

Implementing presentation in Web GIS application with emphasis on user experience

Cecilie Tunsli Lorentzen Andersen

Master's thesis in Software Engineering at
Department of Computing, Mathematics and Physics,
Bergen University College
Department of Informatics,
University of Bergen

June 2015



Acknowledgements

I would like to thank my supervisor Torill Hamre for your cooperation with guidance and constructive feedback along the work on this thesis. Thank you for reading and commenting all the drafts during this process.

Then I would like to thank the test users, system developer Aleksander Vines, leading scientist Lasse Petterson and research scientist Kjetil Lygre. This work could not have been done without your participation and contribution.

I would also like to thank the open source community for the important work in providing the software technology including the documentation, help and guidance online.

Contents

Acknowledgements	ii
Contents	iii
List of Figures	vii
List of Tables	ix
Abbreviations	xi
1 Introduction	1
1.1 Background	1
1.1.1 Geographic Information System domain	1
1.1.2 Oceanographic DataBase (ODB)	2
1.1.3 Web GIS for marine research	2
1.2 Goals	3
1.2.1 Sub-goals	3
1.3 Research question	4
1.4 Thesis structure and outline	4
2 Background	7
2.1 Geographic technology	7
2.1.1 Geographic data	7
2.1.2 Geographic information system on the Web	7
2.2 The Web GIS	8
2.2.1 Geographic data store	8
2.2.2 Geographic data accessor	9
2.2.3 Web GIS client	9
2.3 Web GIS technologies	10
2.4 Usability engineering	11
2.4.1 Usability engineering for GIS	11
3 Problem analysis and requirements	13
3.1 Problem description	13
3.1.1 Problem decomposition	13
3.1.2 Problem definition	15
3.2 Requirements	15
3.2.1 Web GIS client	15

3.2.2	Scope of Web GIS client	17
3.3	Methodology and considerations	18
3.3.1	Usability methodology	18
3.3.2	Development methodology	18
3.3.3	Development considerations	19
4	Software stack	21
4.1	Software stack	21
4.2	GIS standards	21
4.2.1	Open Geospatial Consortium (OGC)	22
4.3	Geographic data store	22
4.3.1	Data store standards and compliance	22
4.4	Geographic data accessors	23
4.4.1	OGC web services (OWS)	23
4.4.2	Web Map Service	23
4.4.3	Web Feature Service	25
4.4.4	Other relevant OGC concepts	25
4.5	The Web GIS client application	26
4.5.1	Selection criteria	26
4.5.2	Client technologies	27
4.5.3	JavaScript unit testing framework	29
4.5.4	Web mapping technologies	31
4.5.5	Additional technologies and libraries	32
4.5.6	JavaScript IDE	33
4.5.7	Build tool and code generators	33
4.5.8	The complete software stack	34
5	Usability engineering	35
5.1	Usability methodology and standards	35
5.1.1	ISO/IEC 25010 standard for system quality and measurement	36
5.1.2	Nielsen heuristics	36
5.1.3	5Es usability attributes	37
5.2	User-centred design process	39
5.2.1	Domain research	40
5.2.2	Conceptual development	41
5.3	Prototyping and implementation	42
5.4	Usability study and evaluation	43
5.4.1	Usability testing and evaluation methods	43
5.4.2	Usability testers	46
5.4.3	Planning usability testing	47
5.4.4	Iterative usability testing	47
5.4.5	Usability evaluation	50
6	Data store and data accessor configuration	51
6.1	Geographic data store	51
6.1.1	Create and populate PostGIS database	52
1.	Create new geometry column	54

2. Populate lon/lat data	54
6.1.2 Complete geographic data store	55
6.2 Data accessor	55
6.2.1 Import data to GeoServer	55
6.2.2 Providing geographic data through GeoServer	57
6.2.3 Providing non geographic data through GeoServer	57
7 Design and implementation of prototype	61
7.1 Architecture	61
7.2 User interface design	63
7.3 AngularJS components	64
7.3.1 View	64
7.3.2 Model	65
7.3.3 Directive	65
7.3.4 Controller	66
7.3.5 Service	66
7.4 AngularJS principles and design patterns	66
7.4.1 AngularJS design patterns	67
7.4.2 AngularJS Service	68
7.5 Front-end frameworks and libraries	68
7.5.1 AngularJS	68
7.5.2 Openlayers	69
7.6 Development challenges	70
7.6.1 OpenLayers 3	70
7.6.2 Same origin Policy (SOP)	70
7.7 Client prototype components	71
7.7.1 Directives	71
7.7.2 Controllers	72
7.7.3 Services	73
7.8 Testing and quality assurance	78
7.8.1 W3C standard compliance	78
7.8.2 Unit testing	79
7.9 Prototype development	80
7.9.1 Rewriting Web GIS application	80
7.10 Front-end development environment	81
7.10.1 Continuous Integration	82
8 System demonstration	85
8.1 Web GIS client front page	85
8.2 Display map and its controllers	85
8.3 Query data, search in data	86
8.4 Display query response in map	90
8.5 Display response data in tabular form	90
8.6 Display dataset in chart	90
8.6.1 Display temperature and salinity	92
8.6.2 Display density chart	92
8.6.3 Calculations for density chart	93

8.7	Display temperature contour and contour grid	94
8.7.1	Calculate and display contour layer	95
8.7.2	Calculate and display contour grid	96
8.8	Export functionalities	96
8.8.1	Export dataset	96
8.8.2	Export graph	99
8.9	Other non functional requirements	100
8.9.1	Prevent user from making errors	100
8.9.2	Information and guidance	101
9	Evaluation	105
9.1	Sub-goals evaluation	105
9.1.1	Conduct usability research in GIS and non GIS applications to determine use of usability methods and principles	105
9.1.2	Develop an iterative user-centred design approach to achieve software quality in terms of usability and UX	107
9.1.3	Perform analysis and selections for software stack which best complies to GIS technology and interactive web applications	108
9.1.4	Install, configure and populate GIS backend technologies	108
9.1.5	Develop, design and implement the Web GIS client	109
9.1.6	Evaluate the usability engineering and usability outcome of the new Web GIS client.	111
9.2	Overall objective	114
9.3	Research questions	115
9.4	Evaluation summary	117
10	Discussion and conclusion	119
10.1	Outcome	119
10.2	Further work	120
10.2.1	UE guideline for further work	120
10.2.2	Design, implementation guideline for further development	121
10.3	Lessons learned	123
10.4	Conclusion	124
A	Personas	125
B	Usability test schema	129
C	Filtering in OpenLayers 2 verses OpenLayers 3	135
	Bibliography	139

List of Figures

2.1	The conceptual model of the Web GIS.	9
2.2	Database schema, ER diagram of Web GIS.	10
4.1	Updated conceptual model including client technologies.	34
5.1	Product Quality of ISO/IEC 25010	37
5.2	Quality In Use of ISO/IEC 25010.	38
5.3	5Es usability attributes.	39
5.4	User-centred design process model.	40
5.5	Example of a persona.	42
5.6	Personas for the Web GIS application.	43
5.7	User story aimed for the persona named Keri.	44
5.8	User centred design model.	45
5.9	Usability test procedure.	49
6.1	Import data into GeoServer.	56
6.2	SQL View in Geoserver.	59
6.3	SQL View in Geoserver.	60
7.1	Web GIS client architecture.	62
7.2	Interaction between the MVC components.	62
7.3	Client prototype conceptual model.	63
7.4	Overview of interaction between the AngularJS components.	64
7.5	UML class diagram.	71
7.6	Grunt test GUI in WebStorm.	81
7.7	Software technology in the presentation layer	82
8.1	Map controller interface.	86
8.2	Map controller interface with menu.	87
8.3	Sequence diagram of requesting filtered data.	88
8.4	First GUI design of the bounding box controller interface.	89
8.5	Final GUI design bounding box controller.	89
8.6	View data result set in tabular form.	91
8.7	Zooming and data point label in temperature chart.	92
8.8	Density chart for a set of temperature and salinity data.	93
8.9	Sequence diagram illustrating the density calculation procedure.	95
8.10	Early contour menu before design change.	97
8.11	Display contour layer with latest design changes.	98
8.12	Contour layer with yellow data points in the map widget.	99

8.13	Export data result to text (JSON) formate.	100
8.14	Export functionality in data table.	101
8.15	Example of input validation.	102
8.16	Give system information and guidance.	103
9.1	Usability evaluation statistics Iteration I.	114
9.2	Usability evaluation statistics iteration II.	115
9.3	Usability evaluation statistics iteration III.	116
9.4	Usability evaluation statistics, mean rating for user testing.	117
A.1	Example of a persona, Natalia.	126
A.2	Example of a persona, Mikka.	127
B.1	Example of usability introduction schema.	130
B.2	Example of usability introduction schema.	131
B.3	Example of usability test task schema.	132
B.4	Example of usability test task schema.	133
B.5	Example of questionnaires schema.	134

List of Tables

3.1	Functional requirements of the new Web GIS client.	16
3.2	Usability requirements of the new Web GIS client.	17
5.1	ISO 9241 6 principles of UCD process	39
7.1	Development tools used in implementation	82
9.1	Questionnaire asked during test sessions.	113

Abbreviations

EWG	The extended Web GIS
GIS	Geographic Information System
GUI	Graphic User Interface
IWG	The initial Web GIS
ISO	International Organization for Standardization
ODB	Oceanographic DataBase
OGC	Open Geospatial Consortium
OWS	OGC Web Service
SFA	Simple Feature Access
SDLC	Software Development Life Cycle
SRS	Spatial Reference System
UCD	User-Centred Design
UE	Usability Engineering
UX	User Experience
WFS	Web Feature Service
WMS	Web Map Service
XML	eXtensible Markup Language

Chapter 1

Introduction

1.1 Background

1.1.1 Geographic Information System domain

Geographic data identifies a geographical location on Earth and are used within a wide variety of information systems e.g for environmental monitoring. A Geographic Information System (GIS) is a desktop application where the user can store, view, manipulate and analyse geographic data. GIS is a large domain in the academic science of Geo informatics. Scientists use a desktop application, the customized GIS to explore, visualize and analyse geographic data.

By the evolution of the Internet, Web GIS applications has been introduced as a naturally expansion of GIS. The transformation has made GIS available for everyone. A Web GIS is a GIS application available in a Web browser using data retrieved from a geographic web server on the Internet. Web GIS applications are used for e.g information service for government purposes, customer service in business and environmental monitoring such as this Web GIS.

The evolutionary Internet, where web pages have turned into web services which creates a platform for Web GIS to grow. The Web GIS application can provide the functionality of GIS in addition to adapt modern web principles for intuitive, dynamic and accessible GIS applications. The main functionality is the same but from a software developers point of view, new issues arise when it comes to fulfilling requirements, performance and

security. With all feature complexity in mind Web GIS has become a full-fledged GIS application for Internet users.

Increased application accessibility provide more users from new types of user groups. Users with different experience level with GIS domain applications and usability requirements become more important.

1.1.2 Oceanographic DataBase (ODB)

Nansen Environmental and Remote Sensing Center (NERSC) has a GIS called Oceanographic DataBase (ODB). The ODB GIS was developed [1] as part of a research project. The dataset in the ODB is extensive it contains approx 755 000 stations, 4.6 million temperature and 2.36 million salinity measurements between the 16th of August 1855 to 29th of September 2010 [1].

The ODB application does not cover the expectations of the users. It is a single-user application locked to proprietary software. NERSC wanted to update the ODB. During a master thesis from 2013 [2] a Web GIS application was developed to cover the lack of functionality and mobility of the ODB.

1.1.3 Web GIS for marine research

The Web GIS developed in 2013 [2] replaced the desktop ODB GIS and implemented functionality for a subset of the data in the ODB. The software infrastructure was redone following best practices and standards of Open Geospatial Consortium (OGC). The Web GIS application is a prototype; it has a front-end client available through a web browser and covers the basic needs of querying and analysing data from the ODB.

In this thesis the client side of the Web GIS is extended. The Web GIS is rendering a number of plots, maps and charts to visualize a subset of the ODB dataset. The assignment is to develop a user interface where the users can complete their tasks within reasonable expectation. This work should in term of Web GIS development benefit experienced Web GIS users, scientists at NERSC as well as other users without any extensive training in how to use a Web GIS application. The new Web GIS client should be user-friendly and have available functionality to meet the demand of the users.

1.2 Goals

The overall goal is to develop a Web GIS client, a prototype designed to let simplicity, usability and user experience (UX) [3] be equally important as the functionality of the client application. A challenge developing GIS applications is when geographic data is turned into information which often effect the UX [4].

