

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Lassi Läätö

Implementing Responsive Design in Industrial Dashboard Editor

Examiners: Prof. Ahmed Seffah
D.Sc. Antti Knutas

Supervisors: Prof. Ahmed Seffah
D.Sc. Antti Knutas

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
School of Business and Management
Tietotekniikan koulutusohjelma

Lassi Läätö

Responsiivisen suunnittelun toteuttaminen teollisuuden hallintapaneeli editorissa

Diplomityö

2017

96 sivua, 19 kuvaa, 8 taulukkoa, 7 liitettä

Työn tarkastajat: Professori Ahmed Seffah
 TkT Antti Knutas

Hakusanat: hallintapaneeli, editori, responsiivinen suunnittelu

Keywords: dashboard, editor, responsive design

Tämä lopputyö on tehty ABB Ability™:n tuotteen View:n responsiivisuuden tehostamiseksi sekä editorissa että hallintapaneeleissa. View on hallintapaneelien luontiin tehty editori ja UI SDK. Responsiivinen suunnittelu tarkoittaa internetsivun kykyä sopeutua eri näytön kokoihin ja muotoihin samalla pitäen optimaalisen käyttäjäkokemuksen. Tutkimuskysymyksenä on, miten responsiivinen suunnittelu voidaan implementoida teolliseen hallintapanelieditoriin. Systemaattisella kartoitustutkimuksella selvitettiin, että on vain harvoja artikkeleita, jotka on kirjoitettu vastaavista hallintapaneeleista ja niiden käyttöliittymistä. Lopputulos on, että responsiivista suunnittelua on mahdotonta käyttää sellaisenaan järjestelmissä kuten View. Muita metodeja käytettiin, jotta vastaavaan lopputulokseen päästiin. Työstä tehtiin käytettävyystudkimus, jossa saatiin selville, että järjestelmän käytettävyys joko nousi tai ei laskenut muutosten ansiosta.

ABSTRACT

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Lassi Lääti

Implementing Responsive Web Design in Industrial Dashboard Editor

Master's Thesis

2017

96 pages, 19 figures, 8 tables, 7 appendices

Examiners: Prof. Ahmed Seffah
 D.Sc. Antti Knutas

Keywords: dashboard, editor, responsive design

This master's thesis was conducted to enhance ABB Ability™'s product View, a dashboard editor and UI SDK, to implement a responsive design in both its editor and dashboards. Responsive design means creating a web page layout so that it can respond to viewport shape and size change to keep optimal user experience. The main research question is how to implement responsive design in an industrial dashboard editor. For this, systematic mapping study is presented, showing that only a few articles have been written in visualisation dashboard systems in relation to their user interface and layouts. Outcomes of the research are that responsive design is not implementable to more complex systems like View. Other methods are created to mimic this core functionality to create similar results. An evaluation was done in form of usability study, and outcomes are that improvements that were done during the work of the thesis had either increased or held the level of usability of the product.

ACKNOWLEDGMENTS

This thesis was written at ABB Oy in Pitäjänmäki, Helsinki during semester 2016 – 2017. I would like to thank everyone at ABB for the opportunity to write this thesis, and other team members, especially those who gave guidance or participated in usability evaluation. I would also like to thank my supervisor Antti for his time and effort in making this book as good as it is. Finally, my deepest thanks for Viola for all the support during this long and dark winter.

Omnia dicta fortiora si dicta Latina

Lassi Lääti

May 2017

Espoo

~_(\ツ)_/~

TABLE OF CONTENTS

1	INTRODUCTION	5
1.1	Goals and Objectives	5
1.2	Research Question	6
1.3	Research Process.....	7
1.4	Structure of Thesis	8
2	SYSTEMATIC MAPPING STUDY	10
2.1	Scope Definition	10
2.2	Primary Search.....	12
2.3	Paper Screening	13
2.4	Classification and Abstraction	14
2.5	Data Extraction and Mapping of Results	15
2.6	Analysis and Discussion	18
2.7	Discussing the mapping study results	23
3	RESPONSIVE WEB DESIGN.....	25
3.1	About Responsive Design.....	27
3.2	Implementing Responsive Design	31
3.3	Justifying Responsive Design.....	34
4	IMPLEMENTATION RESPONSIVE WEB DESIGN	38
4.1	Editor.....	39
4.2	Dashboard	45
4.3	Widgets	56
5	EVALUATION OF FEATURES	63
5.1	Evaluation Process	63
5.2	Evaluation Results	66
5.3	Future Developments and Limitations.....	68

6 CONCLUSION.....	71
REFERENCES	74
APPENDICES	

LIST OF ABBREVIATIONS AND ABBREVIATIONS

ACM – Association for Computing Machinery

AJAX – Asynchronous JavaScript And XML

CEPBoard – Collaborative Electronic Performance Board

COCPIT – Collaborative Online Care Pathway Investigation Tool

CPS – Cyber-Physical System

CSS – Cascading Style Sheets

CSV – Comma Separated Values

CV – Critical Value

DOM – Document Object Model

DSR – Design Science Research

HTML – Hypertext Markup Language

IE – Internet Explorer

IEEE – Institution of Electrical and Electronics Engineers

IMG – Image

IoT – Internet of Things

IT – Information technology

JVM – Java Virtual Machine

LUT – Lappeenranta University of Technology

MBUI – Model-based user interfaces

MVC – Model-View-Controller

MWCC – Media Watch on Climate Change

PPI – Pixels per Inch

RWD – Responsive Web design

SASS – Syntactically Awesome Stylesheets

SCSS – Sassy CSS

SDK – Software development kit

SVG – Scalable Vector Graphics

UI – User Interface

UID – Unique Identifier

UML – Unified Modelling Language

URL – Uniform Resource Locator

WSSBI – Web Support systems for Business Intelligence

XML – Extensible Markup Language

1 INTRODUCTION

In the past years the performance of JavaScript -based applications have increased, as more web-based applications enter the market (McAllister 2015). Naturally, the market goes where the users are, and so industrial applications should move to platforms that are used by the target audience. Now that the web as a platform can answer the performance needs of most kind of applications, it is good to consider moving existing applications to the web.

There already are web-based office tools (e.g. Google G Suite and Microsoft Office 365) and drawing tools like Lucidchart (lucidchart.com) that can replace native office tools for business users. These are high-performance applications running purely on web browsers and show what the web as a platform can offer.

When designing for the web, users expect these high-performance applications also work like any other web application, and have all the benefits of the web as a platform, like being able to work on a wide range of different devices. This can be a blessing, but also a huge challenge for developers. The web provides customers with a way to interact with a product on almost any modern web device, but it does not ensure that developers will create a product that can be used on all of those devices.

ABB has created an industrial dashboard editor and User Interface (UI) Software development kit (SDK) called “View”, for the creation of visualisation of process automation production and maintenance dashboards. The software is at the stage of being able to completely replace old operating system native solutions. In this thesis, the implementation of tools for responsive dashboard creation for this application is researched, so that one dashboard can be effectively be used in all supported devices.

1.1 Goals and Objectives

The main goal of this thesis is to research responsive design and responsiveness of an industrial dashboard editor (case introduced in **Chapter 4**), and improve it so that it can work responsively and produce responsive content. The aim is to do work and write a report

on the findings on how the case system was transferred to work as platform independently as possible. Objectives are:

O1: “Editor and dashboards it produces will work on both desktop and mobile”

O2: “Dashboards will mimic responsive design paradigms”

O3: “Thesis works as a documentation of how this is achieved”

Another goal is to add value to the case system and implement tools with which users can create dashboards that can adjust themselves depending on what platform they are used in. This report should work as good documentation of how this is achieved and how the editor itself was changed during working on this thesis.

1.2 Research Question

Responsive web design is researched in the aspect of web-based industrial software that is built at ABB. This application has been created for data visualisation and dashboard creation. The scope focuses on how editor and dashboard it produces can and should implement responsive web design. Hence, the research question is:

RQ: “How to achieve responsiveness and responsive design in industrial dashboard editor?”

The research process into the responsive design is detailed in the following chapters, in both broader sense and in the aspect of this study. The goal is to create an understanding of responsive design as a concept, and how it can be implemented in any given system. Specific techniques are only discussed as examples of how the wanted effects can be or should be implemented, or why some widely-used techniques do not work.

As the case system in hand is big, all the modules and features of it are not introduced. The actual implementation of features that are built into this system to answer the need created by the task in hand is covered in the report written in **Chapter 4**. Implemented features are covered, but no code is presented, as it belongs to ABB. Some features that answer the research question have already been implemented, and they are introduced as part of the report, even though they were not implemented explicitly for this thesis.

1.3 Research Process

In this research, the aim is to answer questions relevant to a real-life problem with the creation of an innovative artifact. This process can be labelled as Design Science Research (DSR) (A. Hevner and Chatterjee 2010). Artifacts are artificial products of the software development lifecycle (Simon 1996).

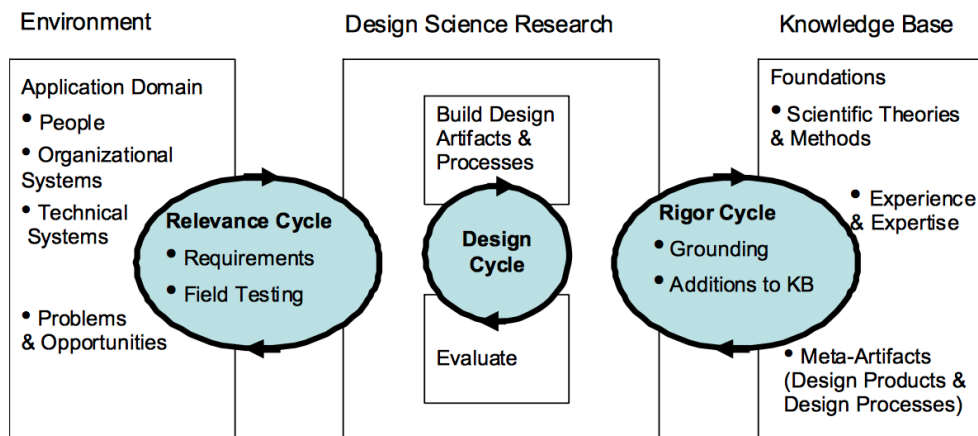


Figure 1 Three cycles of Design Science Research (A. R. Hevner 2007)

Design Science Research can be divided into three cycles (A. R. Hevner 2007): Relevance cycle, Design cycle and Rigor cycle, which can be seen in **Figure 1**. In the Relevance Cycle, a researcher works in The Application Domain to tie together real-life problems with the design of the end-product. In the Design Cycle, a researcher builds and evaluates the artifact. In the Rigor Cycle, a researcher draws knowledge from other computer science research and experiences, which provides the foundation for scientifically significant research.

This model has been taken further by Peffers et al. (2007), by defining six activities of DSR based on Hevner’s (2007) three cycles. All the activities overlap with at least one of the cycles. DSR is not a linear process, and different phases of both models are done when needed. That is why in Hevner’s (2007) research the stages are cycles and in Peffers et al.’s (2007) research they are activities.

Following are the phases of this research, in the scope of both Peffer et al.’s (2007) activities and Hevner’s (2007) cycles. **Problem identification and motivation** (Activity 1, Relevance

and Rigor Cycles) were given by the ABB Ability™ Team. Their motivation was to find out how to enable their product View, an UI (User Interface) SDK (Software development kit) and dashboard editor, to create responsive dashboards. Problem identification can be compressed to one sentence, a research question, which is presented in **Chapter 1.2. Objectives for a solution** (Activity 2, Relevance Cycle) were defined based on the research question at the start of the research project. The objectives are listed in **Chapter 1.1**, and for the implementation part of the project, the required functionalities are defined in **Appendices 1-3**.

The **Design and development** (Activity 3, Design Cycle) of the features of the system were done mostly in parallel with other activities. In this activity, the actual artifacts are designed and created. **Demonstration** (Activity 4, Relevance Cycle) of the work is done by creating example dashboards that show new features, demonstrating how design can solve the given problem. **Chapter 4** has both documentation and demonstration of how the developed features work.

Evaluation (Activity 5, Design Cycle) of the work was done by conducting a usability study among the colleagues. The outcomes and the report of this evaluation can be found in **Chapter 5**. The goal was to observe and prove scientifically that artifacts solve the given problem. The final step is **Communication** (Activity 6, Rigor Cycle), in which the outcomes of the study are communicated to the public. This is done by publishing the thesis and showing the conducted work. The final audience of the thesis are those who read the report from university's public database, other developers of the product, and if new features are added to the end-product, the customers.

1.4 Structure of Thesis

This thesis has been divided into a theoretical and practical part. Second and Third chapters will cover the theoretical part of this thesis. **Chapter 2** presents a systematic mapping study, which is a report of systematically reviewing work related to research question, to find out the interest of field and classify already done work on the subject. This chapter works as related work and background study to the subject. **Chapter 3** is the second part of a theoretical section of this thesis. It is a literature review on responsive web design, going

through different sources and trying to answer what responsive design is, how it can be achieved and why it is important.

Chapter 4 and **Chapter 5** cover the practical part of the thesis. The fourth chapter is a report of how responsive design is achieved in the case system, and it has some discussion on the topic. This chapter covers the implementation part of the thesis and presents the job that has been done for the this and inside the product team between versions 1.2 and 1.4. The fifth chapter covers the evaluation of the thesis work. In that chapter, the evaluation process is introduced and outcomes are discussed based on it. Based on that, future developments that are and should be implemented are presented.

Finally, a conclusion of the thesis is covered in the **Chapter 6**. In that chapter, the research question and objectives are answered again and the thesis summarised.

2 SYSTEMATIC MAPPING STUDY

In this chapter, a systematic mapping study on responsive design is presented. Systematic primary research mapping studies are forms of research, where researcher goes through a systematic process of mapping the research done in specific areas. This is done in following six steps, which are summarised in **Table 1**.

Table 1 Steps of Systematic mapping study (Petersen et al. 2008)

Step	Description	Outcome
Research Scope	Scope and terms of the research are defined.	Search scope and terms.
Primary Search	Test searches are run and outcomes of primary search introduced.	List of studies done based on search terms.
Paper Screening	Papers are screened and the inclusion and exclusion criteria defined.	Reviewed list of included articles.
Classification Scheme	Keywords and abstraction of included articles are described.	Classification scheme definitions and classified articles.
Data Extraction	Included and classified articles are read and data extracted.	Systematic Map.
Analysis	Analysis of the Outcome is presented.	Analysis of Results.

These six steps are covered more in depth in the following sections, which also covers the research done in each step. With this, we can map the interest of the field by studying this specific topic, which in our case is how IT (Information technology) industry has been doing responsive visualisation in different forms and using dashboard systems in solving complex problems.

2.1 Scope Definition

Different research in an area of industrial dashboard editor software and responsive design in this area is covered in this thesis. In selected articles for systematic mapping study, we are

more interested in different layout choices and actual dashboard implementations done in web platforms than data handling between different parts of systems. If there were dashboard system handling issues of other aspects of this research, the article can be included even if it does not include a system built on the web.

The first step of systematic mapping study is scope definition, which will result in usable keywords for database's search engines. Before the final primary search, usage of databases' search engines was clarified. Test searches were performed with each database's advanced search features to ensure that they were used properly.

To find proper keywords test searches were performed and outcomes of them were analysed by using Nails Project. Nails Project is created in Lappeenranta University of Technology to find connections between articles (Knutas et al. 2015).

By using only "responsive web design" in test search, there were a lot of low-quality search results. For example, from the Web of Science 94 articles were found with these keywords. Five most cited of these were either not relevant or in one case not available for download. With "responsive design dashboard" only five articles were found, out of which only one were available for the researcher.

After a couple of test searches, following keywords were selected with proper Boolean operators: ("dashboard" AND "editor") OR ("dashboard" AND "responsive design"). With these keywords, it is expected to find research articles that focus on dashboard systems.

It was also decided to limit the publication year to start from 1990, as it is not possible to find anything related to this study from articles written earlier than that. The first website that had responsive layout was Audi.com in the year 2002 (Kalbach 2012). As Worldwide Web was invented in 1990 by Tim Berners-Lee, it is safe to assume that there cannot be any proper research done on the topic earlier than that.

For this research, it was decided to perform mapping study with four research databases: IEEE (Institution of Electrical and Electronics Engineers) Xplore, Springer Link, ACM (Association for Computing Machinery) Digital Library and Web of Science. These four

archives all provide computer science related journals and articles, advanced search support and free CSV-file (comma-separated values) exporting. **Table 2** lists the search results the from primary search and compares them to the included articles for final analysis.

Table 2 Searches done in databases and their parameters, filters, and results.

Database	Search parameters and filtering	final included/ included in analysis/ first round included
IEEE Xplore	Journals and conference publications from 1990 to present	5 / 5 / 6
ACM Digital Library	Journals and conference publications from 1990 to present	1 / 6 / 13
Springer Link	Journals and conference publications from 1990 to present, with discipline of Computer Science	9 / 19 / 75
Web of Science	Journals and conference publications from 1990 to present	0 / 0 / 2
Total		15 / 30 / 101

2.2 Primary Search

All search results were extracted from the website in a usable format and imported into a spreadsheet. Search results were handled in the following order: IEEE Xplore, ACM Digital Library, Springer Link and Web of Science. Results of primary searches from different databases were compared, and duplicated results were removed to avoid duplicated articles in the final analysis.

After the exclusion of duplicates, 101 articles were left to in next phase, paper screening. A number of papers in different steps can be compared in **Table 2**, in which we can see that Springer Link provided the most articles. It was also apparent in test searches, in which with some keywords Springer Link gave over 500 results.

2.3 Paper Screening

After the primary search was conducted, all included articles were screened, and the following criteria of inclusion and exclusion were applied:

Inclusion criteria for articles were:

- The article discussed dashboard system of some kind.
- The article had some kind of layout discussion on the subject.
- The article discusses its topic in mainly computer science perspective and/or proposes conclusions that have some computer scientific value.

Exclusion criteria for articles were:

- The full article was not readable with current library access for the author.
- The article was not written in English.

From these results keywords, title and abstract were read and the total of 30 articles were selected for the in-depth review based on exclusion criteria explained before. The importance of responsive design was separated as few of the found research articles discussed responsive design in detail.

As seen from **Table 2**, IEEE Xplore and ACM Digital Library provided only a few search results that were close to the research topic. Springer Link in the other hand provided a lot of different search results, differing from research topics, for example, automotive industry (car dashboard production) to case studies related only to business intelligence. Hypothesis at this point was that there is not so much research done, which includes dashboard editors and/or responsive design, or dashboard related products with an emphasis on layout and user interface. Web of Science was added at later stages of study in an effort to add more value to the research, but as seen from **Table 2** none of the articles found there were included in the final analysis.

2.4 Classification and Abstraction

In this section, articles were classified based on different criteria. The point of this is to provide a clear picture of what kind of studies have been done in the area of interest. In theory, classification and data extraction should be conducted in different steps, but in this research, it was found to be hard not combine them. Classification made from abstracts and keywords alone was too hard, so for most of the articles, it was necessary to also read the part where the exact research method was explained and also parts of the conclusions.

Table 3 Classification scheme by research approach (Wieringa et al. 2005)

Evaluation Research	Research has been done to evaluate proposed method/model, and the article describes this evaluation process.
Proposal of Solution	Researchers propose a possible solution for problem and proof for it.
Validation of Research	Solution proposal has been made and researchers describe a validation method made in a controlled environment (for example a laboratory).
Philosophical Paper	Research describe a philosophical opinion.
Opinion Paper	Research describes a researcher's personal opinion.
Personal Experience Paper	The article describes a personal experience of the author.

Multiple classification schemes were selected to find the best for the final systematic map. The first classification scheme was based on a research approach created by Wieringa et al. (2005) which was also used in articles like "Systematic Mapping studies in Software Engineering" (Petersen et al. 2008). Categories of this classification scheme are summarised in **Table 3**.

