and concluded that, except the *jQuery* library, they were hardly used. It was also mentioned that although some frameworks might not be very good with accessibility, they might offer external frameworks that adds accessibility on top. One example is *Bootstrap*²⁰, which has a plug-in²¹ that adds accessibility to all their default plug-ins.

It was suggested recommending *React*²², an up and coming JavaScript framework. This framework was believed to be very accessibility-friendly because it allows for rendering whole or parts of a RIA on the server depending on what device is used. This might also be beneficial for screen readers and other AT. The newest version of *React* also warns the developer when writing inaccessible code, e.g. putting an *onClick* on a *span*.

²⁰ <http://getbootstrap.com/>

²¹ <https://github.com/paypal/bootstrap-accessibility-plugin>

²² <https://facebook.github.io/react/>

7.5.20 Do not recommend specific frameworks at all

On the other hand, there was some scepticism towards suggesting specific frameworks because this is dated information. Frameworks change very quickly. It was suggested rather mentioning some concepts that it is important that the frameworks offers, like for example WAI-ARIA compliance. This will give developers tips on what to search for when finding a suitable framework for a project and make the guidelines more sustainable to time and changes in technology. It was also proposed to be very clear on the fact that one needs to check all plug-ins for accessibility even when using accessibility-friendly libraries. This is an important aspect to mention to prevent developers to use plug-ins from accessibility-friendly frameworks uncritically.

7.5.21 Clarify accessibility benefits of using Web components

It was mentioned that the most important factor about using *Web components* is that if they are made accessible, for example by extending HTML elements, they facilitate accessibility through reuse. It was therefore suggested focusing more clearly on this and moving it to the top of this guideline. This is a good idea which will clarify the use of *Web components* in connection with accessibility. It was also highlighted that the guidelines should say something about how *Web components* can be beneficial for accessibility, e.g. the ability to extend HTML elements. This is advisable being as *Web components* is a quite new technology that not all developers are very familiar with. This way they will quickly understand the link between *Web components* and accessibility. It was also pointed out that it is not granted that *Web components* are accessible. Many of them are terrible when it comes to accessibility. This should be specified clearly in the guideline, perhaps highlighted as a warning to prevent developers missing this and believing *Web components* to always be accessible.

7.6 Suggestions for additional guidelines

In addition to suggestions concerning the guidelines already present, there were some suggestions for additional guidelines that could benefit accessibility. They are presented in the following.

7.6.1 Communicate accessibility across the team

It was suggested having a separate point addressing the importance of communicating accessibility across the team. Perhaps there could be an introduction course or a seminar or something similar. This is a very good suggestion which deserves a place in the guidelines. Exactly how accessibility is communicated may be different for each team and each project. It is however important that everyone understands that accessibility is interdisciplinary, *why* accessibility is important and *how* it affects different sides of web development.

One possible way to communicate accessibility, which was suggested, was to let the members of the team see someone with a disability use the Internet in general or test the system they are building. It was believed this could be an eye-opener and could increase knowledge of how to enhance the experience of these users while at the same time highlight the human aspect as a motivation beyond accessibility as something that something just that “has to be done”.

It was also suggested that it would be useful to explain to the programmer that something in the design has been decided for the sake of accessibility. If this is not communicated it might be overlooked in the implementation because things often change when it is discovered that some of the design does not work in practice.

7.6.2 Where it is necessary to consider accessibility?

There was a proposition to add a point that explains *where* it is necessary to consider accessibility, e.g. HTML, CSS and JavaScript and testing. It was believed this might give the developer a feeling of having an overview of what is needed. Although this is a valid suggestion, one cannot rely on everything being 100% accessible, even within the scope, just by following *these* guidelines. This is especially true because this is just the first iteration of the creation of these guidelines and testing reliability has not yet started. Nevertheless, there could perhaps be some mention of this in the introduction to the guidelines.

7.6.3 Log accessible script modules

It was also suggested adding a point in the *process oriented* guidelines that recommended logging all well functioning and accessible script modules for future use. This is a very good idea. This is a very good suggestion. It has the same benefits of using *accessible Web components* and plug-ins because it facilitates accessibility through reuse. The proposal was to place it with the *process oriented* guidelines. It could also be placed in the *technology oriented* guidelines with the technologies that facilitate accessibility. However, the best place to put it might be with the testing guideline. This gives a natural flow. First testing if something works and then logging it when it is confirmed to function as it should. However, exactly the best place for this point will have to be figured out through further evaluation of the guidelines.

7.6.4 Pay attention to placement of focus

It was mentioned that the guidelines should say something about the importance about paying serious attention to where one is placing *focus* after interactions either with or without re-loadings of a page. There is often re-setting of focus even if the page has not re-loaded. This is very demanding on keyboard users and visually impaired user navigating with keyboard, especially if there are no shortcut links. Focus should be placed where it is most natural to continue using it, not necessarily at the top every time. Having a guideline addressing placement and movement of focus seems like a very good idea, as this is highly relevant for accessibility, especially when it comes to pop-up windows and quickly changing applications. However, this is quite a large area that needs to be examined in detail. It might, at least for now, be enough to refer to WCAG 2.4.3 *Focus order*²³.

7.6.5 Make data available in different ways

It was suggested adding a point saying something about how making data available in different ways benefits accessibility, e.g. a map with information about such things as addresses, distance and directions can also be made available through text. This is however covered quite well in

²³ <http://www.w3.org/TR/WCAG20/#keyboard-operation>

WCAG principle 1²⁴ and whether or not this is a part of the front-end scope is a grey area. On the one hand, it is mainly about visualisation, which for this project has been placed on the outside. On the other hand, it is about making information available to everyone, even through different means and technologies, which might place it within the scope. In any case, it might be valuable to highlight this part of WCAG, being as this is of special importance with RIAs since they include so many different ways of displaying information. It might also be useful doing some further research into this to uncover if there are possibilities beyond what WCAG states.

7.6.6 Merge Web components and framework proposals into one guideline

It was suggested not having *Web components* as a separate guideline because it is not essential to accessibility in the way the other guidelines are (appendix 1, guideline 2.8). Having it as a separate guideline may cause confusion on this point. It was proposed rather placing it together with the recommendations on frameworks and joining these in one guideline. They are similar in the way that they are both technologies that may or may not facilitate accessibility. This will highlight the fact that no *one* library or framework is essential to accessibility. Neither are *Web components*. What *is* essential is to know how to use existing technology and facilitate accessibility thus implemented as efficiently as possible. This is when accessibility-friendly frameworks and *Web components* become powerful tools.

The placement of this guideline is not set in stone. It could be argued that it belongs with the *process oriented* guidelines. Although *Web components* and frameworks are technologies, they are technologies built to make the process easier through for example warnings and reuse of accessible plug-ins or components.

7.7 Suggestions on how to present the guidelines

There was a general dislike of the way the guidelines were presented. It was found confusing and unclear to the extent of reducing comprehension. Some content was presented several places, and some content was presented in the wrong place. It was suggested to re-think where the content was placed and the sequence of the guidelines.

7.7.1 Check-list vs. step by step guide

There was a request for a more orderly presentation form where the most important aspects are presented first and then more detailed information is provided on each point. This was considered a more pedagogical way of presenting the content and it was thought it would make the guidelines more usable so developers can more efficiently go through them. A re-occurring suggestion was to create a collapsed list of guidelines where expanded check-points would provide more information and links to even further explanations from other sources. This would provide both a way of quickly checking if one has remembered everything and more detail on how to ensure accessibility throughout the process. The check-list would function as a tool to make sure one has remembered everything and done everything possible to ensure accessibility. The guide would offer in depth guidance on each point. This might be advisable.

²⁴ <http://www.w3.org/TR/WCAG20/#perceivable>

7.7.2 Presentation of testing tools

There was a suggestion to make the list of testing tools into a table giving an overview of what the different tools can do, e.g. if it checks mark-up, CSS or contrast, if it checks for correct WAI-ARIA use or if it can be used locally or on password protected sites. Another idea was linking the tools to the different guidelines as a sort of indication on what how they help the process.

7.7.3 Tagging the guidelines

It was suggested tagging the guidelines with the professions they are directed at, not just write it at the top of each guideline. This way it would be possible to filter the guidelines and view only guidelines relevant to for one profession. They could also be tagged as *process oriented* and *technology oriented*. This would simplify the process of finding the guidelines that are most relevant in a specific part of the process or for an individual developer. It would increase usability greatly because team members can quickly see what guidelines relate to their profession and start working with them. This will make the use of the guidelines more efficient and straightforward. It will also give the project manager an overview of who is responsible for what, as well as what she herself is responsible for.

7.7.4 More examples and illustrations

There was a call for providing more examples throughout the guidelines in addition to coding-examples. It was thought this would make the guidelines more user-friendly because it will ease comprehension. It was also suggested to provide more illustrations. This is another way of ensuring a more user-friendly explanation of certain aspects of the guidelines. However, finding good and usable examples and good illustrations might prove time-consuming and it is not certain the time schedule for this project allows it. It might also be difficult finding high quality illustrations that can legally be used for free. Doing the illustrations one selves or hiring someone to do them is a possibility, however it is outside the scope of this project.

7.7.5 Prioritised order of guidelines

It was suggested prioritizing the guidelines and placing the top five most important ones first. It was believed this would maintain the users focus. It was thought prioritization would make the guidelines more manageable and easier to deal with. It would also ensure that the most important things are done, not just the easiest thing. It might however be difficult to establish what guidelines are the most important, as reliability has not been established yet. In addition all the guidelines are designed to do their part. It would in any case require further studies.

7.7.6 Tool for using the guidelines actively within a project

Lastly, it was suggested presenting the guidelines in a way so they can be used actively within a project. This could be done for example by offering the possibility of internal discussion and fields for commenting on each guideline within the bounds of a project. This way the team members would be able to discuss the guidelines and the implications of them related to the project they are working on. This facilitates communication of accessibility and it might give a deeper understanding of accessibility to place the guidelines within a context like this. It does however

demand a considerable amount of work and further testing to ensure a high quality, usable tool, and is therefore outside the scope of this project.

8. Implications and discussion of findings

Many comments have been made on the first version of the guidelines (appendix 1). An overview of what works, what does not work, what was missing and what might be unnecessary according to the participants has now been established. These findings will be used to modify and improve the guidelines and present a second version. Many of the suggestions for changes are very interesting, and will in all likelihood improve the quality of both content and presentation of the guidelines. Some can easily and quickly be managed, whereas others need more work or even more studies before being included.

The participants agree on many points. Where they disagree, this seems to have something to do with their background and previous knowledge of accessibility as well as their current work situation. It is for example natural that a programmer is most concerned with the guidelines directly related to programming and that the developers working with bringing accessibility in to their processes requests further detail within the *process oriented* guidelines. It is also natural that the participant who is an expert on accessibility has the most comments on the content, while the ones working with usability and interaction have more comments on the presentation of the guidelines and how to interact with them.

In the following some of the most important things that were discussed with the participants is presented and further examined. *Usefulness and reliability of the guidelines* and *comparison to existing guidelines* is derived from the interview guide. These questions say something about how the participants generally perceived the guidelines and give important clues especially to what seemed to work and received positive attentions. *Explanations of how the guidelines benefit accessibility* and *level of detail* are issues that all the participants mentioned independent of each other and which seemed to be the main weaknesses with the guidelines. When it comes to *prioritizing and estimating the guidelines* and also *including other area of web development*, these are issues that need further discussion and thought processes before making decisions of whether or not to implement them, and if so how to proceed. Lastly there will be a discussion of whether or not the guidelines are in fact relevant in any web development process, or if they are to concrete or out of touch with the “real world”. This is important because it does not only say something about if the guidelines are useful or not, but also to which extent they are useful. This can give indications of the value of this project and the benefits of continuing work in the future.

8.1 Usefulness of the guidelines

The guidelines seemed to be quite useful. All the participants believed they could use parts of them in their next projects and one had already applied some of them in a project and found this useful both for placing focus on accessibility and for learning something about accessibility. The fact that all the participants learned something when reading the guidelines is also an indication of usefulness. It means the guidelines can contribute to building accessibility competence. None of the participants spent a very long time on the evaluation. This is positive because it indicates that the guidelines are able to provide new knowledge in an easy and manageable way. A good indication that the guidelines are working as intended is one participants comment on feeling

more equipped to be responsible for accessibility on a project. Considering that this is the first iteration, these are positive findings, which bode well for future development. It is highly likely that the participants would find the guidelines even more useful and manageable given the opportunity to evaluate the second version, which has taken into account many of the comments they have given.

8.2 Reliability of guidelines

Studying the reliability of the guidelines was not the focus for this project. It is however possible to draw some conclusions on the matter. The fact that these guidelines are built on a literature study reviewing research on how to ensure accessibility, suggests that they probably are, at least to some extent, reliable. Taking the scope into question, the participants also thought applying the guidelines would benefit accessibility. The fact the *accessibility expert* believed this is a strong indication of reliability. This is a very positive finding for the first iteration of these guidelines. Additional research into reliability is essential for further development. This might however not be entirely without problems.

Examining how the technical guidelines benefits accessibility presents no major problems. They are quite easy to link to measurable goals that can be tested and approved, e.g. claiming that WAI-ARIA makes updates available to AT or that HTML5 provides it with better and more understandable semantics and then testing for it. One could apply *Progressive enhancement* and test in different browsers, on different devices, with and without JavaScript and so on. *Device independent methods* are also easily testable. The challenge with some of these guidelines lies with keeping them updated. This has to do with the level of technical concreteness. Though this is a good thing, it will cause the guidelines to become outdated faster than if they were less concrete. This is especially true for use of *Unobtrusive JavaScript*, accessibility-friendly frameworks and *Web components*. These points might however also present other problems when it comes to assessing reliability. They cannot so easily be tested through measurable goals and would have to be evaluated through a process of extensive further studies and testing, development and maintenance of code. Taking measures to facilitate accessibility with *Web components* and accessibility-friendly frameworks will probably make the job easier for developers and in the long term save time. However, it will demand them spending time on research and learning new technology. These tasks needs to be set up against each other to measure time saved versus time used. This will be a long process.

Studying the reliability of the *process oriented* guidelines might be challenging. It has been claimed that accessibility is not a technical issue, but a social one. Literature has shown that adding accessibility is time-consuming and convincing clients and management of its importance is difficult. Tools are often hard and time-consuming to apply and might not even ensure accessibility 100%. This highlights the importance of finding a way to integrate accessibility smoothly into the web development process. Making use of experts and loosing the uncertainty of what is necessary and how to meet requirements for accessibility may go a long way for saving time and money and convincing stakeholders of creating accessible end products. It is certainly easier then arguing that it is the law or that it is the right thing to do, because this is often too vague. It may also be easier

than arguing that accessibility helps everyone or makes an application it more future proof because these arguments are also vague and unclear. It is the authors strong believe that making accessibility efficient and streamlined is the only way of removing all the social issues with accessibility and all the arguments for not implementing it. Having expert testers will save time on learning how to test for accessibility. Having clearly stated accessibility requirements in the specification will help everyone to know what is expected of them and how to meet these expectations. Having the help of an expert to define these accessibility requirements might be of value. Keeping accessibility in mind throughout the development process on the same level as usability *will* create more accessible end products. Although these statements make logical sense, they can be difficult to prove scientifically. It would demand a great deal of time, implementing these guidelines into development processes and examining both processes and results. In addition, how specifically to implement all this in the process will demand a great deal of further examination and might be different from project to project.

8.3 Comparison to existing guidelines

One goal for these guidelines was that they would be easier to apply and more manageable to use than existing guidelines for accessibility. A clear focus has been maintained on technology and how to create accessible RIAs with the standards HTML, CSS and JavaScript.

There was also an attempt to present the guidelines in a way that gives a direct focus on concretely *what* the developer needs to do. For example, instead of saying “Ensure keyboard navigation and screen reader options” these guidelines say:

- Apply WAI-ARIA
- Follow and validate the HTML5 standard
- Use device independent JavaScript methods

These are three very concrete suggestions of what the developers can do to ensure screen reader and keyboard access. It means they do not have to go though the process of figuring out how to ensure this. In theory, they might not even have to know *why* they should follow the guideline; they just have to do it. This is not preferable though, since it has become clear during this project that knowing *why* one is doing something is essential. This is discussed in more detail in the next chapter.

It was believed that having concrete suggestions of what to do in the main guideline before more in depth explanations might make reading and applying the guidelines easier and more efficient. This was confirmed in the interviews. The participants believed these guidelines to be more specific and concrete than existing guidelines and to have a more pedagogical presentation form.

It was also pointed out that it was a benefit having a scope to relate to, because it makes the guidelines less comprehensive. Leaving out things like *alternative text* and *contrast* to make room for focusing on how to deal with the dynamic interactivity of users and responsiveness to different user agents seems to have been beneficial. It gives a quick overview of what is needed to ensure accessibility within the scope. Because of the small scope, these guidelines are able to in depth on things of which WCAG only scratches the surface. This means most of the guidelines address a

large area e.g. *testing for accessibility, applying WAI-ARIA, using HTML5 and using JavaScript unobtrusively*. As such working with the guidelines will take time, but familiarising oneself with them and getting an overview of what is needed should not be too much work. This may prevent the guidelines from discouraging developers and they may seem easier to apply. The downside of the small scope is that the developer cannot be sure accessibility is 100% ensured by only applying *these* guidelines. Others have to support. On the other hand, that is the case with all existing guidelines at the moment.

Another step taken to make the guidelines easier and more manageable to deal with is directing each guideline to one or more specific professions. This is not done in any other guidelines and was much appreciated by the participants. The guidelines are not meant to be used in the same way by all the team members. It is for example the project managers' job to ensure accessibility experience and bringing in accessibility from the beginning, while it might be a front-end programmer or testers' responsibility to test for accessibility. This facilitates quick access to the guidelines relevant to each profession and increases efficiency of use.

An important difference from other guidelines is the *process oriented* part. Many guidelines focus on how to technically ensure accessibility, but not so much on *how* to implement this aspect into a web development process. This is a rather important aspect. For example having an accessibility expert on the team would simplify many of the aspects of implementing accessibility from use of WAI-ARIA, testing with tools and guideline analysis, to focusing on it during planning and design, use of frameworks that facilitate accessibility and reuse of accessible components. The *process oriented* guidelines was as mentioned much appreciated by the participants. They saw the value in getting guidance on how to work with accessibility on a process level. This does not only help with applying guidelines, it can create a completely different way of thinking and approaching accessibility.

8.4 Explanations of how the guidelines benefit accessibility

The participants believed explaining *how* the guidelines benefit accessibility would increase their usefulness. This question has been somewhat difficult to balance in this project. On the one hand, it is desirable that the guidelines should be usable outside the context of the master thesis. On the other hand, including how the guidelines help accessibility in *chapter 10* would cause many redundancies and repetition of the discussions in the literature survey in *chapter 6*. Initially it was decided *not* to place explanations within the guidelines. This is the version that was evaluated. It is an important finding that all the participants requested more explanations within the guidelines. It was expressed that this makes it easier to explain *why* one is doing something, and more difficult *not* to do something if one knows its merits. This supports previous studies of WCAG, which have shown that when developers do not understand why they should use a guideline, the motivation to apply it decreases (Tanaka and Vieira da Rocha, 2011). Tools should promote knowledge transfer and deepen the developers understanding of accessibility (Trewin et. al., 2010). There is reason to believe that further explanations of the consequences of using the guidelines will increase the level of learning. This is especially important because it increases the guidelines' ability to build accessibility competence.