In parallel with developing and implementing the prototype, will research in the domain of developing a user-friendly Web GIS client be conducted. We believe a Web GIS client should be designed by the same principles as world wide web (WWW) applications with emphasis on usability and UX.

The prototype should be built on trusted open source software, which includes good community contribution, support and documentation. This will assure future support of the software and further application maintenance. The client should meet the goals of being testable, maintainable and scalable by following software development principles and best practises.

The client must support the formal requirements of using standardised formats and compliance in GIS technology as discussed in section 4.2. The requirements is derived from the old Web GIS software stack. Functional and non-functional requirements will be further discussed in chapter 3.

1.2.1 Sub-goals

The list of sub-goals underneath form the basis for the overall goal as described above. Every sub-goal is important to achieve for the success of the overall goal.

Sub-goals

- Conduct usability research in GIS and non GIS applications to determine use of usability methods and principles.
- Develop an iterative user-centred design approach to achieve software quality in terms of usability and UX
- Perform analysis and selections for software stack which best complies to GIS technology and interactive web applications

- Install, configure and populate GIS backend technologies
- Develop, design and implement the Web GIS client
- Evaluate the usability engineering and usability outcome of the new Web GIS client

1.3 Research question

We have a Web GIS prototype with a software stack following the standards of OGC and a client serving the end users of the application. The users request data which the client provides by data tables and visualization. The visualization is mainly maps, graph plots or charts visualizing temperature and/or salinity data. The request is fairly simple by a few parameters covering time, bounding box of request and parameter attributes as origin country, name of source and name of vessel. The client has a simple graphic user interface.

A new client will be built using JavaScript libraries. The development of the client will be done by making every effort to meet the demand of the users and by implementing design principles regarding usability and UX.

Based on the background data written in this chapter has several research questions emerged.

- Is it feasible to develop a user-centred Web GIS client by converging user interface design and UX using open source software?
- Is it feasible to develop a user interface providing UX and still offer the functionality of a Web GIS?

1.4 Thesis structure and outline

- Chapter 1: Introduction to the thesis domain.
- Chapter 2: Background information and theory on domain specific technologies.

- Chapter 3: Problem analysis and requirements, including problem decomposition, problem definition, scope of work and methods and methodology for usability and development.
- Chapter 4: Software stack, GIS technology software standards and compliance, Web GIS client technologies analysis and other software technologies used in the new complete software stack.
- Chapter 5: Usability engineering, usability standards and principles, usability methodology and the user-centred design process.
- Chapter 6: Data store and data accessor configuration, including PostGIS and GeoServer installation, configuration and population.
- Chapter 7: Design and implementation of the new Web GIS prototype, implementation details on development environment, implementation challengers, design and architecture of the Web GIS client code.
- Chapter 8: System demonstration, the system functionality is demonstrated with implementation details.
- Chapter 9: Evaluation, sub-goals evaluation for usability and other application requirements, answering research questions and a complete thesis evaluation summary.
- Chapter 10: Discussion and conclusion, discussion on further work and end conclusion.

Chapter 2

Background

This chapter introduces the GIS technology and the conceptual model of the old Web GIS application [2]. Next some web mapping applications are presented to introduce similar GIS applications and in the end some research on usability engineering (UE) is presented to capture the challenge in UE for GIS.

2.1 Geographic technology

2.1.1 Geographic data

Geographic data is also known as spatial data or spatial data. The geographic data identifies a geographical location on planet Earth and are used within a wide variety of information systems on the Internet. Such data are represented as e.g lines, points and raster. Data inside the ODB are represented as points and have x, y variables in addition to a z variable for the depth of the measurement. Geographic data is gathered and saved in geographic databases for the purpose of being manipulated and analysed in GIS.

2.1.2 Geographic information system on the Web

A GIS is a desktop application where the user can store, view, manipulate and analyse geographic data as described in subsection 1.1.1. The complexity of desktop GIS applications is high and requires training and knowledge to manage.

With Internet, GIS has evolved into a powerful technology available to a wide variety of users. User groups other than just domain experts can access Web GIS applications, an increase in such applications are occurring on the Internet. Regular users browsing the Internet for destinations, whether it would be hotels, restaurants or the dentist are browsing Web GIS applications online.

There are many fields of industry that benefit from using Web GIS. For instance applications for government such as public information service like traffic, businesses such as customer service like where to find the nearest bank and Google Maps. Web GIS gathers the geographic data into a map a human can read.

2.2 The Web GIS

As described in subsection 1.1.2, ODB is the predecessor to the Web GIS application. The software stack of the Web GIS system is illustrated in Figure 2.1. The geographic data store provides geographic and non geographic data to the client application through the data assessor. Detailed description is defined in [2]. For further reading on GIS standards and compliance see section 4.2.

2.2.1 Geographic data store

The PostGIS data store is a geographic extension to a PostgreSQL database [5]. The Entity relationship (ER) diagram 2.2 illustrates the database tables inside the PostGIS data store. The entity `station` has parameters describing the measurement provider while `p_salinity` and `p_temperature` contains the data values provided by one instance of the `station` table. These tables are linked by the `absnum` parameter.

The data tables `spatial_ref_sys` and `geometry_columns` are meta data tables containing the geographic components of the database. They meet the compliance of GIS domain standardization and are automatic operated by the geographic data store.

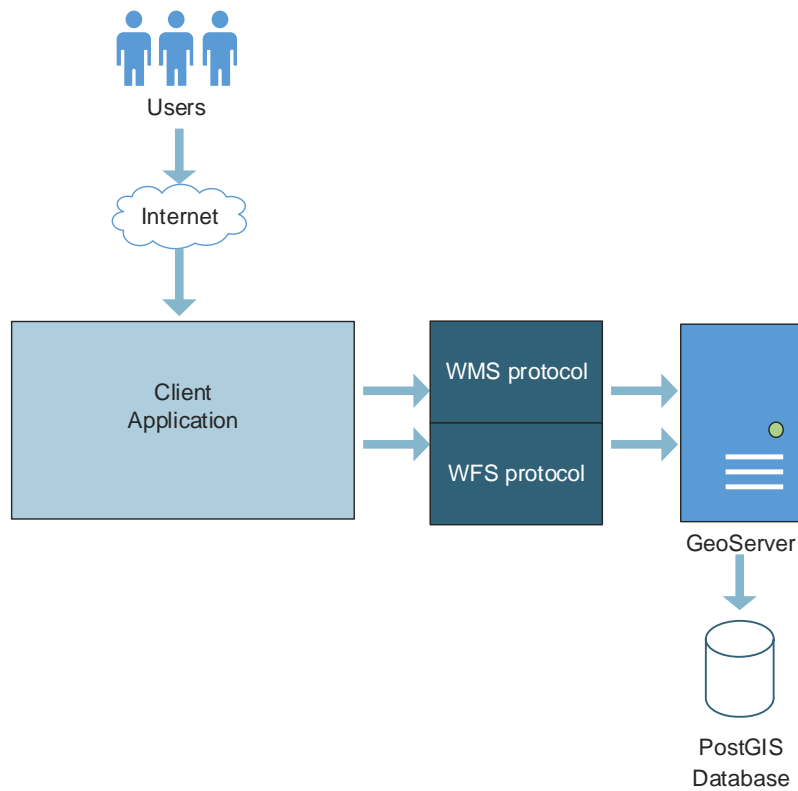


FIGURE 2.1: The conceptual model of the Web GIS.

2.2.2 Geographic data accessor

The Web GIS geographic web service, *Geoserver* [6] has a set of services for managing geographic data. Figure 2.1 illustrates the *Web Map Service* (WMS) and *Web Feature Service* (WFS) protocols for requesting geographic data from the server. GeoServer has native support for PostGIS and communicates through *Simple Feature Access* (SFA). SFA is a standardised interface for geographic data management. See section 4.2 on GIS standards and compliance.

2.2.3 Web GIS client

The Web GIS client is developed with *Java Server Pages* (JSP) as a single page application with asynchronous calls to the geographic data accessor. The client is simple including features for exploring, analysing and visualizing geographic data by a set of parameters. See [2] for more details.

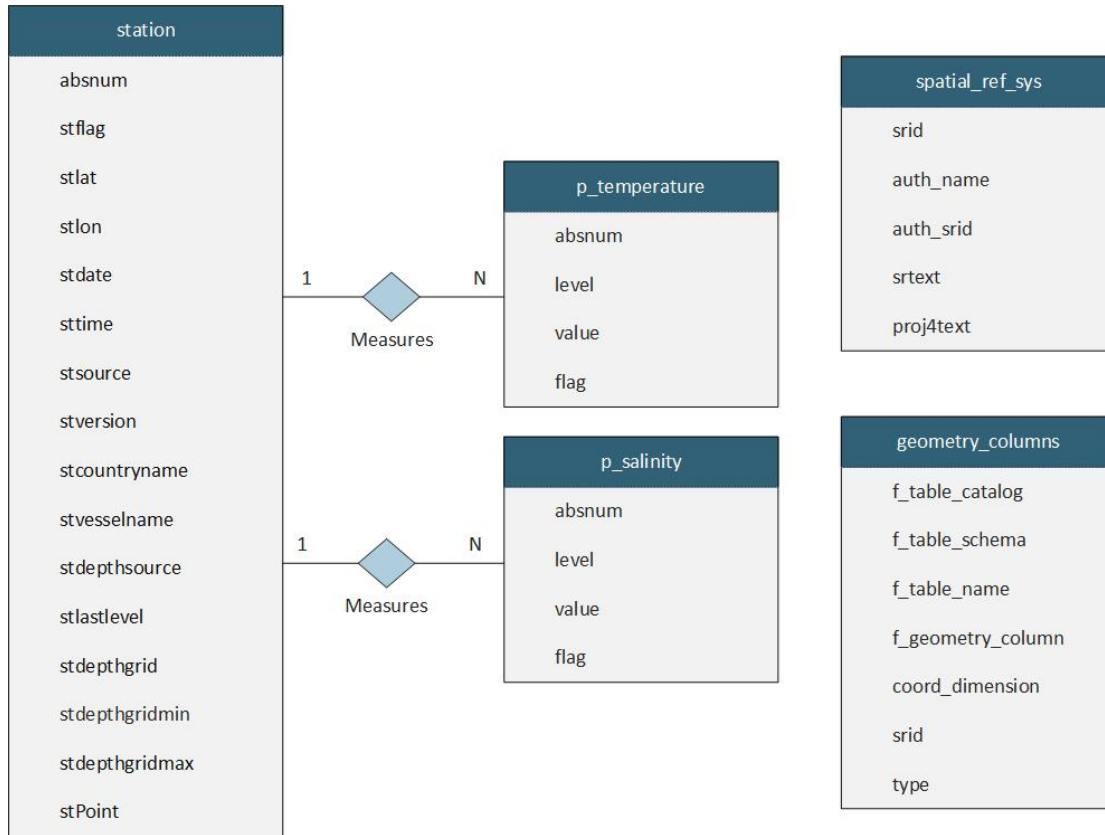


FIGURE 2.2: Database schema, ER diagram of Web GIS.

2.3 Web GIS technologies

The development of Web GIS technologies has expanded the GIS domain. Web-based GIS applications has become more dynamic, interactive and accessible [7] in contrast to single client desktop GIS applications. We think the HTML give GIS applications the opportunity to merge into the convention of developing web applications present on the Web today.

The outcome of the conversion from desktop GIS into the web domain is the capability to interact with distributed multiple and heterogeneous systems and services [7]. GIS applications in the Web domain has evolved into more simplified and user friendly user interface (UI) which meet the demand of a broader set of users than the old GIS desktop applications.

Map.geo.admin [8], Surging seas [9] and Marinexplore [10] are user friendly Web GIS examples with simplified UI. The listed example applications are easy to understand

and scoped to a purpose of use, which we think is a good starting point for developing new web based GIS applications.

Other commercial software have taken the usability concept to a high level. ESRI [11] is continuously improving their UX by employing UX experts, performing usability studies and developing guidelines for usability development [12]. ESRI products have been explored in usability research such as [13] and [14].

2.4 Usability engineering

Usability is a quality attribute which to one extent questions if the system is satisfying the needs of its users. Usability is defined by five attributes, learnability, efficiency, memorability, errors and satisfaction [15]. Each of them can be systemically measured for improvement and evaluation [16]. Usability has been a big domain within product design [17] and web design [18] and still is.

Usability Engineering (UE) is a set of practises on how to perform usability thorough the software development life cycle (SDLC) [16]. UE is a user-centred approach, a process in which techniques and practises are integrated into the SDLC. This process include the users from the beginning of the SDLC to understand their perceptual and cognitive thinking, mental modelling and their tasks in the application [19].