The second classification scheme is a type of contribution that research provides, also used in Petersen et al.'s (2008) article. In this scheme, four different classes are considered: Metric, Model, Framework, and Tool.

The third scheme is an area in which the contribution of the research is applied to. The aim of this research is to find out what kind of articles has been written on the problem in hand, but mostly Computer Science is applied in some secondary field of science, for example, Medical Science. Studies that were not applied to any additional concept outside of Computer Science were classified as such. The following classes were found: Computer Science, Medical Science, Business Intelligence, Education, and Society. With these three classification schemes, articles were checked and first exclusions of articles were done based on their abstracts.

Finally, each article was evaluated based on how closely it discussed the topics related to this research. This was made by looking more closely into the research and trying to answer the following questions:

- Is there discussion about responsive design?
- Is introduced or discussed system web based (preferably HTML)?
- Is there discussion about UI or Layout?
- Does system have an editor of some kind?

With these classification schemes, we know why the article was written, what was researched and how it relates to this study. Multiple rounds of screening were done, and systematic map and analysis were written based on included articles. In the last round of the screening, all rest of education related articles were excluded out of the final analysis, as they did not have any.

2.5 Data Extraction and Mapping of Results

Efforts made while extracting data out of the articles and preparing them for analysis are described in this section. **Figure 2** shows the distribution of areas of research and they are divided by publication year in the beginning of screening phase. It can be seen that in the area of Business Intelligence research, articles' publication distribution is relatively stable. In Computer Science, which represents a more theoretical area of the field in this research, articles have been written since the year 2011. Between 2015 and 2016 publications concerning category "Society" have emerged, meaning that such systems have been discussed in applications such as digital cities. From **Figure 2** it can be also seen that most

of the research done belong to Business Intelligence and Computer Science, and the outcomes are mostly Tools.

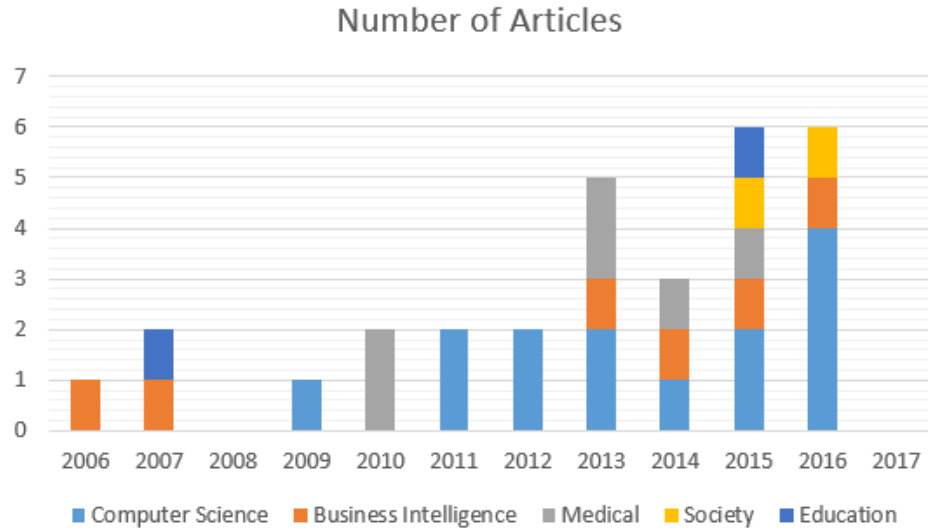


Figure 2 Number of articles per publication year, in the beginning of the screening phase

Figure 3 and **Figure 4** are object oriented design maps, which show the frequency of publications in two dimensions that were included in the final analysis. In y-axis, we have a classification scheme of the research approach. The **Figure 3**'s x-axis shows the outcome of research and **Figure 4**'s x-axis shows the field of application.

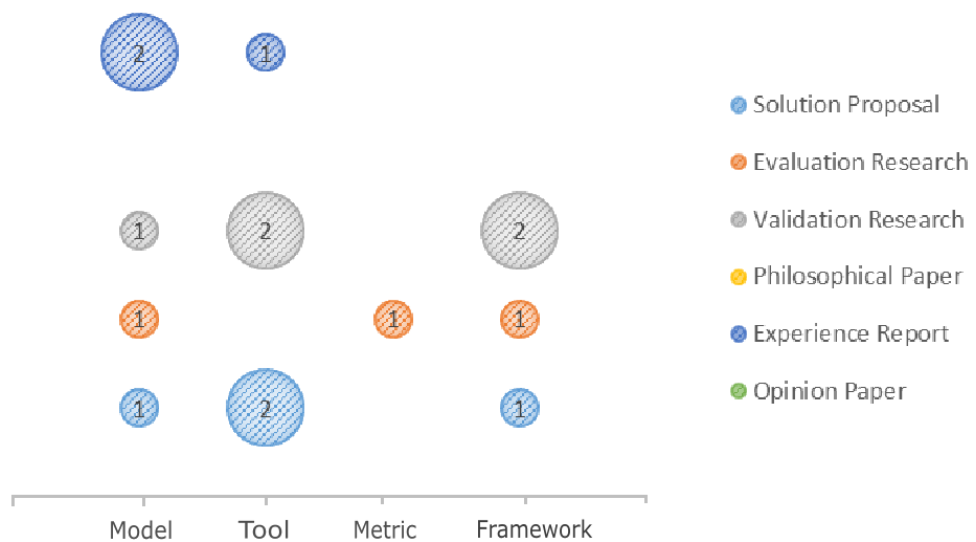


Figure 3 Object oriented map on outcomes of the research

Previous figures give a clear picture of how many of the found articles discussed the wanted topics. The first clear notice is that the only articles that even mentioned responsive design belong to Computer Science class, and there was only three of those. There were four articles included and classified that had a discussion about dashboard editors. The bar of “Having layout discussion” was really low, as many of the articles talking about UI of their systems had a very vague introduction on how it works or just discussed screen captures presented in reports.

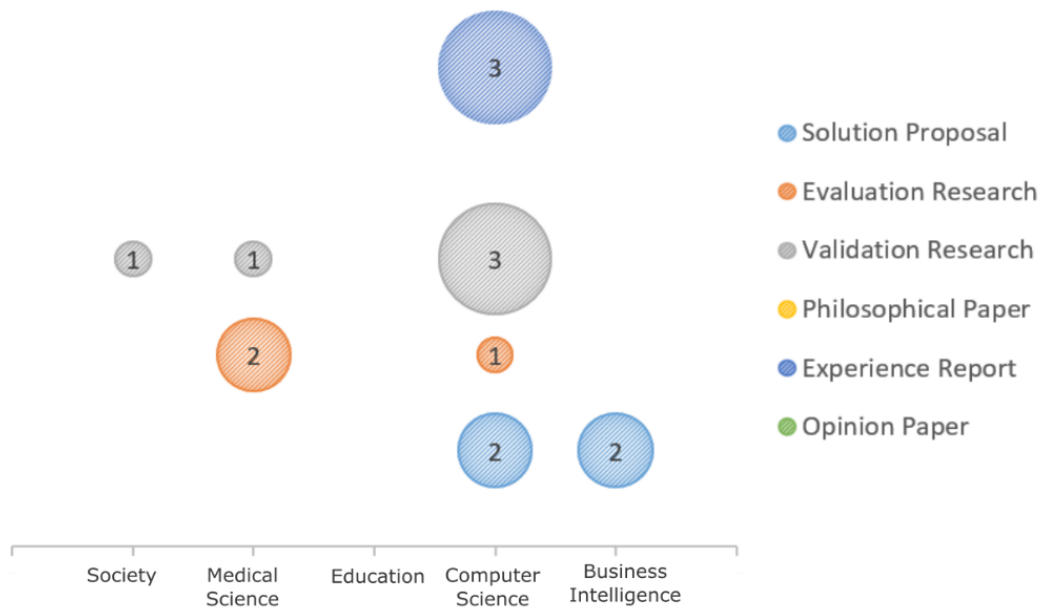


Figure 4 Object oriented map on field of science

Table 4 has a systematic map produced by this mapping study. In this map, we can see citations of different articles related to our research mapped to different criteria. On columns, citations are divided based on their applied field of study and on row, articles are separated based on their relation to this study. The Systematic map and analysis have been divided into three parts:

1. Systems that introduce dashboard editors.
2. Articles that discuss responsive design in context of dashboards
3. Articles with interesting web-based dashboard systems but do not belong to either one of two previous groups but have some discussion on layouts

Table 4 a Systematic Map

15 articles in total	Business Intelligence	Computer Science	Medical Science	Society
Editor		(McKeon 2009) (Gesing et al. 2014) (Pavlov, Knoch, and Deru 2015) (Scharl et al. 2013)		(Balatsoukas et al. 2015)
Responsive Design		(Dayarathna, Withana, and Sugiura 2011) (Gesing et al. 2014) (Desruelle et al. 2016) (Li et al. 2016)		
Web-based Dashboard solution	(Palpanas et al. 2007) (Khan et al. 2014)	(Rimal et al. 2011) (Marin et al. 2015) (Desruelle et al. 2016)	(Mathe et al. 2015) (Dolan, Veazie, and Russ 2013)	(Karagiannidis et al. 2016)

2.6 Analysis and Discussion

Based on the systematic map constructed in the previous phase, we can now conduct an analysis of the researched topic. The areas of interest are divided to clarify research questions' significance in the results.

Editors

Only five articles discussed usage of editors in their dashboard systems. Two of them were experience articles and one was a solution proposal, all of which are Computer Science related. One article was usability evaluation on the medical system and the last article was classified as a validation research, also in the area of Computer Science.

In "Workflows in a Dashboard: A New Generation of Usability" (Gesing et al. 2014) researchers make a solution proposal on a dashboard system for managing different workflows. They have web based workflow dashboards and dashboard editor concepts. Research is ongoing and there is a concept that verifies most of the proposals, but there is no real implementation for the whole system. The proposed system has separated monitoring

and editing sections. In the article Gesing et al. (2014) propose a responsive cross-platform HTML5 based system, but they give no deeper understanding on the topic.

Another system with an editor is introduced in an article “Harnessing the Information Ecosystem with Wiki-based Visualization Dashboards” (McKeon 2009). In it, researcher discusses the implementation of the Dashiki, wiki page for open dashboards. The platform was developed and launched successfully, and the article was written on the results. McKeon (2009) chose to conduct interviews with professionals to improve the platform and take further action in the future. Techniques in this article are relatively old, but they work well as a proof of concept for this research, as the project is fully working and includes collaboration features and data visualisation abilities with techniques of that time.

Research on Media Watch on Climate Change (MWCC) covers work done on a system that can collect data from multiple sources and display it in dashboards that visualise data for example in charts (Scharl et al. 2013). This article is a nice proof of concept and covers wiki page type of editing of documents for users. In this system, users can edit contents of the system instead of layout, but we still include it as it is a system with an editor as it covers collaborative data editing to change visualisations on dashboards.

Another similar system was COCPIT (Collaborative Online Care Pathway Investigation Tool), a system covered in research done by Balatsoukas et al. (2015). In their article, they do usability evaluation on COCPIT, which is Web-Based Integrated Care Pathway Investigation tool. Integrated care pathways or ICPs are process diagrams of steps of treatment. In COCPIT doctors can create these pathways and visualise data from their patients on charts and diagrams to effectively follow their treatment. The article doesn't go into detail on how this system works, instead, introduces outcomes of usability tests and provides guidelines. Based on this research developers should pay attention to modular visualisation and direct manipulation of information. Designers should also pay attention to “the visibility of the system status” and “the match between the system and the real world” (Balatsoukas et al. 2015).

The most interesting article concerning this study is about CEPBoard (Collaborative Electronic Performance Board) by Pavlov, Knoch, and Deru (2015). This article provides a

solution proposal on how to create a dashboard system with widgets, featuring editor with touch/gesture support. Unfortunately, the article written about this research is only two pages long and gives a very brief introduction (research proposal) of the topic, so it does not provide any actual solutions on this matter.

Out of these five articles, it is hard to get any proper solution about the topic of dashboard editors, as all five included articles have flaws on actually giving any proper answers. Article written on CEPBoard (Pavlov, Knoch, and Deru 2015) has described the most promising system but unfortunately, any actual studies are not yet done on the topic.

Responsive Design

In the matter of responsive design, there were three articles that had strict opinions. All of them were classified as Computer Science related articles, covering more technical ways of creating solutions to problems.

In the article by Desruelle et al. (2016) that covers research done on webinos, a framework for ubiquitous systems, researchers admit the need for adaptive user interfaces and opt in for adaptive design principles which are alternative to responsive design. In this framework, researchers have designed a specific interface adaptation engine and evaluator for creating user interfaces for different platforms. While researchers ignore the idea of responsive web design, this article is included as an alternative solution to the problem.

In the article by Dayarathna, Withana, and Sugiura (2011) researchers identify three classes of software clients: “Rich Client”, “Smart Client” and “Thin Client”. In this classification, Rich Clients are heavy clients that aim to be perfect at what they do and in a relatively well-fashioned way. On the other side of the spectrum, Thin Clients are made to do their tasks without any special tricks. This happens also by cutting corners while developing basic functionalities. Smart Clients aim to be in the middle of these two solutions, while providing a functionality and user experience of a rich client, but doing so in a smart and light way. “Responsive and Flexible” is listed in the features in between of Smart and Rich Clients, overlapping both classes. Though not being vital to functionality in most cases, it is one of the most important additional user-interface feature making it a smart adaptation for any application.

Gesing et al. (2014) also list responsive design as one of the key design constraints. This article was also listed in the discussion about editors. Researchers admit that technical details are not in the scope of their research, but they suggest interfaces be done so that they could produce working native mobile applications by using tools like PhoneGap.

As seen here responsive design is not talked in depth in found articles. Same can be observed in research done on a digital city (Li et al. 2016), where the layout of the system is introduced and developers are said to be following responsive web design guidelines, but no concrete proof or advice are represented.

Web Solutions and Layout

In this section rest of the found articles that have some sort of value for this study is listed. There are multiple articles providing proof that data visualisation systems add value to their business, but they mostly discuss the overall architecture of the system instead of giving any specific advice. Rimal et al. (2011) say that user interfaces must be simple, uncluttered and designed according to users' expectations. In the article Rimal et al. (2011) claim that AJAX-based systems (Asynchronous JavaScript And XML) change users' experience so much that it can add value to service. The article is written in 2011, so it is no surprise that researchers only speculate that mobile devices might have a bigger impact in the future.

A research written by Marin et al. (2015) suggests using of model-based user interfaces (MBUI) paradigm to create native user interfaces for ubiquitous systems. They introduce LIZARD, which is a framework for creating MBUI based systems. With this framework, developers can define models that are transformed into full applications. LIZARD generated systems are said to have advantages over RWD applications, as it generates native applications.

Some researchers find the web as a platform to be more effective than ubiquitous application middlewares (for example JVM (Java Virtual Machine) or .NET). In research that covers webinos Desruelle et al. (2016) also point out that by using extensions like PhoneGap a developer can write web applications that use platform independent functionalities like a camera or device storage. Webinos was found to be able to produce Android applications that were over 95.8% pure platform-independent JavaScript code. This platform produced

the largest amount of platform dependant code. Webinos is prototype platform, which was described in the article.

There were multiple articles covering Business Intelligence systems that were included based on their abstract and keywords, but with not much to give for this research when read in-depth. In research that covers Model-driven application development, it is shown how to design in XML (Extensible Markup Language) and produce effective dashboards for Business Intelligence (Palpanas et al. 2007). Although a good idea, the article was written in 2007 and is a bit out of date in a matter of visual capabilities that could be done on this subject. Similar Business Intelligence solutions are identified in the article by (Khan et al. 2014), and they call these systems “Web Support systems for Business Intelligence” or WSSBI in short.

In Medical Science research there was also multiple articles describing the implementation and the importance of such systems of decision supporting on different steps of patients treatment (Mahendrawathi, Pranantha, and Utomo 2010; Moghimi et al. 2013; Huang et al. 2014). These systems are found to improve doctors decision making and in the case of physical therapy patients’ motivation in participating in necessary exercises. Dolan, Veazie, and Russ (2013) made a difference by providing Evaluation research on their dashboard questionnaire system for medical use. In their research, they made patients use their web-based decision dashboard and evaluated usage. As a conclusion, they find that such systems can be easy and provide patients with enough information to make feasible treatment decisions. Mahendrawathi, Pranantha, and Utomo (2010) present screenshots of their system and have a firm introduction to implementation, but unfortunately, as research was written in 2010, the whole service is implemented in PHP and is not designed to work on anything else than desktop.

There was also some research done in educational systems, where it is found that dashboard web systems can be used to boost students learning by making communication more effective and using data collection to improve learning experience (Guo 2015; Gordon and Sensen 2007). They were excluded from this research, as they do not contribute at all to our layout discussion.

In articles classified in the area of society, there was one about CPS-enabled (Cyber-Physical System) sewer mining systems, which included some discussion on their dashboards (Karagiannidis et al. 2016). This system is built on Raspberry Pi based server and the client is using Model-View-Controller (MVC) architecture. Dashboard produces real-time charts and graphs to allow supervisors of plants to monitor the water treatment. The system is said to have a database that collects historical data for data visualisation and has event/alert system for an operator.

2.7 Discussing the mapping study results

The outcome of the systematic mapping study is to find out previous interest in the field. As a result, some articles were found, but no research that answers the specific topic completely. It was found that many dashboard systems have been studied in an academic sense, but not many of those included any discussion about a layout. There was almost no discussion about responsive design in the found articles. There was also a lack of research published about dashboard editors.

Some specific projects have been done on the research of educational and social systems, but they didn't include any interesting discussion of layout either. In these cases, researchers were naturally more interested in how such systems can be implemented to assist learning and teaching. Business Intelligence applications have been studied much more, but the implementation of their interfaces has not been the focus of many articles. In these studies, researchers were more interested in finding out the values to visualise or in handling data instead of the layout of the system.

We can draw the following possible arguments from this research:

1. Database and keyword selection was unsuccessful, research has been done and published but it cannot be found with these search terms.
2. Research on this topic has been done but it is not publicly available.
3. Research on this topic has not been done.

Many articles have been written about how effective web-based dashboard systems are, so it is possible that there is work done on their layouts inside companies that apply these

systems, but this research has not been published. If this is the case, studies like this that are public for all audiences are significant for others interested in the same topic. Another possibility is that implementation of layout techniques is seen as too much of a design decision problem that it is not interesting in academic Computer Science research. As pointed out in Gesing et al's (2014) article, mostly researchers are more interested in other aspects of such systems and only list responsive design as constraint or feature that is left for developers to deal with. Producing responsively designed systems might be seen as something more technical and not well defined to be academically studied.

3 RESPONSIVE WEB DESIGN

In this chapter, responsive [web] design (from now on referred as RWD) is discussed. RWD is a way of designing a web application or website layout to adapt to different screen sizes and to give optimal experience for users with whatever size or shape of a screen. RWD became relevant during the past decade after mobile devices started to have more users online. With products like Microsoft Surface Pro and Microsoft Surface Studio, hardware manufacturers are pushing touchscreens to professional workstations. Computing in all levels of usage is getting a wider range of hardware. For example, Apple who has been praised for being a top manufacturer of creating professional tools in the tech industry, have last updated their desktop computer on December 19th, 2013, and have been concentrating on publishing mobile devices and laptops. Sales of desktop machines have been steadily dropping, while tablets and laptops are taking some of their market share in consumers hands (Titcomb 2017).

Constraints of normal websites are getting broader each year. One could argue that the normal use of one web application could happen in a device that supports any one of the characteristics described in **Table 5**. Devices included in that table are examples of what an everyday customer could buy, excluding all prototypes that should need extra work to get.

The PC market is getting broader, as more manufacturers bring touch screens to their devices. Desktop users can theoretically use resolutions and screen sizes that are their graphics card and monitor output are able to produce. Mobile phones are getting bigger, but so are their display resolutions. Galaxy S7 Edge has a 5.5-inch display with PPI (pixels per inch) of 543. This means that if a developer was to design a website based on absolute pixels, those elements would be extremely small on this kind of smartphone screens. Mobile phones have been evolving in a few years tremendously, and as mobile market also grows steadily, developers need to serve more different audiences if they want to provide their services as broadly as they can.