It was suggested putting *why* one should follow the guideline first on each point. This does however go against some of the comments on the concreteness of the guidelines and that they are easy to use because they pinpoint directly what to do. This might however be solved with an expandable *why*, which can be viewed if necessary.

It has for now been decided not to include a lengthy description of *why* each individual guideline is important and the consequences of using it. Nevertheless, due to strong requests most of the guidelines have been provided with a short explanation of their usefulness. A few of the guidelines are self-explanatory in this regard, e.g. *testing for accessibility*, *communicating accessibility within the team* and *have accessibility expertise on the team*. In addition, an overall explanation has been provided in the introduction.

8.5 Level of detail

It was noted that the level of detail on the different guidelines was somewhat uneven. The problem with this is that it might indicate that the most detailed and comprehensive guidelines are the most important. For example is using *device independent methods* just as important as applying WAI-ARIA though it might not seem that way from the way the guidelines are presented. It is nevertheless the case that the level can never become exactly equal. Some things are more comprehensive, need a more detailed explanation, or are by nature more specific. It could be argued that how to bring accessibility into the web development process is a more diffuse question than how to technically ensure accessibility. It will therefore be a natural consequence that the *process oriented* guidelines are less specific and detailed.

It could also be argued that some guidelines serve more as reminders to web developers rather than giving new information about accessibility. It is for example not necessary to explain a front-end developer how to use HTML5, or an interaction designer what good design practices are. It is natural that the guidelines focus on the aspects of accessibility that are outside of web developers general competence area like applying WAI-ARIA, testing for accessibility and using *Progressive enhancement* and *Unobtrusive JavaScript*. It is nevertheless important that developers become aware that some of the things they already know benefits accessibility, like using standards or making it obvious what is clickable. It may give a sense of not starting from scratch with accessibility, but rather building on existing knowledge. It also highlights how accessibility in fact is a part of usability, and that without usability one cannot have accessibility.

Lastly, not being detailed can both be a curse and a blessing. If there is a lot of detail, it is very easy for the developer to know exactly what to do, and how to do it. This makes the guidelines more usable and, as a result, they might be more used. However, many details may make the guidelines too long and comprehensive. In addition, if there is too much detail on the technical level, they might not be transferable to future projects, or even ongoing projects not using the technology addressed in the guidelines. One example is explaining what features in a framework that facilitates accessibility rather than suggesting specific frameworks. This means the developer has to do their own research on frameworks, but they will know what to look for accessibility wise. At

the same time, the guideline will be much more transferable to other projects. It is also not certain that a detailed description of the process is especially useful or even possible as every process is different. However, giving some general pointers will be both possible and highly valuable.

8.6 Prioritising and estimating guidelines

A prioritised list of guidelines was intentionally not made in this project. Prioritising the guidelines can be smart to make sure developers are not just doing the easy parts and ignore the more complex issues (Rosson et. al., 2005). On the other hand, prioritising means saying something is more important than the other is. Without testing reliability, this is virtually impossible. This needs extensive further research and testing which might enable us to say something about the importance of the different guidelines related to each other. It is however, the authors' theory that prioritising will prove difficult because of the comprehensiveness of many of the guidelines and the fact that they address different types of users and input methods. They may not all be equally important, but they are all designed to be essential to accessibility. Prioritization will always be done at the expense of something. It sends a signal that some things do *not* in fact need to be done. It goes without saying that this *will* not be done in a deadline oriented process where time is of the essence. To ensure an accessible application one has to follow all the guidelines. Although it might be naïve, it is the authors believe that prioritization might reinforce the belief that following accessibility guidelines is not that important in the first place. On the other hand, not prioritizing may result in only the things that can be quickly fixed are done. This may not correspond with what is the most important. In any case, doing a general prioritization may not be the best strategy. In different projects, different things will be important and different things will have to be prioritized. Doing individual prioritizing for each project may be more useful.

There was a suggestion to estimate time use for each guideline. The web development process is often deadline oriented and time is limited. If something can be quickly implemented, it is more likely to be taken into consideration. However, this would need extensive further study. One would have to use the guidelines and log time spent on each many times. It is also highly likely that time spent will be very different from project to project, depending on type of project and level of accessibility knowledge of team members. It might be better *not* to estimate time, rather than estimate something that is not right. Another reason for not estimating time is that it might result in only the guidelines that can be rapidly applied being used. This is especially problematic if time estimation is done, but there is no prioritization.

Despite these issues both time estimation and prioritization of the guidelines is something that may add value and heighten the quality and usability of the guidelines. Therefore this should be examined further in the future before making any definite decisions whether or not to perform these tasks, and if so how to conduct them.

8.7 Including others areas of web development

It was noticed there was little in the guidelines about interaction design and visualisation. Design of *information architecture* was also briefly mentioned with *search engine optimization*. While these are areas where accessibility is very important, this is outside the scope of these guidelines.

It might however cause some confusion having a guideline called *Follow existing principles for good design*. This guideline gives some examples, but does not go further into the matter. This is one of those guidelines that serve as a reminder for the interaction designer and graphic designer. The fact that it is outside the scope could argue for removing this guideline altogether. The reason for keeping it however, is that while it does not really give the developer new knowledge about accessibility, it serves as a reminder that much of what they already know is very good for accessibility. This way the developer might begin the process with the feeling that he already has a head start on accessibility. It is believed that this fits within the *process oriented* part for now. Another reason for keeping this guideline is that it serves as a reminder for future development to include more about these areas in coming versions. This will demand extensive further research. This guideline might then fall under a new headline called something like *Design oriented guidelines*.

8.8 Are the guidelines relevant in any development process?

One participant believed that some of the guidelines could become irrelevant within a web development process because of inability to choose what technology to use. He thought *pure* JavaScript examples become irrelevant when using libraries like *jQuery* or frameworks like *AngularJS*. Others pointed out that the use of *pure* JavaScript examples can be beneficial because it can be related to different frameworks. When updating the guidelines suggestions for how libraries and frameworks can benefit accessibility has mostly replaced suggestion for actual frameworks. It is believed this will make the guidelines more transferrable.

It is difficult create guidelines, and especially examples, that are an exact fit to all processes. This has been an attempt of balancing concreteness and abstraction in the guidelines so that they may be easily applicable and at the same time relevant in many different processes. It is the authors believe that it should be possible to see beyond the examples and apply the guidelines in a way that suits every individual process. It is also important to note that the thought behind for example *Progressive enhancement* and *Unobtrusive JavaScript* is still relevant, and is transferable when using different libraries.

In addition, notice that most of these guidelines are in fact not very *technically* specific. It should be possible in all processes to introduce accessibility from the beginning, hire an accessibility expert, test for accessibility, follow existing design principles, use WAI-ARIA mark-up, follow and validate the HTML5 standard, use *Progressive enhancement*, use *Unobtrusive JavaScript* and make sure methods are independent of input device. Some of these things may be viewed as impossible because it cost money and there is no funding for this. However, if accessibility is a requirement, then funding has to be given for these activities. The same way that if *usability* or a specific functionality or top notch SEO is required, money would have to be spent to ensure this.

When it comes to using technology that facilitates accessibility this will probably be very depending on the type of project and what requirements are set for technology. It is true that the guidelines represent an ideal process whereas the real world will be more complex. The developer is not always at liberty to choose technology. However, if for example a customer who is in dire

need of an accessible website has decided using a technology that impedes this, it is the authors' believe that the consultant company is responsible for making the customer see why choosing another technology might be better. If the consultant firm has specialised themselves on a technology that impedes accessibility, they should consider either not taking on customers who have accessibility as a requirement, or in fact take steps towards familiarising their consultants with a new, more accessibility-friendly technology. It was pointed out that there are often different firms in charge of different modules within a website and that this could impede accessibility. This is true to the extent the other firm is not implementing accessibility. It is however hard to see how that is an obstruction for using the guidelines. Rather, all firms involved should use the guidelines and communicate accessibility, the same way members of the team should.

This has been a fruitful discussion during this project which already has, and may continue to increase the quality of the guidelines, and make them even more relevant in all processes. Something definite about the guidelines relevance in different processes cannot be said at this point. Further studies would be very interesting.

9. Alternation of guidelines

Many changes have been made to the guidelines following the first evaluation process. The most critical weaknesses and the changes that were feasible within the time limits of this project have been prioritized. The following chapter presents what changes have been made starting with the guidelines as a whole before concentrating on the changes made to individual guidelines. The first version of the guidelines can be viewed in *appendix 1*.

9.1 Changes related to the guidelines as a whole

Some changes have been made related to the guidelines as a whole. An *introduction* to the guidelines was added including:

- How the guidelines came about
- Clear statement of scope and where accessibility is considered
- Some consequences of using the guidelines

The guidelines were also linked to relevant existing guidelines and a short description on some of the guidelines of *why* they are good for accessibility was added. In addition the level of detail was adjusted in some degree by adding some details to the *process oriented* guidelines and removing some from the *technology oriented* guidelines. Different and more relevant examples were provided, some of which were made interactive in *Codepen*²⁵. As such developers can play around and try out for themselves. The guideline about *tidy coding* was removed. Some illustrations were added and the main title was changed from *Building accessible RIAs using JavaScript* to *Building accessible Rich Internet Applications*. Some of the titles for the individual guidelines have also been adjusted as a result of editing work.

9.2 Accessibility from the start

There were several suggestions for revising the guideline about introducing accessibility from the beginning of a project. As a result, the following alternations were made:

- Shortened the guideline
- Removed advice about frameworks
- Added part about specification

9.3 Accessibility testing

There were quite a few comments on the guideline about accessibility testing. In response, it was clarified that all points on testing has to be done at key stages in the development process and that testing limited parts of a system applies to all testing. *Simulation* was placed as a part of *expert testing* and it was highlighted that it might be difficult to turn *off* JavaScript and *on* keyboard access in new browsers. It was further added that that it may not always be a problem for users to come to special test facilities for user testing at that it is important that the tasks remain the same, but communication may be different when testing with users with disabilities. A part had been added recommending logging all functioning and accessible script modules when

²⁵ <http://codepen.io/>

they have been thoroughly tested. Lastly, this guideline was aimed at front-end developers as well as testers, and project managers were removed as a target group.

9.4 WAI-ARIA

In the guideline about WAI-ARIA it was highlighted that WAI-ARIA should be not used as a bandage for poor code and that that *Live regions* is not provided by any other language. The distinction between using WAI-ARIA as an aid for functionality that does not exist elsewhere and using it WAI-ARIA as an aid for conveying functionality to AT was also underlined. Finally a warning was given about using the role *navigation* in combination with the HTML5 element `<nav>` and that this does not work well when using *Windows Eyes* version 8 with Internet Explorer (IE).

9.5 HTML5

In the HTML5 guideline widely used input elements like `<search>` and `<email>` and some examples of how correct use of input types benefits accessibility were added. The list of examples of elements was removed and links to more information was provided instead. Lastly a warning was given about how the elements `<section>` and `<article>` do not work well with all screen-readers.

9.6 Progressive enhancement

In the guideline about *Progressive enhancement* it was explained that not all functionality is handled by JavaScript and clarified what *Progressive enhancement* entails for accessibility. In addition the part about *Progressive enhancement in CSS* was removed, as it seemed to be somewhat outdated.

9.7 Unobtrusive JavaScript

The guideline about *Unobtrusive JavaScript* was altered by firstly giving an explanation to why *Unobtrusive JavaScript* is good for accessibility and it is now mentioned that accessibility is only *one* of the benefits to *Unobtrusive JavaScript*. Some of the points were made clearer, more detailed and put into simpler words. An illustration of an event order was added and a warning was given that *Unobtrusive JavaScript* is not necessarily accessible JavaScript.

9.8 Device independent methods

In the guideline about *device independent methods* it has been stated that applications have to be usable with mouse, keyboard and on touch screen and more suggestions for methods has been added. A warning has also been placed in the guideline that *onChange* and *onFocus* goes against WCAG.

9.9 Frameworks

It was debated whether specific frameworks should be suggested in the guidelines. It was decided to *not* suggest specific frameworks, but rather focus on aspects on frameworks that facilitate accessibility. So in this guideline some important accessibility concepts that frameworks should offer has been mentioned and recommendations for most of the specific frameworks has been

removed. The importance of always testing plug-ins and not just rely on the accessibility of a framework has been underlined.

9.10 Web components

In the guideline about *Web components* the explanation of how and why *Web components* can be so beneficial for accessibility, the benefits of reuse of accessible components, has been placed at the top. In addition it has been highlighted that not all *Web components* are accessible, and that they are only as accessible as they are made to be.

9.11 Additional guidelines

Some suggestions were made for additional guidelines. In response to this, the following guidelines were added:

- *Communicate accessibility within the team*: This guideline includes a clear statement that accessibility is interdisciplinary, a mention of the usefulness of explaining to the programmer that something in the design has been decided for the sake of accessibility and videos of people with disabilities using the Web with different AT.
- *Combine WAI-ARIA and HTML5 mark-up*: It was noted that the combination of these two standards should be highlighted. Therefore a separate guideline has been added underlining this.
- *Use technology that facilitates accessibility*: This guideline is a merge of the guideline about frameworks and the guideline about *Web components*.
- *Use accessible modal windows instead of pop-ups*: In the process of finding good examples to illustrate the guidelines, an example of an *accessible modal dialog* using WAI-ARIA was discovered. This led to more research about these modal windows and their benefits to accessibility as opposed to pop-up windows. There is now a chapter about *accessible modal dialogs* in the literature study. However, this guideline was not in place before the evaluation, nor was it a concrete suggestion from the participants. Nevertheless it has some obvious accessibility benefits and directly answers the problems with pop-up windows and has therefore been added to the guidelines.

9.12 Changes made to presentation

Some of the suggestions on how to improve the presentation of the guidelines are only relevant for the digital version, e.g. menu, tags and linking. Where a good substitute is possible, this will be provided, e.g. footnotes instead of links. The guidelines have been tagged for the possibility of filtering them by profession and orientation. The tags will be written beneath the heading of each guideline in the document as indicators. However, the filtering on the website is still under development. It was originally thought to make the guidelines into a *check-list* with the possibility to expand for more details. Instead a sub-menu including each individual guideline was added to the website, functioning as a check-list and a short-cut to each guideline. For users not using JavaScript, a list of links to each guideline is presented at the top of the page. There has also been an attempt to keep content about the same issues in *one* place, and a table was made for presenting the testing tools and their functionalities.

10. Results: Guidelines updated version

In this chapter, the second version of the guidelines for building accessible *Rich Internet Applications* (RIA), which is the result of this project, is presented. A digital version is available at accessibilityagent.no/guidelines. Since the guidelines are designed as a digital document, that version may be preferable to read. The website is under development, but contains all content.

The short version of the guidelines is:

1. Have accessibility expertise on the team
2. Introduce accessibility from the beginning
3. Communicate accessibility within the team
4. Follow existing design principles
5. Test accessibility at key stages
6. Use WAI-ARIA mark-up
7. Follow and validate the HTML5 standard
8. Combine WAI-ARIA and HTML5 mark-up
9. Use *Progressive enhancement*
10. Use *Unobtrusive JavaScript*
11. Make sure methods are device independent
12. Use accessible modal windows instead of pop-ups
13. Use technology that facilitates accessibility

These points will be elaborated and explained in the following.

10.1 Introduction to guidelines

These guidelines are the result of a master thesis studying what issues occurs when working with creating *accessible Rich Internet Applications* (RIA) and how it is possible to resolve some of these issues. They are divided into *process oriented* guidelines and *technology oriented* guidelines. The *process oriented* guidelines suggest some ways to incorporate accessibility into the web development process and recommends methods for testing for accessibility. The *technology oriented* guidelines focuses on front-end and the standard technologies HTML, CSS and JavaScript.

The consequences of following these guidelines are that dynamic and interactive web applications will be usable to more people, especially keyboard users or users of assistive technology (AT). Another consequence is that it will smooth the process of including accessibility into the web development process.

These guidelines are not intended as an alternative to the existing WCAG standard²⁶, but rather as a supplement.

²⁶ <http://www.w3.org/TR/WCAG20/>

10.2 Have accessibility expertise on the team

[Process oriented](#), [Project manager](#)

One or more people on the team should have experience with:

- Existing guidelines and legislation for accessibility (e.g. WCAG)
- Good coding practices to ensure things like keyboard access and understandable navigation for a screen reader
- Different kinds of AT and how they work
- Use of testing tools for accessibility
- Simulation exercises
- User testing with users with disabilities
- Technology that facilitates accessibility

10.3 Introduce accessibility from the beginning

[Process oriented](#), [Project manager](#), [Procurer](#), [Front-end developer](#), [Interaction designer](#), [Graphic designer](#), [Tester](#)

- Follow the accessibility standard WCAG
- Include accessibility on an equal basis as *usability* and *user experience*
- Apply *user-centred design* principles and include users with disabilities

10.3.1 Accessibility in the specification

- There has to be specific, measurable goals for accessibility in the specification. Look at the interface and the planned functionalities and determine what WCAG criteria need to be fulfilled.

10.3.2 Planning

- Include people with disabilities in your target audience.
- Think accessibility when choosing what technology to use. For example a framework should be compliant with WAI-ARIA. Some frameworks have better support for accessibility than others.

10.4 Communicate accessibility within the team

[Process oriented](#), [Project manager](#), [Front-end programmer](#), [Back-end programmer](#), [Interaction designer](#), [Graphic designer](#), [Information architect](#)

Make sure everyone on the team understands:

- That accessibility is interdisciplinary. This means everyone is responsible for accessibility within their professional sphere.
- *Why* accessibility is important
- *How* accessibility affects different sides of web development.

When the design is handed over to the programmer, it is a good idea to explain that something in the design has been decided for the sake of accessibility. If not, it might be overlooked in the implementation phase.

Show the team someone with a disability using the Web. This can deepen the understanding of accessibility and increase motivation. Here are some videos that might be useful.