2.4.1 Usability engineering for GIS

Already in 2001 [20] MacEachren and Kraak stated the lack of Human Computer Interaction (HCI) methods for conducting usability studies within interactive visual applications. There are also lack of UE methods within mapping applications [21]. In research within the field of GIS technologies, a list of studies on Usability Engineering for GIS exists, but still no UE knowledge for GIS has yet been stated [13].

There are a lot of individual research of geographical visualization studies ([14],[13], [19]) all of them stating that usability research within the GIS domain are insufficient. Research of UE through the life cycle of development (user requirements, design and evaluation) is rare [22].

To achieve satisfied users we think it is important to have knowledge of the domain, how the geographic data are managed and the needs of the users. This will prevent simple usability issues and misconception of user tasks early in the development.

UE is more important than ever with the increasing development of GIS. The HTML applies to more users demands better usability. We think the challengers in designing user friendly GIS applications requires good UE techniques and methods thriving towards acceptable usability attributes and UX.

GIS related usability practises which can be used during iterative development is investigated during this thesis. The desired solution is to gain knowledge on how to effectively and seamlessly use UE in developing Web GIS client applications. This include assisting in the design of user interaction and the characteristics of geographic data manipulation [13].

The need of establish a user-centred approach for developing geographical visualisation applications is needed [22]. [22] states that the developers of GIS applications have problems including UE techniques into the design process. They do not know how to include UE into GIS domain applications and further research is needed. The challenge is to package a complex application into a user friendly interface.

Chapter 3

Problem analysis and requirements

This chapter describes the overall goals of this thesis. The goals of the thesis are discussed in section [1.2](#) and will be further investigated to describe the problem objective and state a set of requirements based on the sub-goals in subsection [1.2.1](#). At first the problem decomposition is discussed, then the requirements and the scope of the client applications is stated.

3.1 Problem description

3.1.1 Problem decomposition

The problem statement has two assignments. One part is to research how to develop a Web GIS client application with emphasis of the users. The second part is to implement a Web GIS client on top of the Web GIS architecture described in section [2.2](#). The client will be based on the existing functionality, some will be extended and additional functionality will be developed in cooperation with the users. The problem objectives are elaborated underneath.

Web GIS domain research

The elaboration on the research goal is to assemble a set of UE guidelines for Web GIS client development. These guidelines should take into account the nature of web mapping, geospatial visualization, cartography guidelines and other GIS domain specific areas.

The outcome will be a development process describing the client development, and a roadmap for further development.

User-centred design

In developing with a user-centred approach the users must be involved during the design and implementation steps. A user-centred design approach will contain usability principles and best practises involving usability methods and interface standards to meet the needs of the users and work towards good UX.

Develop Web GIS client

Many web applications today is written in JavaScript. JavaScript frameworks which structure JavaScript code into architectural patterns has gain a lot of popularity in web application development. The Web GIS client is focused on the client code because the server is hosted and managed by the geographic data accessor. The new client implementation involves rewriting the old Web GIS client from JSP to a JavaScript web technology. JSP is a elderly server side technology which also can generate HTML pages. The installation, configuration and population of the geographic data store and accessor must be fulfilled before the work on the new Web GIS client can begin.

The new Web GIS client should provide a user interface that novice and expert users experience as intuitive and easy to use. The elaboration on this goal will include choosing proper JavaScript frameworks and libraries for the client development.

Evaluation of client and development process

In evaluating the UX of the client, usability measures must be analysed to evaluate the client usability and UX. In addition an assessment must be made to evaluate of the influence the user-centred design process has triggered the usability and UX outcome.

The evaluation will include the experience during the work by addressing lessons learned when formulating the final conclusion.

3.1.2 Problem definition

The problem decomposition listed above give an overview of the problem definition of this thesis. The problem definition in addition to the overall goals in 1.2, sub-goals listed in section 1.2.1 underpin the research question in section 1.3. The Web GIS client application requirements are listed in the next section.

3.2 Requirements

Supplementary to the list of sub-goals in subsection 1.2.1, further requirements are classified into functional requirements (FR) and non functional requirements (NFR). Requirements regarding the geographic data base and geographic data accessors will not be discussed in this thesis. Those requirements can be read in [2].

3.2.1 Web GIS client

The client should navigate and analyse a dataset provided by the geographic data store through the geographic data accessor. The functionalities are the same as in [2], they will be further extended during this thesis.

Web GIS client requirements are listed bellow.

- The client application should use open source software and not include any third party software.
- The software technologies should follow the best practices and standardisations as stated in [2].

Standardization of GIS domain services assure GIS operations can be published and seamlessly interoperate with other web frameworks.

FR01	Query data by a set of parameters
FR02	Present queried data inside map in browser
FR03	Present queried data features in tabular form
FR04	Present a map widget with basic map functionality
FR05	Do query based on a specific point on the map
FR06	Offer export functionality of requested data
FR07	Present graphs to illustrate and compare multiple data parameter values
FR08	Calculate and present a graph of density for temperature and salinity features
FR09	Present a contour plot within a range of meters in sea depth

TABLE 3.1: Functional requirements of the new Web GIS client.

Functional requirements

The functional requirements in Table 3.1 are based on the old Web GIS application described in section 2.2. FR01 to FR09 in Table 3.1 describes the requirements.

Personas, a representation of the user roles of the system [23] will be developed to get a classification of the users. Personas will be used during the developing of user stories to understand the needs of the users.

User stories, valuable functionality for the users of the system [23] will be developed as a requirement of what functionality the user wants. User stories are consistent descriptions of the requirements with an associated acceptance criteria.

Non-functional requirements

Non functional requirements (NFR) focus on usability and users interaction with the application. All NFR are stated with the purpose of an outcome in improved UX. This is reflected in the list of NFR in Table 3.2.

Quality models such as FURPS+ decompose quality into classifications which are more measurable and then possible to evaluate [24]. NFR reflects some of the classifications within usability in FURPS+, these regards Human factors, Astehics, Consistency and Documentation [25].

NFR01	Easy to understand interface
NFR02	Easy to learn the Web GIS
NFR03	Consistent user interface
NFR04	Appealing and minimalist design GUI
NFR05	Web GIS should display meaningful error messages
NFR06	Web GIS should prevent the users from making errors
NFR07	Web GIS should be accessible for users little or no experience
NFR08	Give user information on system status
NFR09	Give user information and guidance when interacting with the user interface
NFR10	Use open source software components

TABLE 3.2: Usability requirements of the new Web GIS client.

These requirements states a goal for the application in terms of usability features, but do not display the complete usability feature list.

These requirements will be measured by usability test sessions. Usability study and evaluation of these sessions will determine the outcome of the Web GIS development and answer if these requirements are met in the end.

3.2.2 Scope of Web GIS client

The overall Web GIS client scope is defined to implement a simple and functional Web GIS client. The focus is to pursuit the needs of the users with the best possible UX.

Scope of functionality

- Query data from the database. No altering of data, read only functionality.
- In presentation of the data the scope is to develop a simple query tool for querying data and map/graph plot visualizations to display the data. The visualisation is to be interactive and informative to present data values on demand.
- What visualisation the users are demanding, and how the functionalities will be best presented, is within the scope of developing the best possible UX.

3.3 Methodology and considerations

Agile methodologies and tools in the development process will be implemented to provide structure in the project management and client implementation. Agile methodologies support best practises with artifacts that organizes the development process from planning to delivery.

3.3.1 Usability methodology

UE, usability techniques and methodologies to include in the development process will be assessed. UE will focus on usability assessment and evaluation in the SDLC to improve and implement the best usability principles in the client application. These techniques will be addressed to establish a formula for the client development process in the future.

In addition to develop a Web GIS client, the aim is to develop a UE development process. This will include using known usability techniques. The usability study will be based on known standards definitions for usability in quality standards.

Pragmatic goals including experience, results and consequences of use [26] will assess the usability, and hedonistic goals, stimulation, identification and evocation [27] will assess the users subjective UX. To evaluate the hedonistic and pragmatic goals, a empirical research method [28] will be conducted to extract quantitative and qualitative data to evaluate the usability and UX of the client application.

To assure the goals are met, the UE process will be continuously modified and updated along the development iterations. Issues during the research and design process will be assembled and discussed in the results of this thesis.

3.3.2 Development methodology

The development methodology should be a part of an iterative SDLC to create a user-centred development process. The popular Agile methodology define iterations within timeboxes for software development and can easily be reshaped to include every aspect of a user-centred development process. The manifesto for Agile principles [29] describe the essence of Agile development. Working software and responding to change are two

bullet point of the manifesto for Agile principles, which integrates well with developing a user-centred application to assure valuable feedback from the users.

Other code standards and conventions such as use of standardised technologies, naming conventions, guidelines and other tools for quality assurance and automatic quality checking and testing will be integrated into the development.

User tester contact

The testing and feedback through the development will be in cooperation with the usability testers. Continuously feedback from users are important for the client development. The contact between developer and user should be in person.

Development and implementation

The development cycle involving design, implementation and testing should be iterative an iterative process. Continuous integration and version control are desired to assure the quality and accessibility of a code base.

3.3.3 Development considerations

The development should be performed inside a framework which organises the client code, make it readable, maintainable and scalable. JavaScript frameworks are popular in web application development and a possible technology for the Web GIS client development.

Testing

Testing is the artifact from Agile development with assurance that the code base is running as expected. Test suits should indicate if newly merged code break the current code base. Unit tests should be implemented to assure and assist the quality of the code base.

Code best practises and principles

Use of software design principles and patterns [30] in software development assure good code quality and will be assessed during implementation. Patterns as Model View Controller and the Singleton pattern in addition to single responsibility and dependency-inversion principles will be assessed during client implementation. This will assure modular code by keep it simple stupid philosophy.

Chapter 4

Software stack

This chapter reviews the software stack from data store, to data assessor and client. Standards and compliance of the data store and data accessor are described. The client technology selection criteria are presented, and an analysis of client technologies is conducted. Finally a complete software stack is presented in the end of the chapter.

4.1 Software stack

The software stack is a layered structure that contain a PostGIS data store, a GeoServer data accessor and a web client. This thesis focus on the web client which is built on top of the old Web GIS application [2]. The requirements for the software stack technologies are discussed in chapter 3.

The software stack from the previous Web GIS application [2] is part of the Boundless initiative [31]. Boundless supports the development of the best open source geospatial tools, where the architecture is built on flexibility, scalability and reliability. This modular architecture supports a reliable application where the individually parts easily can be combined and extended.

4.2 GIS standards

The requirements of the software stack is compliance and support of GIS standards. It includes making the geographic data store and web services accessible to the web

client application. *Open Geospatial Consortium* (OGC) is the main standardization for technologies within the GIS domain.

4.2.1 Open Geospatial Consortium (OGC)

OGC [32] is an international industry consortium developing interface standards for GIS technologies. OGC defines standards for storing and exchanging geospatial data. OGC consists of companies, governments agencies and universities which together complies standards to make complicated geospatial technologies, data and services accessible.

The standards are documents describing technical compliance of interfaces or encodings [32]. The standards are specifications to describe interfaces for GIS application development of access and manipulation of geographic data by different technologies [33].

4.3 Geographic data store

PostGIS is the Geographic data store which is completely *Simple feature access* (SFA) compliant [2]. PostGIS accommodates most geospatial database operations in addition to PostgreSQL queries [2].

4.3.1 Data store standards and compliance

SFA is an OGC and International Organization for Standardization (ISO) standard for geographic data stores. SFA specifies interfaces for GIS developers to implement in the purpose of publishing, access and storage for simple features such as point, line, polygons and more [33]. SFA consists of two parts: *Common architecture* and *SQL Option*.

Common architecture

The common architecture describes the simple features geometry. The base geometry class have subclasses for points, curves, surfaces and geometry collections. All geometry objects is referenced to a spatial reference system (SRS). The SRS gives the correct

coordinates for the geometric object in terms of its SRS [33].

SQL Option

The SQL Option specifies a SQL schema for storage, retrieval, update and query of geographic and non geographic data. A collection of features are stored as a feature table including a geometry attribute column. The geometry attribute is mapped to a geometry table containing the geometric data type. The geometric data type is defined by the SFA specifications [2]. The feature table is accessible by Standard SQL operations in addition to supporting querying geographic data by SQL operations [33].

4.4 Geographic data accessors

GeoServer [6] implements several OGC web standards, *OGC web services* (OWS), *Web Map Service* (WMS) and *Web Feature Service* (WFS). GeoServer is actively maintained and updated, reliable, feature rich geospatial data server [2].