Smart TV's are bringing another kind of problem with different operating systems like Android TV, Apple TV, Firefox OS, Tizen OS or WebOS. In some cases, these operating systems are provided to the TV-screen by a pluggable device, in some cases, it is built in

feature. For example, Apple offers Apple TV as an independent device. These devices also offer users an ability to access the internet. The problem with this is that for example Apple TV controller only has 6 buttons and voice input.

In addition, game consoles usually have web browsers. For example, Nintendo 3DS has two low-resolution screens which of only one has touch input. On the other hand, Steam OS controllers provide two touchpads and multiple platform specific buttons. Both devices have their own built-in browser. Most interesting of all might be Nintendo Wii Internet Channel since its browser is based on TV version of the Opera, as presented in the investigation by Perakakis and Ghinea (2015). In this case, an input is taken care of by Nintendo Wii motion controller.

Table 5 Common consumer devices to access the web and their constraints (2016)

Device Family	Common Input Methods	Screens
Desktop or Laptop	Keyboard and mouse/trackpad, optional touch input	Anything from 9-inch laptop to the 80-inch TV screen. Resolutions up to 5K by the end of 2016
Mobile phones and tablets	Touch only, optional hardware buttons for device specific input (back/home), optional keyboard	Common tablets up to 12.9-inch (iPad Pro). Samsung Galaxy S7 Edge 5.5-inch with PPI 534
Smart TVs	P-Pad controller with optional alphabetic and numeric input, optional voice input	Up to 85-inch and 8K resolution
Game Consoles	D-Pad controller with extra buttons and axis inputs, optional touch area input	From Nintendo 3DS 800x240px 3.54-inch up to Steam Link powered PC with top tier PC GPU and high-end TV

Web site and application developers need to either construct their services to function under any kind of device users might access, or then to exclude some of the users based on what device they are able to use at the moment they happen to stumble upon their services.

For services like Spotify who aim to serve as many customers as they can, this has been solved with a web based player and more functional native clients. But for all service providers, this is not possible, as supporting multiple native codebases is expensive and mostly needs experts in their specific fields. Web applications are getting more complex and capable all the time, with functionalities like interacting with Bluetooth-devices (Beaufort 2015).

Browsers and the web as a platform are getting more features all the time that try to get them at the same bar with native applications, for example, Mozilla's WebAssembly, which aims to add native performance to web applications. With this kind of performance, it is guaranteed that in the future web has an even wider change of high-performance software. That added to an even wider range of devices that are able to connect to these services, creates new kind of challenges for developers.

3.1 About Responsive Design

Responsive Web Design is by definition creating content with CSS and HTML that can resize, hide, shrink, enlarge or move based on how it would look on screen (w3school 2017). By this definition, websites that claim to use responsive web design must be written with CSS media queries, which will take the size of the screen to define rules on page elements. Another take on defining RWD is by Luke Wroblewski, the author of the book Mobile First (Wroblewski 2011). He mentions RWD to be implemented with CSS, HTML and "sometimes a bit of JavaScript" (Wroblewski 2011). Then again w3school states the following (w3school 2015):

“

What is Responsive Web Design?

- *Responsive web design makes your web page look good on all devices.*
- *Responsive Web design uses only HTML and CSS.*
- *Responsive web design is not a program or a JavaScript.*

“

The problem with this definition is that it not only defines RWD to be the outcome of specific techniques, it also defines technologies that are supposed to be used in order to achieve wanted outcome. Even in the book “Responsive Web Design (2nd Edition)” (Marcotte 2014) author lists multiple workarounds to add functionality and backwards compatibility with JavaScript.

Because there are many conflicting definitions of this term, from now on in this paper we will ignore those who define responsive web design to be only CSS technique and define it as follows: “Responsive web design is a design pattern, which aims to produce content and containers which can adapt responsively to different screen sizes. Responsive design means developer’s ability to also design web applications, simple or complex so that they can be fully used in a wide range of different devices. “

In a broader sense, responsive design also takes into account other aspects of different client devices. In his book “Responsible Responsive Web Design” Scott Jehl defines a term “Responsible Design”, a paradigm of actively thinking of developing in the lowest possible constraint (Scott Jehl 2014). An example of this is to design services to work at small touchscreen phones in a high latency mobile network. In many cases, this might be the actual case of the applications usage most of the time. This also means abstracting input methods and forgetting platform specific actions, like mouse hovering which is impossible to emulate (natively) on touchscreens.

The popularity of RWD in IT industry has also influenced the amount of research in this area. Many researchers (Jiang et al. 2014; Lee et al. 2015; López et al. 2016) have been going through different methods and libraries with which RWD can be implemented in products and papers about using RWD in different projects (Mohorovičić 2013; Otsuka et al. 2015; Kittivaraporn et al. 2014). Most of these papers have one thing common: they all mention designing “Mobile first”. “Mobile first” is a term originating from a former chief design architect of Yahoo, Luke Wroblewski in his book Mobile First. It essentially means designing web application by thinking how it first behaves on small mobile screens, then expanding it to a desktop.

Transferring existing desktop web applications UI to mobile ones can be a very time-consuming job if it is not designed to work on mobile devices. This was found out in an article written by Voutilainen et al. (2015) in their project on opendata.fi. They wrote a paper about their project, in which they designed a website and then created a mobile interface for it afterwards. They were able to use existing libraries to convert UI to work on different screen sizes, but they also mentioned that this was not possible in some cases. When designing user interfaces, it should be taken into consideration that there might be a simpler way to represent the whole system and build on that, revealing more functions and information as more room gets freed.

Mohorovičić (2013) has written about implementing responsiveness on a website project. He introduced a different way of thinking: Desktop first. Essentially it is exactly the same thing as Mobile first, but instead of starting the design from mobile, in the Desktop First design is started from showing everything important and then deciding what to hide and collapse. In this way of thinking, elements are not hidden and to be docked in layout, but instead visible and to be collapsed or hidden. It doesn't matter from which side developer will start looking at the product, content prioritising needs to be done in order to figure out which content to get out of the screen when space runs out.

Sometimes just stacking hidden content can be done multiple ways, as long as users can either find when he wants to see it or the content are shown automatically when users need it. Usually, a safe way to visualise collapsible content is to use methods that are already familiar to the users (Majid et al. 2015).

In an article released by Microsoft Mobile, RWD is summarised in three concepts: Modularization, Minimalism and Single-Sourcing (Katajisto 2015). This supports Mobile first way of thinking, where websites should be thought of as modular building blocks that can be shown, hidden and moved as needed for automatically creating different layouts for different screen sizes.

Mostly papers mention RWD in the context of Mobile First and mobile applications, but a new range of devices has entered consumer's web habits in past years. As it is pointed out in many papers, adapting to different devices doesn't necessarily mean only adapting to

smaller mobile phone screens. In TV related research by Perakakis and Ghinea (2015), it was pointed out that many users nowadays use different d-pad like input methods for navigation in different systems. Examples of devices with internet browsers are Smart TVs or gaming consoles (for example Steam OS console controlled by Steam Controller or Sony PlayStation console). In their research, they have different suggestions on how to use navigation methods that work well for huge screens and d-pad input. They defined the following principles for TV-optimized web design (Perakakis and Ghinea 2015):

- Minimum font size of 22px
- P-Pad navigation
- Paging is better than scrolling
- Autofocus on navigation
- Clear focus on selected navigation item
- Don't place important elements too far sides of screen

In their research, they found out that by following HTML5 standard and using common elements defined in it, a developer can have the best possibility to successfully support different TV browsers.

Clark (2015) mentions how important it is when designing for mobile applications to position the elements into the screen the right way. He calls this “The Rule of Thumb”, the way of inserting the most important elements (those which need to be accessible by means of input) in the range of user's thumb. He also calls users “One eyeball and one thumb”, meaning that when entering mobile applications author of the site mostly has only user's half attention, as he/she in many cases uses a mobile application while not paying full attention, but also with one thumb, referring the way user holds mobile phone in his hand.

Touch and mobile devices might have revolutionised input methods in application development in the early 2010s, but in the future developers might also need to think about those users which don't touch their devices physically at all while visiting their website. There are already devices like Leap Motion and Microsoft Kinect which will detect motion and gestures of users who stand or sit in front of the device. This kind of interaction with technology is not common yet but might become mainstream in the future. Another implementation of such technology can be found in some mobile phones, where a sensor

detects hand motion above the screen or the movement of the eyes, like in some Samsung products.

Research done in this matter by Hespanhol et al. (2012) found out that while comparing different air gestures, the dwelling was perceived by users as the most intuitive. On the second place was grabbing in air, followed by lassoing and pushing. The research was done by making subjects to play a card game with Microsoft Kinect. In this game players needed to use these gestures, and their performance was evaluated.

Although RWD has become the buzzword of web design, there are alternatives. One of these philosophies to create multiplatform web applications is “Adaptive design”, as described by Aaron Gustafson (2015) in the book “Adaptive Web Design”. A key difference between adaptive design and RWD is that in adaptive design, a developer creates multiple static layouts that are used differently in different devices. This method has benefits over RWD if a content of the application needs to be drastically different depending on the viewport size or shape, as it is more beneficial to create totally a different layout.

RWD is found in many papers as an optimised and effective way of creating a generic website for multiplatform purposes. But as found out later in this thesis, should not be considered as a “magic bullet” that hits all the targets.

3.2 Implementing Responsive Design

Websites are usually created by using HTML (Hypertext markdown language) for marking element structures, CSS (Cascading Stylesheet) for marking style attributes for HTML elements and JavaScript for creating additional functionality. As mentioned earlier, purely done RWD layout by some definitions only uses HTML and CSS to create fluid, responsive containers for different devices and screen sizes. When this is not possible, JavaScript needs to be added to ensure the wanted functionality.

Some of the easiest ways to get started with creating RWD containers are public domain CSS libraries like Bootstrap and Foundation. Some of these libraries require additional JavaScript functionality, for example, Foundation provides navigation tabs and complex

dropdown menus. These libraries provide CSS classes and JavaScript pre-sets to create responsive elements.

In the simplest form, RWD consist of three main ingredients (Marcotte 2014):

1. Flexible, grid-based layout
2. Flexible images and media
3. Media queries

Flexible images and media mean making media that is shown at screen take a proper amount of space compared to other elements, with priorities in mind. Secondly, this media should look good in all sizes. With images, this can be achieved by using vector graphics, described in detail in the last section of **Chapter 4.3**.

The grid-based layout is usually achieved by defining the whole layout in rows and columns, and giving elements some relative sizes to viewport so that they take size properly. This can be done with tables, flexboxes or in future CSS grid.

CSS Media queries are blocks that rule out parts of CSS rule by defining a master rule to them. Media queries are defined by first starting a new CSS rule with string `@media`, then it's given a media type and finally the actual query to do on that media. All the media types are defined in w3.org specifications. Media types are print, screen, speech and all. Print defines rules for printers, which is queried when a user wants to print out the website. The screen is most of the computer screens. Speech means screen readers. All refers to all media types. Final media query consists of media features, for example “min-resolution: 300dpi” or “width: 600px”.

Flexible Grid means dividing the layout to be constructed out of rows and columns, that then are collapsed to different orders based on screen sizes (with CSS media queries, ideally). To achieve flexible grids a designer must think of the size of the elements in the web page in relation to each other. This can be done by first understanding how a unit system works in CSS.

In CSS units can be either relative or absolute. Absolute fonts are pixels (px), points (pt), pc (picas), centimetres (cm), millimetres (mm), or inches (in). These fonts are not by nature relative to any attribute of the application (like viewport size) so they are not usable as is in responsive applications. How to size them is explained in **Table 6**.

Table 6 CSS Units

Unit	Based on	Description
em	Relative	Font unit that is relative to its direct parent's font size (1em equals the parent elements absolute font size).
rem	Relative	Font unit that is relative to its root elements font size (1em equals the root elements absolute font size).
%	Relative	A unit that is directly relative to its parent elements corresponding property.
ex	Font-Family	A unit that is relative to x-height (fonts lower case height) of the current font or one-half em.
ch	Font-Family	A unit that is relative to font families '0'-character width.
px	Absolute	A pixel is 1/96 th of an inch on the screen. Represents a pixel on a screen, but is not.
cm, mm, in	Absolute	centimetres, millimetres and inches on the screen.
pt	Absolute	1/72 th of an inch on the screen. Used in real life print media, represents a point in printed paper.
pc	Absolute	12 points. A pica represents a character in the font.
vh, vw	Viewport	Viewport height and width. They are relative to 1% of the either width or height of the viewport.
vmin, vmax	Viewport	Minimum and maximum value selector of vh and vw. IE does not support vmax.

So-called relatively sized units are ideal for simple responsive applications. Some of them are relative to some parent container like em, rem or %. These units size themselves into some context, for example, em to its parent's size. Ethan Marcotte (2014) in his book prefers sizing context container in pixels and using em and rem based on it. This is ok for simple

websites, but for case system in this thesis, we need something more sophisticated. The solution is described in **Chapter 4.3**.

Additionally, there is `ch` and `ex`, which are units that can be relative to the container's font family. In these cases, if a designer wishes, he/she can size the container to be exactly sized based on how much text he/she wants in the container in question. For example, setting container's width to `40ch` means that the container can hold 40 times '0'-characters that it uses. If a font family is monospaced (each character takes exactly the same amount of width) that means that whatever text the container holds, in whatever font size, its' width is 40 characters of that text it holds. The same effect can be achieved with the `ex`, which takes its height from the font's lowercase characters' height.

Finally, there are units based on viewport size. Units `vh` and `vw` are relative to viewport height and width and represent the percentage of those measures. For example, if viewport's width is `500px`, and element is marked to have width of `50vw`, that element's width is `250px`. If the viewport at any stage is resized, those elements relative to it will too. Finally, `vmin` and `vmax` take the minimum and maximum value of `vh` and `vw`. It is notable that `vmax` is not supported in Internet Explorer. These units' way of representing measures of elements can be responsive to elements.

With these units, designers can create responsive and flexible grids by using relative units of measurement. Usually, this is best done applying it to Marcotte's (2014) ingredient three, which is media queries.

3.3 Justifying Responsive Design

Typically, responsive design and Mobile first have positive outcomes on the design of user experience, as a designer really has to think of the content that is visible to users and prioritise on what to show (Kim 2013). This can produce more informative and easy to understand applications for users.

Lestari, Hardianto, and Hidayanto (2014) found out in their research that responsiveness of the websites did not affect users' ability to understand website's content based on the

homepage of the website. In their research, they found out also that responsive layout was able to reduce users' scrolls required while reading content, while non-responsive website required fewer actions (clicks and scroll) from users to find information with navigations. In this light, responsive websites are harder to navigate if there is a lot of content. Bloated content on the screen can make websites very hard to read and understand (Kim 2013).

With responsive design website, developers don't need to maintain multiple codebases (Kim 2013; Olson 2013). This also means no need for server side redirects for multiple instances of the same site (Olson 2013). In some cases, one unified codebase for every platform is not the most optimal case. Responsive websites might add a lot of additional code to support different web page sizes, which makes website's size bigger. Additional JavaScript functionality adds bloat which might add critical load time on low bandwidth networks. The same bloat might also add performance issues (Kim 2013). Additional bloat on codebase might also slow down development if a project is not structured correctly. Having two completely different layouts let users load only the code (JavaScript and CSS) that is needed for his/her platform (Olson 2013).

To fight the problem of polluting codebase with adding platforms to support, web development community has created projects like SASS (syntactically awesome stylesheets) or Jade, which are pre-processors for existing web design languages. Aims of these tools are to simplify codebase and add functionality to native web languages that help developers to produce simpler multiplatform code. For example, SASS is a pre-processor for CSS, which adds features like variables. Jade is HTML pre-processor which simplifies syntax and adds features like iterators, variables and templates. These pre-processors aim to make template languages act more like programming languages.

While these pre-processor extensions to existing languages might make the development of responsive web applications easier, they add bloat to production pipeline. While CSS, JavaScript and HTML don't need to be parsed or compiled, pre-processed template languages and extensions need to be processed before they can be used. To automate this process, developers have created task managers like Gulp, which will take care of compiling the source. This adds another layer of complexity to web development, which could be done just by writing a source file.

If a developer chooses to use widely spread frameworks for responsive design, it is possible to maintain a cleaner database by using tools to check for layout errors. With one modular and responsive layout per view, the layout can be tested to work on different platforms. In research done at University of Sheffield and Allegheny College researchers introduce ReDeCheck, a prototype of a tool that can automatically check for any layout errors and CSS faults on sites created with popular RWD frameworks like Bootstrap and Foundation (Walsh, McMinn, and Kapfhammer 2015).

It is notable that sometimes what seems to be a preferable design might not have the best performance. Balagtas-Fernandes et al. (2009) found out that users preferred layout form is not usually the most efficient. In their study users like more a tabbed tablet layout of the test application, but when after testing it, both empirical observations and user feedback showed that scrollable one-page version was significantly faster and easier to navigate. This was because the test device used had a trackball for scrolling, which made usage of scrollable layout easier. In the same study, it was noted that users found long pressing was a counterintuitive method of opening context-menu compared to hardware menu button. This is an example of how designing for every device can affect negatively on user experience because pushing unified way of interacting with the application can be difficult for users who are used to the platform's native user experience.

It can be that sometimes when users are already familiar with the non-responsive layout of the site they prefer more pinching and zooming of desktop layout than learning to navigate the responsive one (Kim 2013). Forcing users to learn a new layout unwillingly also goes against two of five usability principles: Familiarity and Consistency. Hiding elements also might put a developer in a temptation of removing functionality completely for a smaller interface (Majid, Kamaruddin, and Mansor 2015).

Making a web page responsive without designing for it may hurt the user experience more. If not designed properly, an application designed for "Mobile first" can compromise the desktop's user experience. Completely different layout for mobile and desktop can add freedom for a designer to create the optimal layout for each platform. (Olson 2013)

Small screen space will create different problems entirely for web designers. For example, it is typically harder to insert a banner advertisement, if that is what users want (Olson 2013).

In 2017, Google will start giving a penalty for a bad advertisement that will disrupt user experience on the website. Getting lower search engine visibility will itself grant great harm to websites' popularity. ("Helping Users Easily Access Content on Mobile" 2016)

4 IMPLEMENTATION RESPONSIVE WEB DESIGN

This thesis was written for ABB Ability™ platform product cpmPlus View. Ability™ is ABB's platform for Industrial Internet of Things. View, part of the cpmPlus product family, is a web-based dashboard editor and UI SDK for this product, written in JavaScript, and created for data visualisation and management. cpmPlus, project originally started in 1997, is a family of products built around the RTDB database, developed in ABB.

For practical part of this thesis, this product is presented, and improvements implemented to it to make it responsive. The goal is to make the editor of the application responsive to different screen sizes, and usable for both mouse and touch inputs. The dashboards that product outputs should be able to, as result, be configured to respond to screen size and shape, and be used on both mobile and desktop platforms. The desktop versions should be usable on all screen sizes, viewed on TV also.

Figure 5 shows the editor in whole. At the top, there is a Top bar -element, hosting company logo, product name, some buttons and username. Rest of the space is reserved for a window manager, which is introduced more in-depth in the next section of this chapter. Inside it, we have five windows: two at left are tree navigation window, which can be used to open new dashboards or inspect equipment model (abstract data model used by the system). Under it is a property grid, editor window used to edit any properties in the system at runtime. In the middle, there is a tab window, which hosts two dashboards. At right, there is dashboard hierarchy window, which shows active dashboards widget hierarchy. Window with blue borders has been undocked from tab window and is active/in front (hence blue).

View includes implementation of Component – Entity architecture, reflection, serialisation, own class structure and inheritance and wrappers for events.