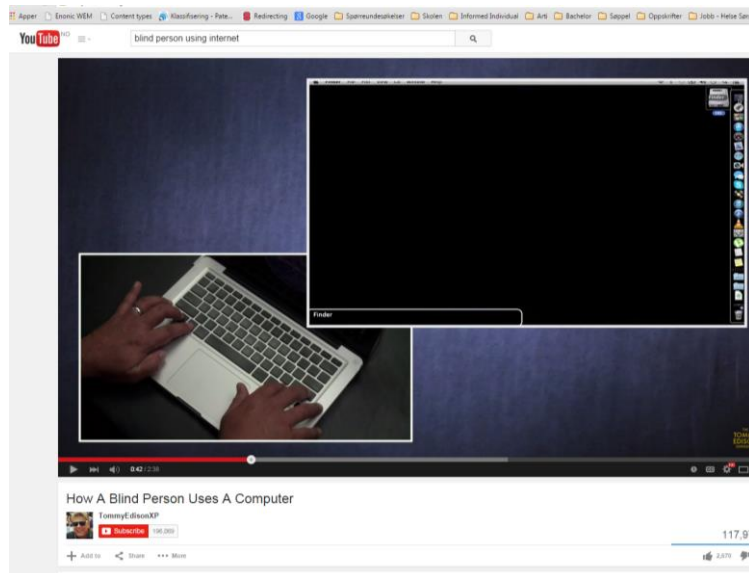


Figure 7: Video of blind person using computer (YouTube, 2013)



Figure 8: Video of blind person using iPhone 4S (YouTube, 2012)

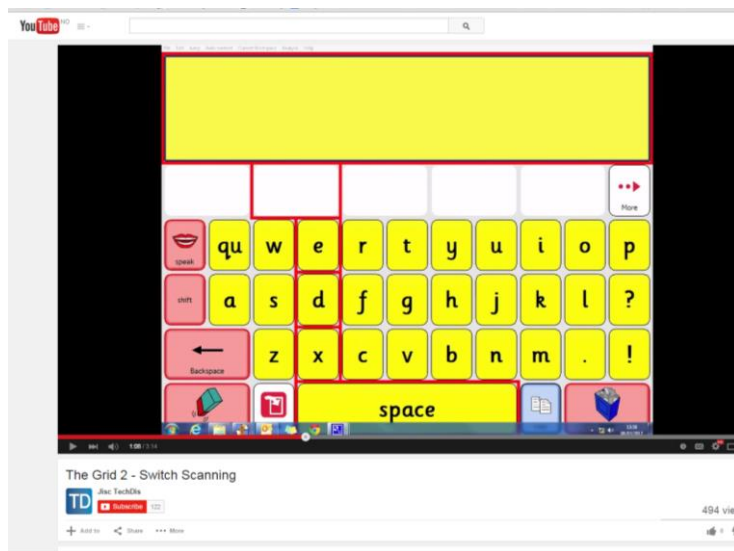


Figure 9: Video of scanning software (YouTube, 2012)

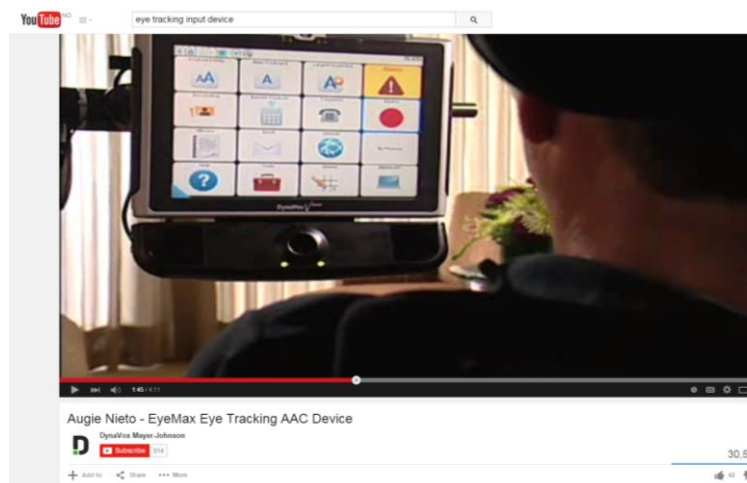


Figure 10: Video of person using computer with eye-scanning (YouTube, 2009)

10.5 Follow existing design principles

[Process oriented](#), [Graphic designer](#), [Interaction designer](#), [Information architect](#)

Good design and user experience go a long way in making a site more accessible. An application that is usable for everyone is the most efficient way of ensuring it can be used by someone with a disability. Pay extra attention to ease of navigation, scalability and responsive design, font-type font-size and contrast.

Some examples of good design principles are:

- Know the main things people want to do on your site and make them obvious and easy
- Save the user steps where possible
- Make it easy to recover from errors
- Know what questions the user is likely to have and answer them on the FAQ page
- Tell the users what they want to know, even things like shipping cost and parking fee

- The language should be clear and concise and needless words omitted
- Have a clear visual hierarchy where the most important things come first
- Use conventions, for example the placing of the search field in the top right corner
- Break the site up in clearly defined areas such as navigation, news feeds, content and external links
- Make it obvious what is clickable
- Avoid too much noise by not having everything on the same page, split it up
- Help the user find her way on the website by using things like clear labels in the navigation and breadcrumbs and make sure the search functionality is optimised

10.6 Test accessibility at key stages

[Process oriented](#), [Tester](#), [Front-end developer](#)

Test using the following methods:

- Automatic accessibility testing
- Expert based testing
- User testing

Do short task-based sessions and focus attention on one aspect at the time, e.g., how accessible the forms are.

Key stages for testing are every time something new and essential to the application is developed during the:

- Designing phase (Start testing with sketches and work your way from there)
- Development phase (Use automatic testing tool from your first HTML line)

All the different ways of testing for accessibility described below should be applied throughout the process.

10.6.1 Automatic tools

Table 3 gives an overview of some automatic tools for testing accessibility and their functionalities.

Tool	Examine s data tables	Checks structur e	Check s alt- text	Check s labels	Checks contras t	Examine s WAI- ARIA	Simulate s screen reader	Usable on local/passwor d protected sites
Web Accessibility Evaluation Tool	No	Yes	Yes	Yes	No	No	No	No

(WAVE)²⁷								
WAVE	No	Yes	Yes	Yes	No	No	No	Yes
Toolbar for Firefox²⁸								
Accessibility Evaluator for Firefox²⁹	No	Yes	Yes	Yes	No	No	No	No
Web Accessibility Checker³⁰	No	Yes	Yes	Yes	No	No	No	No
WCAG Contrast Checker for Mozilla³¹	No	Yes	Yes	Yes	Yes	No	No	No
WebAim Colour Contrast Checker³²	No	No	No	No	Yes	No	No	No
Juicy Studio Accessibility Toolbar³³	Yes	No	No	No	Yes	Yes	No	No
Fangs Screen Reader Emulator for Firefox³⁴	No	No	No	No	No	No	Yes	No
Firefox Accessibility Extension³⁵	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Web Accessibility Toolbar (WAT)³⁶	Yes	Yes	Yes	No	Yes	Yes	No	No

Table 3: Automatic testing tools and their functionalities

10.6.2 Expert based testing

There are many different ways of doing expert based testing. The most common ones are:

²⁷ <http://wave.Webaim.org/>

²⁸ <https://wave.Webaim.org/toolbar/>

²⁹ <https://addons.mozilla.org/en-us/firefox/addon/accessibility-evaluation-toolb/>

³⁰ <http://achecker.ca/checker/index.php>

³¹ <https://addons.mozilla.org/en-us/firefox/addon/wcag-contrast-checker/>

³² <http://Webaim.org/resources/contrastchecker/>

³³ <https://addons.mozilla.org/en-US/firefox/addon/juicy-studio-accessibility-too/>

³⁴ <https://addons.mozilla.org/en-us/firefox/addon/fangs-screen-reader-emulator/>

³⁵ <http://firefox.cita.illinois.edu/>

³⁶ <http://www.paciellogroup.com/resources/wat/>

- Heuristic evaluation³⁷
- Cognitive walk-through³⁸
- Consistency inspection³⁹
- The personas method⁴⁰
- **Simulation**

10.6.3 Simulation

Simulation is used to simulate what it would be like to use an application with some kind of disability. There are several ways of simulating:

Turning of the following things in the browser:

- Style sheets
- Images
- Sound
- JavaScript
- Java
- Support for Flash/Silverlight
- Pop-up windows

Be aware that it might be difficult to turn of JavaScript in new browsers.

Navigate using only the keyboard. Try out a few familiar websites first to get used to keyboard navigation. Be aware that and it might be difficult to turn on keyboard access in new browsers.

Turn off the display and navigate the application using a screen reader. Possible screen readers to test with are:

- Apple screen reader (free software)
- VoiceOver
- Demo version of JAWS or the
- NVDA for Windows (free software)
- Firefox Screen-reader simulator

10.6.4 User testing

User testing with people with disabilities is a bit different from user testing with users without disabilities. To ensure the highest possible quality of the testing keep this in mind:

- User may be dependent on large equipment of AT (e.g. *Braille* keyboard, foot mouse)
- User should use their own equipment which they are comfortable with
- User might need to perform the testing in their home or workplace (but they might also be perfectly able to come to a testing facility)

³⁷ http://en.wikipedia.org/wiki/Heuristic_evaluation

³⁸ http://en.wikipedia.org/wiki/Cognitive_walkthrough

³⁹ <http://www.usabilityfirst.com/glossary/consistency-inspection/>

⁴⁰ http://en.wikipedia.org/wiki/Persona_%28user_experience%29

Be aware that the tasks need to be the same, but there might be some differences in communication, e. g. saying *choose link* instead of *click on link*, talk about content rather than colour, or structural placement rather than visual placement.

10.6.5 Log all accessible script modules

Make sure you log all well functioning and accessible script modules. It facilitates accessibility in the future through reuse.

10.7 Use WAI-ARIA mark-up

[Technology oriented](#), [Front-end developer](#)

ARIA stands for *Accessible Rich Internet Applications*. WAI-ARIA is designed to aid in making dynamic and interactive web applications more accessible. It provides semantics and conveys updates and changes to assistive technology (AT). It can be used for:

- Adding functionality that does not exist elsewhere, e.g. conveying updates to assistive technology
- Conveying functionality that exist elsewhere, but can not be passed on to AT otherwise, e.g. *extend/collapse* or *sliders*

WAI-ARIA should be *not* used as a bandage for poor code.

The following will give an introduction to:

- WAI-ARIA roles
- WAI-ARIA states and properties
- WAI-ARIA Live regions
- WAI-ARIA use of *tabindex*

10.7.1 WAI-ARIA roles

Use WAI-ARIA role attributes to define a widgets role to AT. The role given by the WAI-ARIA role attribute trumps the role of the native element. The WAI-ARIA specification⁴¹ maintains a *list of roles*⁴².

JavaScript is often used to insert custom modal dialogs (usually divs) instead of opening a new browser window. The WAI-ARIA role *dialog* can be used on elements used as containers for the dialog to inform screen reader users that a custom dialog is being inserted (or made visible via CSS). In the following example a *div* element has a *role* attribute of *dialog*. A dialog is an application window that is designed to interrupt the current processing of an application in order to prompt the user to enter information or require a response.

⁴¹ <http://www.w3.org/TR/wai-aria/>

⁴² <http://www.w3.org/TR/wai-aria/#roles>

By using the role *dialog* and the WAI-ARIA attributes *aria-labelledby* and *aria-describedby* on the same element, we provide additional information to AT.

```
<div id="myDialog" role="dialog"
  aria-labelledby="myTitle"*
  aria-describedby="myDesc">*
  <div id="myTitle"> Save changes?</div>
  <button id="saveMe" type="button"> Yes </button>
  <button id="discardMe" type="button">No </button>
  <button id="neverMind" type="button">Cancel </button>
</div>
```

*The *aria-labelledby* attribute is similar to *aria-describedby* in that both reference other elements to calculate a text alternative. However, a label should be concise, whereas a description is intended to provide more detailed information. Depending on the type of screen-reader that is used, the attributes *aria-labelledby* and *aria-describedby* will either be announced upon opening the dialog, or they will be available to read via the virtual cursor.

This example follows the following important rules:

1. If you use the role “dialog”, you *must* use *aria-labelledby* to point to the element containing the visible dialog title. The *aria-labelledby* attribute must be *on the same HTML element* as the role “dialog” attribute.
2. If your dialog has one main descriptive text, you must use *aria-describedby* to point to its element, also on the *same* element that has role “dialog”.

10.7.2 Document landmark roles

Document landmarks are a subset of regular roles. Use them to help screen reader users understand the role of a section on a page and help orientate themselves within the document.

WAI-ARIA defines the following document landmark roles:

- *Article* (content that makes sense in its own right such as a complete blog post)
- *Banner* (site-orientated content, such as the title of the page and the logo)
- *Complementary* (supporting content for the main content)
- *Contentinfo* (information about the content)
- *Main* (main content)
- *Navigation* (content that contains the links to navigate this or related documents)
- *Search* (contains a search form to search the site)

The following example specifies the landmark roles⁴³ of *banner*, *navigation*, *main* and *contentinfo*. Landmark roles make navigation between areas of your page more efficient for

⁴³ http://www.w3.org/TR/wai-aria/roles#landmark_roles

different kinds of users. They help convey basic semantic intent and can serve as hooks and helpers for other technologies.

```
<header role="banner">*
  <p>Put company logo, etc. here.</p>
</header>
<nav role="navigation">*
  <ul>
    <li>Put navigation here</li>
  </ul>
</nav>
<main role="main">*
  <p>Put main content here.</p>
</main>
<footer role="contentinfo">*
  <p>Put copyright, etc. here.</p>
</footer>
```

*Within any document or application, do not mark more than one element each with the *main*, *banner* and *contentinfo* role.

WARNING: The WAI-ARIA role *navigation* does not work well with the HTML5 `<nav>` element when using *Windows Eyes* version 8 with Internet Explorer (IE). Using only `<nav>` is enough because this HTML5 element has achieved a wide approval and is well handled.

10.7.3 States and properties

Use WAI-ARIA *states* and *properties* in combination with *roles*. Changes in *states* or *properties* will notify the user of an AT that a change has occurred and help understand how to interact with the widget. The state identifies a unique configuration of information for an object. For example, the *aria-checked* property has three state values; *true*, *false* and *mixed*.

In the dialog example above extra information was provided to the AT user by using WAI-ARIA properties *aria-labelledby* and *aria-description*. By further adding the state *aria-hidden* screen-readers are prevented from interacting with the rest of the page once the modal dialog is open.

```
<main id="mainPage" role="main"
  aria-hidden="true">*
  <div id="modal" role="dialog"
    aria-hidden="false" **
    aria-labelledby="modalTitle" **
    aria-describedby="modalDescription"> **
  </div>
</main>
```

*When the modal dialog is displayed, the mainPage is marked with `aria-hidden='true'` to prevent screen readers from interacting with it once the modal dialog is open.

**By using state *aria-hidden* and properties *aria-labelledby* and *aria-describedby*, this information is conveyed to the AT.

A good rule of thumb is that elements that change how they look often have changing states. The dialog changes by going from being invisible to visible. So we are going to use `aria-hidden="true"` on the dialog initially and change its value when it is shown.

```
function showModal(obj) {  
    document.getElementById('mainPage').setAttribute('aria-hidden','true');  
    document.getElementById('modal').setAttribute('aria-hidden','false');
```

It is possible to use CSS selectors to show and hide the dialog based on the *aria-hidden* value. Then there is no need to change the CSS class in the code, only the value of *aria-hidden*.

```
[aria-hidden=true] {visibility: hidden;}
```

There is a *full list of states and properties*⁴⁴ to help define accessible widgets in the WAI-ARIA specification.

10.7.4 Live regions

WAI-ARIA is the *only* language capable of conveying updates to assistive technology. *Live regions* allow elements in a document to be announced if there are changes, without the user losing focus on their current activity. The *aria-live* property has a value indicating one of three verbosity levels in a region:

- Off: This indicates that the region is not live
- Polite: This indicates that it is not necessary to respond until user completes their current activity
- Assertive: This value is a higher priority than normal, but does not necessarily interrupt the user immediately

They are written as such:

```
<ul aria-live="off">  
<ul aria-live="polite">  
<ul aria-live="assertive">
```

Other important properties that can be used when defining *live regions* are:

⁴⁴ http://www.w3.org/TR/wai-aria/states_and_properties#state_prop_taxonomy

Aria-atomic: Indicates if AT should present all or only part of the changed region to the user. It has the values *true* or *false*. If this property is set to *true*, AT should present the entire region as a whole.

In the following example, all elements within an unordered list will be announced in their entirety when the region is spoken, unless another element further down the chain overrides the *aria-atomic* property.

```
<ul aria-atomic="true"  
  aria-live="polite">
```

Aria-busy: Prevents AT announcing changes before the updates are complete. It has the values *true* or *false*. If multiple parts of a live region need to be loaded before changes are announced to the user, the *aria-busy* property can be set to *true* until the final part is loaded, and then set to *false* when the updates are complete.

```
<ul aria-atomic="true"  
  aria-live="polite"  
  aria-busy="true">
```

Aria-relevant: Indicates what changes are considered relevant within a region. Accepts a space separated list of the following property values:

- Additions: Nodes are added to the DOM within the region.
- Removals: Nodes are removed from the DOM within the region.
- Text: Text is added or removed from the DOM.
- All: All of the above apply to this region.

In the absence of an explicit *aria-relevant* property, the default is to assume there are text changes and additions:

```
<ul aria-relevant="text additions">
```

The following example would only announce changes if nodes are added to the DOM within the region. If there are text changes, or nodes are removed within the region, the user will not be notified.

```
<ul aria-relevant="additions"  
  aria-atomic="true"  
  aria-live="polite">
```

10.7.5 Tabindex

The HTML *tabindex* attribute can be used to set tab structure on a page. Do not use *tabindex* as a replacement for a logical reading order. If mark-up has a logical structure you do not need

tabindex for interface elements that are already in the keyboard tab order, such as buttons, links and form elements.

WAI-ARIA extends *tabindex* to be:

- used on all visible elements
- given focus through scripting

WAI-ARIA allows a negative value (typically -1) to be specified for elements that should not appear in the keyboard tab order, but can be programmatically focused.

Tabindex is relevant every time you want to set focus to something, not just when using script. Use it for all widgets that have a series of components that need keyboard access, such as a tree. For example, a menu widget where the menu itself is in the tab order but the menu items is not. Instead, the menu items could be programmed so they can be navigated using cursor keys. This way, users do not have to tab through all items in the menu, and can better navigate the document.

The following example uses a *tabindex* attribute value of 0 to put a `<div>` element into the tab order so that a keyboard user can navigate to the element:

```
<div tabindex="0">
...
</div>
```

The following example uses a negative *tabindex* attribute value, so that the element is not placed in the tab order, but can receive programmatic focus:

```
<a href="accessibilityagent.no" id="progaccess" tabindex="-1">
42 reasons to implement accessibility
</a>
```

The following snippet of JavaScript selects the element defined above, and uses the *focus* method to place focus on the element:

```
var objDiv = document.getElementById('progaccess');
objDiv.focus();*
```

*Focus on the element

Related guidelines:

[WCAG Guideline 4.1.2 Name, Role, Value](#)⁴⁵

⁴⁵ <http://www.w3.org/TR/WCAG20/#ensure-compat>

10.8 Follow and validate the HTML5 standard

[Technology oriented](#), [Front-end developer](#)

Standards should be used because AT is developed according to standards. The HTML5 standard has stronger semantics than any older standard, which is beneficial for accessibility because it helps AT determining what the different elements are.

Declare doctype as such:

```
<!DOCTYPE html>
```

Validate using [Validator.nu \(X\)HTML5 Validator](#)⁴⁶ or [W3C mark-up validation service](#).⁴⁷

Using input types like *numbers*, *url* and *email* will provide the right keyboard popping up on touch screens and is also beneficial for live form validation.



Figure 11: Touch keyboard and input field with input type="number"
(Johansson, 2010)

⁴⁶ <https://html5.validator.nu/>

⁴⁷ <http://validator.w3.org/>

WARNING: Be aware that it may be using `<section>` and `<article>` because they have caused troubles for some screen readers.

Read more about HTML5 at the W3C Schools Introduction to HTML5.⁴⁸

Related guidelines:

[Difi UU-skolen: Kodestandarder](#)⁴⁹(Norwegian)

[Difi UU-skolen: Tekst og struktur](#)⁵⁰(Norwegian)

10.9 Combine WAI-ARIA and HTML5 mark-up

[Technology oriented](#), [Front-end developer](#)

Neither WAI-ARIA nor HTML5 is fully supported in all browsers and on all devices. It is therefore smart to combine them to get a fuller support. This is no problem to do. The code example on *Document landmark roles* did this. This is illustrated in figure 12.