4.4.1 OGC web services (OWS)

OWS is the standard for web applications using the HTTP protocol. WMS and WFS are OWS services which publishes layers provided by GeoServer. Each OWS contains a subset of layers and can be configured by a workspace. The workspace constrains the availability of the published layer on the server.

4.4.2 Web Map Service

WMS is the standard used to request map data from a GIS enabled database. The interface standard specifies a set of features about the requested images. WMS supports a list of output formats, including PNG, JPEG, TIFF, SVG and KML. The operations of WMS are *GetCapabilities*, *GetMap*, and *GetFeatureInfo* [34]. The operations in interest for this thesis are listed underneath. All WMS operations offer exception reporting with XML by default. See the WMS Implementation Specification [34] for more details.

GetCapabilities

The **GetCapabilities** provides general information request info about the WMS map server. The mandatory parameters in a WMS request to GeoServer are **SERVICE=wms** and **REQUEST=getcapabilities** as described in listing 4.1. The request retrieves information about the layers available, what image types it can serve, list of coordinate systems etc.

LISTING 4.1: WMS request.

```
1 http://localhost:8080/geoserver/wms?service=wms&version=1.1.1&↵  
  request=GetCapabilities
```

GetMap

GetMap request the map image. OGC specifies a list of WMS parameters in the **GetMap** operation.

- **VERSION** The version of the request
- **REQUEST** The name of the request
- **LAYERS** The list of layers in the request
- **STYLES** The list of styles in the request
- **CRS** The coordinate system type
- **BBOX** The size of bounding box
- **WIDTH**
- **HEIGHT**
- **FORMAT** Output format of map

Other parameters include **TRANSPARENT**, **BGCOLOR** among others [34]. Figure 8.1 illustrates a map image response from a **GetMap** request.

GetFeatureInfo

The **GetFeatureInfo** request provide information about a map image returned by a **GetMap** operation. Clicking a point on a map **GetFeatureInfo** queries feature information about the current point(I,J) on the map. The operation parameters are all the same as *GetMap* except **VERSION** and **REQUEST**, including parameter **I** and **J**. The parameter **I**=pixel.column and **J**=pixel.row from the requested map. The server should response the features within (I,J).

4.4.3 Web Feature Service

WFS is a standardization for managing vector format geographic data through HTTP. The WFS request is written in Geography Markup Language (GML). WFS supports output formats GML2, GML3, Shapefile, JSON, JSONP and CVS. They are specified in **OUTPUTFORMAT=**json parameter [35].

GetFeature The request for features is instantiated by a query. The response is a set of features. The Common Query Language (CQL) described in the next subsection can be used to constrain the **GetFeature** request. Parameters are what kind of **SERVICE=WFS**, what **VERSION=2.0.0** (current service version), request **REQUEST=getFeature** and return a response for the layer in the **TYPENAMES=namespace:featuretype** parameter [35]. Other parameters can be included too but these are not relevant in this thesis.

4.4.4 Other relevant OGC concepts

Filters

GeoServer can narrow the response of a request by filters. Filters sets the condition for the search in data store features. Filters are used in WMS to request features presented in the map and in WFS to return the requested features. These are used in the request to WMS **GetMap** and WFS **GetFeature** request as filter parameters. GeoServer is compatible with CQL filters. CQL filters can use all the filter functions available in GeoServer [36].

To constrain the response from GeoServer CQL filter parameters can be appended to the request. The filter converts into a SQL query where the filtered parameters are defining the query string. "WHERE", "AND", "OR" are statements used to filter the query. A CQL filter specifies the query is defined in Code Listing 4.2. This query filter the request to select states that have between 100 000 and 150 000 inhabitants. CQL filter by parameters in database, can compare entities in database or by filter functions or geometric filters.

LISTING 4.2: CQL filter.

```
1 http://localhost:8080/geoserver/wms/kml?layers=topp:states&↵
  CQL_FILTER=LAND_KM+BETWEEN+100000+AND+150000
```

4.5 The Web GIS client application

The client application developed in this thesis is based on the requirements defined in chapter 3. The client application should support high interactivity in a user friendly manner to meet requirements of a satisfying UX. The main feature is a map widget displaying geographic data. The client should contain map controllers as zooming/panning and a layer-switcher. All data should be stored in the geographic data store and accessed through OWSs (OGC Web Service) from the data accessor.

4.5.1 Selection criteria

Several client technologies were investigated to meet the functional and non functional requirements listed in chapter 3. The selected criteria underneath is stated to meet the requirements in section 3.2.

Standard compliance

The client technology should implement the latest web standards by World Wide Web Consortium (W3C) such as HTML5 [37] and CSS3 [38]. Web standards will assure the content of the client application will be supported and displayed as desired.

OGC standards must be addressed to assure the application interoperability. OGC offers a complete list, the most important in this client application are OWS, OGC standard created for WWW applications. The main standards in this application are inherited by the old Web GIS application. Such as WMA, WFA and SFA [2].

Open source

The technology has to be open source, with a license that allows the tools to be free to use, modify and shared. Open source software (OSS) is a requirement 3.2 for reasons that OSS is more secure, often quickly fixed and updated [39], to prevent vendor lock-in and prevent costs.

Community and documentation

The choice of technology has to be widely supported by a community actively developing the technology. The documentation has to be available and accessible both from the community and from other sources such as books, videos, articles etc. Future support of the technology is a key criteria. Active development and an established open source community predicts good future support.

4.5.2 Client technologies

JavaScript language in web client applications enhances the UX with dynamic user interface features. JavaScript is a widely used language with support for most web technologies used today.

Using a JavaScript web framework will help systematise the code in a structure and give some design choices to make the applications easier to understand, test, expand and maintain [40]. JavaScript frameworks for web applications has gained a wide popularity the last years. It may involve investing time and effort to learn new paradigms on writing JavaScript to simplify and attach behaviours to the client code.

The web frameworks in this analysis are three of the most popular JavaScript frameworks [41]. They provide many of the same features but have some differences from each other.

AngularJS

AngularJS [42] is a JavaScript framework for developing extendible web applications. It is built to be easy to test and maintain as it scales. AngularJS is an open source framework licensed under The MIT License [42]. AngularJS was released in 2009 and has become one of the most popular JavaScript frameworks available. AngularJS is maintained by Google, it has an active community and had in 2013 the fourth largest contributions to their repository [43]. AngularJS has a Model View Controller (MVC) design pattern using data binding, client side templates and dependency injection to build a structure for developing web applications [40]. AngularJS decreases the complexity of JavaScript code by directives which attach behaviours to the document object model (DOM).

BackboneJS

BackboneJS [44] is a lightweight JavaScript library which provides a minimal set of structure offering an architectural structure extendible to fit your requirements. It decouples concerns and makes code more maintainable [45]. BackboneJS is created by Jeremy Ashkenas and released in 2010 [45]. BackboneJS has an active community with tutorials and information about how to get started. BackboneJS is mature and popular, and has extensions available to build upon.

EmberJS

EmberJS [46] is a JavaScript framework based on MVC pattern for developing web applications. EmberJS provides the patterns, components and tools to manage important tasks like code modules, state and data flow [46]. EmberJS provides best practices and structural code templates for building ambitious and testable applications. It relies on convention over configuration [47] to structure the application [46]. EmberJS is the newest framework in this analysis, although released in 2012 [48] it has a large community on GitHub [49].

Conclusion

AngularJS is a JavaScript framework providing support and structure for JavaScript web applications. BackboneJS also offers structure to JavaScript code but provides a

looser convention of how to structure your code. BackboneJS is easy to get started because of the lightweight library, AngularJS is a more extensive framework to get a grip of considering the AngularJS jargon. EmberJS is the largest of the three and built for ambitious web applications. In comparison with AngularJS both frameworks have extensive libraries for building feature rich, navigations, testable, readable and maintainable web applications. EmberJS and AngularJS are not as easy as BackboneJS to learn, but have more support for more complex applications.

Because of the maturity, the extent of features and active community support is AngularJS the choice of web framework developing Web GIS client in this thesis. AngularJS and EmberJS are two of a kind, but AngularJS is built with testing in mind and have a simplicity and structure in code which give cleaner code. It is also easy to learn in terms of knowledge of MVC and is the most popular JavaScript framework for complex web applications. AngularJS also has the biggest community, and good documentation and tutorials to help getting started.

4.5.3 JavaScript unit testing framework

Unit tests are building blocks for automated testing of software. There are many JavaScript testing frameworks available. This analysis evaluates three popular high quality standalone test frameworks.

As described in the JavaScript analysis, AngularJS is developed to be testable. AngularJS has developed it's own test runner, Karma [50]. Karma is a fast, simple and stable test runner which provide support for several JavaScript testing libraries, some of them described in the analysis underneath.

QUnit

QUnit [51] is an easy to use framework first developed as part of jQuery before evolving into its own name and API in 2009. QUnit is independent of any JavaScript library (it runs completely standalone), and is used by jQuery projects as well as other JavaScript projects [51]. QUnit is released under the Massachusetts Institute of Technology (MIT) license [52]. QUnit runs in the browser and provide easy testing of DOM manipulation. QUnit is simple to get started, attach two files to the HTML file, the test runner

`qunit.js` and a CSS file `qunit.css` [53].

Mocha

Mocha [54] is a feature rich flexible test framework. Any assertion libraries are supported among other third party as libraries for Behavioural Driven Development (BDD) testing [55], TDD testing, interface including QUnit's assertions and Jasmine assertion library [56]. The flexibility provide complexity to configure it in a desired way. Mocha is run through a commando line interface and built on top of node.js, which is specially suited for node.js applications.

Jasmine

Jasmine [57] is a popular BDD testing framework. JUnit [58] was the original JavaScript unit testing framework until it became Jasmine. It does not rely on any JavaScript frameworks, browsers or the DOM [57]. Jasmine is developed by Pivotal Labs and is licensed under the MIT License. The syntax of Jasmine is easy to understand. Jasmine has extensive documentation with description code examples and relevant books with tutorials on Jasmine.

Conclusion

All test frameworks are standalone frameworks suited to write unit tests for the new Web GIS client and all of them integrates great with Karma. QUnit do have extended functionality for DOM testing but this is available as a plug-in with Jasmine. QUnit has a smaller range of assertions and do not add third party assertions libraries as good as Jasmine.

Mocha is the newer framework with some lack of documentation. Mocha is very flexible with extensive set of features which make it less beginner friendly. A simpler framework as Jasmine is more desirable to get fast up and running with tests without depending on any browser to run as desired.

Jasmine is the preferred choice, it provide an easy to get started package which get the testing environment up and running without much experience. Jasmine do provide

descriptive syntax for BDD testing, is easy to get started, popular and well as suited for beginners.

4.5.4 Web mapping technologies

Two web mapping libraries were analysed in regard to implementing PostGIS as back end and GeoServer as web server. The web mapping libraries should implement OGC standards, WMS and WFS and a supportive contribution to the development. The web mapping technologies bellow are supporting the services GeoServer can provide.

Leaflet

Leaflet [\[59\]](#) is an open source JavaScript library for displaying interactive maps in a web browser. Leaflet are lightweight library for displaying simple interactive maps. Leaflet take advantage on HTML5 and CSS3 and create a modern user interface with simplicity, performance and usability in mind.

OpenLayers

OpenLayers [\[60\]](#) is a JavaScript library for interactive maps in web applications. It enables displaying and editing geographical data from WMS and WFS sources. OpenLayers is open source, FreeBSD licence and working towards a new release with highlights as WebGL, HTML5 and CSS3 features. OpenLayers builds rich geographic web applications with no server side dependencies. PostGIS, GeoServer and OpenLayers are proved to be a solid software stack in GIS development [\[61\]](#).

Conclusion

OpenLayers is feature rich and more complex than the lightweight Leaflet library. Leaflet is designed to provide simple interactive maps while OpenLayers are more suited for GIS applications as our Web GIS application. Leaflet do have a lot of plug-ins to make up for OpenLayers features, but is designed to be a lightweight library with a simple and stable code base.

OpenLayers wide support for OGC standards and GIS functionality make OpenLayers the better option for the new Web GIS client. OpenLayers supports a wide range of OGC standards compliant sources and data formats and is pluggable with AngularJS [62] OpenLayers is the more mature library and has a lot of documentation to show for.

4.5.5 Additional technologies and libraries

Technologies and libraries supporting the client code with templates of user interface design elements and presenting tables and charts was analysed. There were alternatives but not comparable to the ones listed underneath. They are both popular and the most used in their respective domains.