In this chapter, new developments concerning responsiveness of View are presented. A full list of constraints and requirements are listed in **Appendices 1-3**. As the implementation of responsive design to the product is represented in following sections, the IDs of the following are mentioned in the text. They are referred to prefixes C (Constraint), F (Functional

requirement) and NF (Non-Functional Requirement). Changes presented in this thesis are implemented in versions 1.3 and 1.4.

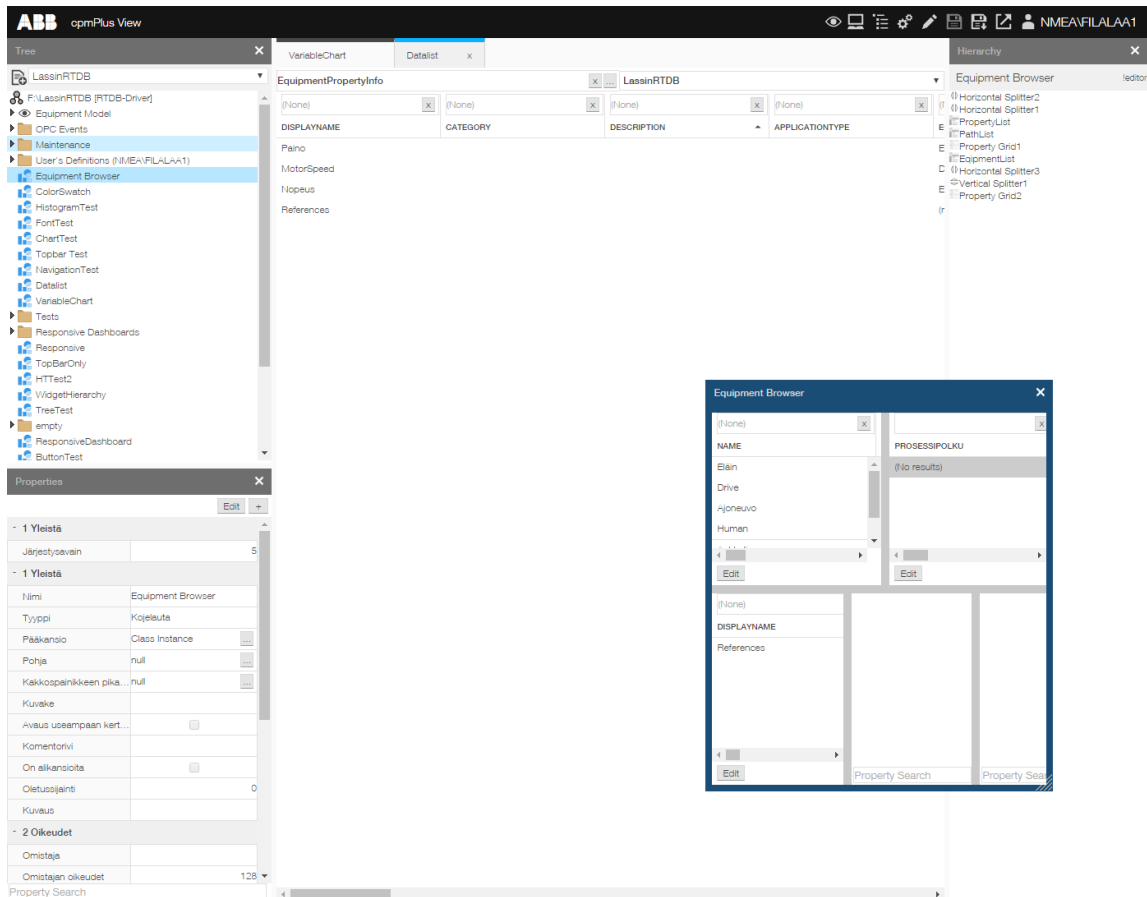


Figure 5 View 1.4

4.1 Editor

In this section, fast introduction to Views Editor is given, and the parts that belong to it. View can be separated in editor and runtime. The editor has a window manager, which can host different windows, that contain either dashboards or editor specific tools. These tools are essentially just widgets that are hosted in special containers outside of the dashboards. The runtime is a host of one dashboard, which works as separate “application”. The runtime can be either running inside editor desktop, or its own browser tab, completely freely from the editor.

In **Figure 6** improvements made to the top bar of the window between 1.2 and 1.3 can be seen. Top bar was the only part of the editor that did not work responsively, so it was replaced with more modern version. The original top bar inside 1.2 was created with custom split-windows that hosted images. These split-windows were configured to have constant width unless resized with a mouse. This design was good for a time when the system was designed to work only on the desktop. In future versions of the product, it might be the case that different users have a different number of buttons in this bar, so it was necessary, in addition, to add a more responsive element to host this element, to replace it all together.

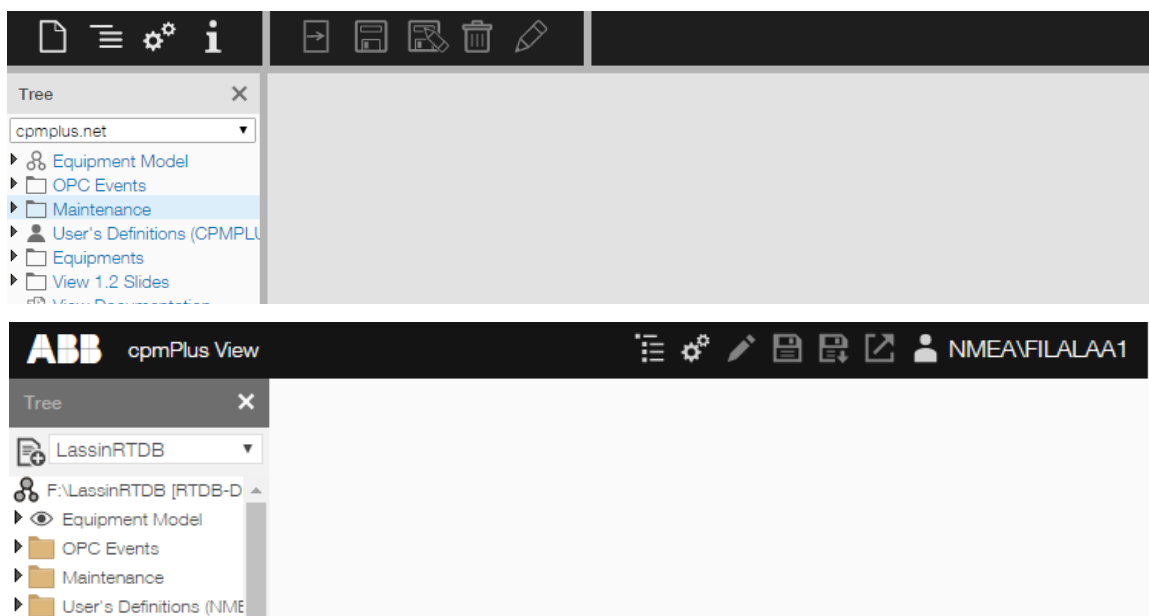


Figure 6 Top bar improvements between 1.2 and 1.3

The top bar itself is just a top bar -widget, instantiated outside window-manager and dashboards by the app itself. The design of the system is modular so that widgets work in any container they are appended into. In the same way, other editor windows host widgets in special containers that can drive editor. For example, Tree-window is just special container Tree-Widget, which opens new dashboards to Desktop-window (or tab-window inside it). Widgets are covered more in depth in section Widgets4.3.

During the development of 1.3 toolbar-component was created. Toolbars are used in several widgets, most notably as the main visual component of the top bar -widget, and are built to be a responsive version of the old implementation of toolbars in 1.2. This component, among

other things it does, replaces the old implementation of toolbar specified split-windows, used in the top bar –widget of 1.2, which is visible in **Figure 6**.

Window manager

Window manager takes care of handling editor windows. The whole system is hosted in different windows that user can move and dock to create different layouts for different needs. All windows are in either tab-window, split-window or float-window.

In addition, there is a desktop, which works as a base for editor itself. It is a special container inside navigation application that acts as a split-window in a case that it can be used as a split target. It also hosts every new dashboard that is requested to be opened from tree window, but it always holds these windows in a tab container. In case if there is only one dashboard in a desktop container, it looks like it would be hosted inside a tab container.

Split-windows are window containers that always have two panels. This panel can either contain one window or group of windows. If multiple windows are dropped into the same panel, they form a group that is represented as a tab-window. Split-windows have two panels, which are separated by a split line. The split of the panels can be either horizontal or vertical, and split lines distance to each side of the split windows sides are relative.

Float-windows represent a window that is placed on a screen temporarily. A floating window has holds its absolute distance to window manager's top and left sides and its absolute size inside window manager. A floating window can be moved by selecting its caption (top side) and dragging. It can also be resized by either (with a mouse) hovering over to any side, and when mouse pointer changes to operating system specific resize cursor, resizing that side. There also is resize area on the bottom right, which can be selected with either mouse or touch input. Other windows cannot be docked into float-window, but float-window can be docked into split or tab -windows.

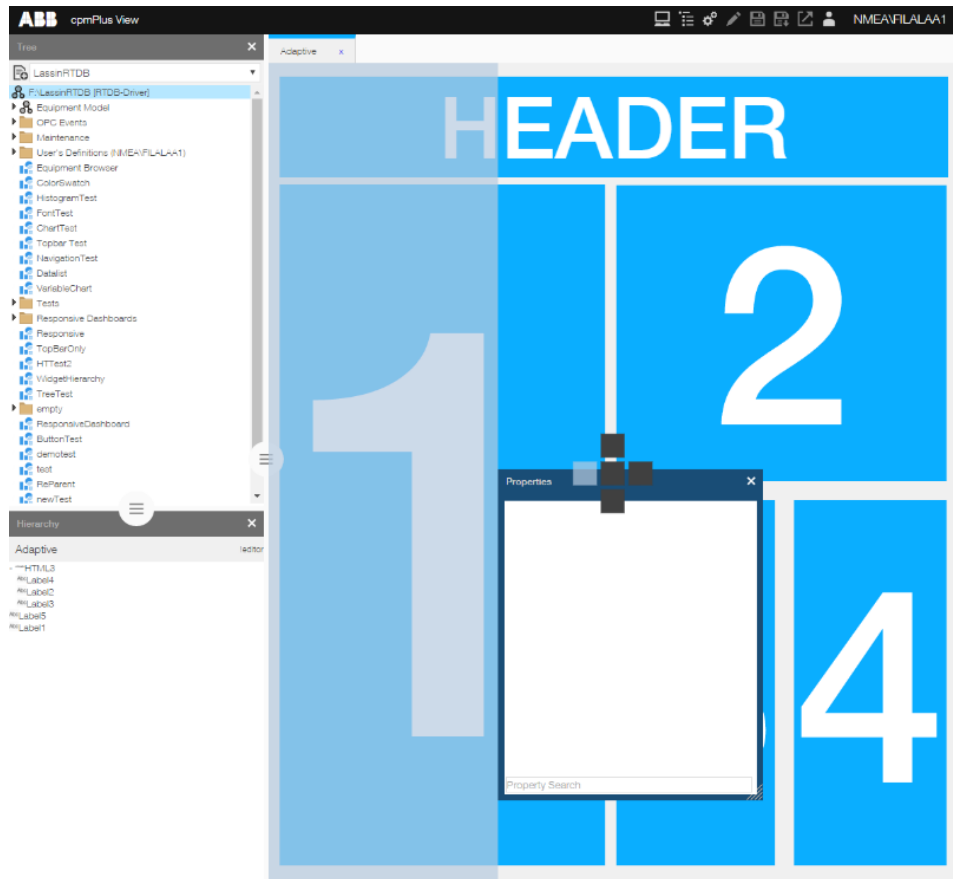


Figure 7 Float-window being docked to split-panel containing a tab view

When a floating window is docked directly into another split-window, a tab-window is produced. Tabbed window is hosted into its split panel, but has tabs out of which user can select the active one. The active tab is the window that is shown in the panel. A window can be detached from tab-window by either dragging it out of the tab-button container with a mouse or holding with touch input for 150ms. Normally, when a tab is detached from tab-window, and only one tab is left, tab-window is destructed and the last tab left is represented as docked float window. This is the case in every other case than a desktop container.

In **Figure 7** process of a window being docked to the left side of the desktop container is shown. In the same window, moving handles are created for split-windows (F-14). As mentioned before, split-windows can be resized by dragging split line, which is a small margin between two split-panels. With mouse input, this can be easily done, and when a mouse hovers over the split-line cursor is changed to represent the direction split-line can be moved. With touch, split-line is hard to drag. For touch, sizing handles were created so that

they are easier to grab with a finger. These handles can be toggled to be visible, as they might be distracting when not used.

Input Methods

Supporting input methods for multiple or all different devices is very tricky in web JavaScript. Since CSS version 3 specification there has been pointer type of events, which support both touch and mouse input. For View, a `cDOMEvents` class has been created to abstract the registering of input methods for all devices (F-11). By giving element that events need to be registered, a developer can register different abstract listeners that will give unified outputs for pointer, zoom and drag events.

In base class created for all structures in case system, event handling is being created. These events are registered in the construction phase of each structure by defining them with a member function that takes a name and additional information as a parameter. In `cDOMEvents`, this functionality of the system is used as a layer on top of existing HTML event listener functionality.

Events `In`, `Out`, `Leave` and `Enter` for both mouse and pointer are not included in `cDOMEvents`, as they cannot be represented in touch at all. Instead, different gestures that can be represented in both mouse and touch.

Supported events include `click`, `contextmenu`, `pointerdown`, `pointermove`, `pointerup`, `pointerlongpress`, `dragstart`, `dragend`, `dragover`, `drag`, `drop` and `zoom`. From these events `click`, `contextmenu`, `drag` and `drop` are currently used only as fall through functions, just passing native event through systems event system. Pointer events for `down`, `move` and `up` are used as a common layer on top of either mouse and input events, or pointer events if used browser supports it. `Zoom` can either be fired when the mouse wheel is used, or when a user interacts element with two fingers pinch and zoom.

The input system is meant to not only provide functionality where it does not exist but also limit some functionality that is emulated when it is not asked for. **Figure 8** shows how Google Chrome emulates right mouse click while long pressing with “`GesturesLongPress`”-event. From this figure, which is a screenshot of Chrome Developer Console, we can see

Chrome triggering an unwanted event when emulated touch-input is pressed, and after about 600ms triggers right mouse down function. To reflect this, cDOMEvents has its own “GestureLongPress”, which can be configured to call context menu in case one is not opened (F-12).

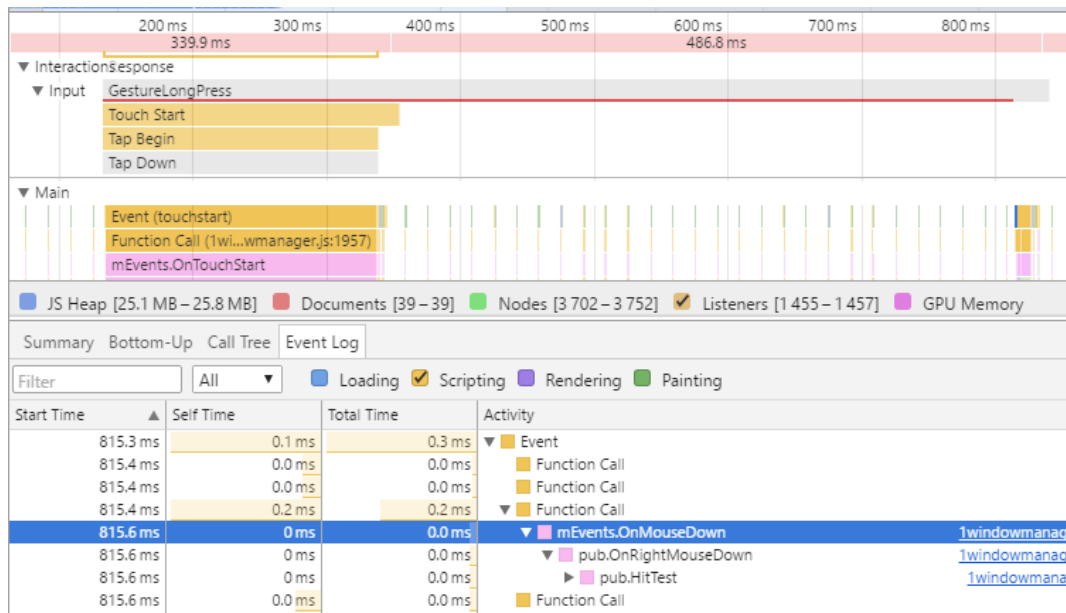


Figure 8 Google Chrome simulating long press event

It was noticed by using this system, that if moving of some element rely on mouse getting pressed down, then moved and finally being released, if somehow the client happens to freeze for a moment and the actual element is not under the pointer when the mouse is released, the element does not receive the pointerup-event. To fix this, a developer would either need to implement a listener for global pointer up event or check otherwise if the mouse is still up while the pointer is moving. To tackle this problem, cDOMEvents class distributes information on its pointer move event if any button is pressed.

Event handling with separate class is effective, but in some cases not necessary. In some cases, where there is no need for multiple listeners per element, for example, if some element works as only click listener, a function is bound to elements onclick-property. This way developer does not need to take care of garbage collection either, as the listener is cleaned up after the element is destroyed.

The editor itself works as a shell to host other pieces. Because its role in the whole system is this, it does not need to have any other responsive functionalities, as they are implemented in other parts of the system.

4.2 Dashboard

In this section, the responsiveness of the system is discussed on a scale of one dashboard. Widgets are placed and docked by the user inside the editor. For the responsiveness and this thesis, multiple new improvements have been made to the system to support responsive layouts.

Layout Object

A layout object is an interface between user given widget positioning input and scaling motor(s) (F-7). Ultimately, Layout is just an abstract class that is inherited by different layout objects. In View, there is currently one scaling motor, that calculates widget containers size and scale based on anchors. Different layouts inherit from this object and can specify parameters to define final widgets look on a dashboard. Each widget follows set by their parents' layout. If a container is set to be responsive, all widgets that inherit their position from it will follow this parent widgets layout. A responsive layout is an object inheriting layout object that is specified to handle its children responsively. The relationship between widget-containers, dashboards, layout and scaling is represented in **Figure 9** with UML (Unified Modelling Language).

Layout Object has two main functions: refreshing its own state and updating DOM-elements (Document Object Model element) of its children. In traditional responsive systems, responsiveness is done by blocking CSS-rules in media queries that are used by selecting them based on screen viewport size. In different layouts, actual DOM-element positioning decision is made by first using given input from a user (placement inside the editor) and scaling motor calculations and then making the decision based on a size of each elements scope. In responsive layout, the decision to make is either to use selected scaling motors given position or use collapsed position. For traditional "Rubber Layout", the decision is to always follow scaling motor's given positions.

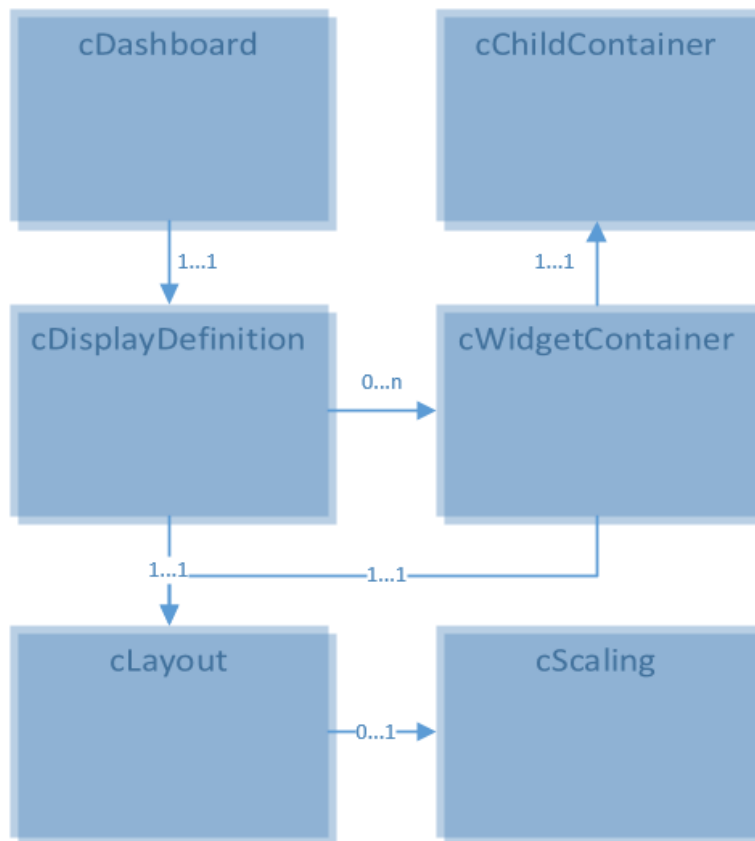


Figure 9 UML of widget-dashboard-layout relationship

For this system, the decision to use layout object to host properties to drive layouts was made so that the layout of the dashboards would not be only tied to default Rubber Layout. The old implementation already had separation of actual scaling motor between the rest of the system, so it was natural to create sort of an interface to handle user inputs. Layout object can be seen as a unified container of the layout-related properties for both dashboard and widget container and is serializable by View’s property system.