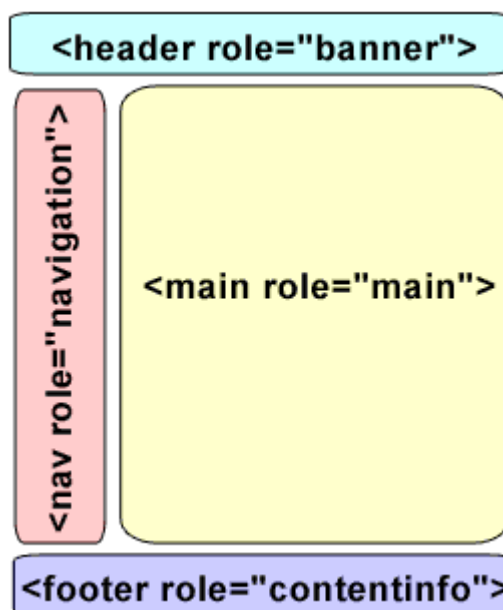


Figure 12: Combination of WAI-ARIA and HTML5 elements (Web accessibility eClass, undated)

⁴⁸ http://www.w3schools.com/html/html5_intro.asp

⁴⁹ <http://uu.difi.no/veiledning/nettsider/uu-skolen/kodestandarder>

⁵⁰ <http://uu.difi.no/veiledning/nettsider/uu-skolen/tekst-og-struktur>

Table 4 gives a brief overview of HTML5 elements and their WAI-ARIA equivalent.

HTML5	WAI-ARIA	Description
<u><header></u>	role="banner"	Introduction to a page or section. Can contain a heading (H1-H6), site logo, navigation.
<u><nav></u>	role="navigation"	Can be used for various types of navigation such as site navigation, sub-navigation, breadcrumbs, previous/next links.
<u><footer></u>	role="contentinfo"	Describes the page or a section of the page. A page's footer may contain author name, copyright info, privacy policy, etc.
<u><aside></u>	role="complementary"	Information that is tangentially related to the main page content, but can be read separately. Visually you might see this as a sidebar.
<u><article></u>	no equivalent	Independent item such as a blog post, article, etc. Think of it as something that could be independently picked up and moved around, such as blog posts in a RSS feed.

Table 4: HTML5 elements and WAI-ARIA equivalents

10.10 Use Progressive enhancement

[Technology oriented](#), [Front-end developer](#), [Graphic designers](#), [Interaction designers](#)

Progressive enhancement allows everyone to access the basic content and functionality of a web page, using any browser or Internet connection, while also providing an enhanced version of the page to those with more advanced browser software or greater bandwidth.

10.10.1 The core of Progressive enhancement

The core of *Progressive enhancement* is separating:

- Content and basic functionality (HTML)
- Presentation (CSS)
- Behaviour (JavaScript)

10.10.2 The principles of Progressive enhancement

The principles of *Progressive enhancement* are:

- Basic content should be accessible to all web browsers and devices
- Basic functionality should be accessible to all web browsers and devices
- Sparse, semantic mark-up contains all content
- Enhanced layout is provided by externally linked CSS
- Enhanced behaviour is provided by unobtrusive, externally linked JavaScript
- End-user web browser preferences are respected

10.10.3 The process of Progressive enhancement

1. Start with marking up the content. Make sure the mark-up conveys the greatest level of detail about the content it wraps around. HTML5 and WAI-ARIA goes a long way to serve this purpose. This is essential for offering a basic experience to:

- Search engines
- People on mobile devices
- People on old browsers

2. Create a separate CSS file and link to it as such:

```
<link rel="stylesheet" type="text/css" href="main.css">
```

This is for users who have:

- Basic CSS support
- Lack support for JavaScript

3. Create a separate JavaScript file and link to it as such:

```
<script src="main.js"></script>
```

Add JavaScript in an *unobtrusive* way (See next guideline).

Related guidelines:

[Difi UU-skolen: Utforming og presentasjon](#)⁵¹(Norwegian)

[WCAG technique SCR24: Use Progressive enhancement to open new windows on user request](#)⁵²

10.11 Use Unobtrusive JavaScript

[Technology oriented](#), [Front-end developer](#)

Unobtrusive JavaScript is about catering for people whose browser lack JavaScript support, or if JavaScript fails. This means not making JavaScript a requirement for a functional application. It is also about avoiding unnecessary movements on a web page, unintuitive widget functionality and unfamiliar controls. This is an advantage to everyone, but especially benefits keyboard and screen-reader users. This section also describes some general good coding practises with other benefits besides accessibility.

WARNING: Note that *Unobtrusive JavaScript* is not necessarily accessible JavaScript and is no guarantee for keyboard and screen reader access.

The seven rules of *Unobtrusive JavaScript*:

1. Do not make any assumptions
2. Find your hooks and relationships

⁵¹ <http://uu.difi.no/veiledning/nettsider/uu-skolen/utforming-og-presentasjon>

⁵² <http://www.w3.org/TR/WCAG20-TECHS/SCR24.html>

3. Use CSS to traverse the DOM
4. Understand browsers and users
5. Understand events
6. Play well with others
7. Work for the next developer

10.11.1 Do not make any assumptions

Do not expect JavaScript to be available, and do not expect the intended mark-up to be there.

Four things to keep in mind are:

- Do not expect browsers to support certain methods and have the correct properties, but test for them before accessing them
- Do not expect the correct HTML to be at your disposal, but check for it and do nothing when it is not available
- Keep functionality independent of input device
- Expect other scripts to try to interfere with the functionality and keep the scope of the scripts as secure as possible

10.11.2 Find your hooks and relationships

Before starting to plan a script:

- Look at the HTML the script will be enhancing
- Consider what is the best way of letting the script interact with it
- Consider the hooks and relationships in the HTML

HTML hooks are:

- Unique IDs (in valid HTML). Access them with the DOM method *getElementById*.
- HTML elements that can be retrieved with *getElementsByTagName* and CSS classes.

Regarding HTML relationships ask the following questions:

- How can I reach this element the easiest way and with the least steps traversing the DOM?
- What elements do I have to alter to update all the child elements which should be changed?
- What attributes does one element have that I can use to link to another element?

10.11.3 Use CSS to traverse the DOM

Using CSS to traverse the DOM is more effective, takes fewer resources and will not create any unnecessary dependency on JavaScript. In the example below, we perform a similar styling of images first with CSS and then with JavaScript.

```


```

```
.css-border{
  width: 200px;
  height: 200px;
  border:5px solid;
  border-radius:100px;
  border-bottom-left-radius:0;
}
```

```
(function(){
  var images = document.querySelectorAll('img');
  for (var i = 0; i < images.length; i++){
    images[i].style.border = '5px solid';*
    images[i].style.borderBottomLeftRadius = 0;*
    images[i].className='css-border';**
  }
})();
```

* Don't do this. Leave precise style details to the stylesheet

** It is best practice to dynamically manipulate classes via the className property

In many cases, and where possible, it really is best practice to dynamically manipulate classes via the className property since the ultimate appearance of all of the styling hooks can be controlled in a single *stylesheet*. That way the JavaScript code also becomes cleaner since instead of being dedicated to styling details, it can focus on the overall semantics of each section it is creating or manipulating, leaving the precise style details to the *stylesheet*.

Read more about using dynamic styling information in the article *Using dynamic styling information*⁵³.

Use JavaScript to enhance CSS

JavaScript can interact with *stylesheets*, allowing you to write programs that change a document's style dynamically.

There are three ways to do this:

- By working with the document's list of *stylesheets*—for example: adding, removing or modifying a *stylesheet*.

⁵³ https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model/Using_dynamic_styling_information

- By working with the rules in a *stylesheet*—for example: adding, removing or modifying a rule.
- By working with an individual element in the DOM—modifying its style independently of the document's *stylesheets*

```
(function(){
  var images = document.querySelectorAll('random-color-border');
  for (var i = 0; i < images.length; i++){
    var randomColor = '#'+(~~(Math.random()*(1<<24))).toString(16);*
    images[i].style.borderColor = randomColor;
  }
})();
```

* Dynamism can be implemented with the support of scripting. This is a progressive enhancement of the styling, not affecting functionality.

Play with this example of using JavaScript to enhance CSS in *Codepen*⁵⁴.

10.11.4 Understand users and browsers

You need to understand:

- how browsers work
- how browsers fail
- what users expect to happen

Do not wander too far from the way browsers work and how users expect them to work. Consider the following:

- Will the interface work independent of input device, and if not, what should be the fallback?
- Is the interface following rules of the browser or the rules of the rich interface? Is it for example possible to navigate a multi level menu with cursors or is tabbing required? Are some keyboard shortcuts overlooked? Try to keep the conventional keyboard shortcuts.⁵⁵
- What necessary functionality is dependent on JavaScript?

10.11.5 Understand events

Event handling helps with separating the JavaScript from the HTML and CSS, and also goes a bit further.

- The elements in the document are placed there to wait for handlers to listen to a change happening to them. When it happens the handlers retrieve an object (normally a parameter called *e*) that tells them what happened to what and what can be done with it.

⁵⁴ <http://codepen.io/anon/pen/wazZRR>

⁵⁵ <http://www.howtogeek.com/114518/47-keyboard-shortcuts-that-work-in-all-Web-browsers/>

Event handling does not only happen to the element you want to reach, but also to all the elements above it in the DOM hierarchy. (This does not apply to all events. *Focus* and *blur* do not do that.) This allows you to assign one single event handler to for example a navigation list and use the event handling's methods to reach the element in question. This technique is called *event delegation* and it has several benefits:

- You only need to test if a single element exists, not each of them
- It is possible to dynamically add or remove new child elements without having to remove or add new event handlers
- It is possible to react to the same event on different elements

The event handling follows an event order when there are elements inside elements as such:

```
<!DOCTYPE html>
  <html>
    <body>
      <h1>This example uses the addEventListener() to demonstrate event
order</h1>
      <div id="myDiv">
        <button id="myBtn">Click me</button>
      </div>
    </body>
  </html>
```

```
document.getElementById("myDiv").addEventListener("click", myFunction);
document.getElementById("myBtn").addEventListener("click", myFunction);
```

```
function myFunction() {
  document.getElementById("myDiv").insertAdjacentHTML('beforeend', '<p>You clicked
' + this + "</p>");
}
```

This will output:

You clicked [object HTMLButtonElement]

You clicked [object HTMLDivElement]

Play with this example of event listeners in *Codepen*.⁵⁶

There are two different event order models:

Event capturing: the outer element event takes place first. This means it starts capturing events from the outer element.

⁵⁶ <http://codepen.io/anon/pen/YXGvbb>

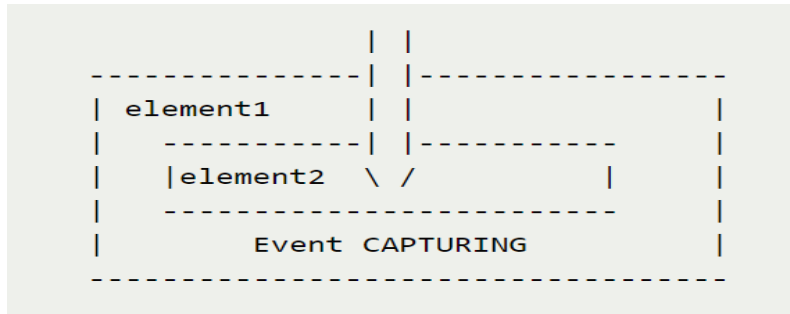


Figure 13: W3C event model - Event capturing (Quirksmode, undated)

Event bubbling: the inner element event takes place first. It starts capturing events from the inner element.

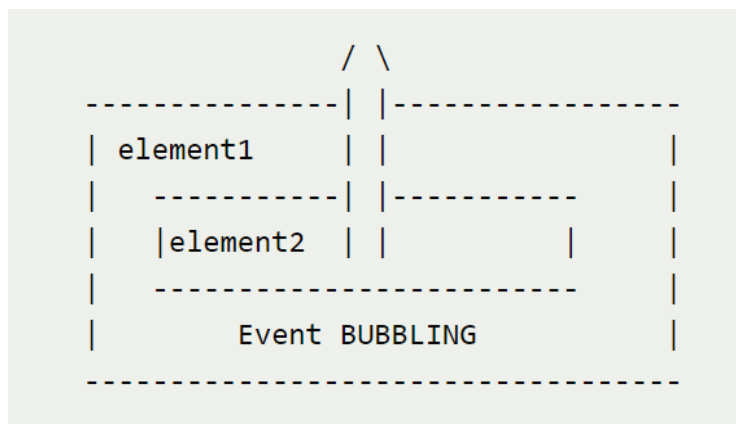


Figure 14: W3C event model - Event bubbling (Quirksmode, undated)

Event capturing and *event bubbling* can be combined so that events are first captured until it reaches the target element and then bubbles up again.

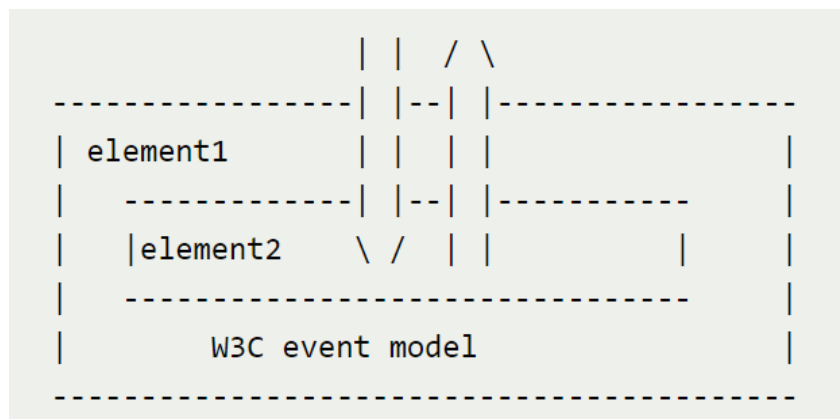


Figure 15: W3C event model - Combination of event capturing and event bubbling (Quirksmode, undated)

10.11.6 Play well with others

There will hardly ever be only one script used in a document. Make sure your script does not interfere with others, and make your script difficult to interfere with.

- The code should not have global function or variable names that others scripts can override
- Instantiate every variable using the *var* keyword
 - Declared variables are constrained in the execution context in which they are declared. Undeclared variables are always global.
 - Declared variables are created before any code is executed. Undeclared variables do not exist until the code assigning to them is executed.
 - Declared variables are a non-configurable property of their execution context (function or global). Undeclared variables are configurable (e.g. can be deleted).

Because of these three differences, failure to declare variables will very likely lead to unexpected results. Thus it is recommended to always declare variables, regardless of whether they are in a function or global scope.

```
var nav = document.querySelector('nav');*  
  
function init() {  
  if (nav.classList.contains('nav-hide')) {**  
    showNavigation();  
  }  
}  
  
function showNavigation() {  
  if (nav.classList.contains('nav-hide')) {  
    nav.classList.remove('nav-hide');  
    nav.classList.add('nav-show');  
  }  
}  
  
function hideNavigation() {  
  if (nav.classList.contains("nav-show")) {  
    nav.classList.remove("nav-show");  
    nav.classList.add("nav-hide");  
  }  
}  
  
init();
```

* The script has a global variable called `nav` which can be accessed from all the functions `init()`, `showNavigation()` and `hideNavigation()`. The functions can access the global variable and each other by name.

**HTML5's `classList` functionality (IE10+) makes adding and removing classes easy. You can feature detect if the browser supports it by using `if ("classList" in document.documentElement)`.

Play with this example of declaring variables in *Codepen*⁵⁷.

The object literal

Avoid all global code by wrapping the code in an object using the *object literal*. That way you turn the functions into methods and the variables into properties.

Define the methods and variables with a name followed by a colon and separate each of them from the others with a comma.

```
var myScript = {
  nav:document.querySelector('nav'),
  init:function(){
    myScript.showNavigation();
  },
  showNavigation:function(){
    if(myScript.nav.classList.contains('nav-hide')){
      myScript.nav.classList.remove('nav-hide');
      myScript.nav.classList.add('nav-show');
    }
  },
  hideNavigation:function(){
    if(myScript.nav.classList.contains('nav-show')){
      myScript.nav.classList.remove('nav-show');
      myScript.nav.classList.add('nav-hide');
    }
  }
}

myScript.init();
```

Play with this example of *Object literal* in *Codepen*⁵⁸

These methods can be accessed from outside and inside the object by pre-pending the object name followed by a full stop, illustrated by the example below. The drawback with this pattern is that the name of the object needs to be repeated every time it is accessed from another method. In addition, everything put inside the object is publicly accessible.

⁵⁷ <http://codepen.io/anon/pen/rVWBzy>

⁵⁸ <http://codepen.io/anon/pen/qdqWPo>

Module pattern

If you want to only make parts of the script accessible to other script in the document it is possible use the module pattern:

```
var myScript = function() {
  // these are private
  var nav = document.querySelector('nav');
  var init = function() {
    myScript.showNavigation();*
  }

  // these are public
  return {**
    showNavigation: function() {
      if (nav.classList.contains('nav-hide')) {
        nav.classList.remove('nav-hide');
        nav.classList.add('nav-show');
      }
    },
    hideNavigation: function() {
      if (nav.classList.contains('nav-show')) {
        nav.classList.remove('nav-show');
        nav.classList.add('nav-hide');
      }
    },
    init: init
  }
}();

myScript.init();
```

* Problem: to access one public method from another or from a private method you need to go through the verbose long name.

** Public methods and properties wrapped in a return statement and using the object literal

Play with this example of *Module patterns* in *Codepen*⁵⁹

You can access the public properties and methods that are returned the same way as in the object literal. The problem is that to access one public method from another or from a private method you need to go through the verbose long name again (the main object name can get rather long).

Module pattern – return an object with synonyms

To avoid repeating long, verbose names, define the methods as private and only *return an object with synonyms*.

⁵⁹ <http://codepen.io/anon/pen/jPVNzq>

```

var myScript = function() {
  // these are all private methods and properties
  var nav = document.querySelector('nav');

  var init = function() {
    showNavigation();*
  }

  var showNavigation = function() {
    if (nav.classList.contains('nav-hide')) {
      nav.classList.remove('nav-hide');
      nav.classList.add('nav-show');
    }
  }

  var hideNavigation = function() {
    if (nav.classList.contains('nav-show')) {
      nav.classList.remove('nav-show');
      nav.classList.add('nav-hide');
    }
  }

  //return objects with synonyms
  return {**
    showNavigation: showNavigation,
    hideNavigation: hideNavigation,
    init: init
  }
}();

myScript.init();

```

* We can now access showNavigation with its short name.

** Return public pointers to the private methods and properties you want to reveal

Play with this example of returning an object with synonyms in *Codepen*⁶⁰.

This allows for a consistency in coding style and gives the possibility to write shorter synonyms when they are revealed.

Anonymous function

To avoid revealing any methods or properties, it is possible to wrap the whole code block in an anonymous function and call it immediately after it was defined.

⁶⁰ <http://codepen.io/anon/pen/bdBbYG>

In the example below we use an anonymous function to progressively enhance the page by creating a print button only if the browser supports it. Notice how defensive the script is, we don't assume anything.

```
<p id="printThis">
  Thank you for your order. Please print this page for your records.
</p>
```

```
(function() {*
  if (document.getElementById) {**
    var printThis = document.getElementById('printThis');**
    if (printThis && typeof window.print === 'function') {***
      var printButton = document.createElement('input');****
      printButton.setAttribute('type', 'button');****
      printButton.setAttribute('value', 'Print this now');****
      printButton.onclick = function() {****
        window.print();****
      };
      printThis.appendChild(printButton);****
    }
  }
})();*
```

* To avoid leaving any global variables behind, we wrap the whole functionality in an *anonymous function* and immediately execute it - this is done with `(function(){})()`

** We test for DOM support and try to get the element we want to add the button.

*** Then test if the element exists and if the browser has a window object and a print method

**** We create a new click button and apply `window.print()` as the click event handler

***** The last step is to add the button to the paragraph.

Anonymous function is a great pattern for functionality that just needs to be executed once and has no dependency on other functions. This will make the code work well for the user and the machine it is running on as well as other developers.