Bootstrap

Bootstrap [63] is a HTML, JavaScript and CSS framework for developing responsive web client applications. Bootstrap has reusable HTML elements, CSS components and templates, and jQuery plug-ins.

Angular UI-Grid

UI-Grid [64] is a native AngularJS implementation for displaying data in tables. It is configurable and has a template for CSS, and behaviours such as filtering, export, sorting and many more.

Highcharts

Highcharts [65] is a JavaScript library which supports a large set of different charts which can dynamically be added and removed. It is based on native browser technologies and is fully compatible with modern browsers.

4.5.6 JavaScript IDE

WebStorm [66] is a popular JavaScript IDE and the desired choice. WebStorm is a powerful lightweight IDE. WebStorm have coding assistance, error detection, refactoring, code completion and seamless tool integration. WebStorm supports a productive development environment with good support for JavaScript, HTML, CSS. [67] states WebStorm has great advantages over the other tested JavaScript, HTML and CSS IDEs. Other JavaScript IDEs was Netbeans [68] and Aptana [69]. WebStorm has good support for AngularJS and a good choice for this web project.

4.5.7 Build tool and code generators

Yeoman

Yeoman is a code generator. When installed, Yeoman generates code on demand and cover a hole range of projects [70]. The generator is ran by the `yo` command to scaffold new code of the project. E.g when a new AngularJS controller is generated, test classes with implemented tests are also generated.

Grunt

Grunt [71] is a build tool. A JavaScript task runner perform the build, deployment and unit testing of the project. The workflow of the tasks are configured in the Gruntfile. Grunt has a long list of tasks as plug-ins. Commands in project: `serve`, `build`, `test`. Grunt run a watch task when project is launched into the browser. CSS, HTML and JavaScript changes are continuously relaunched and displayed in the browser.

Bower

Bower [72] is a component installer. Bower manages every library and its dependencies when installing through Bower. Bower keep track of the files in a manifest file `bower.json`. When new JavaScript libraries are installed, Bower make sure they are included in the necessary files in the project.

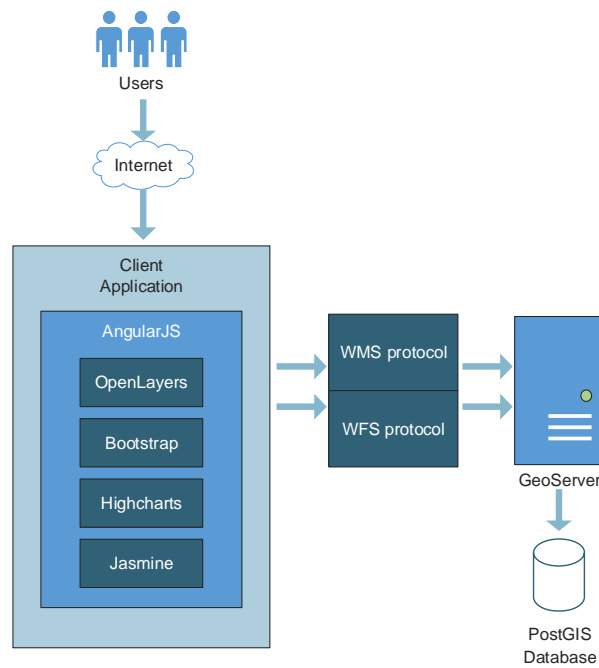


FIGURE 4.1: Updated conceptual model including client technologies.

Others

Git is a version control tool with GitHub as the major online provider. Travis is a continuous integration (CI) tool for building the codebase whenever pushed to GitHub.

4.5.8 The complete software stack

The software stack described in chapter 3 include a PostGIS geographic database and GeoServer web server which is serving the client geographic and non geographic data. During this chapter the software stack is updated with client technologies by the selection criteria in subsection 4.5.1. Figure 4.1 illustrates the new complete software stack. AngularJS is the HTML5 and CSS compliant JavaScript web framework for interactivity and presentation. Karma and Jasmine are the testing libraries for unit testing to meet the requirement of a good enough test coverage. OpenLayers is implementing the map widget and compliant OGC standards WMS, WFS. Highcharts assign charts for geographical data exploration and analysis.

Chapter 5

Usability engineering

This chapter describe the UE and the User-centred design (UCD) process used implementing the Web GIS client. It is introduced by the usability methodology and standards used as base for the usability studies. Then the usability methodology is discussed. The rest of this chapter discusses step by step the UCD process including conceptual development, prototyping, implementation, usability studies, usability analyse and evaluation in the end.

5.1 Usability methodology and standards

UE is implementing methods for interaction between humans and computers by get to know the users goals, domain, tasks and work practices by focusing on how the users actually work rather than change the way they work [73]. It is critical to implement usability principles which are easy to use and get the usability tasks done [74].

Principles and practises of known HCI research has been used in the Web GIS client development. HCI promotes interfaces based on principles combined from computer science, philosophy and design. UCD is a set of practises originating from HCI [21]. UCD is implemented as a design process that focus on the users and their tasks. The UCD process developed is influenced by a range of studies [75], [76] and [77],and embraces the usability evaluation of the system [78].

The Web GIS client was designed based on conversation with usability testers during a empirical usability evaluation method. To analyse the outcome of the empirical usability

testing a quantitative and qualitative analysis was conducted. The quantitative method was based on the users perception interacting with the system, while the qualitative analysis was based on the user comments during the usability testing. Both methods accumulated the users' accurate client interaction experiences, and was always performed straight after a test session.

The usability attributes used in the UCD process is based on the heuristics from Nielsen, ISO standard 25010 [79] and the 5Es from Quesenbery discussed in the next subsection.

5.1.1 ISO/IEC 25010 standard for system quality and measurement

ISO/IEC 25010 describes usability as a set of attributes, which help specific users achieve a set of goals efficient and effective [79]. The bold text with darker background in Figure 5.1 and Figure 5.2 specifies the entities used in this thesis. Figure 5.1 classifies usability as an entity where the attributes are quality factors for good product quality, which is an important in achieving good UX. The models are dependent on each other by the definition of *Learnability*, "degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use" [79].

A UCD process has been developed to define, evaluate and measure usability classifications and attributes of ISO/IEC 25010. The attributes are picked to meet the Non Functional Requirements (NFR) in subsection 3.2.1. [4] argues ISO 25010 is applicable for evaluation of GIS applications by extending the model with new attributes specific for the GIS domain. The usability attributes specified in the figures have been adapted into the UCD process to accommodate usability and UX of the Web GIS client.

5.1.2 Nielsen heuristics

Heuristics are rules of thumb for usability developed by Jakob Nielsen [16]. [80] lists a set of 10 heuristics used as a guideline for usability evaluation described in subsection 5.4.1. A subset of the heuristics was used during the prototyping; Error prevention, Aesthetic and minimalist design and Help and documentation. Error preventing was important to help the users input and feedback on non valid input to improve the work-flow efficiency.

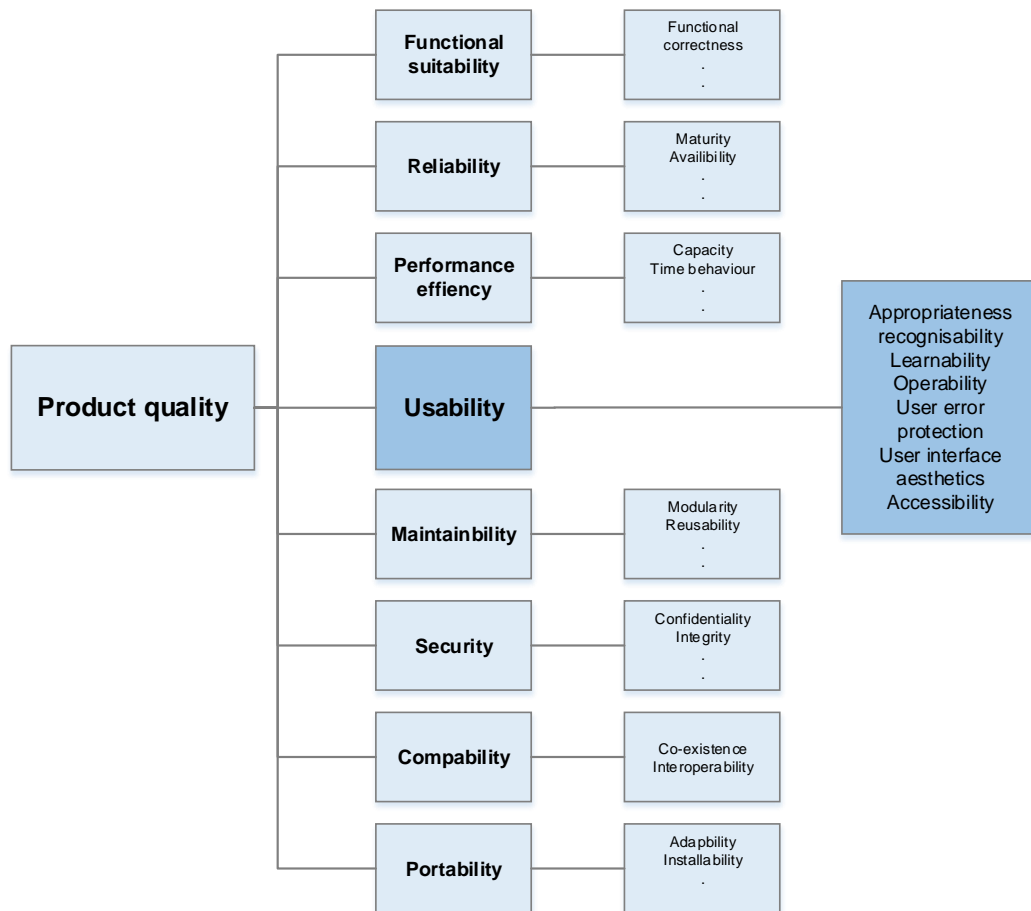


FIGURE 5.1: Product Quality of ISO/IEC 25010

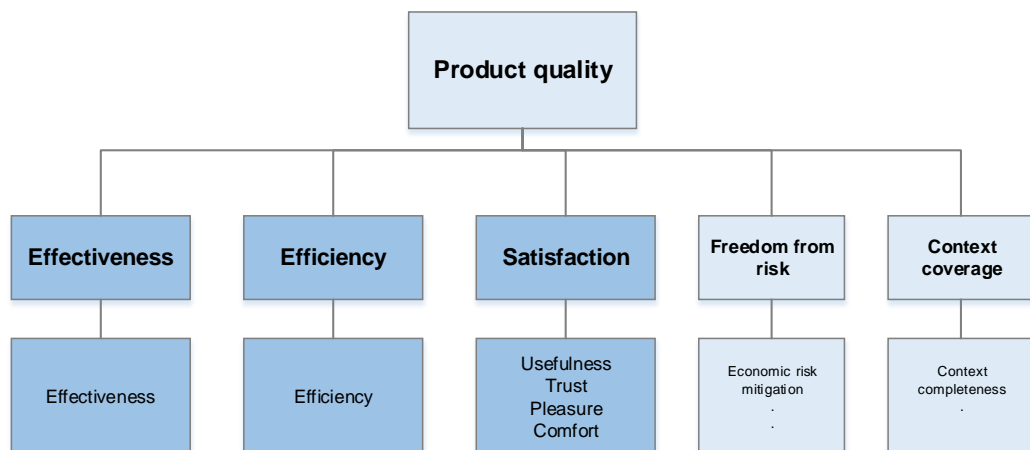


FIGURE 5.2: Quality In Use of ISO/IEC 25010.

Minimal design was implemented to eliminate all non necessary design elements which may take attention from the purpose of the application. Help and documentation was used to inform the users of the applications functionality and user tasks during the work-flow. The remaining heuristics are equally important but not implemented due to the scope of time developing the Web GIS client.

5.1.3 5Es usability attributes

5Es is a set of usability qualities attributes classified by Whitney Quesenbery [81]. Quesenbery partition some of the usability attributes from ISO 25010 and are implemented to fulfil the NFR of the client. Understanding the usability aspect is understanding what the usability depends on [82]. Figure 5.3 illustrates the balance between the usability quality attributes used. The attributes, easy to learn and error tolerant was the mostly used, after engaging and effective ("how the users accieve their goal accuratly" [82]). Efficient, the amount of time in terms of number of clicks before a user task is achieved was least used. The 5Es was used as a guideline for what make the client "work for the users" [82]. All of the 5Es attributes are applicable for Web GIS applications and can be used to balance what the users think is the most important usability attribute of the Web GIS application. The attributes are discussed further in the user interface evaluating in subsection 5.4.



FIGURE 5.3: 5Es usability attributes.