Widget Inheritance

In version 1.2, View did not support inheritance of the widgets, so this was necessary for layouts and responsiveness to be implemented. Making a widget child of another widget changes the scaling scope of the child widget to be parent widget instead of a current dashboard (F-4). Inheritance of the widget is implemented to be able to mimic Marcotte’s (2014) RWD Ingredient “Flexible Grid Layout”, which is implemented as a responsive layout object.

There are two different ways to think of widget inheritance: how elements are placed inside DOM and how to store objects in JavaScript. Multiple ways of both problems were considered. In widget level, inheritance relation of the widgets can be parsed by looking at widget containers ParentWidget-property. There is following design decisions made on this:

- Information of the parent is stored inside a container, as containers might be initialized and positioned before actual widgets.
- Each container holds information of the actual widget, as they are serializable and individually identifiable with UIDs (Unique Identifier).
- If a widget has ParentWidget-property of “null”, it is positioned inside the current dashboard. There is no use of storing information of dashboard, as every container already has “Runtime”-property having that information.
- List of all containers can be found in dashboards runtime handled in an array structure. Additionally, the dashboard has lookup table generated for widget hierarchy, which holds UIDs of each widget as a key to an array of child widget UIDs for fast lookup of inheritance.
- Information about the parent is stored as a property of the component, hence it can be serialised.

Each parent object holds its children inside its child container. The exception is, that every child of a dashboard is already in the root of dashboards DOM-element, as dashboard itself does not have sibling elements, or any other contents than actual children (widgets). It is natural that dashboards DOM-Element is itself child container element of its children. For widgets, their children are stored inside child container that is a sibling element with widget container inside DOM.

Multiple different ways of storing children were considered, but rationale in final decision is, that by simply researching already written implementations of older widgets it was seen that many of them use in their reinitialization methods technique of emptying contents innerHTML (given parent element's contents) completely. In other words, old implementations of widgets rely on the fact that in old system container that was given for widget to make its implementation to. Considering this it was either necessary to create another div inside widget container and call this contents of widget etc. and make child containers as a sibling of this element.

Additionally, child-containers were created, as a reservation for future use of using all children as a unit (with used container element) as a separate group than their parent element. This way children could be found. Child container was also implemented for simply having able to use HTML native overflow functionality to create a scrollable list of child widgets anywhere on the dashboard. Usually, when the parent of these widgets is a dashboard, scroll is going to be in a scope of the whole dashboard, but this can also be implemented in children of a widget. Using of native HTML features and elements is the best way to create scrolling for ubiquitous systems, as browser implementations of the standard specification may vary.

Having separation of child container and widget also gives freedom for layout developer to select whether to follow calculated bounds by scaling motor of a widget with children or not. In the implementation of responsive design, this is a desirable effect, as we want to follow selected scaling motors given positions only if the screen size is not under some user defined limits. When these limits are exceeded, we wish to collapse children container to another size. More information about the responsive layout is given later in this chapter.

Widget inheritance editing

View is essentially a dashboard editor, so tools needed to be created to easily modify the hierarchy of widgets (F-5). All the widget containers are at the same level inside parent container as their child containers. Every widget container is responsible for its own child container. The widget is given its own container and is responsible for this element. This way containers are still separated from the widget implementation, and the widget can still be cleaned by simple emptying parent elements innerHTML-property.

In editor dashboard designer, can easily change the parent of the widget container by either selecting reparent tool from editor menu while one widget is selected or selecting the tool from widget's context menu. Additionally, a developer can change the ParentWidget-property directly from widgets properties. There is an error dialogue presented for a developer if:

- a) Reparenting process causes a loop in parent chain
- b) Reparenting process causes 200 widgets deep parent chain
- c) Developer tries to parent widget to itself

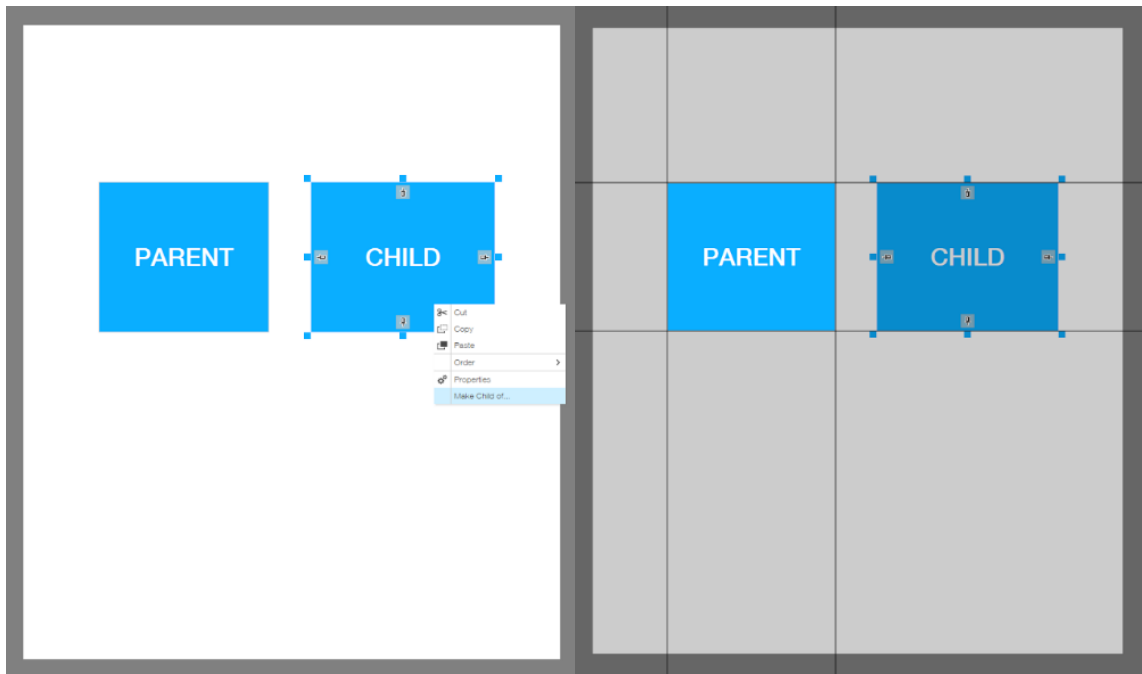


Figure 10 Process of re-parenting a widget inside the editor

After a widget has been made children of another widget, scaling scope of its anchors is changed to parent widgets bounds instead of dashboards. **Figure 10** visualises how label widget “child” has been made child widget of another label widget called “parent”. Constant anchors that are not hooked to other widgets are by default holding their positions to outer bounds of its new parent.

To visualise the hierarchy of widgets inside a dashboard a widget was created, which can be seen in **Figure 5** (F-6). This widget was then implemented inside editor window so it could be used as a development tool while creating new dashboards. A widget was created for just visualising hierarchies and has only one additional function, which is widget selection when in edit mode. For future work, it is advised to redo the widget with implementing already made tree widget with widgets as nodes.

Widget container anchors and Rubber Layout

In View, default container positioning is done with “Rubber Layout”. This name comes from an idea that each dashboard is made of rubber, and each widget then placed on it can be positioned with four anchors. These four anchors are on four sides of widget container rect.

For each anchor, there are three possible different anchors to select from: pinned, anchored (constant) or floating.

Pinned anchor is a relative anchor that holds its relative position on a dashboard. For left and right side of the container, an anchor that has pinned type will hold its position relative to parent container's width. For top and bottom anchors on widget container rectangle, pinned anchors will hold their position relatively to parent container's height.

Constant anchors (or anchored sides as they are called in case system) are types of anchors than hold their absolute distance to their opposite end of the parent container. These anchors can also be set to any other container inside the same parent. In View, this process is called “hooking” to another widget. For example, if top anchor of a container is pinned to another container, it will hold its absolute distance to that containers bottom side, even if that distance is negative.

Floating anchors will take no action in resizing or moving the container. If all sides of a container have floating anchors, the widget will float freely inside its container. This means it will not resize and will hold absolute position to top and left sides of the container. If opposite side of a container has been anchored (either pinned or constant) freely floating side will not take any action. This means it will hold the same size, and another side of the container will take care of moving in that axis.

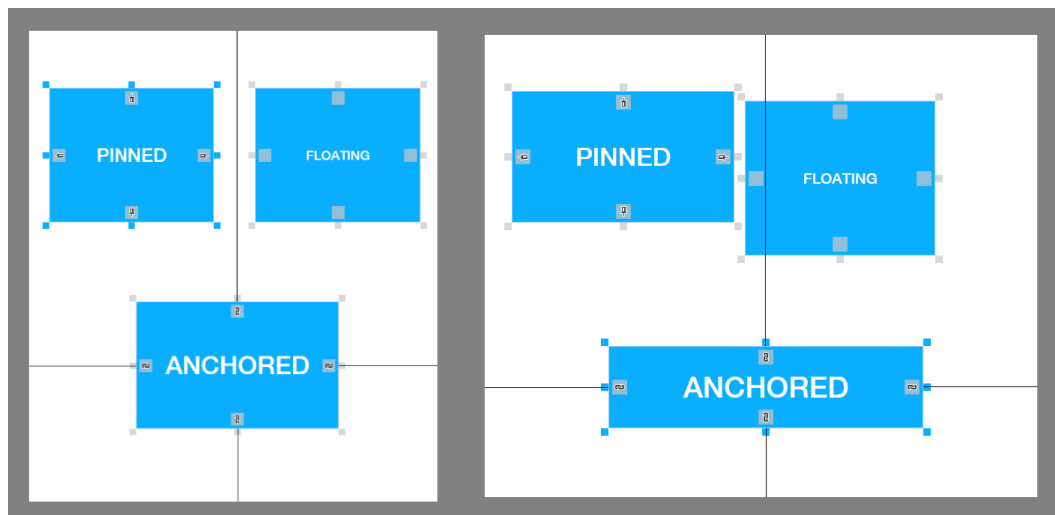


Figure 11 Three label-widgets inside dashboard with different anchors, responding differently to dashboard size change

Out of these three anchoring options Pinned anchoring works best for keeping the overall layout of the dashboard (or any particular parent container) in any size. Constant anchors are mostly used to have other widgets hold constant distance to another widget, that has been forcefully made to hold its size using one floating size. For some widgets, this is necessary, as all of the widgets are old and not from the beginning designed to work responsively.

This pinning system (or so-called “Rubber Layout”) can work well with different sizes, but when the aspect ratio of the dashboard changes radically it is necessary to reorder all widget containers. In some cases, it is necessary to hold the position of the containers in their positions on every device. For this, case system has built in setting for always holding dashboards aspect ratio. In this case, the system will generate empty space insides that need it and hold the dashboard as is in full size in the middle of the screen.

Rubber Layout has been a default layout style for case system. In this layout mode scaling motor calculates new positions for each widget based on their anchor relationships, and widgets are placed in parent DOM-Element exactly where scaling motor tells them to go. In some cases, reordering the widget containers is not a bad thing, and for this, a responsive layout was created.

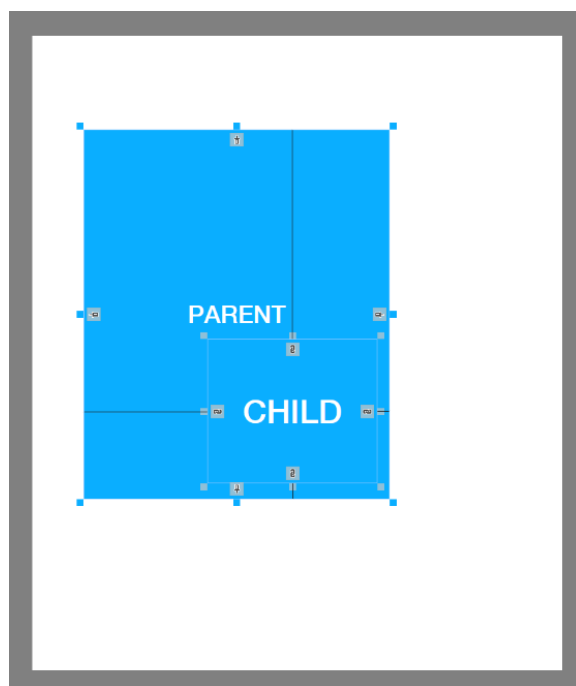


Figure 12 A child widget inside parent

Responsive Layout

The responsive layout was created to emulate responsive design layout of dashboard inside dashboard element (F-10). As mentioned many times in this thesis, the final ingredient of RWD “Media Queries” is not suitable for this the system, due to its limitations (Marcotte 2014). Fortunately, we have been able to implement a system that mimics this by restricting the scope of scale inside parent container of each widget, and the creating responsive layout objects for each container that hold scaling and positioning properties of each widget. Responsive layout builds on already a defined Widget inheritance system so that by changing parent’s layout to be responsive we can make all widgets that are direct child elements to act responsively based on responsive widgets size or aspect ratio.

In practice, Responsive Layout is a combination of freely positioned widgets, positioned with Rubber Layout, and fixed positions by placing widgets in line while in wide or narrow aspect ratios. With widget inheritance, widgets can form a grid that acts as fluid grid container. Every widget holds its own children in their own scope, allowing every children container to react to their own state. This should allow a designer to create different layouts quite freely.

Responsive layout is built to follow given scaling motors rules unless given limits are exceeded. The user can define both maximum and minimum limits, with both absolute value in pixels and aspect ratio. For minimum limits, if parent size falls below given limits, the mobile (narrow) view is used. When parent bounds exceed maximum limits, wide view is used for child widgets.

Responsive layout views are made to follow the common case of collapsing elements on top of each other in case either horizontal or vertical space is in use. The most optimal way of doing this is by calculating y or x offset for each childNode inside parents DOM-element. These child nodes can be accessed as HTMLCollection from DOM-element with DOM references children function. To make this process faster, children are sorted inside parent element so that the offset calculation takes linear time.

While in responsive mode, widget follows size instructions given by the user. In both narrow and wide mode, the user gives a size of a widget as aspect ratio, which is just a text property

that is formed by two numbers, separated with “:”-character. In the narrow mode, all of the children widget affected will take the full width of their parent widget, and height of the widget is calculated with the size property. In wide mode, the same procedure is done with the full height of the parent widget size.

Example use case of responsive layout

To better grasp the idea, an example of how this system is used is presented. **Figure 13** holds screen-capture of example, that is discussed in this section. In this dashboard, there is a total of six unique widgets. Three of those are children of the dashboard, which has a responsive layout and three of them are made children of these three. These three widgets that work as a base of the dashboard layout are the Label-widget with text “HEADER”, Label-widget with number 1 in it and invisible panel widget hosting three other labels. Panel widget also has responsive layout.

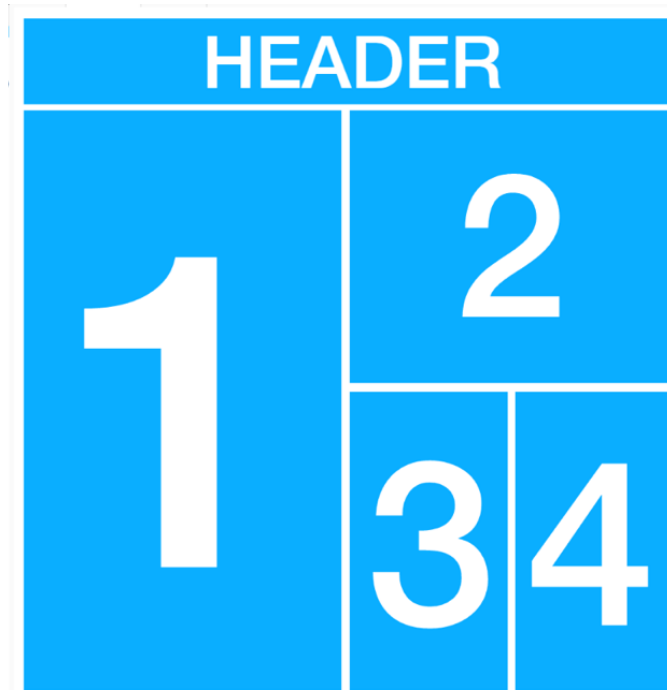


Figure 13 Example dashboard layout

In this example, we are concentrating on a layout that can be easily configured to work with both mobile and desktop. This example also shows layouts systems capability to collapse in multiple layers.

Figure 14 shows how the previously mentioned dashboard can adapt to provide a different layout for different screen sizes. In two first pictures from left dashboard has been collapsed in three different parts. In this dashboard, in the first level of widget hierarchy (widgets that have children of responsive dashboard layout) are set to collapse when dashboard widget is smaller than an absolute value of 800px. Header, first label and panel have collapsed on top of each other. On the second level, labels that have numbers 2, 3 and 4 are still placed with their anchors, but inside their parent, which is now taking more space.

The second level of widget hierarchy, children containers of panel-widget, are set to collapse when their parent element is taking less than 600px of width. The interesting part of this is that panel widget would have already been collapsed unless layouts “collapse before parent”-property would not have been set false, as it would have in many cases been less than 600px of width way before the dashboard width is smaller than 800px. This example of an adaptive layout is taken from company style guide and is probably used in many places to present the effects of responsive layout.

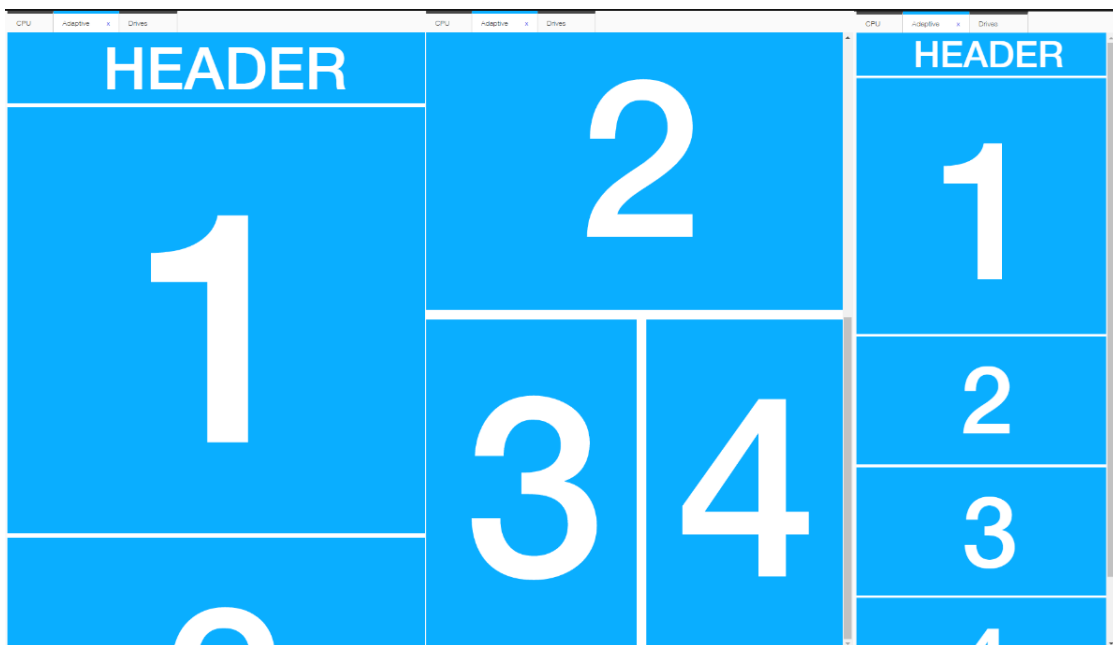


Figure 14 Example dashboard collapsed in two stages

In **Figure 14** the most rightmost layout image shows how all the widgets have been collapsed after the dashboards width drops lower than 600px. This example works as proof of concept. In many cases, a designer of a dashboard would probably want to have better aspect ratio

adjustment for labels presenting numbers 3 and 4, depending on what these widgets host in their real-life case.