Play with this example of anonymous functions in *Codepen*⁶¹.

Read more about *Progressive enhancement vs. Graceful degradation* on W3C⁶².

⁶¹ <http://codepen.io/anon/pen/WvoeKr>

⁶² http://www.w3.org/wiki/Graceful_degradation_versus_progressive_enhancement

10.11.7 Work for the next developer

Think about the developer who has to take over once this code is in production. Consider the following:

- Are all the *variable* and *function* names logical and easy to understand?
- Is the code logically structured? Is it possible to "read" it from top to bottom?
- Are the dependencies obvious?
- Are areas that might be confusing commented?

The HTML and CSS of a document is much more likely to change than the JavaScript as these make up visual output. It is therefore a good idea not to have any *class* and *ID* names or *strings* that will be shown to the end user buried somewhere in the code, but separate it out into a configuration object instead. This way maintainers know exactly where to change these without having to alter the rest of your code.

```
.show{display:block}
```

```
.hide{display:none}
```

```
<button onclick="myScript.showNavigation()">Show navigation</button>
```

```
<button onclick="myScript.hideNavigation()">Hide navigation</button>
```

```
<nav id="nav" class="show" >
```

```
  Page navigation
```

```
</nav>
```



```

var myScript = function() {
  var config = {
    navigationID: 'nav',
    visibleClass: 'show',
    invisibleClass: 'hide'
  };

  var nav = document.getElementById(config.navigationID);

  var showNavigation = function() {
    nav.classList.add(config.visibleClass);
    nav.classList.remove(config.invisibleClass);
  }
  var hideNavigation = function() {
    nav.classList.remove(config.visibleClass);
    nav.classList.add(config.invisibleClass);
  }
  return {
    showNavigation: showNavigation,
    hideNavigation: hideNavigation
  }
}();

```

* We define a *configuration object* that contains all hardcoded references to the classes used in our html, instead of having references to css classes spread all around out JavaScript code.

Try this example for yourself in this *Codepen*.⁶³

10.12 Make sure methods are device independent

[Technology oriented, Front-end developer](#)

Device independent JavaScript methods are not dependent on a specific device input. Using a combination of the following methods will ensure accessibility both from mouse, keyboard, touch and assistive technology. Ensuring keyboard access goes a long way for ensuring access to different AT.

Input methods for mouse are:

- *onMouseOver*
- *onMouseOut*
- *onClick* (Mostly works with the Enter key)

Input methods for keyboard are:

- *onFocus*

⁶³ <http://codepen.io/anon/pen/oXzMOj>

- *onBlur* (Especially practical for live validation)
- *onChange* (use with caution, goes against WCAG)
- *onSelect* (use with caution, goes against WCAG)
- *onkeypress*
- *onkeydown*
- *onkeyup*

Input methods for touch are:

- *touchstart*
- *touchmove*
- *touchend*
- *touchenter*
- *touchleave*
- *touchcancel*

Without *device independent methods* users may have problems with:

- Navigation
- Content being hidden from them
- Lack of control over automated content changes

Read more in *Accessible JavaScript –JavaScript event handlers*⁶⁴ or in Luke Wroblewskis book *Mobile and Multi-device design*.⁶⁵

WARNING: Use *onChange* and *onSelect* with caution as they go against WCAG 3.2.1. and 3.2.2

Related guidelines:

[Difi UU-skolen: Tastaturnavigering](#)⁶⁶ (Norwegian)

[WCAG Guideline 2.1 Keyboard accessible: Make all functionality accessible from a keyboard](#)⁶⁷

10.13 Use accessible modal windows instead of pop-ups

[Technology oriented, Front-end developer](#)

Modal windows should be used instead of pop-up windows, and should be made accessible with WAI-ARIA. In a modal dialog screen-readers will speak the title, the text, and the currently focused button automatically. In the following example the title is called “modalTitle” and the text is called “modalDescription”.

The first line declares a container that encompasses the whole dialog. HTML5 still does not have a proper dialog element that is supported everywhere. Therefore use the WAI-ARIA role “dialog”.

⁶⁴ <http://Webaim.org/techniques/javascript/eventhandlers>

⁶⁵ http://static.lukew.com/MobileMultiDevice_LukeWsm.pdf

⁶⁶ <http://uu.difi.no/veiledning/nettsider/uu-skolen/tastaturnavigering>

⁶⁷ <http://www.w3.org/TR/WCAG20/#keyboard-operation>

```

<main id="mainPage" role="main"
  aria-hidden="true">
  <div id="modal"
    role="dialog"
    aria-labelledby="modalTitle"
    aria-describedby="modalDescription"
    aria-hidden="false">
    <div id="modalDescription" class="screen-reader-offscreen">
      Beginning of modal dialog window example. Escape will cancel and close the
      window.
    </div>
    <h1 id="modalTitle">Modal dialog window example</h1>
    <p>
      These are the onscreen instructions that are not attached explicitly to a focusable
      element.
      Can screen reader users read this text with the virtual cursor?
    </p>
    <button id="modalCloseButton" title="Close modal dialog window">*
      
    </button>
  </div>
</main>

```

*The *modalCloseButton* is the only button and the first focusable element in the dialog. It is therefore the best place to set initial keyboard focus (as in: I can just press “return” and carry on with my task). Make sure each active dialog has a focused descendant element that has keyboard focus. When dialog is closed, you must also set focus back to the element that opened the dialog, or any other useful element from which the user of your app will most likely continue working.

Remember that WAI-ARIA merely provides semantic information. It does not automatically introduce certain types of behaviour. You have to use JavaScript to do that. To ensure accessibility remember the following:

- Set focus on the first keyboard focusable element within the dialog
- Trap keyboard focus and create a natural tab order inside the dialog
- Provide an escape route
- Restore focus after closing the dialog

```

function showModal(dialog) {

    // save current focus
    focusedElementBeforeModal = document.activeElement;

    // Focuses the close button
    dialog.find('button:last-of-type').focus()

    document.getElementById('mainPage').setAttribute('aria-hidden','true');
    document.getElementById('modal').setAttribute('aria-hidden','false');
}

function hideModal() {

    // Moves focus back on closing the dialog
    focusedElementBeforeModal.focus();

    document.getElementById('mainPage').setAttribute('aria-hidden','false');
    document.getElementById('modal').setAttribute('aria-hidden','true');
}

```

For more information on how to implement accessible modal dialogs and more complex examples read Marco Zehes article *Advanced ARIA Tip #2: Accessible modal dialogs*⁶⁸ and/or look at an example by Greg Kraus called *The Incredible Accessible Modal Window*⁶⁹, also available on GitHub.⁷⁰

10.14 Use technology that facilitates accessibility

[Technology oriented](#), [Front-end developer](#)

It has been said that accessibility is not a technical issue, it is a social one. The technical aids are there and it is very possible to create accessible *Rich Internet Applications* (RIA). However, it may take more time and it demands skills and knowledge about accessibility. This is expensive. Therefore using and promoting technology that facilitates accessibility and makes it "fast and easy" is important.

10.14.1 Libraries and frameworks

Use a framework that facilitates accessibility with:

- WAI-ARIA compliance
- Server rendering
- Warnings of inaccessible coding

⁶⁸ <https://www.marcozehe.de/2015/02/05/advanced-aria-tip-2-accessible-modal-dialogs/>

⁶⁹ <http://accessibility.oit.ncsu.edu/training/aria/modal-window/version-2/>

⁷⁰ <https://github.com/gdkraus/accessible-modal-dialog>

- Components with keyboard support

Some frameworks might not be very good with accessibility, but offer external frameworks which adds accessibility on top.

A few possibilities are:

- *React* (server rendering, warnings when writing inaccessible code)
- *jQuery* (WAI-ARIA compliance and keyboard support)
- *Bootstrap* (Has a plug-in that adds accessibility to all their default plug-ins)

WARNING: Remember that you always have to check that plug-ins is accessible. Do not rely on the framework alone.

10.14.2 Web components

Creating accessible *Web components* will facilitate accessibility in the future through reuse of these components. How accessible a *Web Component* is, is up to the developer. Consider many of the same things as when creating RIAs in general:

- Add WAI-ARIA
- Resizable text
- Give it an alternative text
- Make sure it works without audio
- Ensure keyboard functionality
- Ensure it is manageable to operate with a screen reader

To learn about *Web components* start by reading the W3C introduction.⁷¹

W3C also provides some good examples of *Web components*.⁷²

When you want to start using *Web components* the *official website*⁷³ gives a general overview of:

- specifications used in *Web components*
- libraries you can use to create your own *Web components*
- browser support
- general discussion and community

The *Web components wiki*⁷⁴ gives information and provides links to tutorials for:

- HTML imports⁷⁵
- Custom elements⁷⁶
- The Shadow DOM⁷⁷

⁷¹ <http://www.w3.org/TR/2013/WD-components-intro-20130606/>

⁷² <https://dvcs.w3.org/hg/Webcomponents/raw-file/57f8cfc4a7dc/samples/index.html>

⁷³ <http://Webcomponents.org>

⁷⁴ <http://www.w3.org/wiki/WebComponents/>

⁷⁵ <http://www.html5rocks.com/en/tutorials/Webcomponents/imports/>

⁷⁶ <http://www.html5rocks.com/en/tutorials/Webcomponents/customelements/>

There are many other resources for information and examples.^{78 79 80 81}

You can also read more about creating accessible *Web components* in the article *Accessible Web components*.⁸²

⁷⁷ <http://www.html5rocks.com/en/tutorials/Webcomponents/shadowdom/>

⁷⁸ <http://customelements.io/>

⁷⁹ <http://component.kitchen/components/pveyes/Web-resume>

⁸⁰ <http://www.w3.org/standards/techs/components>

⁸¹ <http://w3c.github.io/Webcomponents/spec/custom/>

⁸² <https://www.polymer-project.org/articles/accessible-Web-components.html>

11. Conclusions

A set of accessibility guidelines has now been created, evaluated and updated. A second version of the guidelines has been presented as the result of this project. In the following chapters the conclusions drawn from this study and answers to the research questions is presented. The research questions were:

RQ 1: What problems exist with RIA accessibility?

RQ 2: What can be done to avoid RIA accessibility problems?

RQ 3: What makes accessibility guidelines manageable for developers?

11.1 What problems exist with RIA accessibility?

Several issues concerning *Rich Internet Application* (RIA) accessibility have been uncovered. They can be divided in to three areas:

- *Social issues* which include the difficulties with convincing colleagues, clients and management of the importance of accessibility, the fact that accessibility is a massive, complicated and time-consuming subject and competence and knowledge about accessibility is low. Lastly, there is also the problem that some people are still questioning the need for accessibility.
- *Tool issues* are related to the difficulties developers face when working with accessibility guidelines and automatic testing tools. They are time-consuming and difficult to apply and get familiar with and it can be difficult to understand how to use them. In addition they are not always 100 % reliable or valid.
- *RIA issues* address technical issues that cause problems for users. They include assistive technologies (AT) having trouble keeping up with Web 2.0, updates not being detected, non-existing semantics, standard violations and errors, navigation problems, problems with keyboard access, pop-up windows and over-engineered user interfaces.

It is important to note that although these are the problems discovered during this project, it does not mean that this list is exhaustive. There may be other problems yet undiscovered. These might reveal themselves during further research and development of the guidelines.

11.2 What can be done to avoid RIA accessibility problems?

Most of the RIA accessibility issues discovered in the literature study can be avoided using one or more of the solutions presented in the guidelines. The solutions are both technically oriented and process oriented. The technical issues and solutions are rather straight forward. Doing one thing has direct impact on another. When it comes to some of the social issues the answer seems to lie with facilitating accessibility and creating a smooth process. Difficulties with tools may not be so easy to avoid. The tools are, to some extent, comprehensive and complex because of the complex nature of accessibility. Having expertise deal with these aspects seems like the only reasonable solution. The following is a short version of the second version of the guidelines, which was the end result of this project.

1. Have accessibility expertise on the team
2. Introduce accessibility from the beginning
3. Communicate accessibility within the team
4. Follow existing design principles
5. Test accessibility at key stages
6. Use WAI-ARIA mark-up
7. Follow and validate the HTML5 standard
8. Combine WAI-ARIA and HTML5 mark-up
9. Use *Progressive enhancement*
10. Use *Unobtrusive JavaScript*
11. Make sure methods are device independent
12. Use accessible modal windows instead of pop-ups
13. Use technology that facilitates accessibility

Just as with the accessibility issues it is important to note that this list of *solutions* neither is exhaustive. There will most likely be other solutions to some of the same issues, and solutions addressing issues not brought up here. This does however serve as a basic understanding of what ensuring accessibility entails in a *front-end* perspective and in a *process* perspective.

11.3 What makes guidelines manageable for developers?

Some re-occurring things have come up that seems to make guidelines more manageable for developers. This indicates that there are some steps one can take to ensure usability of guidelines. Guidelines that give concrete advice specifically on *what* to do to ensure accessibility seem to be quite straightforward to apply. Guidelines should also be short and easy to quickly get familiar with. A check-list with concrete advice offering the possibility of quickly checking that everything is in order seems preferable. Links from there to find more information if necessary was appreciated. This makes the guidelines easy to relate to and efficient to apply. Knowing *why* something is essential for accessibility has been expressed as important, both in the literature and by the participants. It seems this makes guidelines clearer and easier to understand.

Guidelines that have a scope also seem to be more manageable than guidelines that try to cover everything. It makes them smaller and less time-consuming to get familiar with. It was recognized however that having a scope means one cannot rely on the guidelines ensuring 100% accessibility in all aspects. It seemed it was advisable to direct each guideline towards a profession within the team. It gives the developers a sense of what they are responsible for, and it is easier to relate to a shorter set of guidelines directly linked to ones own profession. This means they do not have to manage a full set with technical language or technology they may be unfamiliar with or unsure of.

It also seems guidelines that are prioritised are very useful for developers. This means they can know what is most important to ensure accessibility and make priorities according to this. It has however been discussed whether or not prioritising is a good idea. Guidelines that are time estimated are also useful, because it lets the user know how much time to set aside for working

with the guidelines. Nevertheless, time estimation may be hard and very different from project to project.

When the guidelines guide developers in how to work with them during the development process, they become more manageable. It helps developers understand how to use the guidelines and how to incorporate them in their work. In addition, having knowledge about accessibility will always make a set of guidelines easier to apply. Someone with knowledge will not have to spend time learning about the different areas of accessibility, but rather use a guideline set as insurance that they have remembered everything. It will work as a support tool for them, rather than a way of learning everything they need to know about accessibility. It means they will more quickly familiarise themselves with the guidelines and be able to use them more efficiently.

12. Contribution to research and future work

This project represents the beginning of development of guidelines for how to create accessible *Rich Internet Applications* (RIA). Developing guidelines is a long process demanding several iterations and continued maintenance and updates. Some things should be done before the next round of interviews and others might need a few more rounds of further studies. There are also some questions that have come up during this process, which would need extensive further studies. The following chapters present proposals for possible ways to continue work with these guidelines in the future.

12.1 What to do before the next round of interviews?

Before the next round of interviews a lengthier explanation of *why* each individual guideline is good for accessibility and the consequences of following it should be provided along with more links to additional information. It would also be possible to make *all* the examples interactive.

12.2 What to study before the next round of interviews?

Before the next round of interviews some studies should be made of what is required for a framework to facilitate accessibility and tools that were recommended by the participants should be checked out. The extent of the issues with *<article>* and *<section>* and compatibility issues with WAI-ARIA, HTML5, AT and browsers should be studied along with when it is smart to use the different types of input methods. An examination should also be done of the possibilities of making data available in different ways beyond the statements of WCAG and use of JavaScript to hide content and *if or how* this benefits accessibility. Lastly, *focus order* is an important issue that has not been given much attention in this project.

Recommendations for *focus order* should therefore be further studies so they can be placed in the guidelines.

12.3 What needs further studies?

Some issues that have emerged during this project that require extensive further studies across several iterations of guideline development. These are:

- Reliability of guidelines
- Prioritization of guidelines
- Time estimation of guidelines
- Further development of the *process oriented* guidelines
- Guidelines related to other aspects of web development
- Number of users with disabilities for accessibility testing
- Further development of website where the guidelines are presented
- Presenting the guidelines to be used actively internally within a project

12.3.1 Reliability of guidelines and prioritization

Establishing reliability beyond what has been possible during this project is essential for further development. One way to test reliability of the guidelines is to apply them to a development

process and afterwards submit the application to extensive accessibility testing. To establish the guidelines' involvement in making the application accessible one would have to create a set of testable hypotheses linked to the guidelines. For example:

- Application of WAI-ARIA semantics makes it easy to navigate an application with a screen reader.
- Use of device independent JavaScript methods makes an application easy no navigate through keyboard
- Use of *Progressive enhancement* will provide a useful application even if JavaScript and CSS is turned off

A severe lack of involvement of target group in the research of RIA accessibility has been noted (Dell Anhol Almeida and Calani Baranauskas, 2012). This method would directly involve both the users of the guidelines, i. e. the web developers, and the beneficiaries of the guidelines, i. e. the end user with some sort of disability. Once some grade of reliability has been established, one can easier prioritize the guidelines and say which the most important ones are.

Not all the guidelines are fit to be evaluated for reliability this way. Especially with the *process oriented* guidelines it would be difficult to apply this method. In the following chapter suggestions of possible ways of further development and evaluation the *process oriented* guidelines is presented.

12.3.2 Further development of the process oriented guidelines

One possible way to work with the *process oriented* guidelines would be to do a qualitative study of how IT companies bring accessibility into their development process and compare this to the level of accessibility on the applications they develop. Taking this a step further would be to apply the *process oriented* guidelines to a development process and analyse the result comparing it with an application developed without use of these guidelines. This would have to be done quite a few times to be able to say something definite. However, a few rounds of analysis would be able to give some indications.

Other possibilities are further literature studies and asking accessibility experts to evaluate the *process oriented* guidelines. It would also be possible to do interviews with accessibility experts to establish their view of what is needed in a development process to ensure accessibility. If for example seven different experts express some of the same views, this would be a good indication of reliability.

12.3.3 Guidelines related to other aspects of web development

Extending the guidelines to address other areas of web development like *graphic* and *interaction design* and *information architecture* is a large project. It might start out in the same way to this project did with a thorough literature study as a basis for a first version of a set of guidelines with subsequent interviews as an initial evaluation. Some of these guidelines might further be tested for reliability with measurable hypotheses similar to the method proposed earlier in this chapter.

12.3.4 Guidelines' relevance in different development processes

Further studies of how relevant the guidelines would be in different web development processes would be very interesting. This would however take some time. The guidelines would have to be applied in many processes and their relevance would have to be thoroughly analyzed based on type of process, accessibility requirements in the project, team members knowledge about accessibility, what technology is used and many other parameters. Nevertheless, studies of this would be extremely valuable to estimate the value and usefulness of the guidelines.