1	The design is based upon an explicit understanding of users, tasks and environments
2	Users are involved throughout design and development
3	The design is driven and refined by user-centered evaluation
4	The process is iterative
5	The design addresses the whole user experience
6	The design team includes multidisciplinary skills and perspectives

TABLE 5.1: ISO 9241 6 principles of UCD process

5.2 User-centred design process

UCD work towards usability quality in products by including the users in an iterative development process, to meet the requirements, needs, wants and limitations of the users [21]. UCD is based on ISO 9241-210 [83] standard Part 210: Human-centred design for interactive systems. The standard describe 6 principles of user centred design illustrated in Table 5.1.

Figure 5.4 illustrates the UCD process. The models starts with domain research then continues with design, prototyping and implementation of the application. In the end usability studies and evaluation are performed. The big arrows illustrates the main steps while the thinner arrows indicates the iterative process where short-cuts can occur. The activities in the UCD process are discussed during this chapter.

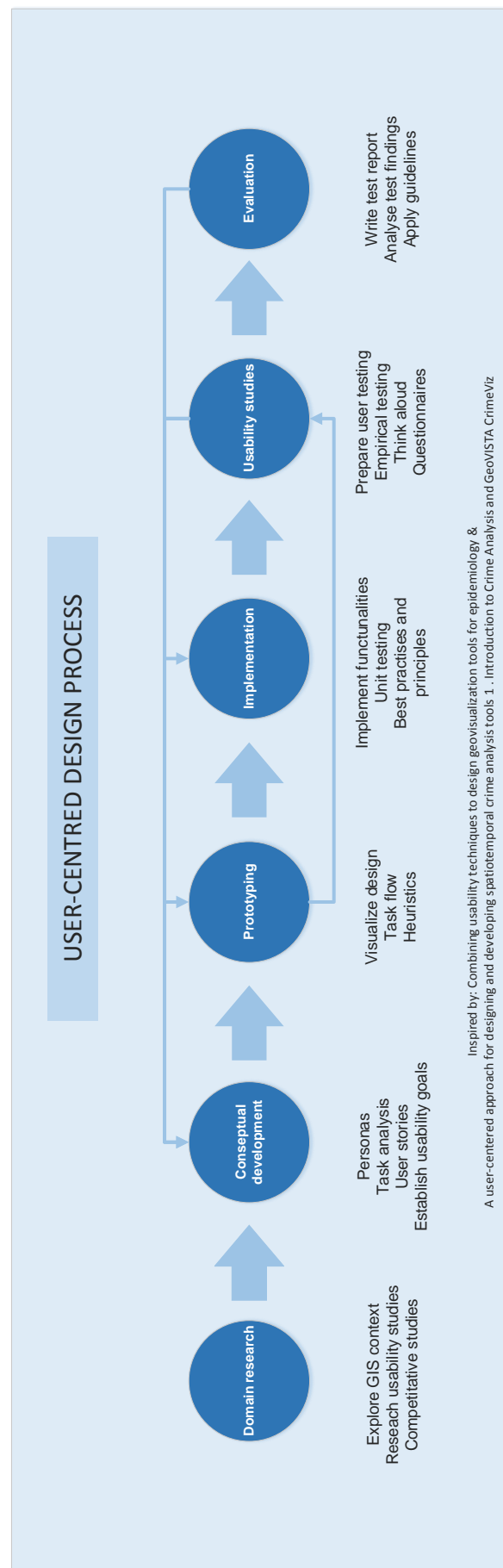


FIGURE 5.4: User-centred design process model.

5.2.1 Domain research

The first activity involved studying the GIS and Web GIS domain, study the software technology, and look for similar web applications to understand the work process of these. Next, UE methods, principles and practises were investigated to adapt development activities which would be suitable for the Web GIS client development. UE domain research concerned with both GIS and non GIS domain were studied. The goal was to get to know both GIS and UE domain and converge both domains to the benefit of this thesis.

5.2.2 Conceptual development

The goal of using a user-centred approach with usability evaluation was to confirm if the Web GIS client objective is fulfilled i.e if the users can complete their tasks with satisfaction [84]. The methods used should assure the use of technology and visualisation methods of the dataset is appropriate for the set of users [21].

Design decisions

The first user interface design and features were based on the old Web GIS client [2] and similar applications on the Internet listed in section 2.3. New user task were continuously developed based on feedback from the usability testers through the UCD iterations.

Design decisions in terms of usability were based on standards and principles from section 5.1, broken down to make the usability evaluation measurable.

Personas

A persona is "an imaginary representation of a user role" [23]. The personas should feel as real persons for the developer. Including the personas Web GIS experience, what their user tasks, goal using the Web GIS and a portrait photo to give the "person" a face. Three personas were developed in early conceptual development. Each representing different Web GIS expertise and purpose of using the application. A persona, Dr. Keri Sofie Larsen is illustrated in Figure 5.5.



FIGURE 5.5: Example of a persona.

The personas are classified as novice, regular and expert Web GIS users as illustrated in Table ???. The developed personas is in Appendix A.

User role	Web GIS expertise	Name
Oceanographer	Expert	Keri Sofie Larsen
Environmental scientist	Regular	Mikka Heikenen
Student	Novice	Natalia Geyer

FIGURE 5.6: Personas for the Web GIS application.

Design user stories

Figure 5.7 illustrates a user story related to the specific persona illustrated above. User stories is a brief description of a unique user task, focusing on the functionality of the client. The user stories were developed through task analysis. Functionalities were divided into smaller subtasks in cooperation with the users. The user stories are based on the knowledge of the users, the domain, geographical data and presentation/visualization of the specific user task. Discussion and agreement about proper user tasks arose from the conversations during usability testing.

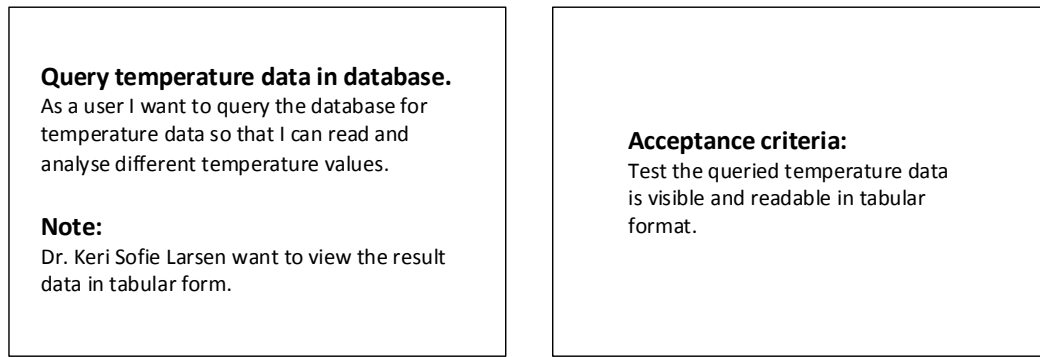


FIGURE 5.7: User story aimed for the persona named Keri.

5.3 Prototyping and implementation

Prototyping was performed in iterations before it was implemented. Paper prototypes was developed and evaluated in the first iteration. The paper prototypes was a sketch tool to elicit visual design options and to evaluate the positioning of user interface controllers before implementing them. The paper prototype evaluation were conducted using Nielsen's heuristics as listed in [80].

The next section discusses the methods used during the usability- study and evaluation of the implemented prototype.

5.4 Usability study and evaluation

In UE, the main purpose is to examine the satisfaction of early design decisions [85], while learning about the concrete user mental model, tasks and goals interacting with the Web GIS client. Usability- study and evaluation of the prototype has been done in three iterations. Figure 5.8 illustrates the usability studies step in the UCD process by an enlarged orange circle.

5.4.1 Usability testing and evaluation methods

The usability evaluation methods were carefully chosen early in the project. The criteria for picking a method were based on following; number of usability testers, time of scope and effect. Usability study and evaluation were conducted on both paper prototype and

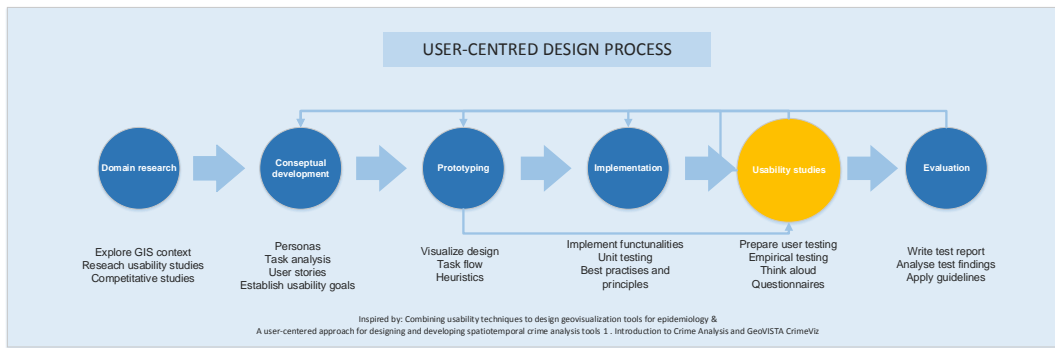


FIGURE 5.8: User centred design model.

interactions with the implemented client prototype.

Assessment Methods

- The paper prototype was iteratively inspected by Nielsen’s heuristics [80], the user interface was adjusted to meet the non functional requirements listed in table 3.2. Heuristic evaluation is a effective way to eliminate usability issues early at least cost possible [16].
- Empirical lab studies were performed at every test session. Lab studies is a very popular method for usability testing and an important method to test UX in a early phase of product development [86]. Observation revels real user tasks and they suggests new functions and features [16]. Empirical testing gets closer to the users to understand the users visual interpretation of tools and interaction paradigm [85].
- Thinking aloud, properly the most valuable UE method [16] was used during the lab studies. It is a method where instant feedback is received, which enables to explore the cognitive thinking and perceived subjective thoughts and feelings about the user interface. Thinking aloud enable us as developers to understand the user perception and misconceptions interacting with the prototype [16] while thinking aloud during problem solving. Heuristics and thinking aloud are a complementary set of usability methods to reduce usability problems [16] .
- Questionnaires was completed by users after every usability test session. A set of questions with multiple choice illustrated in Figure B.5 in Appendix B gathering

feedback from the users. Questionnaire is regarded as one of the most effective methods to measure both pragmatic and hedonistic goals [27].

There exists a wide range of UE methods which also could be considered such as focus groups [16], but that will require a large group of usability testers over a longer period of time. Next subsection discusses the choice of usability testers, how many used and why.

5.4.2 Usability testers

We had three usability testers during the development. Jakob Nielsen argues that a large number of usability testers do not necessarily find more usability issues than a smaller set of testers. Three usability testers will reveal many of the same usability issues [87]. Approx 70% of the issues at first iteration and the remaining two iterations can improve the usability issues with as much as 38 % each [16].

The domain of Web GIS applications, its geographical data and user tasks demand a thorough selection of usability testers. The usability testers had different personal background and domain expertise. The testers were both domain- and technical representatives, in addition to having experience level in similar GIS applications from novice to regular and expert.

The usability testers have not been available from the start of the project, they were first introduced to the Web GIS client when prototype 1 was implemented. The new client design and architecture had to be implemented before the first usability study could be conducted.

[85] emphasises that close collaboration between developer and users give a better understanding of context in specific domain tasks. The users were cooperative and inspiring in sharing their perception, subjective feelings and ideas of improvements. Conversations with the users resulted in improved usability by several user interface changes. The users' role in improving the client application affected the outcome of this thesis.

5.4.3 Planning usability testing

Every usability test session was carefully planned in advance. A usability test document was designed and used throughout the three iterations of the UCD process. The document contains basic information about the test process, the tools, methods used and the purpose of the usability testing. The document is attached in Appendix B.

Evaluation measurements were implemented into the test plan, measurements to fulfil the system requirements listed in section 3.2. Usability standardisation and 5Es discussed in section 5.1 classified the user tasks in the empirical testing and questions in the empirical testing. Figure B.3 and Figure B.4 in Appendix B illustrates the user tasks to meet the usability attributes in 5Es. Classifications of quality attributes in Figure 5.3 were designed to fulfil the usability attributes of ISO 25010. The empirical test sheet with the questionnaire schema is shown in Figure B.5 in Appendix B.

5.4.4 Iterative usability testing

The first usability test was performed in iteration 1 of the UCD process. The old system would preferably been usability tested to inherit functional design decisions but it was not accessible. The first interaction with the users was performed by a Google spreadsheet, asking general questions to get to know the users. The collected data was used designing the first prototype.