In **Figure 15** another dashboard is shown, in this case, shown how to section of it can be designed to work in both wide and narrow mode. It has to be noted first that in both cases (left and right) dashboard has been cropped to fit better this document. In left, the dashboard is wide (small height and big width) and in right dashboard is narrow (small width and bigger height). In this dashboard, the designer has configured a header area, which has a white background, and blue area, which takes rest of the space and would probably host some widgets in real life example.

In this example, we want to examine header area, and how it has been configured to act responsively. First of all, header area has been set to not collapse to responsive mode before actual dashboard has been collapsed. In mobile view (narrow), dashboard header area is set to take 6:1 aspect ratio of space, and never collapse. In wide view, dashboard takes 1:4 space, and is set to collapse when its aspect ratio exceeds 1:1 (which in this case happens always in wide view). When this happens, header collapses to its wide space, in which case widgets that are its children are set to act responsively in narrow mode (stack on top of each other). Because by default widgets in the stack take 1:1 space in width and height, widgets are forming a nice stack of blocks.

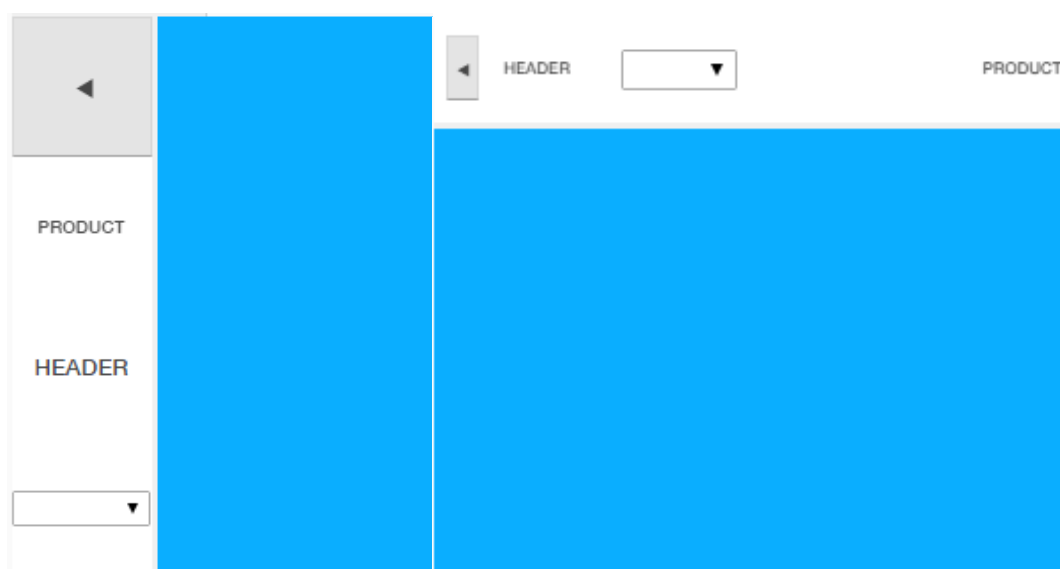


Figure 15 Dashboard header element collapsing in two directions

These examples show how View's responsive layout can be used to make dashboards that can be used in any system.

4.3 Widgets

In this section responsiveness of View in the scope of a widget is described. It is notable that all the separate elements in a while system are represented with elements that are built as widgets. For example, editor windows such as Tree, Properties and Hierarchy window are just windows that host a specified widget. These widgets are designed so that they can be placed in a dashboard also. It is important that all the widgets are built with responsiveness in mind, as at some point they might be needed to be used as an editor tool.

As mentioned, an individual widget might be hosted in a widget container inside dashboard or inside a window container. Because of this, when constructing a widget, the developer always need to keep in mind that parent element that is given to constructor of widget might be of any shape or size. Additionally, when designing responsive elements, the designer must understand that he cannot ever rely on viewport size. The widget must always be designed to respond to given parent element's size.

Usable DOM-elements

Concerning responsive elements in View, two different structures are used: Flex based grid and Table containers. Elements of which CSS display-property has been set to flex are so-called flexboxes. Children of flexbox are flex-items, and they are driven by CSS properties. Flex items can be positioned and sorted from CSS. These are both techniques that enable the creation of Fluid Grid, which is the first ingredient of RWD (Marcotte 2014).

Figure 16 shows two different versions of same Tree-widget. One on the left is more a traditional looking desktop version, where mouse input is preferred. It is hosted in the editor window and in most editor layouts, always show in the left of the screen for fast dashboard opening and navigation. On the right side, there is the exact same widget, but this time it is using different layout mode. This layout mode can be toggled from widget properties. It is designed to be easier touched with finger.

This touch layout mode uses Flex properties. In this case, flex is used to wrap items automatically using flex-wrap. Items are also easily aligned or justified. Flex has the ability to fill itself with child elements with grow and shrink -properties, without calculating it with JavaScript. Additionally, flex items can be easily sorted with just changing the order property, without changing the actual order of the DOM-elements. The downside of using Flex is that the support of it is marked only “partial” on Internet Explorer 11 (“Can I Use Flexbox?” 2016). Additionally, according to some tests, using table is on some browsers faster than flex (Frain 2014).

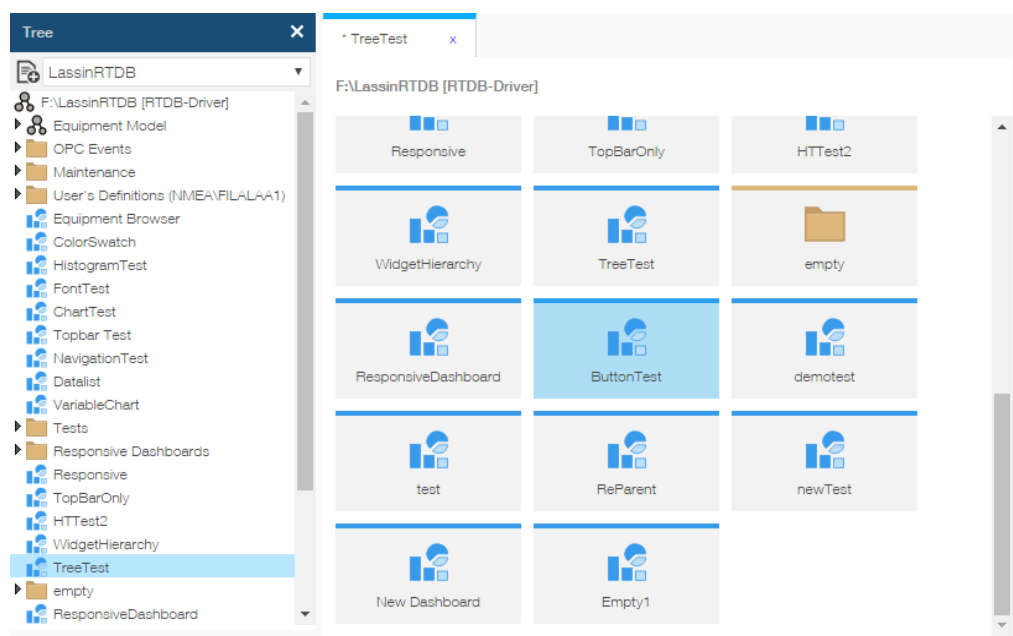


Figure 16 Two instances of same tree widget, another one with touch optimised layout mode

Table based containers, like toolbar element used in View, rely on special properties of CSS display property table. Table elements can be vertically aligned and can fill their container automatically. Inside table, if one element has been defined to take 100% of the width, and rest of the elements have some fixed width, the one with 100% will always take the rest of the space. By using this special feature of the table, toolbars used in widgets are created (F-8).

The toolbar is a component that is easily configurable with single JavaScript-object, built with general item objects and is able to hold its content responsively, and even collapse to

some hidden state if the user wants so. In the constructor of toolbar item, the developer can specify factory functions for two different DOM-elements, the element that host the item and “Hidden DOM-Element”, the element in which the content is moved whenever the given collapsing rule is filled. Responsiveness of toolbar containers can be seen in action in **Figure 17**, in which toolbar is implemented in the top bar -widget.

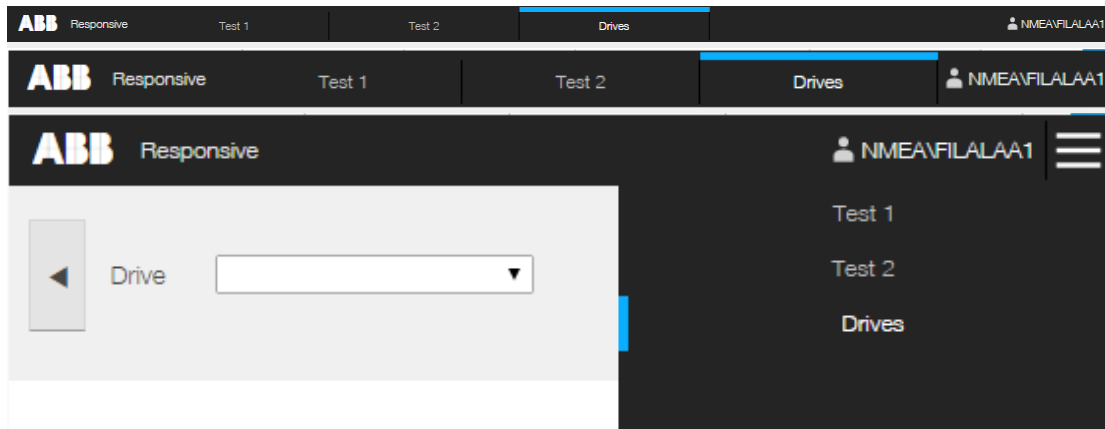


Figure 17 Responsive Top bar –widget

Fonts

Fonts and typography are one of the basic elements of any application and are part of the flexible media in RWD ingredients (Marcotte 2014). “Font” is in a case of systems reflection defined as its own property type. In View, Font properties have been generated to handle one widgets fonts. In widget definition, as many font properties can be defined as needed. In the definition of one font property, a developer needs to give this property a prefix, so each property can be separated from each other.

To register font properties for the widget, the developer uses the following function:

ABB.Mia.Font.AddFontProperties(object, defaults, category, prefix);

In this, the object is linked to object itself that properties are installed into. Defaults will set custom default values for properties if needed. The category parameter will set a custom category for each added property. Property categories are used in, for example, property grid widget, to categorise and sort properties. Finally, a prefix can be used to separate multiple font properties from each other.



Figure 18 Custom editor for font-properties

In View, a developer has the option to write custom editors for properties. In this case, if an entity has a property of type `FontProperties`, custom editors presented in **Figure 18** is presented (F-3). In this figure, the above line is minified version of this editor, used in the property grid. In it, the user can configure the most used properties, `FontSize`, `FontFamily` and `FontColor`. By clicking the icon on the right, full font property editor is opened.

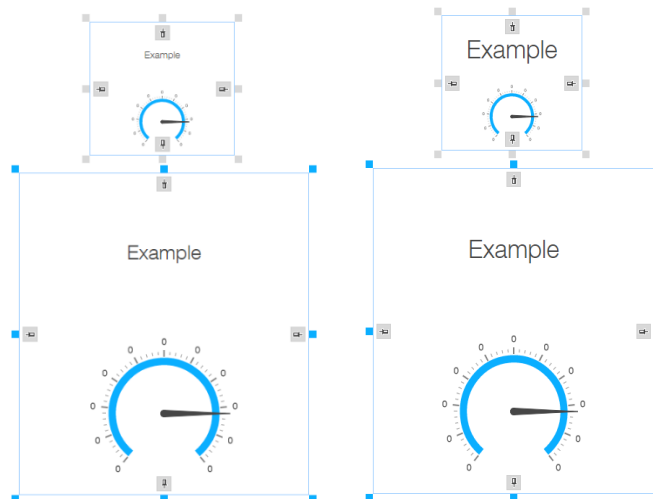


Figure 19 Adjusted font scaling restricting vs non-restricting dashboard

Figure 19 has an example of adjusted font scale restricting on a dashboard scale (F-1). On the left side of the picture, there are two widgets that are on the same dashboard, with unified font scaling. In this example, designer wishes to have all the fonts inside dashboard to be scaled with a common target, whether they are hosted in dashboard widget or not. On the right side, two widgets that are inside same dashboard have different font target that they scale on. In this second example, two instances of the same dashboard both scale inside their own container instead of using root dashboard size as scaling target (F-2). Constraining font adjustment to not find the most utmost dashboard can be useful when dashboard represents not part of the other dashboard but its own view.

Adjusted scaling is calculated as a pixel size, relative to adjusted roots width or height, depending which one is smaller. The size of adjusted root is transferred to a plain number by comparing it to the default size, which is 1024 x 576 pixels. Adjusted root means the first dashboard that intersects font adjustment inheritance. This root can be either root dashboard or dashboard that is placed on another dashboard as a widget. The font is calculated:

*$min(\text{root_width} / \text{default_width}, \text{root_height} / \text{default_height}) * \text{size}$* .

Additionally, a developer can define another font scaling types. For a label, it was necessary to create maximum scaled font, in which font scales to always fit the widget container. This has been achieved by placing table container inside widget container. Inside another container, the actual text container is placed. When upscaling font container the outer container scales up too. With binary search the biggest possible font size is found, where the text container still fits in the widget container. This effect can be achieved with a text container only, inside a widget container. Three containers have been used, so that a text label can be centred both vertically and horizontally inside a widget container, using CSS tables “vertical-align” and text-align properties.

Images

The second part of the Marcotte’s (2014) RWD Ingredients is “Flexible Media”, which also covers images. Image handling responsively can be seen the same way as widgets: there are a container and its contents. In general, there are two different ways to handle images effectively inside a web page: pixels and vector graphics. Pixels are the older way of

representing images on the web, but as found out in earlier chapters, they cannot scale up without losing quality. Vector graphics are created for this, as they represent paths, shapes, and areas with filled and stroked colours. This means they can be of any size, and still look sharp in browser viewport.

A common way of representing images in vector graphics are images in SVG-format (Scalable Vector Graphics). SVG is free format, that is based on XML marking language. SVG has its own tag in HTML5 standard and is supported in all of the modern browsers. Vector graphics were created to create an image that is never going to be pixelated, in any size of the viewport. There are multiple ways to view vector images inside a browser viewport, following as an example (Coyier 2013):

- using IMG-tag (Image) and SVG-image as a source (works in IE9)
- using SVG image as a CSS background-image property (works in IE9)
- using inline SVG-image
- using SVG as an object

Just placing SVG file contents inside an HTML page does not do, as we want to download elements dynamically. This can be done with embed element. By inserting the whole SVG-element as embed-element inside HTML, the SVG-attributes can be changed at the runtime of the application. With this, for example, toggleable buttons with different colours in different states are possible. Unfortunately, after testing this solution, embedding the whole SVG-image inside HTML page was seemingly slower inside Internet Explorer.

To tackle this problem, View has a dynamic image `cImage`-class to take care of this (F-13). For normal use, the server can handle SVG attribute changing by URL (Uniform Resource Locator) parameters. Supported parameters are shown in the following table. An example of loading an SVG-image of an alarm bell, with a size of 75 times 75 pixels, an opacity of 0.9, fill colour of blue, stroke colour of red and line width of 0.4:

```
/alarm_18_75x75!o0.9!fb!blue!sred!lw0.4.svg
```

Image parameters are given to URL inside image name, right before the file format. All image parameters are separated with exclamation marks. All of the supported URL parameters and their image counterparts are listed in **Table 7**.

Server changes the SVG-attributes, saves new a copy to cache and serves image to the browser. It can be used as IMG-tag source. When the user toggles image, another image is served from the server, and both are saved to browser cache so that they don't need to be fetched in future. This optimisation provides an efficient SVG image using for case system needs, so no other image system was seen to be developed for this thesis.

This system surely has some fallbacks, as changing a colour property linearly during time would cause a server to create multiple static images to be served. But for images that have only a few states, these images can be cached on both server and client side can be fetched/used very efficiently.

Currently, for icon purposes, this system has been more than enough, as it takes advantage of both flexibility of vector graphics and efficiency and browser support of pixel graphics. For full vector image support, a user can always take HTML-widget, of which inner contents can be easily manipulated inside dashboard editor.

A special case of loading spinner is handled by its own class, which uses CSS3 animation to show a spinning image. This class provides spinner and loading text, and it is easy to append in any part of the system by only providing element it is inserted into.

Table 7 Supported parameters of the server modified SVG-image.

Class member	SVG attribute	URL parameter
SVGFill	fill	f[colour code]
SVGStroke	stroke	s[colour code]
SVGStrokeWidth	stroke-width	lw[float in scale [0.0,1.0]]
SVGOpacity	opacity	o[float in scale [0.0, 1.0f]]
Size	viewport-size	[integer]x[integer]

5 EVALUATION OF FEATURES

In this chapter, the evaluation process and the outcomes of evaluation are covered. It was necessary to do separate evaluation process for all the features to create an academic understanding of how outcomes benefit the product. In following sections, evaluation is introduced, outcomes of it discussed and future developments presented.

5.1 Evaluation Process

Evaluation of the product was done by conducting a small usability study between versions 1.2 and 1.4. This study was performed in following steps:

1. The two-part survey was created for evaluator group, a cognitive walkthrough on creating a very simple dashboard. The first part is for both versions, second part helps to implement some responsiveness features that only exist in the new version.
2. Evaluator group was asked to inspect version 1.2 and try to follow instructions on the first part of the walkthrough. After this, they answered a small survey on the usability of the product.
3. All evaluators were asked to do the same walkthrough again, this time with the new version and doing both parts. The second part of the task implements some new responsive features to the dashboard that was already created. After this, they were asked to fill out the same survey again.

After steps two and three, the evaluator was given a tablet, so that he/she could also test and evaluate how version in hand works in authentic touch. Additionally, they were informed on how to use touch emulation on their browser debugger for additional testing.

A survey that evaluation group was asked to fill can be found in **Appendix 4**, and cognitive walkthrough steps can be seen in **Appendix 5**. The questions of this survey are divided into three categories based on ISO 9241-11 standard (Brooke et al. 1996):

- Effectiveness – Users ability to complete tasks using the system, and the quality of the output of those tasks.

- Efficiency – The level of resource consumed in performing the tasks.
- Satisfaction – Subject reactions to using the system

Evaluation group consisted of five co-workers from the office. Based on the answers given by the evaluation group, we hope to reach an understanding of how changes done during the process of this thesis have affected the product in general.

Although the sample size was small, statistical significance was still tested with Two Tail Wilcoxon Signed-Rank test to find if some of the findings achieve statistical significance, hence they can be mathematically proven to be significant. Wilcoxon Signed-Rank test is equivalent to the paired T-Test with non-parametric data (Wohlin et al. 2012). In this case, we use two-tail test instead of one, because we are expecting possibly either increase or decrease in values (Ruxton and Neuhäuser 2010).

In this test, we use Critical Value (CV) of 0.1, although it is admitted that smaller critical value would lead to more reliable results. Unfortunately, that is only possible with higher sample size. All the answers were tested per question. The statistical significance has been marked in the last column of **Table 8**. Hypotheses are:

H_0 : There is no change between the versions.