12.3.5 Time estimation for each guideline

To estimate time spent on each guideline one would have to study developers applying them to their processes and register how much time every developer spend on every guideline. By doing this for a period, it might be possible to come up with an indication of how much time is spent on an average. However, given the comprehensive nature of some of these guidelines, it might be difficult. Different people work in different ways and have different knowledge levels, which will have impact on the time spent. It will also be different from project to project. There will be many variables to take into account in a study like this.

12.3.6 Number of users with disabilities for accessibility testing

Figuring out how many users is needed for accessibility testing is a complex task. It may be different from project to project and also depends on the users doing the testing. However, interviewing people who are experts in user testing with users with disabilities, for example the employees at *MediaLT*⁸³, will be a good place to start. Other than that one has to do user testing, record number of errors found, and analyze it against the full number of errors discovered when the project is finished and the user testing is over. Having users with different kinds of disabilities test the system is a good place to start, but it is also important to remember that even if someone has the same disability it does not mean they use the Web in the same way or have the same level of IT competence.

12.3.7 Further development of website where the guidelines are presented

The website where the guidelines are available should at least live up to the standards in the guidelines. As of now this website is still under development. Finalizing the website has not been top priority when concluding this project within the deadline. It needs be consistently tested for usability and accessibility while it is being updated and maintained.

12.3.8 Presenting the guidelines for active internal use

Presenting the guidelines in a way so they can be used actively within a project would mean having to design and create an interactive tool providing possibilities for internal discussion of each guideline within the bounds of a project. The tool would also have to be evaluated in terms of usability and usefulness to developers. This would be a very interesting project, which could have a great deal of value for facilitating use of these guidelines.

⁸³ <http://www.medialt.no/>

12.4 Most important contribution to web accessibility research

Guidelines for RIA accessibility are still in early stages (Dell Anhol Almeida and Calani Baranauskas, 2012). Guidelines focusing only on the *front-end* technologies and especially JavaScript are an important contribution, because it focuses on the most complex areas of web development and accessibility; maintaining accessibility in highly dynamic and interactive applications. Nevertheless, although not gathered in this fashion, this is something there already is quite a lot of knowledge about.

It is therefore the authors believe that the most important contribution to the field of web accessibility research of this project is:

1. Strong indications for a need for *process oriented* accessibility guidelines. All participants genuinely appreciated this and mentioned they had not seen something like it, including the accessibility expert.
2. The beginning of the creation of a set of *process oriented* accessibility guidelines and suggestions for continued work with these guidelines.

Other fields within web development like *programming*, *graphic* and *interaction design* and *information architecture* are well established and well integrated in the process. Accessibility, although not a new field, does not have the same traction. It seems there are uncertainties as to how to integrate it into the process, and guidelines on how to do this would be a welcome solution to this issue. It is therefore particularly important that work should continue with studies and development of the *process oriented* guidelines. It is highly likely that making the process of accessibility smooth and easy will solve many of the social issues with web accessibility and as a result future applications will attain a higher level of accessibility.

13. Reference list

AbleTech: Assistive Technologies Inc. (2010). Software solutions. Retrieved from <http://abletech.ca/products/computer-access/software-solutions/>

Almeida, L. D. A., & Baranauskas, M. C. C. (2012, November). Accessibility in rich internet applications: people and research. In *Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems* (pp. 3-12). Brazilian Computer Society.

Assistive Technology (2014, February 26th) In Wikipedia. Retrieved March 3rd 2014 from http://en.wikipedia.org/wiki/Assistive_technology

Berget, G. & Moseid, T. (2012). Mind the Gap: Mellom oppsøkende bibliotekjenester og det universelt utformede bibliotek. I: R. Audunson (Red.). *Krysspeilinger: Perspektiver på bibliotek- og informasjonsvitenskap*. Oslo: ABM-Media. s.185-204

Bingham, C., Clarke, L., Michielsens, E., & Van de Meer, M. (2013). Towards a social model approach? British and Dutch disability policies in the health sector compared. *Personnel Review*, 42(5), 613-637.

Braga, J., Damaceno, R., Leme, R., & Dotta, S. (2012, January). Accessibility Study of Rich Web Interface Components. In *ACHI 2012, The Fifth International Conference on Advances in Computer-Human Interactions* (pp. 75-79).

Brajnik, G. (2009, October). Validity and reliability of web accessibility guidelines. In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility* (pp. 131-138). ACM.

Brown, A., Jay, C., Chen, A. Q., & Harper, S. (2012). The uptake of Web 2.0 technologies, and its impact on visually disabled users. *Universal Access in the Information Society*, 11(2), 185-199. doi: 10.1007/s10209-011-0251-y

Buckler, C. (2009, September 22nd). *Progressive enhancement and graceful degradation: an overview*. [blog post]. Retrieved from <http://www.sitepoint.com/progressive-enhancement-graceful-degradation-basics/>

BuiltWith. (2014). JavaScript usage statistics. Retrieved March 17th 2014 from <http://trends.builtwith.com/docinfo/Javascript>

Cascading Style Sheets. (2014, September 14th). In Wikipedia. Retrieved October 1st 2014 from http://en.wikipedia.org/wiki/Cascading_Style_Sheets

Connor, J. O. (2012). *Pro HTML5 accessibility*. Apress.

Cooper, M. (2007, May). Accessibility of emerging rich web technologies: web 2.0 and the semantic web. In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)* (pp. 93-98). ACM. doi: 10.1145/1243441.1243463

Cooper, M. (2014, March 20th) The W3C blog [Blog post]. Retrieved from <http://www.w3.org/blog/2014/03/wai-aria-expands-Web-accessibility/>

CRPD. (2006). *UN Convention on the Rights of Persons with Disabilities*. United Nations.

Difi (2015, January 19th). Kodestandarder. Retrieved from <http://uu.difi.no/veiledning/nettsider/uu-skolen/kodestandarder>

Fadeyev, D. (2008, August 30th). Usability tip: use verbs as labels on buttons. Retrieved from <http://usabilitypost.com/2008/08/30/usability-tip-use-verbs-as-labels-on-buttons/>

Faulkner, S. (2012, July 6th). *Notes on Web components + ARIA*. [blog post]. Retrieved from <http://www.paciellogroup.com/blog/2012/07/notes-on-Web-components-aria/>

Fentek Industries. (2015). Foot controlled, no hand mouse. Retrieved from <http://www.fentek-ind.com/nh-mouse.htm#.VV-Mifntmkp>

Fergus Ferguson, R., & Heilmann, C. (2013). *Beginning JavaScript with DOM Scripting and Ajax: Second Edition*. Apress.

Fernandes, N., Batista, A. S., Costa, D., Duarte, C., & Carriço, L. (2013, May). Three web accessibility evaluation perspectives for RIA. In *Proceedings of the 10th International cross-disciplinary conference on web accessibility* (p. 12). ACM. doi: 10.1145/2461121.2461122

Foley, A., & Regan, B. (2002). Web design for accessibility: Policies and practice. *AACE Journal*, 10(1), 62-80.

Garcia-Izquierdo, F. J., & Izquierdo, R. (2012). Is the browser the side for templating?. *Internet Computing, IEEE*, 16(1), 61-68. doi: 10.1109/MIC.2011.81

Gustafson, A. (2008, October 7th). Understanding progressive enhancement. *A List Apart*, 269. Retrieved from <http://alistapart.com/article/understandingprogressiveenhancement>

Gustafsson, A. (2008, October 21st). Progressive enhancement with CSS. *A list apart*, 270. Retrieved from <http://alistapart.com/article/progressiveenhancementwithcss>

Heilmann, C. (2007). *The seven rules of Unobtrusive JavaScript*. [blog post] Retrieved from <http://icant.co.uk/articles/seven-rules-of-unobtrusive-javascript/>

Hoffman, A. (2014, May 13th). Accessibility: The missing ingredient. *A list apart*, 395. Retrieved from <http://alistapart.com/article/accessibility-the-missing-ingredient>

HTML. (2014, September 14th). In Wikipedia. Retrieved October 1st 2014 from <http://en.wikipedia.org/wiki/HTML>

HTML5. (2014, September 18th). In Wikipedia. Retrieved October 1st 2014 from <http://en.wikipedia.org/wiki/HTML5>

JavaScript (2014, April 25th). In Wikipedia. Retrieved April 28th from <http://en.wikipedia.org/wiki/JavaScript>

Johansson, R. (2010, April 12th). HTML5 input types. [blog post]. Retrieved from http://www.456bereastreet.com/archive/201004/html5_input_types/

Johansson, R. (2010, January 26th). Unobtrusive JavaScript is not necessarily accessible JavaScript. [blog post]. Retrieved from http://www.456bereastreet.com/archive/201001/unobtrusive_javascript_is_not_necessarily_accessible_javascript

Kern, W. (2008). Web 2.0-end of accessibility? analysis of most common problems with Web 2.0 based applications regarding Web accessibility. *International Journal of Public Information Systems*, 4(2).

Krug, S. (2006). *Don't make me think: A common approach to Web usability*. Berkeley: New Riders.

Kvale, S., & Brinkmann, S. (2009). *Interviews: Learning the craft of qualitative research interviewing*. Sage.

Lazar, J., Dudley-Sponaugle, A., & Greenidge, K. D. (2004). Improving web accessibility: a study of webmaster perceptions. *Computers in Human Behavior*, 20(2), 269-288. doi:10.1016/j.chb.2003.10.018

Lazar, J., Feng, J. H., & Hochheiser, H. (2010). *Research methods in human-computer interaction*. John Wiley & Sons.

Lawson, B. and Faulkner, S. (2011, May 25th). HTML5 and accessibility. MSDN Magazine. Retrieved from <http://msdn.microsoft.com/en-us/magazine/hh204741.aspx>

Lawson, B. (2009, January 15th). *Redesigning with HTML and WAI-ARIA*. [blog post]. Retrieved from <http://www.brucelawson.co.uk/2009/redesigning-with-html-5-wai-aria/>

Lawson, B. (2014, March 24th). *Notes on accessibility of Web components*. [blog post]. Retrieved from <http://www.brucelawson.co.uk/2014/notes-on-accessibility-of-Web-components/>

Lemon, G. (2008, August 1st). Introduction to WAI-ARIA. Opera Software [blog post]. Retrieved from <https://dev.opera.com/articles/introduction-to-wai-aria/>

Mankoff, J., Faith, H., & Tran, T. (2005, April). 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* (pp. 41-50). ACM.

McAllister, N. (2012, March 23rd). Web components: New hope fro Web designers. [blog post]. Retrieved from <http://www.infoworld.com/article/2619725/Web-development/Web-components--new-hope-for-Web-designers.html?page=1>

Merayo, R. V. (2011). Rich Internet Applications (RIA) y Accesibilidad Web. *Hipertext. net*, (9), 2.

Moreno, L., Martínez, P., Ruiz, B., & Iglesias, A. (2010). Toward an equal opportunity web: Applications, standards, and tools that increase accessibility. *Computer*, (5), 18-26. doi: 10.1109/MC.2010.370

Nederlof, A., Mesbah, A., & Deursen, A. V. (2014, May). Software engineering for the web: the state of the practice. In *Companion Proceedings of the 36th International Conference on Software Engineering* (pp. 4-13). ACM. doi: 10.1145/2591062.2591170

Nielsen, J. (1994). *Usability engineering*. Elsevier.

Park, E. J., Lim, Y. W., & Lim, H. K. (2014). A Study of Web Accessibility of Websites Built in HTML5-Focusing on the Top 100 Most Visited Websites-. *International Journal of Multimedia & Ubiquitous Engineering*, 9(4).

Quirksmode. (undated). Event order. Retrieved from http://www.quirksmode.org/js/events_order.html

RehabMart. (2012, May 1st). Assist Pediatrics with Assistive Technology. Retrieved from <http://library.rehabmart.com/?filename=assist-pediatrics-with-assistive-technology>

Rosson, M. B., Ballin, J. F., Rode, J., & Toward, B. (2005). "Designing for the web" revisited: a survey of informal and experienced web developers. In *Web Engineering* (pp. 522-532). Springer Berlin Heidelberg. doi: 10.1007/11531371_66

Rosson, M. B., Ballin, J., & Rode, J. (2005, September). Who, what, and how: A survey of informal and professional web developers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on* (pp. 199-206). IEEE.
doi: 10.1109/VLHCC.2005.73

Sandnes, F. E. (2011). *Universell utforming av IKT-systemer: Brukergrensesnitt for alle*. Oslo: Universitetsforlaget

Sutton, M. (2014, February 5th). *Accessibility and the Shadow DOM*. [blog post]. Retrieved from <http://substantial.com/blog/2014/02/05/accessibility-and-the-shadow-dom/>

Switch (2014, March 1st) In Wikipedia. Retrieved March 3rd 2014 from http://en.wikipedia.org/wiki/Computer_accessibility

Tanaka, E. H., & Da Rocha, H. V. (2011, October). Evaluation of web accessibility tools. In *Proceedings of the 10th Brazilian Symposium on on Human Factors in Computing Systems and the 5th Latin American Conference on Human-Computer Interaction* (pp. 272-279). Brazilian Computer Society.

Tollefsen, M. (2011). Universell utforming som prosess, virkemiddel og mål i utvikling av ny programvare. Media LT. Retrieved May 20th 2014 from <http://www.medialt.no/universell-utforming-som-prosess-virkemiddel-og-maal-i-utvikling-av-ny-programvare/1016.aspx#gen30>

Trewin, S., Cragun, B., Swart, C., Brezin, J., & Richards, J. (2010, April). Accessibility challenges and tool features: an IBM Web developer perspective. In *Proceedings of the 2010 international cross disciplinary conference on web accessibility (W4A)* (p. 32). ACM. doi: 10.1145/1805986.1806029

UX Movement. (2011, March 19th). Why modal windows have killed popup windows. Retrieved from <http://uxmovement.com/forms/why-modal-windows-have-killed-popup-windows/>

WebAim. (2012, October 12th). Motor disabilities. Retrieved from <http://Webaim.org/articles/motor/motordisabilities>

WebAim. (2012). Screen reader user survey #Results. Retrieved from <http://Webaim.org/projects/screenreadersurvey4/>

WebAim (2013, October 24th). Accessible JavaScript. Retrieved from <http://Webaim.org/techniques/javascript/other>

WebAim (2013, August 28th). Visual Disabilities. Retrieved from <http://webaim.org/articles/visual/blind>

Web accessibility eClass. (undated). Retrieved from http://www.d.umn.edu/~lcarlson/eclasses/accessibility/103_structure/1.3_lecture_landmarks.html

Web Accessibility Initiative. (2014, February 12th). In Wikipedia. Retrieved from http://en.wikipedia.org/wiki/Web_Accessibility_Initiative

Web Accessibility Initiative. (2014, June 12th). WAI-ARIA Overview. Retrieved from <http://www.w3.org/WAI/intro/aria>

World Wide Web Consortium. (2014, February 12th). In Wikipedia. Retrieved March 3rd 2014 from http://en.wikipedia.org/wiki/World_Wide_Web_Consortium

WHO. (2011). World report on disabilities. Retrieved from http://whqlibdoc.who.int/publications/2011/9789240685215_eng.pdf?ua=1

Wright, T. (2012). *Learning JavaScript: A Hands-On Guide to the Fundamentals of Modern JavaScript*. Addison-Wesley.

W3C (2011, October). Strategic plan for Web accessibility. Retrieved from <http://www.w3.org/WAI/impl/>

W3C (2002, October 11th) Implementation Plan for Web Accessibility. Retrieved from <http://www.w3.org/WAI/impl/expanded.html>

W3C (2005, September). Introduction to web accessibility. Retrieved from <http://www.w3.org/WAI/intro/accessibility.php>

W3C (2008, Desember 11th). Web Content Accessibility Guidelines (WCAG) 2.0. Retrieved 2014 from <http://www.w3.org/TR/WCAG20/>

W3C (2013, June 6th). Introduction to Web components. Retrieved from <http://www.w3.org/TR/2013/WD-components-intro-20130606/>

W3CSchools. (2014). JavaScript Tutorial. Retrieved from <http://www.w3schools.com/js/DEFAULT.asp>

Yesilada, Y. Brajnik, G. and Harper, S. 2009. How much does expertise matter?: a barrier walk-through study with experts and non-experts. In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility* (pp. 203-210)

YouTube (2013, August 16th). How a blind person uses the computer [Video clip]. Retrieved from <https://www.youtube.com/watch?v=UzffnbBex6c>

YouTube (2012, April 10th). How blind people use Twitter & YouTube on the iPhone 4S [Video clip]. Retrieved from <https://www.youtube.com/watch?v=c0nvdiRdehw>

YouTube (2012, July 11th). The Grid 2 – Switch scanning [Video clip]. Retrieved from <https://www.youtube.com/watch?v=1k-OMnJdxT0>

YouTube (2009, February 24th). Augie Nieto - EyeMax Eye Tracking AAC Device [Video clip]. Retrieved from <https://www.youtube.com/watch?v=gDKFNqgmtZ4>

Zehe, M. (2015, February 5th). Advanced ARIA Tip #2: Accessible modal dialogs. [blog post]. Retrieved from <https://www.marcozehe.de/2015/02/05/advanced-aria-tip-2-accessible-modal-dialogs/>

Appendix 1: Guidelines 1st version

1. Process oriented guidelines

1.1 Have accessibility expertise on the team

(Project manager)

One or more people on the team should have experience with:

- Use of testing tools for accessibility
- Different kinds of assistive technology and how they work
- Good coding practices to ensure things like keyboard access and understandable navigation for a screen reader
- Simulation exercises
- User testing with users with disabilities

1.2 Implement accessibility from the beginning of a project

(Project manager)

Include accessibility throughout all the process phases:

- Information gathering
- Planning phase
- Designing phase
- Development phase
- Testing phase
- Maintenance

Include accessibility on an equal basis as:

- Usability
- User experience (UX)

Keep accessibility in mind when deciding what libraries and frameworks to use. Some frameworks will make the implementation of WAI-ARIA easier:

- Dojo (Open source library that uses the WAI-ARIA specification)
- *jQuery* (Several widgets and plug-ins have accessibility built in)
- Bindows (Section 508 compliance. Not open source)
- Fluid Infusion (Sits on top of the *jQuery* toolkit. Applies user centred design principles to provide an inclusive toolkit)

Apply user-centred design process and include users with impairments.

- Iterations
 - Test with representative users

- Make changes
- New iteration

This process will help you:

- Better the interface
- Understand more of the users need

Test with both users with and without disabilities.

1.3 Test accessibility at key stages

(Project manager and testers)

Key stages for testing are:

- Designing phase (Start testing with sketches and work your way from there)
- Development phase (Use automatic testing tool from your first HTML line)

Test using the following methods:

- Automatic accessibility testing
- Expert based testing
- Simulation
- User testing

1.3.1 Automatic tools

Some examples of automatic tools you should use are:

[Web Accessibility Evaluation Tool \(WAVE\)](#)⁸⁴

Checks structure, alternative text for images, labels for form controls and so on.

[WAVE Toolbar for Firefox](#)⁸⁵

Runs WAVE directly within Firefox and ensures 100% private and secure accessibility reporting.

Can check:

- Intranet
- Password-protected Web pages.
- Dynamically generated Web pages.
- Sensitive Web pages.

[Accessibility Evaluator for Firefox](#)⁸⁶

Supports Web developers in testing their Web resources for functional accessibility features based on the iCITA HTML Best Practice.

⁸⁴ <http://wave.Webaim.org/>

⁸⁵ <https://wave.Webaim.org/toolbar/>

⁸⁶ <https://addons.mozilla.org/en-us/firefox/addon/accessibility-evaluation-toolb/>

[W3C Mark-up Validation Service](#)⁸⁷

Checks the validity of your mark-up for Web documents written in HTML, XHTML, SMIL, MathML, etc.