The test sessions were performed in a ordinary office space familiar to the testers were they could feel comfortable. One by one, each tester had enough time to conduct the test procedure. All the test sessions were voice recorded.

User interface evaluation proceeded in an iterative design process. Interactive user testing was constantly improving the user interface with feedback from the testers. Design and implementation issues was encountered by discussions on how to improve and redesign as the users desired. Feature implementations was inspected and tested to find any design issues, some UI elements were redesigned or further tested in later user testing.

Figure 5.9 illustrates step by step activities during the test process. The steps are described further in this section.

Usability test procedure

1. Short introduction

The initial step introduced the test procedure for the tester, including the purpose of the usability testing, the steps and information due to this current usability testing. See Appendix B for a written version of information regarding the testing session.

2. Pre-test interview

A pre-test interview was then conducted to introduce the current test and answer any questions the tester had. The introduction was aimed to gather the experiences from any previous test sessions.

3. User-testing

Earlier design decisions in the user interface, map operations, search functions, chart plots, density and contour features were assessed during the empirical testing sessions. The empirical testing method, think aloud included testing tasks to investigate the user interface. The test tasks was completed one by one until every task was conducted. The tester could freely ask for guidance if misconception did occur. Appendix B Figure B.3 and Figure B.4 in Appendix B illustrates testing tasks for iteration 3.

The feedback from the users was used as a qualitative measurement to evaluate the satisfaction with the client and its functionalities. User stories, functionality was build of the user feedback during the think aloud session.

4. Post-test questionnaire

The post-test questionnaire define the user subjective preferences [16] after experiencing the Web GIS client. The test user answered the questionnaire post-testing with interaction experiences fresh in mind.

Questionnaires as a quantitative analysis expressed the UX on the system, and could be systematically measured by computation average and comparison of the answers.

5. Debrief

The debrief was designed to capture experiences during the test session. Questions and answers from both parties to evolve and adapt the process the best possible way.

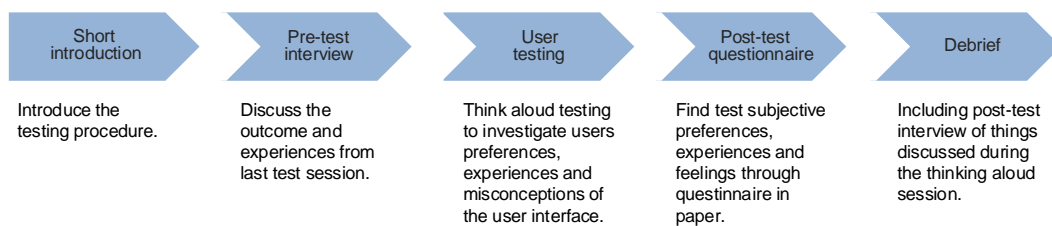


FIGURE 5.9: Usability test procedure.

5.4.5 Usability evaluation

Usability analysis and evaluation include methods where developers get to know what the users are satisfied with and how to conduct further design decisions to meet the satisfaction and goals of the users. All the test results were analysed in the end of each iteration. The analysis was based on observation, oral feedback and questionnaires. The recordings from each test session was analysed and feedback for each test session was noted to the respective test schema. Usability issues, proposal in new or change for user tasks were collected in a backlog. The backlog was analysed every iteration and added to a prioritised list for future implementation. The usability evaluation in the end of each iteration was directed towards a tool or a operation, interface or layout.

Quantitative and qualitative methods were used in the analysis and end evaluation of every iteration. The qualitative evaluation gave a good indication of the usability testers cognitive thinking, their experiences when interacting with the client in both pragmatic and hedonistic goals. The quantitative evaluation based on the questionnaires validated the UCD outcome iteration after iteration. Quantitative usability test data evaluated the usability of the website, if the UCD did give the outcome of better usability. The evaluation results are discussed in chapter 9.

Chapter 6

Data store and data accessor configuration

This chapter describes step by step how the PostGIS data store and GeoServer data accessor were installed, populated and configured. First the OpenGEO Suite [\[61\]](#), a complete development platform for geographic data including installers, documentation and code examples was installed. Then population and configuration starting with PostGIS data store and then the GeoServer is described during this chapter.

6.1 Geographic data store

Even if the PostgreSQL database enabled with PostGIS extension was installed during the installation of OpenGeo package some configuration was needed to make it complete. First step was to create a PostGIS database using the PostGIS template [\[88\]](#) inside the PostgreSQL graphical user interface (GUI) pgAdmin. This template spatially enables the database by configure extensions for PostGIS functions and data types. It includes the mandatory tables for managing PostGIS functionality, `spatial_ref_sys` and `geometry_columns`.

6.1.1 Create and populate PostGIS database

Create database tables

The next step was to create the database tables. Code listing 6.1 describes how the station table was created including column names and data types. The complete database schema, `station`, `p_salinity` and `p_temperature` are illustrated in Figure 2.2. All data tables have the field `absnum`, it is a primary key in `station` and foreign key in `p_salinity` and `p_temperature`.

LISTING 6.1: SQL syntax for creating the `station` table.

```
1 CREATE TABLE station
2 (
3   absnum integer ,
4   stflag smallint ,
5   stlat numeric (8 ,5) ,
6   stlon numeric (9 ,5) ,
7   stdate date ,
8   sttime time without time zone ,
9   stsource character varying (12) ,
10  stversion smallint ,
11  stcountryname character varying (40) ,
12  stvesselname character varying (40) ,
13  stdepthsource integer ,
14  stlastlevel smallint ,
15  stdepthgrid smallint ,
16  stdepthgridmin smallint ,
17  stdepthgridmax smallint
18 )
```

Import data into database tables

The data were copied into the database tables from simple text files using the PostgreSQL function `copy`. In order to use the `copy` function PostgreSQL needed the correct file system permissions, given by the properties of the text file `copy` is reading from.

The text files contained the subset of data, one file for each table. The text had a delimiter “—” (dash) to separate the parameter values. Code Listing 6.2 specifies the SQL command copying the parameters from a text-file into the `station` table. Some issues arose in the *copy* process, consisting mainly corrections of empty spaces indicate tab-indentation. The *copy* operation was a success if no error message appeared running the query.

LISTING 6.2: SQL syntax COPY data.

```
1 COPY station FROM 'C:\\DatabaseData2003\\station2003.txt' WITH ↵  
   DELIMITER '|';
```

The dataset imported into the database were measurements of temperature and salinity within the year of 2003. This subset contained sufficient data to perform Web GIS operations within satisfactory performance and client requirements.

Add geometry column in station table

At this point the database is populated with required database tables. The `station` table was populated with the SFA point type to represent the latitude and longitude [2]. This step of the process was performed by instructions described in [2]. The steps to create and populate the geometry columns in the `station` table is described next.

LISTING 6.3: `station` table including the new point variable.

```
14 CREATE TABLE station  
15 (  
16   absnum integer ,  
17   stflag smallint ,  
18   stlat numeric (8 ,5) ,  
19   stlon numeric (9 ,5) ,  
20   stdate date ,  
21   sttime time without time zone ,  
22   stsource character varying (12) ,  
23   stversion smallint ,  
24   stcountryname character varying (40) ,
```

```
25  stvesselname character varying (40) ,
26  stdepthsource integer ,
27  stlastlevel smallint ,
28  stdepthgrid smallint ,
29  stdepthgridmin smallint ,
30  stdepthgridmax smallint ,
31  stPoint POINT
32 )
```

1. Create new geometry column This step created a new geometry column in the `station` table. The data in this column were stored as the SFA type `point`.

Code Listing 6.4 illustrates the SQL syntax in this operation. The parameters include schema name, table name, column name, type of srid, type and dimension of geometry. The `Srid` parameter defines the SRS used for the `station` table. It is also a foreign key to the `spatial_ref_sys` table [89]. This Web GIS `srid, 4326` is a parameter value of the European Petroleum Survey Group (EPSG). EPSG:4326 is the projection of the SRS World Geodetic System 1984 (WGS84) [90]. WGS84s coordinate system is Cartesian Coordinates (X, Y, Z), it's widely used, e.g as the coordinate system for the Global Position System (GPS) [91].

LISTING 6.4: SQL syntax to add Geometry coloumn.

```
1 SELECT AddGeometryColumn (
2  'public', 'station', 'stPoint', 4326 , 'POINT', 2
3 );
```

2. Populate lon/lat data When the `station` table was successfully populated, lon/lat pairs were given geometry encodings. The steps are illustrated in Code Listing 6.5, where `stPoint` is initialised for every lon/lat pair in the `station` table. The `ST_SetSRIS()` function sets the Spatial Reference System Identifier (SRID) of the geometry columns containing lon/lat values. The SRID was defined as a geometry type in step 1, `ST_Point` to the EPSG:4326 projection.

LISTING 6.5: SQL syntax set Point for every lat/lon pair.

```
1 UPDATE public.station
2 SET "stPoint" = ST_SetSRID ( ST_Point (stlon , stlat ), 4326);
```

6.1.2 Complete geographic data store

Populating and configuring the database as described in the previous section finalized the initial work on the PostgreSQL PostGIS database. The remaining part was to run some test queries to confirm the database geographic functionalities. One example query is listed Code Listing 6.6. Running some test scripts with success concluded the PostGIS database to be correct configured as a geographical data store for the Web GIS application.

LISTING 6.6: SQL syntax to test PostGIS data store.

```
1 SELECT "stPoint"
2 FROM station
3 WHERE ST_Distance ("stPoint" , ST_GeomFromText ('POINT (100 200)←
      ', 4326))
4 < 10
```

6.2 Data accessor

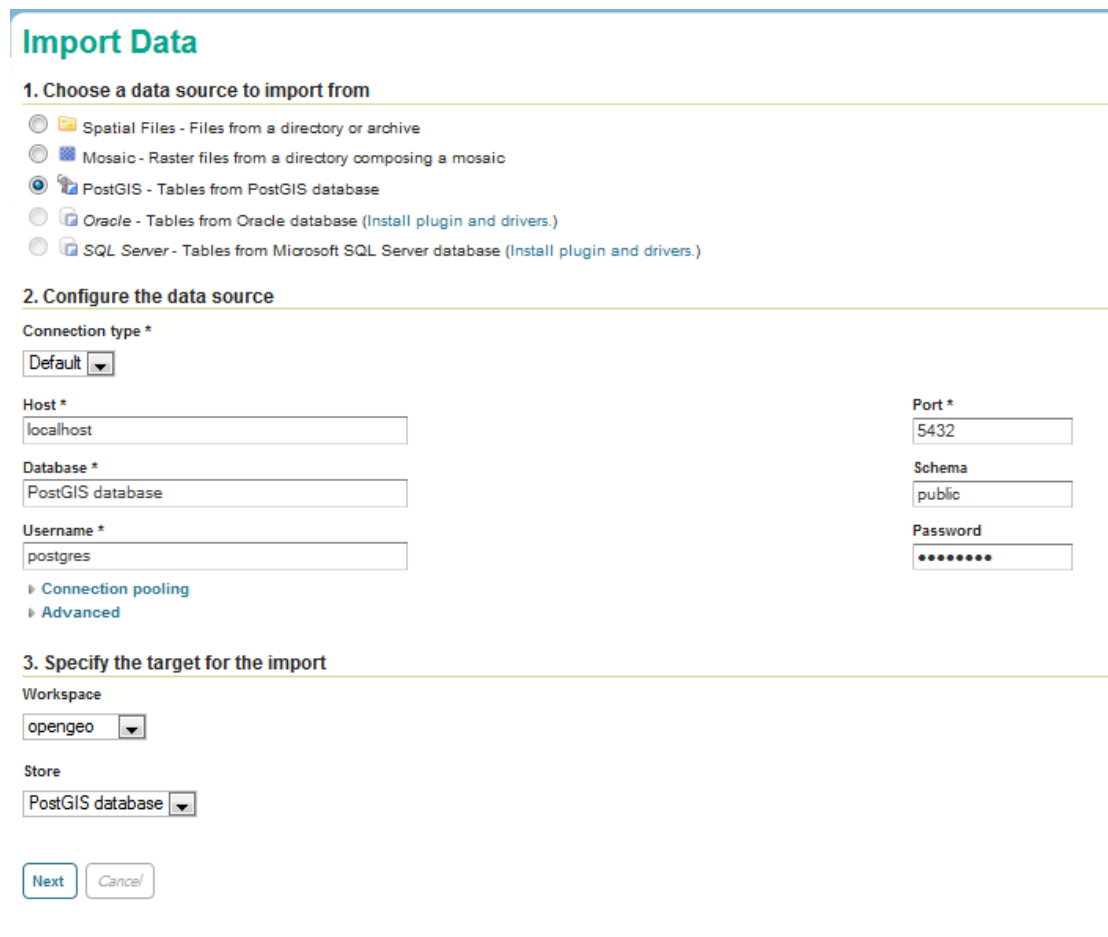
The installation of GeoServer web server was included in the OpenGEO suite. GeoServer is the web server providing the OWS support between GeoServer and the Web GIS client. Figure 2.1 illustrates the data flow between the PostGIS database, GeoServer and the Web GIS client.