H_1 : There is statistically significant change between versions.

According to test, if tested statistic is equal or less than the critical value, null hypothesis H_0 can be rejected. Otherwise, the results of the test are not statistically significant. The actual change between versions is described as an average increase or decrease **Table 8**.

Table 8 Survey results in categories of ISO 9241-11

Usability Characteristics	Quality metrics	1.2		1.4		Change in Mean	Significance p <= 0.1
		Mean	Std. Deviation	Mean	Std. Deviation		
Effectiveness	It is effective to create dashboards with the editor.	3,33	0,75	3,5	0,76	+0,17	
	Dashboards produced with editor work well.	3,17	0,90	3,33	0,75	+0,16	
	All the tools inside editor seem to work as I expect them to.	2,33	0,75	2,67	0,75	+0,34	
	The system reports and recovers from error states.	2,83	0,90	3,17	0,37	+0,34	
	Provided widgets seem to scale well with the dashboard.	3,00	1,00	3,67	1,11	+0,67	x
Efficiency	There is enough information (for example description texts on properties) on how to use tools inside the editor.	2,17	1,34	2,17	1,34	+0.00	
	Tools, their amount, and placement on editor did not disrupt my workflow.	2,17	0,37	2,50	0,76	+0,33	
	Complexity and learning curve of the tools are not high.	3,00	1,41	2,67	1,37	-0,33	x
	Tools are easy to find in the editor.	3,00	0,82	3,50	1,12	+0,50	
	There are enough tools for me to create dashboards without scripting.	3,00	1,15	3,00	1,15	0.00	x
Satisfaction	Dashboards that I create work well on both mobile and desktop.	2,33	0,47	2,83	0,69	+0,50	
	This version (both editor and produced dashboards) works well with a touch interface.	1,50	0,50	3,67	0,75	+2,17	x
	I think this version implements the principles of RWD in its interface.	3,50	0,96	4,50	0,50	+1.00	x
	I like the interface of the system, I find it pleasant and modern.	3,50	0,76	4,00	0,58	+0,50	
	Overall, I feel productive when working with the system.	2,67	0,94	2,67	0,75	+0.00	

5.2 Evaluation Results

Based on the feedback given by the evaluation group we can form following assumptions:

1. On the average, the product had either hold its level or improved in almost every aspect, although it cannot be statistically proved on every question with a sample of this size.
2. Most variances in answers were given when asked about the efficiency of the product, in both old and new versions.
3. The biggest improvement was done in the overall look of the interface and capabilities to create responsive dashboards and usage of touch in editor and dashboards. Both are statistically significant results.
4. The effectiveness of the tool increased between versions, and evaluation group felt that widgets scaled well with the new version.
5. The addition of new features slightly added a complexity of the product, which affected negatively to the learning curve.

Table 8 shows average, distribution of values and change. Raw values given by evaluation group can be found in **Appendix 6** and **Appendix 7**. Based on the feedback given by evaluation group, the current state of the product is that it is quite capable of creating dashboards. The new features made a huge difference in capabilities of responsive dashboard creation.

The old version of the product was not designed to support touch interface usage. During the development of the newer versions, there has been a huge effort in transforming the editor in more touch friendly and usable software. Even with time used in stabilising the new version that evaluation group were testing with, the editor was found not perfectly stable, but usable with a touch interface.

Overall, satisfaction with the product has had the biggest difference. Four out of five areas in this category saw improvements. The biggest increase out of all questions was also about touch capabilities, which saw an increase of 2.17. Also, the question of RWD implementation had a significant increase between versions.

Worst points in both old and new versions were given in structure of the editor and simplicity. There have been efforts in creating the editor that is usable and capable of creating dashboards, but it was found, based on this experiment, still quite confusing to use in hands of inexperienced user. With too vague instructions evaluators had problems in performing simple tasks. This probably caused Efficiency to either not increase or have a bigger deviation in experience among evaluation group.

Four out of five questions in the category of Efficiency had a deviation of over 1.1. From this, we can assume that there was a huge difference in how evaluators felt the efficiency of the product. There is a possibility that for this test, the sample was chosen poorly, as evaluators had different backgrounds. Some had experience with the product, while others only knew of its existence. It was noted that for some, the editor was easy to use and they figure out the task very easily, as for some the tasks were quite difficult, as the users were not able to make the connection between the instructions and actual features on screen. In the future, it is suggested to hold quantitative usability testing for those who have used both versions extensively, to find out significant results in this category.

For some questions, for example concerning widget capabilities and scripting, the evaluation group found that the test provided did not give enough insight to give a proper answer. It is notable, that for this question the survey design was too loose and poor, as the scope of walkthrough did not specifically guide on product scripting, as it would have taken too much of the testers time, and probably required additional training. For this, qualitative research in the future is also strongly advised.

It was found, that the way of using property descriptions to inform the user how the tools are affecting the dashboard, does not scale well with the increase of unique properties and tools inside the editor. For some of the evaluators, it was very unclear that this description system was in place at all. Some of the evaluators found that the old version and its tools were easier to understand than the new one, simply because it had fewer functionalities. This caused part of the evaluation group also feel that the interface was less confusing in the older version. One of the evaluators pointed out, that the placement of some buttons (main toolbar in particular), were “non-standard” and ”somewhat confusing”. When asked about the learning curve in general, dispersion among the evaluation group was high.

Overall, changes seen in questionnaire between versions is seen as positive change, and useful feedback when moving forward with development. The task of improving the capabilities in the scope of responsiveness can be seen as filled. Although this feedback survey might not be statically especially significant, it is directional and provides fast feedback for developers in finishing the release of 1.4.

5.3 Future Developments and Limitations

This section is reserved for a conversation on functionalities and ideas that could or should be further investigated or developed, but that were either outside of the scope of this thesis work or should be implemented in future on top of now developed functionalities.

During the work done for this thesis, we changed the original system by defining clear objects for different parts of layout and adding modularity to the system. Layouts are right now designed to support multiple different scaling motors, but currently only one exists. Responsive layout supports user-selected scaling motor, but change is not visible because no other exists. The decision to add support for more scaling motors was made to keep and enhance the modularity of the system, but unless there is a need for additional scaling motors this change was unnecessary.

The anchoring system is very tightly bound to rubber layout but is coded deeply inside widget container. It is recognised that this tie could be broken, but due time limits of master thesis and new test cases this would need it are left for future developments. The editor itself expects anchors to exist, and their presence in widget placement inside of the dashboard is very natural. The need of additional scaling motors or replacing of the current implementation is discussable but not that relevant at the moment.

Current toolbar system does use property type of function, which is currently not serialised and can only be seen inside the editor. The decision to use this property type was made in the case at some point toolbar system could be editable and usable as a widget. This could be done easily in future, but a decision of how to use current properties needs to be first made. There is also need to create more built-in support for vertical toolbars, of which can

be made currently but are not tested or used, which could lead to findings of additional needed changes in future.

It was inspected during evaluation stage that the current usage of description based information sharing to users is a bit confusing, and new ways of guiding users to help their work should be investigated. During the development of the product, while this thesis was written, the number of tools and complexity of the system has risen, so there is new need of communicating with users on how to use different parts of the system to lower the learning curve for new users.

In usability evaluation of the product, it was noted that, although the tools were effective and working, producing usable layouts, they were not yet user-friendly enough. In the future, if the product is meant to be used widely not as an engineer only tool, more usability testing and layout design for the editor should be considered to make engineering tool more user-friendly.

The current implementation of layouts and inheritance seem to cause unnecessarily many full refreshes on the whole dashboard the while in both loading and runtime, which could be limited to a certain amount. To optimise dashboard refreshes there is already some work done to profile refreshes to signal the scope of which the changes has been made (full refresh, container size didn't change etc.). To fully incorporate this more testing needs to be done. For cDOMEvents, it could be possible to also implement resize event for widget size change, which could be used as a different type of refresh functionality on widgets to further optimise the whole system.

Font scaling is currently always bound to dashboard scale, and it can be restricted to the scope of a dashboard inside another dashboard. Widget inheritance was introduced as a new functionality after current implementation of font scaling was designed and implemented, so after this thesis it is probable that target scale of adjusted scaling should be changed from scope of dashboard size to current root parent containers size, with a default of dashboard and widget not restricting to find root container. This would move font scaling calculations from dashboard scope to containers altogether.

Usability evaluation itself is not probably statistically very significant, as the group size was small, but it gives some estimate and direction. In future, it is suggested that proper usability studies are held to get more information. Especially usability should be tested in an aspect of figuring out how learning curve and simplicity of the tools can be improved to help users better understand the system.

6 CONCLUSION

In this research, it can be seen that, although, in web development, developers usually tend to find loopholes to create the lightest way to find the solution. In addition, sort of unsophisticated methods needs to be used to create support for a complex system to create and emulate wanted functionality, as an example media queries. The product introduced in this thesis shows that responsive layouts can be produced in such systems, and that responsive web design paradigm can be implemented in a system that does not rely only on CSS.

Implementing responsive design into a product of this scale can give value to the customer, as the same dashboard can be designed to work for devices with many screen sizes. The responsiveness of website, in general, makes dashboards future-proof, as the difference in smallest and highest possible resolution size gets bigger.

The question of how to achieve responsiveness and responsive design in industrial dashboard editor (**RQ**) is answered by the description of new features in View, described in **Chapter 4**. Examples of these are different responsive containers that mimic the fluid grid and introduction of image class that handles vector graphics. The main ingredient of responsive web design is media queries, which was found to be impossible to incorporate in a product of this complexity (Marcotte 2014). The effect was replicated with separate layout objects, which are currently handled in JavaScript. It is admitted that this is not as optimised as calculating positions in CSS, but unfortunately, current limitations of CSS prevent its usage in the current system. With all these implemented to the system, **O2** is achieved.

The implementation chapter also works partially as **O3**, supplemented with comments within the new source code, and future additions in the product documentation. **O1** is seen as fulfilled, although as it was seen in usability evaluation, stabilisation of the product is still an ongoing effort before the release. However, with new developments, user satisfaction in the product has risen according to usability evaluation.

Creators of the term “Responsive design” have probably created it at a time when web techniques were more limiting. Using media queries is an efficient way to create adjusted

layouts for different screen sizes in a very browser light way. This technique, which is already mentioned in early 2010, is now seen as very limited as users expect their applications to do much more than that. As seen from related work, researchers have already noticed that desktop, laptop and mobile phones are not the only devices that the internet is consumed nowadays. Widescreen monitors and TVs are gaining popularity, and so even more devices need to be supported to gain the badge of “working everywhere” in web development.

Responsive design was found to be surprisingly faulty as a paradigm and term. Creating responsive applications as they are defined right now is impossible in the scope of this research. Many tricks and fall-back methods are needed to create functionality where it does not exist yet. In conclusion, we have created responsive dashboards inside a dashboard editor, but in a way that they are not meant to be created. With these new features, View remains competitive, as the art of dashboard creation is mixed with modern web development.

Extensions like Typed JavaScript transpilers (e.g. Typescript) has gained some movement lately, and in the future, it could be useful for the development of View too. To take advantage of modern web language extensions, SCSS (Sassy CSS) is already in use in the production of View. Web development takes big steps in a short amount of time.

During the writing process of this thesis, element queries have emerged. A public domain JavaScript plugin (Hodgins 2016) and spec written by Tommy Hodgins (published 4. Feb 2017) show how to create element queries in CSS. In this technique, a developer could create a truly responsive HTML element by defining media queries in the scope of an individual DOM-element, so that responsiveness could be achieved without JavaScript. It is safe to guess that this specification is going to be added in web standards at some point in time, but until then we are stuck with current implementations.

Element queries can solve the responsiveness of system in a single widget scale, but even with them we are still stuck with how to implement responsiveness of dashboard layout with user input (i.e. editor). For that current implementation is sufficient, as it provides a drag and drop interface with more in-depth properties for tweaking. Adding this kind of a third party

developed library as a key dependency to the project of this scale would not be such a wise move, so we discard these possibilities.

In the future, responsive web design should be researched more using element queries in systems like View. When CSS standard supports element queries, they could provide a more optimised way of handling layout changes. This should happen when the product can be released with these features on all supported browsers.

A state of huge systems made in JavaScript is a turning point, as late developments have added needed functionalities and performance to JavaScript to create and maintain functional industrial applications. Quirks of JavaScript have made it hard to work with at times, and perhaps not as easy to learn, which is its reputation, but in exchange, it gives developers the possibility to create ubiquitous applications in an agile way (Park et al. 2015). JavaScript has become a “native” language of web development, and in the rise of Internet of Things (IoT) web as a platform is becoming essential to master if developers want to fully take advantage of it.

REFERENCES

Balagas-Fernandez, Florence, Jenny Forrai, and Heinrich Hussmann. 2009. "Evaluation of User Interface Design and Input Methods for Applications on Mobile Touch Screen Devices." In *IFIP Conference on Human-Computer Interaction*, 243–246. Springer. http://link.springer.com/chapter/10.1007/978-3-642-03655-2_30.

Balatsoukas, Panos, Richard Williams, Colin Davies, John Ainsworth, and Iain Buchan. 2015. "User Interface Requirements for Web-Based Integrated Care Pathways: Evidence from the Evaluation of an Online Care Pathway Investigation Tool." *Journal of Medical Systems* 39 (11): 183. doi:10.1007/s10916-015-0357-5.

Beaufort, François. 2015. "Interact with Bluetooth Devices on the Web | Web." *Google Developers*. July. <https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web>.

Brooke, John, and others. 1996. "SUS-A Quick and Dirty Usability Scale." *Usability Evaluation in Industry* 189 (194): 4–7.

"Can I Use Flexbox?" 2016. *Can I Use*. June 8. <http://caniuse.com/#search=flexbox>.

Clark, Josh. 2015. *Designing for Touch. A Book Apart*. <https://abookapart.com/products/designing-for-touch>.

Coyier, Chris. 2013. "Using SVG." *CSS-Tricks*. March 5. <https://css-tricks.com/using-svg/>.

Dayarathna, Miyuru, Anusha Withana, and Kazunori Sugiura. 2011. "Infoshare : Design and Implementation of Scalable Multimedia Signage Architecture for Wireless Ubiquitous Environments." *Wireless Personal Communications* 60 (1): 3–27. doi:10.1007/s11277-011-0256-0.

Desruelle, Heiko, Simon Isenberg, Andreas Botsikas, Paolo Vergori, and Frank Gielen. 2016. "Accessible User Interface Support for Multi-Device Ubiquitous Applications:

Architectural Modifiability Considerations.” *Universal Access in the Information Society* 15 (1): 5–19. doi:10.1007/s10209-014-0373-0.

Dolan, James G., Peter J. Veazie, and Ann J. Russ. 2013. “Development and Initial Evaluation of a Treatment Decision Dashboard.” *BMC Medical Informatics and Decision Making* 13 (1): 51. doi:10.1186/1472-6947-13-51.

Frain, Ben. 2014. “CSS Performance Test: Flexbox v CSS Table – Fight!” *BenFrain*. April 1. <https://benfrain.com/css-performance-test-flexbox-v-css-table-fight/>.

Gesing, S., M. Atkinson, R. Filgueira, I. Taylor, A. Jones, V. Stankovski, C. S. Liew, A. Spinuso, G. Terstyanszky, and P. Kacsuk. 2014. “Workflows in a Dashboard: A New Generation of Usability.” In *2014 9th Workshop on Workflows in Support of Large-Scale Science*, 82–93. doi:10.1109/WORKS.2014.6.

Gordon, Paul MK, and Christoph W. Sensen. 2007. “Seahawk: Moving beyond HTML in Web-Based Bioinformatics Analysis.” *BMC Bioinformatics* 8 (1): 208. doi:10.1186/1471-2105-8-208.

Guo, Philip J. 2015. “Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming.” In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, 599–608. UIST ’15. New York, NY, USA: ACM. doi:10.1145/2807442.2807469.

Gustafson, Aaron. 2015. *Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement*. 2 edition. San Francisco, CA: New Riders.

“Helping Users Easily Access Content on Mobile.” 2016. *Official Google Webmaster Central Blog*. Accessed September 21. <https://webmasters.googleblog.com/2016/08/helping-users-easily-access-content-on.html>.

Hespanhol, Luke, Martin Tomitsch, Kazjon Grace, Anthony Collins, and Judy Kay. 2012. “Investigating Intuitiveness and Effectiveness of Gestures for Free Spatial Interaction with

Large Displays.” In *Proceedings of the 2012 International Symposium on Pervasive Displays*, 6. ACM. <http://dl.acm.org/citation.cfm?id=2307804>.

Hevner, Alan, and Samir Chatterjee. 2010. *Design Research in Information Systems*. Vol. 22. Integrated Series in Information Systems. Boston, MA: Springer US. doi:10.1007/978-1-4419-5653-8.

Hevner, Alan R. 2007. “A Three Cycle View of Design Science Research.” *Scandinavian Journal of Information Systems* 19 (2): 4.

Hodgins, Tommy. 2016. “CSS Element Queries Repository.” *Github*. December 6. <https://github.com/tomhodgins/element-queries-spec>.

2017. “CSS Element Queries [Level 1].” February 4. <https://tomhodgins.github.io/element-queries-spec/element-queries.html>.

“HTML Responsive Web Design.” 2017. *w3school*. February 13. https://www.w3schools.com/html/html_responsive.asp.

Huang, Kevin, Patrick J. Sparto, Sara Kiesler, Asim Smailagic, Jennifer Mankoff, and Dan Siewiorek. 2014. “A Technology Probe of Wearable In-Home Computer-Assisted Physical Therapy.” In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, 2541–2550. CHI '14. New York, NY, USA: ACM. doi:10.1145/2556288.2557416.

Jiang, Wei, Meng Zhang, Bin Zhou, Yujian Jiang, and Yingwei Zhang. 2014. “Responsive Web Design Mode and Application.” In *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, 1303–6. doi:10.1109/WARTIA.2014.6976522.

Kalbach, Jim. 2012. “The First Responsive Design Website: Audi (circa 2002).” *EXPERIENCING INFORMATION*. July 22. <https://experiencinginformation.com/2012/07/22/the-first-responsive-design-website-audi-circa-2002/>.

Karagiannidis, L., M. Vrettopoulos, A. Amditis, E. Makri, and N. Gkonos. 2016. "A CPS-Enabled Architecture for Sewer Mining Systems." In *2016 International Workshop on Cyber-Physical Systems for Smart Water Networks (CySWater)*, 1–6. doi:10.1109/CySWater.2016.7469056.

Katajisto, L. 2015. "Creating Support Content for Responsive Websites at Microsoft Mobile." In *2015 IEEE International Professional Communication Conference (IPCC)*, 1–4. doi:10.1109/IPCC.2015.7235787.

Khan, Mohammed Khalid, Muhammad Sohail, Muhammad Aamir, B. S. Chowdhry, and Syed Irfan Hyder. 2014. "Web Support System for Business Intelligence in Small and Medium Enterprises." *Wireless Personal Communications* 76 (3): 535–48. doi:10.1007/s11277-014-1723-1.

Kim, Bohyun. 2013. "Chapter 4: Responsive Web Design, Discoverability, and Mobile Challenge." *Library Technology Reports* 49 (6): 29–39.

Kittivaraporn, J., J. Chokdeeanan, T. Yaophrukchai, and T. T. Sunetnanta. 2014. "HRM Portal: Human Resource Management Portal." In *Student Project Conference (ICT-ISPC), 2014 Third ICT International*, 183–86. doi:10.1109/ICT-ISPC.2014.6923246.

Knutas, Antti, Arash Hajikhani, Juho Salminen, Jouni Ikonen, and Jari Porras. 2015. "Cloud-Based Bibliometric Analysis Service for Systematic Mapping Studies." In , 184–91. ACM Press. doi:10.1145/2812428.2812442.