[Web Accessibility Checker \(AChecker\)](#)⁸⁸

You select guidelines you want AChecker to test against and it produces a report of all accessibility problems for your selected guidelines. AChecker identifies three types of problems:

- Known problems
- Likely problems
- Potential problems

[WCAG Contrast Checker for Mozilla](#)⁸⁹

Tests compliance with the contrast levels, brightness and shine in the colour combination of foreground and background of textual content based on the requirements of WCAG 1 and WCAG 2.

[WebAim Color Contrast Checker](#)⁹⁰

Checks that the colour contrast passes the WCAG contrast ratio requirements.

[Juicy Studio Accessibility Toolbar](#)⁹¹

Enables developers to examine WAI-ARIA live regions roles and properties, examine data tables, and determine if the colour contrast is sufficient.

[Fangs Screen Reader Emulator for Firefox](#)⁹²

Renders a text version of a Web page similar to how a screen reader would read it.

1.3.2 Expert based testing

There are many different ways of doing expert based testing. The most common ones are:

- Heuristic evaluation
- Cognitive walk-through
- Consistency inspection
- The personas method

1.3.3 Simulation

Use simulation to get an idea on what an application is like to use for someone with a disability.

There are several ways of simulating:

- Turn of the following things in the browser:
 - Style sheets

⁸⁷ <http://validator.w3.org/>

⁸⁸ <http://achecker.ca/checker/index.php>

⁸⁹ <https://addons.mozilla.org/en-us/firefox/addon/wcag-contrast-checker/>

⁹⁰ <http://Webaim.org/resources/contrastchecker/>

⁹¹ <https://addons.mozilla.org/en-US/firefox/addon/juicy-studio-accessibility-too/>

⁹² <https://addons.mozilla.org/en-us/firefox/addon/fangs-screen-reader-emulator/>

- Images
- Sound
- JavaScript
- Java
- Support for Flash/Silverlight
- Pop-up windows
- Navigate using only the keyboard (try out a few familiar Websites first to get used to keyboard navigation)
- Turn off the display and navigate the application using a screen reader. Do:
 - Short task-based sessions
 - Focus attention on one aspect at the time (e.g. how accessible the forms are)
 - Possible screen readers to test with are:
 - Apple screen reader (free software)
 - VoiceOver
 - Demo version of JAWS or the
 - NVDA for Windows (free software)
 - Firefox Screen-reader simulator

1.3.4 User testing

User testing with people with disabilities is a bit different from user testing with users without disabilities. To ensure the highest possible quality of the testing keep this in mind:

- User may be dependent on large equipment of AT (e.g. *Braille* keyboard, foot mouse)
- User should use their own equipment which they are comfortable with
- User should perform the testing in their home or workplace

1.4 Follow existing design principles

(Graphic designers and interaction designers)

Some examples of good design principles are:

- Know the main things people want to do on your site and make them obvious and easy
- Save the user steps where possible
- Make it easy to recover from errors
- Know what questions the user is likely to have and answer them on the FAQ page
- Tell the users what they want to know, even things like shipping cost and parking fee
- The language should be clear and concise and needless words omitted
- Have a clear visual hierarchy where the most important things come first
- Use conventions, for example the placing of the search field in the top right corner
- Break the site up in clearly defines areas such as navigation, news feeds, content, external links
- Make it obvious what is clickable
- Avoid too much noise by not having everything the same page, split it up

- Help the user find her way on the Website by using things like clear labels in the navigation and breadcrumbs and make sure the search functionality is optimised

2. Technology oriented guidelines

2.1 Follow and validate the HTML5 standard

(Programmers)

Declare doctype as such:

```
<!DOCTYPE html>
```

Validate using Validator.nu (X)HTML5 Validator⁹³ or W3C mark-up validation service.⁹⁴

Some of the most interesting new features of HTML5 are new semantic elements like:

- *Header* - Introduction to a page or section.
- *Footer* - Describes the page or a section of the page.
- *Article* - Independent item such as a blog post, article, etc.
- *Section* - Defines sections in a document, such as chapters, headers, footers etc.

It also has new form controls like:

- *Number*
- *Date*
- *Time*
- *Calendar*
- *Range*

There is a strong support for graphics with:

- *Canvas*
- *Figure*
- *Svg*

There is a strong support for multimedia with:

- *Video*
- *Audio*

Read more about HTML5 at the W3C Schools Introduction to HTML5.⁹⁵

2.2 Apply Progressive enhancement

(Programmers, graphic designers, interaction designers, content managers)

The core of Progressive enhancement is separating:

- Content (HTML)
- Presentation (CSS)

⁹³ <https://html5.validator.nu/>

⁹⁴ <http://validator.w3.org/>

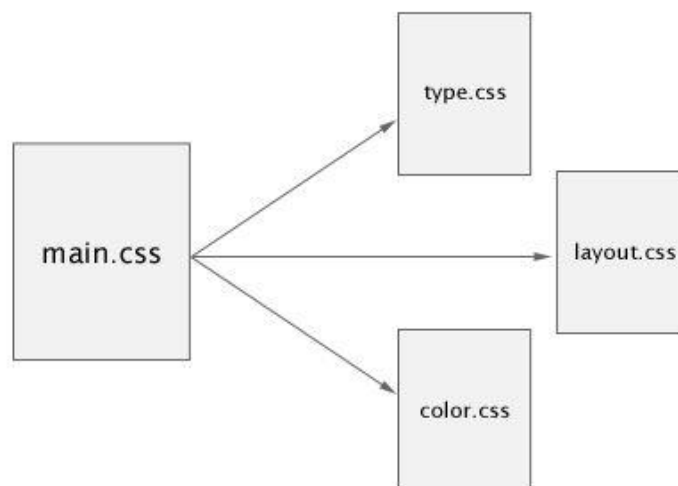
⁹⁵ http://www.w3schools.com/html/html5_intro.asp

- Behaviour (JavaScript)
- 4. Start with marking up the content. Make sure the mark-up conveys the greatest level of detail about the content it wraps around. HTML5 and WAI-ARIA goes a long way to serve this purpose. This is essential for offering a basic experience to:
 - a. Search engines
 - b. People on mobile devices
 - c. People on old browsers
- 5. Create a separate CSS file and link to it as such:
`<link rel="stylesheet" type="text/css" href="main.css">`. This is for users who have:
 - a. Basic CSS support
 - b. Lack support for JavaScript
- 6. Create a separate JavaScript file and link to it as such:
`<script src="myscripts.js"></script>`. Add JavaScript in an unobtrusive way (See next | guideline).

2.2.1 Separate your stylesheets

Start your project with a standard set of files including the following:

- type.css
- layout.css
- color.css
- screen.css
- print.css



This way you can test designs in the most standards-compliant browsers available, and then provide fixes for older or other browsers.

- Start with a main.css file included via a link element:
 - `<link rel="stylesheet" type="text/css" href="main.css" />`

- Break it into separate calls to contextual stylesheets:
 - `<link rel="stylesheet" type="text/css" href="type.css"/>`
 - `<link rel="stylesheet" type="text/css" href="layout.css"/>`
 - `<link rel="stylesheet" type="text/css" href="color.css"/>`

Since there is no media type declared, problematic browsers will read any styles in these three files.

Use a browsers' basic understanding of CSS against it by:

1. Moving all of the styles that *layout.css* contained into a new stylesheet named *screen.css*.
2. Update the content of *layout.css* is to import *screen.css*. Problematic browsers will not detect this because they do not understand the *@import* directive.
3. Declare which media that this stylesheet is for by adding a media type to the *@import* declaration: *@import 'screen.css' screen*. (The use of single quotes (') instead of a double quotes (") around the stylesheet name is a trick for getting IE5/Mac to ignore a stylesheet.)
4. The same technique can be used for example for print- or mobile-specific rules: *@import 'mobile.css' mobile*

2.3 Apply Unobtrusive JavaScript

(Programmers)

The seven rules of *Unobtrusive JavaScript*:

1. Do not make any assumptions
2. Find your hooks and relationships
3. Leave traversing to the experts
4. Understand browsers and users
5. Understand events
6. Play well with others
7. Work for the next developer

2.3.1 Do not make any assumptions

Do not expect JavaScript to be available, and do not expect the intended mark-up to be there.

Four things to keep in mind are:

- Do not expect browsers to support certain methods and have the correct properties, but test for them before accessing them
- Do not expect the correct HTML to be at your disposal, but check for it and do nothing when it is not available
- Keep functionality independent of input device
- Expect other scripts to try to interfere with the functionality and keep the scope of the scripts as secure as possible

2.3.2 Find your hooks and relationships

Before starting to plan a script:

- Look at the HTML the script will be enhancing
 - See what is the best way of letting the script interact with it
 - Consider the hooks and relationships in the HTML

HTML hooks are:

- Unique IDs (in valid HTML). Access them with the DOM method `getElementById`.
- HTML elements which can be retrieved with `getElementsByTagName` and CSS classes.

Regarding HTML relationships ask yourself the following questions:

- How can I reach this element the easiest way and with the least steps traversing the DOM?
- What elements do I have to alter to update all the child elements which should be changed?
- What attributes does one element have that I can use to link to another element?

2.3.3 Leave traversing to the experts

Use CSS to traverse the DOM. It is more efficient than JavaScript. To illustrate:

```
var n = document.getElementById('nav');
if(n){
  var as = n.getElementsByTagName('a');
  if(as.length > 0){
    for(var i=0;as[i];i++){
      as[i].style.color = '#369';
      as[i].style.textDecoration = 'none';
    }
  }
}
```

is the same as

```
#nav a{
  color:#369;
  text-decoration:none;
}
```

Dynamically assign classes to elements higher up in the DOM hierarchy or alter IDs. For example if you add a class to the body of the document using the DOM, the designer gets a chance to define both the static and dynamic version of the document.

JavaScript:

```
var dynamicClass = 'js';
var b = document.body;
b.className = b.className ? b.className + ' js' : 'js'; (shorthand if/else)
```

CSS:

```
/* static version */

#nav {
  ....
}

/* dynamic version */

body.js #nav {
  ....
}
```

2.3.4 Understand browsers and users

You need to understand:

- how browsers work
- how browsers fail
- what users expect to happen

Do not diverge too far from the way browsers work and how users expect them to work. Consider the following:

- Will the interface work independent of input device, and if not, what should be the fallback?
- Is the interface following rules of the browser or the richer interfaces it came from? Is it for example possible to navigate a multi level menu with cursors or is tabbing required?
- What necessary functionality is dependent on JavaScript?

2.3.5 Understand Events

Event handling helps with separating the JavaScript from the HTML and CSS, but also goes a bit further.

- The elements in the document are there to wait for handlers to listen to a change happening to them.
- When it happens the handlers retrieve an object (normally a parameter called e) that tells them what happened to what and what can be done with it.

Event handling does not only happen to the element you want to reach, but also to all the elements above it in the DOM hierarchy. (This does not apply to all events. *Focus* and *blur* do not do that.)

This allows you to assign one single event handler to for example a navigation list and use the event handling's methods to reach the element in question. This technique is called *event delegation* and it has several benefits:

- You only need to test if a single element exists, not each of them
- It is possible to dynamically add or remove new child elements without having to remove or add new event handlers
- It is possible to react to the same event on different elements

2.3.6 Play well with others

There will hardly ever be only one script used in a document.

- Make sure the script does not interfere with others
- Make the script hard to interfere with
 - The code should not have global function or variable names that other scripts can override
 - Instantiate every variable using the *var* keyword

```
var nav = document.getElementById('nav');
function init(){
    // do stuff
}
function show(){
    // do stuff
}
function reset(){
    // do stuff
}
```

The script above has a global variable called *nav* and functions called *init*, *show* and *reset*. The functions can access the variable and each other by name:

```
var nav = document.getElementById('nav');
function init(){
    show();
    if(nav.className === 'show'){
        reset();
    }
    // do stuff
}
function show(){
```

```

    var c = nav.className;
    // do stuff
}
function reset(){
    // do stuff
}

```

- Avoid all global code by wrapping the code in an object using the object literal. That way you turn the functions into methods and the variables into properties.
- Define the methods and variables with a name followed by a colon and separate each of them from the others with a comma.

```

var myScript = {
    nav:document.getElementById('nav'),
    init:function(){
        // do stuff
    },
    show:function(){
        // do stuff
    },
    reset:function(){
        // do stuff
    }
}

```

- These methods can be accessed from outside and inside the object by pre-pending the object name followed by a full stop.

```

var myScript = {
    nav:document.getElementById('nav'),
    init:function(){
        myScript.show();
        if(myScript.nav.className === 'show'){
            myScript.reset();
        }
        // do stuff
    },
    show:function(){
        var c = myScript.nav.className;
        // do stuff
    },
    reset:function(){
        // do stuff
    }
}

```

```
}  
}
```

The drawback with this pattern is that the name of the object needs to be repeated every time it is accessed from another method. In addition, everything put inside the object is publicly accessible. If you want to only make parts of the script accessible to other script in the document it is possible use the module pattern:

```
var myScript = function(){  
    // these are all private methods and properties  
    var nav = document.getElementById('nav');  
    function init(){  
        // do stuff  
    }  
    function show(){  
        // do stuff  
    }  
    function reset(){  
        // do stuff  
    }  
  
    // public methods and properties wrapped in a return  
    // statement and using the object literal  
    return {  
        public:function(){  
  
            },  
        foo:'bar'  
    }  
}();
```

You can access the public properties and methods that are returned the same way as in the object literal. The problem is that to access one public method from another or from a private method you need to go through the verbose long name again (the main object name can get rather long). To avoid this, define the methods as private and only return an object with synonyms.

```
var myScript = function(){  
    // these are all private methods and properties  
    var nav = document.getElementById('nav');  
    function init(){  
        // do stuff  
    }  
    function show(){
```



```

    // do stuff
    // do stuff
}
function reset(){
    // do stuff
}
var foo = 'bar';
function public(){

}
// return public pointers to the private methods and
// properties you want to reveal
return {
    public:public,
    foo:foo
}
}());

```

This allows for a consistency in coding style and gives the possibility to write shorter synonyms when they are revealed.

To avoid revealing any methods or properties, it is possible to wrap the whole code block in an anonymous function and call it immediately after it was defined.

```

(function(){
    // these are all private methods and properties
    var nav = document.getElementById('nav');
    function init(){
        // do stuff
        show(); // no need for pre-pended object name
    }
    function show(){
        // do stuff
    }
    function reset(){
        // do stuff
    }
})();

```

This is a great pattern for functionality that just needs to be executed once and has no dependency on other functions. This will make the code work well for the user and the machine it is running on as well as other developers.

2.3.7 Work for the next developer

Think about the next developer who has to take over once this code is in production. Consider the following:

- Are all the variable and function names logical and easy to understand?
- Is the code logically structured? Is it possible to "read" it from top to bottom?
- Are the dependencies obvious?
- Are areas that might be confusing commented?

The HTML and CSS of a document is much more likely to change than the JavaScript as these make up visual output. Therefore it is a good idea not to have any class and ID names or strings that will be shown to the end user buried somewhere in the code, but separate it out into a configuration object instead.

```
myscript = function(){
  var config = {
    navigationID:'nav',
    visibleClass:'show'
  };
  var nav = document.getElementById(config.navigationID);
  function init(){
    show();
    if(nav.className === config.visibleClass){
      reset();
    };
    // do stuff
  };
  function show(){
    var c = nav.className;
    // do stuff
  };
  function reset(){
    // do stuff
  };
}();
```

That way maintainers know exactly where to change these without having to alter the rest of your code.

2.4 Apply tidy coding

(Programmers)

Tidy coding relates to *Progressive enhancement* and *Unobtrusive JavaScript*.

- Define global behaviours in the JavaScript file that attach them to the ID and class hooks in the HTML file. This way they will cascade through the HTML file.

2.5 Make sure methods are independent of input device

(Programmers)

Device independent JavaScript methods are not dependent on a specific device input. Using a combination of the following methods will ensure accessibility both from mouse, keyboard and AT.

- *onMouseOver*
- *onMouseOut*
- *onClick*
- *onFocus*
- *onBlur*
- *onChange*

2.6 Apply WAI-ARIA mark-up

(Programmers)

2.6.1 Use frameworks and libraries with built-in WAI-ARIA support

Using a framework with built-in or support of WAI-ARIA will save time. Possible frameworks are

- *jQuery*
- *Fluid Infusion*
- *Dojo*
- *Bindows*

2.6.2 WAI-ARIA roles

Use WAI-ARIA role attributes to define a widgets role to assistive technology (AT). The role given by the WAI-ARIA role attribute trumps the role of the native element.

```
<input type="image"  
  src="thumb.gif"  
  alt="Effectiveness"  
  role="slider">
```

In the example above, an input element has a role attribute of slider. The role exposed to AT will therefore be slider. The WAI-ARIA specification⁹⁶ maintains a [list of roles](http://www.w3.org/TR/wai-aria/#roles)⁹⁷.

2.6.3 Document landmark roles

Document landmarks are a subset of regular roles. Use them to help screen reader users understand the role of a section on a page and help orientate themselves within the document. WAI-ARIA defines the following document landmark roles:

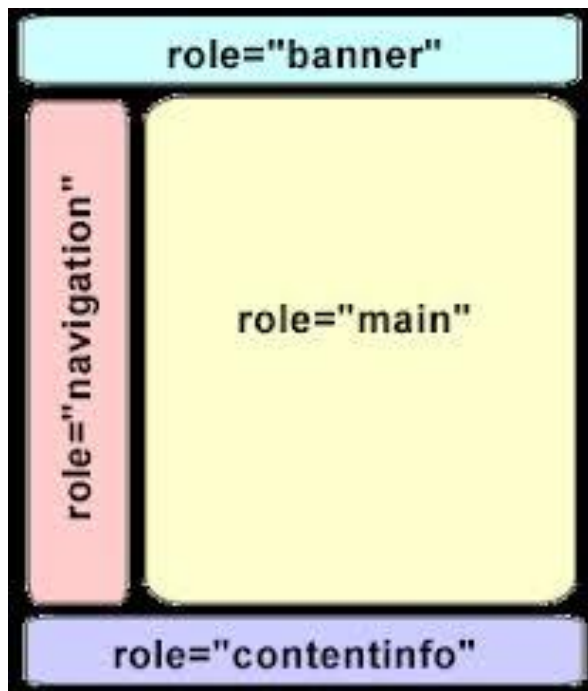
⁹⁶ <http://www.w3.org/TR/wai-aria/>

⁹⁷ <http://www.w3.org/TR/wai-aria/#roles>

- Article (content that makes sense in its own right such as a complete blog post)
- Banner (site-orientated content, such as the title of the page and the logo)
- Complementary (supporting content for the main content)
- Contentinfo (information about the content)
- Main (main content)
- Navigation (content that contains the links to navigate this or related documents)
- Search (contains a search form to search the site)

The following example specifies the landmark roles of *banner*, *navigation*, *main* and *contentinfo* to create the page structure shown below.

```
<div role="banner">  
...  
</div>  
<div role="navigation">  
...  
</div>  
<div role="main">  
...  
</div>  
<div role="contentinfo">  
...  
</div>
```



2.6.4 States and properties

Use WAI-ARIA *states* and *properties* in combination with *roles*. Changes in *states* or *properties* will notify the user of an AT that a change has occurred and help the user understand how to interact with the widget. The state identifies a unique configuration of information for an object. For example, the *aria-checked* property has three state values; *true*, *false* and *mixed*.

```
<input type="image"
  src="thumb.gif"
  alt="Effectiveness"
  role="slider"
  aria-valuemin="0"
  aria-valuemax="100"
  aria-valuenow="42"
  aria-valuetext="42 percent"
  aria-labelledby="effective">
```

In the slider example above, various aria-properties are included to describe the widget:

- *aria-valuemin*: Stores the lowest value a range may have
- *aria-valuemax*: Stores the highest value a range may have
- *aria-valuenow*: Stores the current value in a range
- *aria-valuetext*: Stores readable text to help the user understand the context. For example, "30 dollars"
- *aria-labelledby*: Stores the id attribute of a text label containing an appropriate prompt for this widget

Some properties can be updated through scripting. For example, the *aria-valuenow* and *aria-valuetext* properties of the slider widget would be updated when the thumb is moved. There is a [full list of states and properties](#)⁹⁸ to help define accessible widgets in the WAI-ARIA specification.