Lee, J., I. Lee, I. Kwon, H. Yun, J. Lee, M. Jung, and H. Kim. 2015. "Responsive Web Design According to the Resolution." In *2015 8th International Conference on U- and E-Service, Science and Technology (UNESST)*, 1–5. doi:10.1109/UNESST.2015.11.

Lestari, Devita Mira, Dadan Hardianto, and Achmad Nizar Hidayanto. 2014. "Analysis of User Experience Quality on Responsive Web Design from Its Informative Perspective." *International Journal of Software Engineering and Its Applications* 8 (5): 53–62.

- Li, Zhongli, Shiai Zhu, Huiwen Hong, Yuanyuan Li, and Abdulmotaleb El Saddik. 2016. "City Digital Pulse: A Cloud Based Heterogeneous Data Analysis Platform." *Multimedia Tools and Applications*, November, 1–24. doi:10.1007/s11042-016-4038-2.
- López, A. J. I., I. L. Zorrozua, E. S. Ruiz, M. Rodríguez-Artacho, and M. C. Gil. 2016. "Design and Development of a Responsive Web Application Based on Scaffolding Learning." In *2016 IEEE Global Engineering Education Conference (EDUCON)*, 308–13. doi:10.1109/EDUCON.2016.7474572.
- Mahendrawathi, E., D. Pranantha, and Johansyah Dwi Utomo. 2010. "Development of Dashboard for Hospital Logistics Management." In *2010 IEEE Conference on Open Systems (ICOS 2010)*, 86–90. doi:10.1109/ICOS.2010.5720069.
- Majid, E. S. A., N. Kamaruddin, and Z. Mansor. 2015. "Adaptation of Usability Principles in Responsive Web Design Technique for E-Commerce Development." In *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, 726–29. doi:10.1109/ICEEI.2015.7352593.
- Marcotte, Ethan. 2014. *Responsive Web Design*. 2nd Edition. A Book Apart. <http://alistapart.com/article/responsive-web-design>.
- Marin, Ignacio, Francisco Ortin, German Pedrosa, and Javier Rodriguez. 2015. "Generating Native User Interfaces for Multiple Devices by Means of Model Transformation." *Frontiers of Information Technology & Electronic Engineering* 16 (12): 995–1017. doi:10.1631/FITEE.1500083.
- Mathe, Z., A. Casajus Ramo, F. Stagni, and L. Tomassetti. 2015. "Evaluation of NoSQL Databases for DIRAC Monitoring and beyond." In *21st International Conference on Computing in High Energy and Nuclear Physics (chep2015), Parts 1-9*, 664:42036. Bristol: Iop Publishing Ltd.

McAllister, Neil. 2015. "Google Hits TurboFan Button on Chrome's JavaScript Engine." *The Register.co.uk*. July 7.

https://www.theregister.co.uk/2015/07/07/google_turbofan_javascript_compiler/.

McKeon, Matt. 2009. "Harnessing the Information Ecosystem with Wiki-Based Visualization Dashboards." *IEEE Transactions on Visualization and Computer Graphics* 15 (6): 1081–1088. doi:10.1109/TVCG.2009.148.

Moghimi, Fatemeh Hoda, Richard De Steiger, Johnathan Schaffer, and Nilmini Wickramasinghe. 2013. "The Benefits of Adopting E-Performance Management Techniques and Strategies to Facilitate Superior Healthcare Delivery: The Proffering of a Conceptual Framework for the Context of Hip and Knee Arthroplasty." *Health and Technology* 3 (3): 237–47. doi:10.1007/s12553-013-0057-4.

Mohorovičić, S. 2013. "Implementing Responsive Web Design for Enhanced Web Presence." In *2013 36th International Convention on Information Communication Technology Electronics Microelectronics (MIPRO)*, 1206–10.

Olson, Derek. 2013. "Choosing between Responsive Web Design and a Separate Mobile Site to Improve Mobile Visitors' Experience." *Foraker.com*. April 17. <https://www.foraker.com/blog/choosing-between-responsive-web-design-and-a-separate-mobile-site-to-improve-mobile-visitors-experience>.

Otsuka, S., K. Kato, and V. Klyuev. 2015. "Academic Laboratory Website Development." In *2015 17th International Conference on Advanced Communication Technology (ICACT)*, 791–94. doi:10.1109/ICACT.2015.7224903.

Palpanas, Themis, Pawan Chowdhary, George Mihaila, and Florian Pinel. 2007. "Integrated Model-Driven Dashboard Development." *Information Systems Frontiers* 9 (2–3): 195–208. doi:10.1007/s10796-007-9032-9.

Park, Thomas H., Brian Dorn, and Andrea Forte. 2015. "An Analysis of HTML and CSS Syntax Errors in a Web Development Course." *Trans. Comput. Educ.* 15 (1): 4:1–4:21. doi:10.1145/2700514.

Pavlov, Vladimir, Sönke Knoch, and Matthieu Deru. 2015. "CEPBoard: Collaborative Electronic Performance Board and Editor for Production Environments in Industry 4.0." In *Proceedings of the 4th International Symposium on Pervasive Displays*, 239–240. PerDis '15. New York, NY, USA: ACM. doi:10.1145/2757710.2776801.

Peppers, Ken, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. 2007. "A Design Science Research Methodology for Information Systems Research." *Journal of Management Information Systems* 24 (3): 45–77.

Perakakis, E., and G. Ghinea. 2015. "Responsive Web Design for the Internet Connected TV: The Answer to More Smart TV Content?" In *2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 38–42. doi:10.1109/ICCE-Berlin.2015.7391286.

Petersen, Kai, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. "Systematic Mapping Studies in Software Engineering." In *12th International Conference on Evaluation and Assessment in Software Engineering*. <http://www.diva-portal.org/smash/record.jsf?pid=diva2:836140>.

"Responsive Web Design Introduction." 2015. *w3school*. May 18. https://www.w3schools.com/css/css_rwd_intro.asp.

Rimal, Bhaskar Prasad, Admela Jukan, Dimitrios Katsaros, and Yves Goeleven. 2011. "Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach." *Journal of Grid Computing* 9 (1): 3–26. doi:10.1007/s10723-010-9171-y.

Ruxton, Graeme D., and Markus Neuhäuser. 2010. "When Should We Use One-Tailed Hypothesis Testing?" *Methods in Ecology and Evolution* 1 (2): 114–17. doi:10.1111/j.2041-210X.2010.00014.x.

Scharl, A., A. Hubmann-Haidvogel, M. Sabou, A. Weichselbraun, and H. P. Lang. 2013. "From Web Intelligence to Knowledge Co-Creation: A Platform for Analyzing and Supporting Stakeholder Communication." *IEEE Internet Computing* 17 (5): 21–29. doi:10.1109/MIC.2013.59.

Scott Jehl. 2014. *Responsible Responsive Design*. A Book Apart. <https://abookapart.com/products/responsible-responsive-design>.

Simon, Herbert A. 1996. *The Sciences of the Artificial*. MIT press. https://www.google.com/books?hl=fi&lr=&id=k5Sr0nFw7psC&oi=fnd&pg=PR9&dq=The+Sciences+of+Artificial&ots=-v4IkFDNHu&sig=6iLYr7SUo6CCcx-5uxfGymbd_7M.

Titcomb, James. 2017. "Tablet Sales to Fall for Third Successive Year as PC Market Stabilises." *The Telegraph*, November 1, sec. 2017. <http://www.telegraph.co.uk/technology/2017/01/11/tablet-sales-fall-third-successive-year-pc-market-stabilises/>.

Voutilainen, J. P., J. Salonen, and T. Mikkonen. 2015. "On the Design of a Responsive User Interface for a Multi-Device Web Service." In *2015 2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 60–63. doi:10.1109/MobileSoft.2015.16.

Walsh, T. A., P. McMinn, and G. M. Kapfhammer. 2015. "Automatic Detection of Potential Layout Faults Following Changes to Responsive Web Pages (N)." In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 709–14. doi:10.1109/ASE.2015.31.

Wieringa, Roel, Neil Maiden, Nancy Mead, and Colette Rolland. 2005. "Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and a Discussion." *Requir. Eng.* 11 (1): 102–107. doi:10.1007/s00766-005-0021-6.

Wohlin, Claes, Per Runeson, Martin Höst, Magnus C. Ohlson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. 1st ed. Springer-Verlag Berlin Heidelberg.

Wroblewski, Luke. 2011. *Mobile First*. 1st ed. Book Apart.
<https://abookapart.com/products/mobile-first>.

Appendix 1. Table of implementation constraints

ID	Constraint	Description
C-1	Requires Internet Connection	The application requires internet access to the data source to get data and authenticate the user.
C-2	IE11 Support	Supports IE11 version, which is widely used in windows 7 based computers within system
C-3	iOS 7 Safari Support	Support for iPad
C-4	Chrome 51 on Android	
C-5	Support all modern browsers	Support for latest versions of Microsoft Edge, Mozilla Firefox and Google Chrome.
C-6	Touchscreen devices	The system can be used by touch only device.
C-7	Desktop and Laptop devices	The system can be used by mouse and keyboard.
C-8	Runtime usable in mobile device	Runtime of the dashboard must be able to made usable in common mobile device.
C-9	WebSocket connection	Client window creates one web socket connection per data source.
C-10	Reflectable	Wherever applicable, all new functionalities must use already existing reflection system.
C-11	Serializable	Wherever applicable, new functionalities must use already existing serialisation system.

Appendix 2. Table of non-functional requirements for implementation

ID	Requirement	Description
NF-1	Handle error conditions	New functionalities must handle their own error conditions
NF-2	Optimized	No new feature must not add unnecessary overhead
NF-3	Maintainable	Final outcome must be maintainable and well documented
NF-4	Optimized communication	New functionalities don't add unnecessary communication to server
NF-5	Legacy support	Adding new functionality should not break old dashboards
NF-6	Reuse old codebase	New functionalities must use as much old codebase as possible
NF-7	Follow Code Style Guide	All new code follow company code style guide
NF-8	Resource Sharing	Resources should be shared between all open dashboards and editor

Appendix 3. Table of functional requirements for implementation

ID	Requirement	Description	Area
F-1	Adjustive font scaling	Ability to scale font relative to parent dashboard	Fonts
F-2	Adjustive font restriction	Ability to select whether font scales on parent dashboard	Fonts
F-3	Font editor	Better visual editor for fonts	Fonts
F-4	Widget Inheritance	Ability for widget container to host another widget container	Scaling Motor
F-5	Inheritance editing tools	Easy editing of children-parent relationship	Scaling Motor
F-6	Widget hierarchy tree	Create widget that can visualise hierarchy of given dashboard	Layout
F-7	Layout Object	Abstract object that works as a layer between scaling motor and user, and which layouts inherit from	Scaling Motor
F-8	General scalable container	Container type that can scale	Layout
F-9	Rubber Layout	Make old anchor based layout its own layout object	Layout
F-10	Responsive Layout	Create separate layout that will mimic the responsive layout	Layout
F-11	Common Input	Editor must be usable with both mouse and touch input	Input

Appendix 3. (continues)

F-12	Right click emulation for touch	Create a way to emulate right click for touch
F-13	Scalable images	Optimised scalable images
F-14	Split window sizing handle for touch	Split-line handle that can be dragged easier with finger

Appendix 4. Usability Evaluation survey

Value	Description
1	I strongly disagree
2	I disagree
3	I cannot say
4	I agree
5	I strongly agree






	1	2	3	4	5
1. It is effective to create dashboards with the editor.					
2. Dashboards produced with editor work well.					
3. All the tools inside editor seem to work as I expect them to.					
4. The system reported and recovered from error states.					
5. Provided widgets seem to scale well with the dashboard.					
6. There is enough information (for example description texts on properties) on how to use tools inside the editor.					
7. Tools, their amount, and placement on editor did not disrupt my workflow.					
8. Complexity and learning curve of the tools are not high.					
9. Tools are easy to find in the editor.					
10. There are enough tools for me to create dashboards without scripting.					
11. Dashboards that I create work well on both mobile and desktop.					
12. This version (both editor and produced dashboards) works well with a touch interface.					
13. I think this version implements the principles of RWD in its interface.					
14. I like the interface of the system, I find it pleasant and modern.					
15. Overall, I feel productive when working with the system.					

Appendix 5. Cognitive walkthrough instructions

When you are answering first part of this survey and testing 1.2, open browser and navigate to *[link is censored in published work]*

When you are answering to second part of this survey and testing 1.4, open a browser and navigate to *[link is censored in published work]* (in office network)


PART 1

- 1) From Tree-window on left, open data-source dropdown and select `$/dashboard/data/displays`
- 2) Create a new dashboard. On 1.2 right click an empty area and select `new/dashboard`.
On 1.4, you can either do that or select  a “new node” -icon next to data source selection
- 3) With your new dashboard opened, go to edit mode by pressing  edit-button
- 4) In edit mode, find  Widgets-icon and open Widgets list.
- 5) In this step, we are creating a dashboard in **Error! Reference source not found.**
 - a) From Widgets selection, drag widgets and place them like in **Figure 1**. Create three labels and resize them to create the same formation.
 - b) Change anchors to the following formation. You can change anchors by tapping anchor icon when the widget is selected in edit mode.
 - i) Header label (the one on top) should be pinned () on left, right and top while floating on the bottom.
 - ii) Both left and right labels below header are pinned on the bottom, left and right, and hooked() to header label on top. You can anchor widget by first changing the anchor type to link/hook and then dragging pointer from anchor to widget you are aiming to hook at.
- 6) You can change label texts and colours to whatever you like on properties menu.
- 7) You can now go ahead and resize the browser window, to see how the dashboard acts in different aspect ration



When you are doing this exercise with 1.4, proceed to **PART 2**, in which we apply some new features to dashboard to make it more responsive.

Appendix 5 (continues)

PART 2

1. In the  edit mode, select one of the labels and go to font properties by selecting the button on the right of the compact Font-property editor in property grid:



2. Set label fonts to be responsive
 - a. On the left and right Label, we can use “Adjust”-face-type, which works on labels that are pinned and/or hooked on each side. By selecting face-type to adjust, label font size will be relative to dashboard size
 - b. On the header, we have one side floating. For that, easiest way to get a responsive text is Scaled font face type. Scaled font always takes full size inside its container. To limit headers size, set max font size property to be something appropriate.
3. Enable responsive layout, which mimics fluid grid
 - a. Still, in the  edit-mode, unselect all widgets by pressing grey area outside the dashboard, and from property grid, change Layout to be “Responsive Layout”.
 - b. Go and select label on top. Scroll to the bottom of the property-grid and open editor of “Widget Container” for this widget. Inside widget container editor, navigate to editor of the Rubber-Layout
 - i. **Note:** Layout of this widget drives positions of the widgets that inherit this widget’s scope. This widget acts responsively, when its parent’s layout is responsive (which in this case is dashboard)
 - c. In this property’s editor, find “Narrow Flow Size”. This is the setting that drives the size of the container when the container is in the fluid grid. Set it to 8:1
 - i. **Note:** This means the ratio between x and y. In narrow mode (like in mobile screen), the widget is scaled 8 times in width compared to 1 times in y. In wide mode, width is always 100% of the container size.
4. Exit  -edit mode and inspect how the layout of the dashboard changes when the browser window is resized.

Appendix 5 (continues)

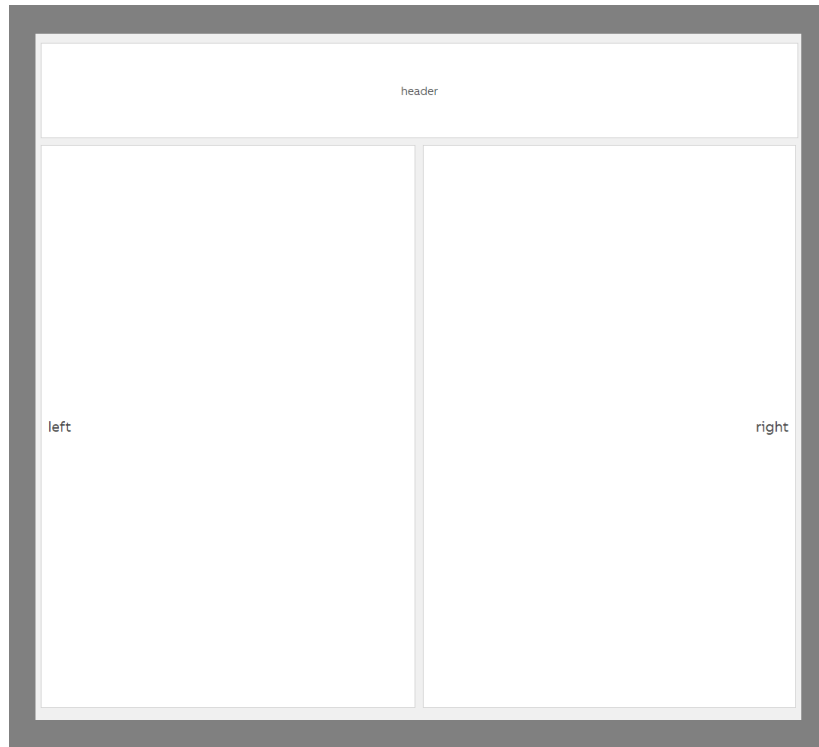


Figure 1 Result of this test

Appendix 6. Evaluation Matrix on View 1.2

Usability Characteristics	Quality metrics	Evaluator 1	Evaluator 2	Evaluator 3	Evaluator 4	Evaluator 5	Evaluator 6
Effectiveness	It is effective to create dashboards with the editor.	2	4	3	4	4	3
	Dashboards produced with editor work well.	2	4	3	4	2	4
	All the tools inside editor seem to work as I expect them to.	2	2	4	2	2	2
	The system reports and recovers from error states.	1	3	4	3	3	3
	Provided widgets seem to scale well with the dashboard.	2	4	2	4	2	4
Efficiency	There is enough information (for example description texts on properties) on how to use tools inside the editor.	2	5	1	1	2	2
	Tools, their amount, and placement on editor did not disrupt my workflow.	3	2	2	2	2	2
	Complexity and learning curve of the tools are not high.	2	5	1	4	2	4
	Tools are easy to find in editor.	2	4	3	4	3	2
	There are enough tools for me to create dashboards without scripting.	4	4	3	1	4	2
Satisfaction	Dashboards that I create work well on both mobile and desktop.	2	3	2	2	2	3
	This version (both editor and produced dashboards) works well with a touch interface.	2	2	2	1	1	1
	I think this version implements the principles of RWD in its interface.	2	4	3	4	5	3
	I like the interface of the system, I find it pleasant and modern.	2	4	3	4	4	4
	Overall, I feel productive when working with the system.	2	4	2	2	4	2

Appendix 7. Evaluation Matrix on View 1.4

Usability Characteristic	Quality metrics	Evaluator 1	Evaluator 2	Evaluator 3	Evaluator 4	Evaluator 5	Evaluator 6
Effectiveness	It is effective to create dashboards with the editor.	3	4	4	4	2	4
	Dashboards produced with editor work well.	3	4	4	4	2	3
	All the tools inside editor seem to work as I expect them to.	3	2	4	2	2	3
	The system reports and recovers from error states.	3	3	4	3	3	3
	Provided widgets seem to scale well with the dashboard.	3	5	3	5	2	4
Efficiency	There is enough information (for example description texts on properties) on how to use tools inside the editor.	2	5	1	2	1	2
	Tools, their amount, and placement on editor did not disrupt my workflow.	2	4	2	2	2	3
	Complexity and learning curve of the tools are not high.	2	5	1	4	2	2
	Tools are easy to find in editor.	2	5	4	4	2	4
	There are enough tools for me to create dashboards without scripting.	4	4	3	1	4	2
Satisfaction	Dashboards that I create work well on both mobile and desktop.	3	3	2	4	2	3
	This version (both editor and produced dashboards) works well with a touch interface.	4	2	4	4	4	4
	I think this version implements the principles of RWD in its interface.	4	5	4	5	5	4
	I like the interface of the system, I find it pleasant and modern.	3	5	4	4	4	4
	Overall, I feel productive when working with the system.	3	4	3	2	2	2