2.6.5 Live regions

Use *live regions* to allow elements in a document to be announced if there are changes, without the user losing focus on their current activity. The *aria-live* property has a value indicating one of three verbosity levels in a region:

- **Off**: This indicates that the region is not live
- **Polite**: This indicates that it is not necessary to respond until user completes their current activity
- **Assertive**: This value is a higher priority than normal, but does not necessarily interrupt the user immediately

⁹⁸ http://www.w3.org/TR/wai-aria/states_and_properties#state_prop_taxonomy

They are written as such:

```
<ul aria-live="off">
<ul aria-live="polite">
<ul aria-live="assertive">
```

Other important properties that can be used when defining *live regions* are:

Aria-atomic: Indicates if AT should present all or only part of the changed region to the user. It has the values *true* or *false*. If this property is set to *true*, AT should present the entire region as a whole.

In the following example, all elements within an unordered list will be announced in their entirety when the region is spoken, unless another element further down the chain overrides the *aria-atomic* property.

```
<ul aria-atomic="true"
    aria-live="polite">
```

Aria-busy: Prevents AT announcing changes before the updates are complete. It has the values *true* or *false*. If multiple parts of a live region need to be loaded before changes are announced to the user, the *aria-busy* property can be set to *true* until the final part is loaded, and then set to *false* when the updates are complete.

```
<ul aria-atomic="true"
    aria-busy="true"
    aria-live="polite">
```

Aria-relevant: Indicates what changes are considered relevant within a region. Accepts a space separated list of the following property values:

- Additions: Nodes are added to the DOM within the region.
- Removals: Nodes are removed from the DOM within the region.
- Text: Text is added or removed from the DOM.
- All: All of the above apply to this region.

In the absence of an explicit *aria-relevant* property, the default is to assume there are text changes and additions:

```
aria-relevant="text additions"
```

The following example would only announce changes if nodes are added to the DOM within the region.

```
<ul aria-relevant="additions"
    aria-atomic="true"
    aria-live="polite">
```

If there are text changes, or nodes are removed within the region, the user will not be notified.

2.6.6 Tabindex

The HTML *tabindex* attribute can be used to set tab structure on a page. Do not use *tabindex* as a replacement for a logical reading order. If mark-up has a logical structure you don't need *tabindex* for interface elements that are already in the keyboard tab order, such as buttons, links and form elements.

WAI-ARIA extends *tabindex*:

- To be used on all visible elements
- To be given focus through scripting

Originally *tabindex* only accepted a positive value between 0 and 32767. WAI-ARIA allows a negative value (typically -1) to be specified for elements that should not appear in the keyboard tab order, but can be programmatically focused.

Use this for all widgets that have a series of components that need keyboard access, such as a tree. For example a menu widget where the menu itself is in the tab order but the menu items is not. Instead the menu items could be programmed so they can be navigated using cursor keys. This way, users do not have to tab through all items in the menu, and can better navigate the document.

The following example uses a *tabindex* attribute value of 0 to put a *<div>* element into the tab order so that a keyboard user can navigate to the element.

```
<div tabindex="0">  
...  
</div>
```

The following example uses a negative *tabindex* attribute value, so that the element is not placed in the tab order, but can receive programmatic focus.

```
<div id="progaccess" tabindex="-1">  
...  
</div>
```

The following snippet of JavaScript selects the element defined above, and uses the *focus* method to place focus on the element.

```
var objDiv = document.getElementById('progaccess');  
// Focus on the element  
objDiv.focus();
```

2.7 Create or apply accessible Web components

(Programmers)

To learn about *Web components* start by reading the W3C introduction.⁹⁹

⁹⁹ <http://www.w3.org/TR/2013/WD-components-intro-20130606/>

W3C also provides some good examples of *Web components*.¹⁰⁰

When you want to start using *Web components* the official Website¹⁰¹ gives a general overview of:

- specifications used in *Web components*
- libraries you can use to create your own *Web components*
- browser support
- general discussion and community

The *Web components* wiki¹⁰² gives information and provides links to tutorials for:

- HTML imports¹⁰³
- Custom elements¹⁰⁴
- The Shadow DOM¹⁰⁵

There are many other resources for information and examples.^{106 107 108 109}

Creating accessible *Web components* will facilitate accessibility in the future through reuse of these components.

How accessible a Web Component you use or create is, is up to you. You have to consider many of the same things as when creating RIAs in general:

- Add WAI-ARIA
- Resizable text
- Give it an alternative text
- Make sure it works without audio
- Ensure keyboard functionality
- Ensure it is manageable to operate with a screen reader

You can read more about creating accessible *Web components* in the article *Accessible Web components*.¹¹⁰

¹⁰⁰ <https://dvcs.w3.org/hg/Webcomponents/raw-file/57f8cfc4a7dc/samples/index.html>

¹⁰¹ <http://Webcomponents.org>

¹⁰² <http://www.w3.org/wiki/WebComponents/>

¹⁰³ <http://www.html5rocks.com/en/tutorials/Webcomponents/imports/>

¹⁰⁴ <http://www.html5rocks.com/en/tutorials/Webcomponents/customelements/>

¹⁰⁵ <http://www.html5rocks.com/en/tutorials/Webcomponents/shadowdom/>

¹⁰⁶ <http://customelements.io/>

¹⁰⁷ <http://component.kitchen/components/pveyes/Web-resume>

¹⁰⁸ <http://www.w3.org/standards/techs/components>

¹⁰⁹ <http://w3c.github.io/Webcomponents/spec/custom/>

¹¹⁰ <https://www.polymer-project.org/articles/accessible-Web-components.html>

Appendix 2: Screen shots - digital 1st version of guidelines

[About guidelines](#) [About author](#) [Contact](#) [Guidelines](#)

Building accessible RIAs using JavaScript

1: Process oriented guidelines

- [1.1: Have accessibility experience on the team](#)
- [1.2: Implement accessibility from the beginning of a project](#)
- [1.3: Test accessibility at key stages](#)
- [1.4: Follow existing principles for good design and user experience](#)

2: Technology oriented guidelines

- [2.1: Follow and validate the HTML5 standard](#)
- [2.2: Apply progressive enhancement](#)
- [2.3: Apply unobtrusive JavaScript](#)
- [2.4: Apply tidy coding](#)
- [2.5: Make sure methods are independent of input device](#)
- [2.7: Apply WAI-ARIA mark-up](#)
- [2.8: Create or apply accessible web components](#)

1.1: Have accessibility experience on the team

(Project manager)

One or more people on the team should have experience with:

- Use of testing tools for accessibility
- Different kinds of assistive technology and how they work
- Good coding practices to ensure things like keyboard access and understandable navigation for a screen reader
- Simulation exercises
- User testing with users with disabilities

1.2: Implement accessibility from the beginning of a project

(Project manager)

Include accessibility throughout all the process phases:

- Information gathering
- Planning phase
- Designing phase
- Development phase
- Testing phase
- Maintenance

Include accessibility on an equal basis as:

- Usability
- User experience (UX)

Keep accessibility in mind when deciding what libraries and frameworks to use. Some frameworks will make the implementation of WAI-ARIA easier:

- Dojo (Open source library that uses the WAI-ARIA specification)
- jQuery (Several widgets and plug-ins have accessibility built in)
- Bindows (Section 508 compliance. Not open source)
- Fluid Infusion (Sits on top of the jQuery toolkit. Applies user centred design principles to provide an inclusive toolkit)

Apply user-centred design process and include users with impairments.


- Iterations
 - Test with representative users
 - Make changes
 - New iteration

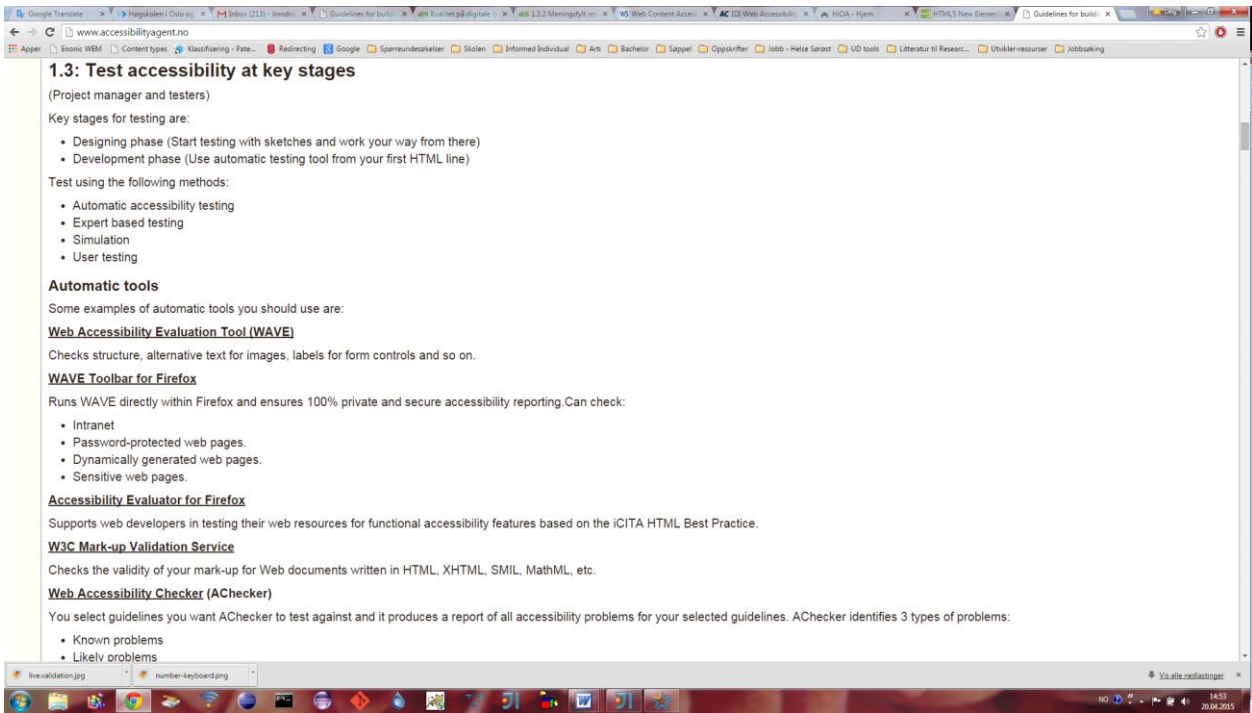
This process will help you:

- Better the interface
- Understand more of the users need

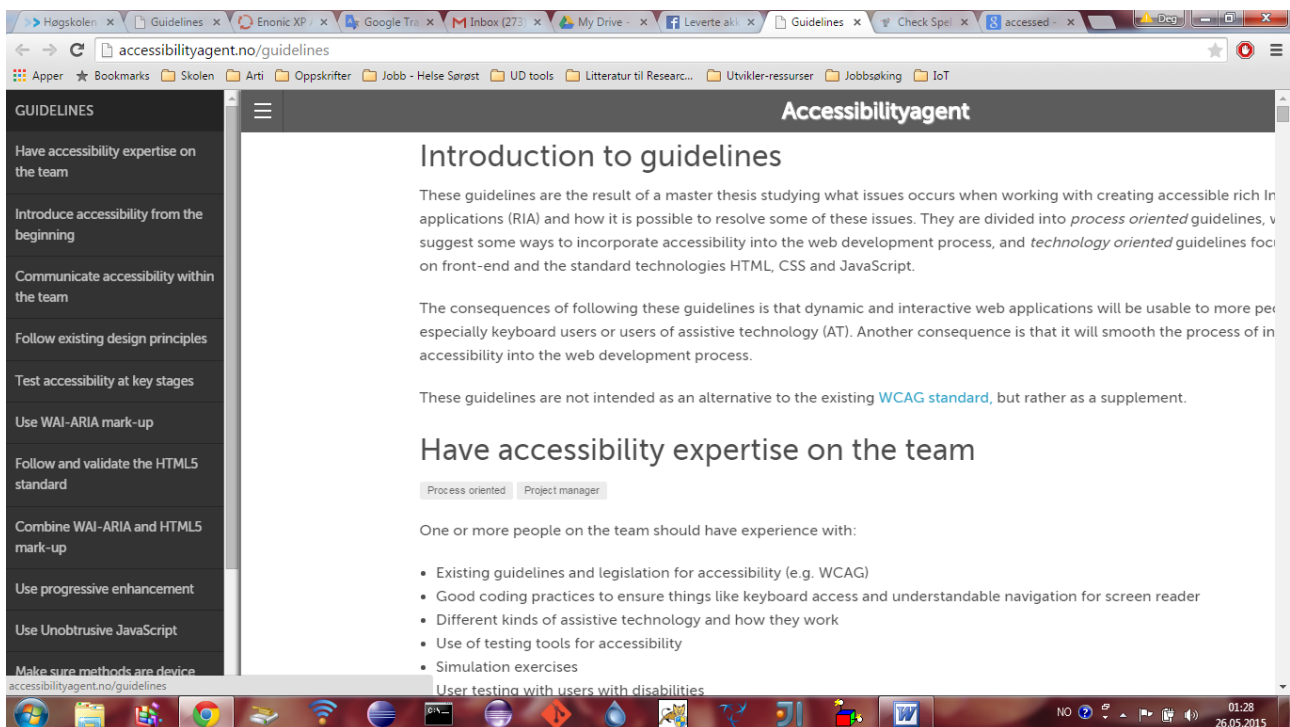
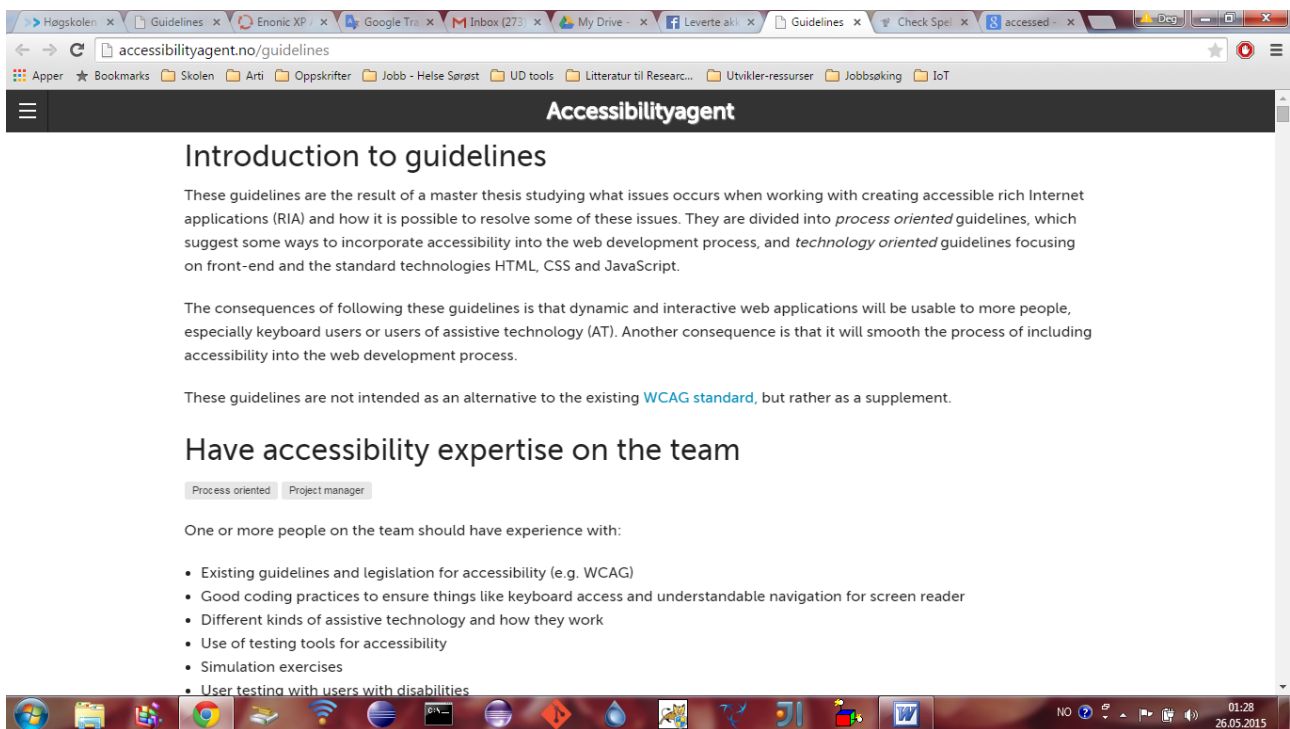
Test with both users with and without disabilities.

[Top of page](#)





Appendix 3: Screen shots - digital 2nd version of guidelines



Use CSS to traverse the DOM

Using CSS to traverse the DOM is more effective, takes fewer resources and will not create any unnecessary dependency on JavaScript.

In the example below, we perform a similar styling of images first with CSS and then with JavaScript.

```

1 
2 

```

```

1 .css-border{
2   width: 200px;
3   height: 200px;
4   border: 5px solid;
5   border-radius: 100px;
6   border-bottom-left-radius: 0;
7 }

```

```

1 (function(){
2   var images = document.querySelectorAll('img');
3   for (var i = 0; i < images.length; i++){
4     images[i].style.border = '5px solid';
5     images[i].style.borderBottomLeftRadius = 0;
6     images[i].className = 'css-border';
7   }
8 })();

```

4-5 Don't do this. Leave precise style details to the stylesheet

Use accessible modal windows instead of pop-ups

Technology oriented Frontend developer

Modal windows should be used instead of pop-up windows, and should be made accessible with WAI-ARIA. In a modal dialog screen-readers will speak the title, the text, and the currently focused button automatically. In the following example the title is called "label" and the text is called "description".

The first line declares a container that encompasses the whole dialog. HTML5 still does not have a proper dialog element that is supported everywhere. Therefore use the WAI-ARIA role "dialog".

```

1 <main id="mainPage" role="main"
2   aria-hidden="true" >
3   <div id="modal"
4     role="dialog"
5     aria-labelledby="modalTitle"
6     aria-describedby="modalDescription"
7     aria-hidden="false" >
8     <div id="modalDescription" class="screen-reader-offscreen">
9       Beginning of modal dialog window example. Escape will cancel and close the window.
10    </div>
11    <h1 id="modalTitle">Modal dialog window example</h1>
12    <p>
13      These are the onscreen instructions that are not attached explicitly to a focusable element.
14      Can screen reader users read this text with the virtual cursor?
15    </p>
16    <button id="modalCloseButton" title="Close modal dialog window">
17      
18    </button>
19  </div>
20 </main>

```