6.2.1 Import data to GeoServer

GeoServer has native support for importing PostGIS database tables, shapefiles, Geo-TIFF files as well as other formats, and enables simple importing of geographic data.

Geographic data are imported through the GeoServer *Layer Importer* interface or the *Layer Importer* REST API.

The *Layer Importer* interface in Figure 6.1 was used to import the `station` table into GeoServer. The table was stored as a layer and was given a set of properties necessary to publish the layer.



The screenshot shows the 'Import Data' form in GeoServer. It is divided into three main sections:

- 1. Choose a data source to import from:** This section contains five radio button options:
 - Spatial Files - Files from a directory or archive
 - Mosaic - Raster files from a directory composing a mosaic
 - PostGIS - Tables from PostGIS database** (This option is selected)
 - Oracle - Tables from Oracle database (Install plugin and drivers.)
 - SQL Server - Tables from Microsoft SQL Server database (Install plugin and drivers.)
- 2. Configure the data source:** This section contains several input fields:
 - Connection type ***: A dropdown menu with 'Default' selected.
 - Host ***: A text input field containing 'localhost'.
 - Database ***: A text input field containing 'PostGIS database'.
 - Username ***: A text input field containing 'postgres'.
 - Port ***: A text input field containing '5432'.
 - Schema**: A text input field containing 'public'.
 - Password**: A text input field containing masked characters (dots).
 - Below these fields are two links: 'Connection pooling' and 'Advanced'.
- 3. Specify the target for the import:** This section contains:
 - Workspace**: A dropdown menu with 'opengeo' selected.
 - Store**: A dropdown menu with 'PostGIS database' selected.
 - At the bottom are two buttons: 'Next' and 'Cancel'.

FIGURE 6.1: Import data into GeoServer.

While choosing a data source to import from, a set of connection parameters is prompted by GeoServer to connect to the data source. Then the *Import Loader* scans the data source, the PostGIS database for available geographic enabled data tables and prepare for import. GeoServer saves the PostGIS connection as a data store to use in future. Figure 6.1 illustrates the *Importer Layer* interface while importing data to GeoServer.

The *Layer Importer* also supports other data formats as GeoTIFF and Shapefiles. Other database supported are Oracle and Microsoft SQL Server.

6.2.2 Providing geographic data through GeoServer

As Figure 2.1 illustrates, GeoServer provides geographic data to the Web GIS client through WMS and WFS. WMS and WFS protocols are OGC standardisations for providing geographic data as described in subsection 4.4. The `station` table imported to GeoServer had to be configured to include the right geographic parameters including the SFA point. Geographic data layers in GeoServer are available by HTTP request, see Code Listing 4.2.

6.2.3 Providing non geographic data through GeoServer

The solution presented in [2] leverage the WFS protocol with special GeoServer capabilities, SQL Views. A SQL View is a virtual table based on the result of a SQL query to existing tables. Using parameter substitution inside GeoServer, the WMS Vendor parameter `VIEWPARAMETERS` can be used to replace part of the SQL View query [2]. Non geographic data, measurement values of temperature and salinity is paired with their station parameter which incorporate their respective geographic data.

Create SQL View

A SQL View is created in the GeoServer user interface. Figure 6.2 and Figure 6.3 illustrates a SQL View created in GeoServer. Code Listing 6.7 shows a SQL syntax to create a new SQL View. This SQL view was used in the functionality described in section 8.7. The parameters in line 7 describes the dynamic parameters within a range between `%low%` and `%high%`. Figure 6.2 illustrates the regular expression which validate the input parameter to allow only positive floating numbers. The parameters are used in the implementation code to dynamically set parameters for the view. See Code Listing 6.8 for implementation usage. Line 4 specifies parameter list for a contour layer based on the parameters `%low%` and `%high%`.

LISTING 6.7: SQL syntax to create SQL View in GeoServer.

```
1 SELECT p_temperature.level, p_temperature.value, station."↵
    stPoint", station.stcountryname, station.stvesselname, ↵
    station.stsource
```

```
2 FROM p_temperature
3 JOIN station
4 ON p_temperature.absnum = station.absnum
5 WHERE p_temperature.level BETWEEN %low% AND %high%
```

LISTING 6.8: SQL View used in implementation.

```
1 var contourParams = {
2     LAYERS: 'opengeo:stationTemperatureAtLevel',
3     TILED: false,
4     VIEWPARAMS: 'low:' + low + ';high:' + high
5 };
6 var contour = new ol.layer.Image({
7     title: 'Contour at' + low + ' to ' + high + 'm',
8     source: new ol.source.ImageWMS({
9         url: 'http://localhost:8080/geoserver/wms', //url ←
10         Geoserver request
11         params: contourParams,
12         serverType: 'geoserver',
13         visible: true,
14         opacity: 0.2
15     })
16 });
```

This is a high level description of SQL Views. A more detailed description, some comments about an alternative solution and the security aspect of using SQL Views can be read in [2]. Several SQL Views was created and used for different purposes during this client implementation.

Edit SQL view

Update the definition of the SQL view and its metadata

View Name

allTemperaturesAtLevel

SQL statement

```
SELECT p_temperature.level, p_temperature.value,  
station."stPoint", station.stcountryname,  
station.stvesselname, station.stsource  
FROM p_temperature  
JOIN station  
ON p_temperature.absnum = station.absnum  
WHERE p_temperature.level BETWEEN %low% AND  
%high%
```

SQL view parameters

Guess parameters from SQL Add new parameter Remove selected

<input checked="" type="checkbox"/>	Name	Default value	Validation regular expression
<input type="checkbox"/>	low	5.0	^[d\.]+-eE]+\$
<input type="checkbox"/>	high	10.0	^[d\.]+-eE]+\$

☐ Escape special SQL characters

FIGURE 6.2: SQL View in Geoserver.

Attributes

Refresh ☐ Guess geometry type and srid

Name	Type	SRID	Identifier
absnum	Integer		<input checked="" type="checkbox"/>
stPoint	Point	4326	<input type="checkbox"/>
level	BigDecimal		<input type="checkbox"/>
value	BigDecimal		<input type="checkbox"/>
flag	Short		<input type="checkbox"/>

Save

Cancel

FIGURE 6.3: SQL View in Geoserver.

Chapter 7

Design and implementation of prototype

This chapter describes the implementation steps of the Web GIS client. The Web GIS software technologies are introduced in chapter 4. The geographical technologies with installation, configuration and population of the data store and data accessor is presented in chapter 6. This chapter starts by describing the architecture and design of the Web GIS client. Then AngularJS components, principles and patterns are defined before the client components are presented. In the end testing and quality assurance are discussed.

7.1 Architecture

Figure 7.1 highlight the three architectural layers in the Web GIS client. The prototype components are divided by single responsibility principles. Each component should have one responsibility to attain a modular applications as described in subsection 7.4. This architecture decouple the components from each other, make it more testable and loosely coupled.

The components are defined in a UML class diagram in Figure 7.5. It illustrate the scope of the components, the view, controllers, services and its dependencies to each other. The next sections describe the components and its purpose in more detail.

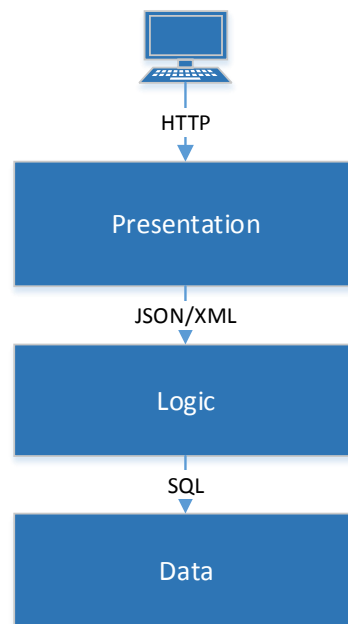


FIGURE 7.1: Web GIS client architecture.

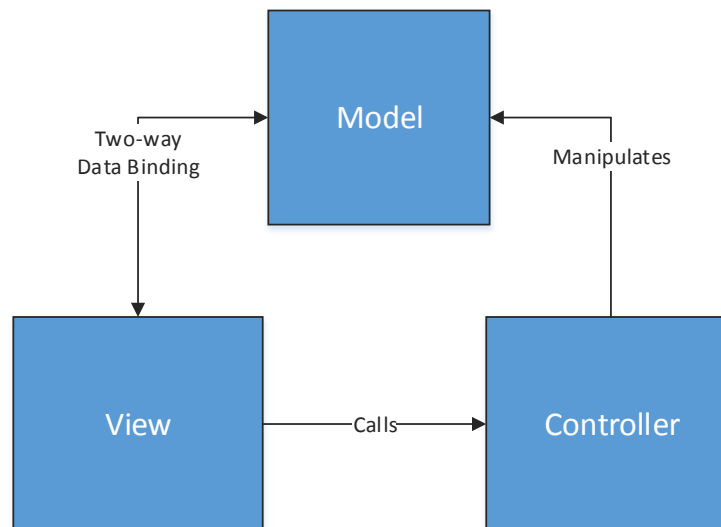


FIGURE 7.2: Interaction between the MVC components.

Each component is designed to be simple and modular implementing software design patterns and principles described in subsection 7.4.1.

The design implements the MVC pattern which separate the application into model, view and controller components such as described in 7.4.1. Figure 7.2 divides the components of the application into the AngularJS MVC pattern.

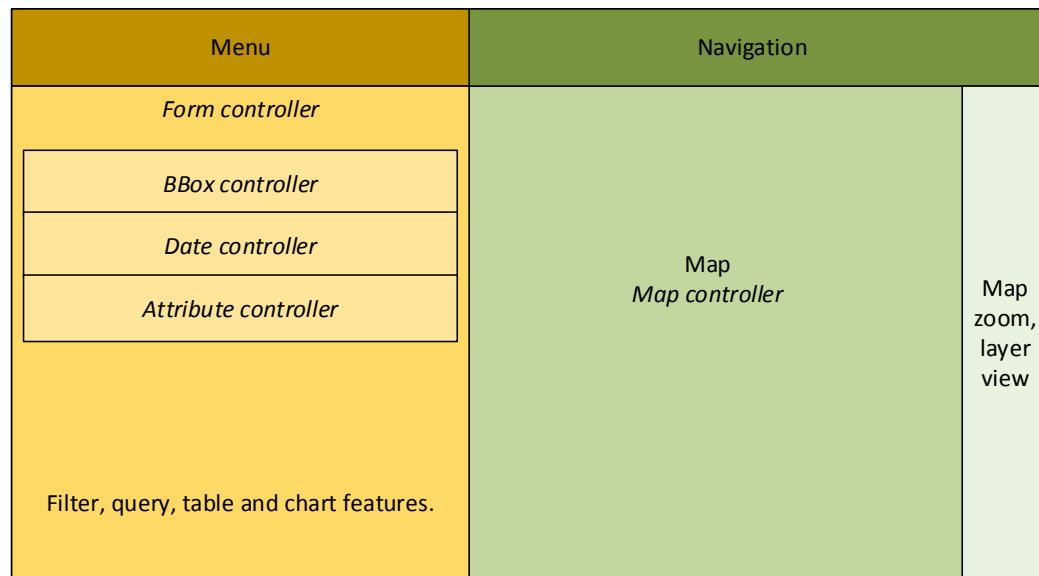


FIGURE 7.3: Client prototype conceptual model.

7.2 User interface design

The client prototype is designed as a simple web page application as specified in Figure 7.3. Each section representing the user interface components. The navigation menu is in the top left of the page and whenever clicked the menu is opened/closed by the AngularJS Bootstrap Accordion directive [92]. The map has an absolute position in the center of the view and is accessible whenever needed.

7.3 AngularJS components

An AngularJS module is a container including the components for a specific part of the application. Figure 7.4 illustrates the module in the architecture of the AngularJS components. The module calls a `config()` method to set up the routes paths for the application. The routes includes controller and HTML template.

The prototype is a one module application. In a larger more complex application it would be proper to have a module for feature as a container for its components.

The Web GIS client **Model**, **View** and **Controller** components are introduced in the next sections.

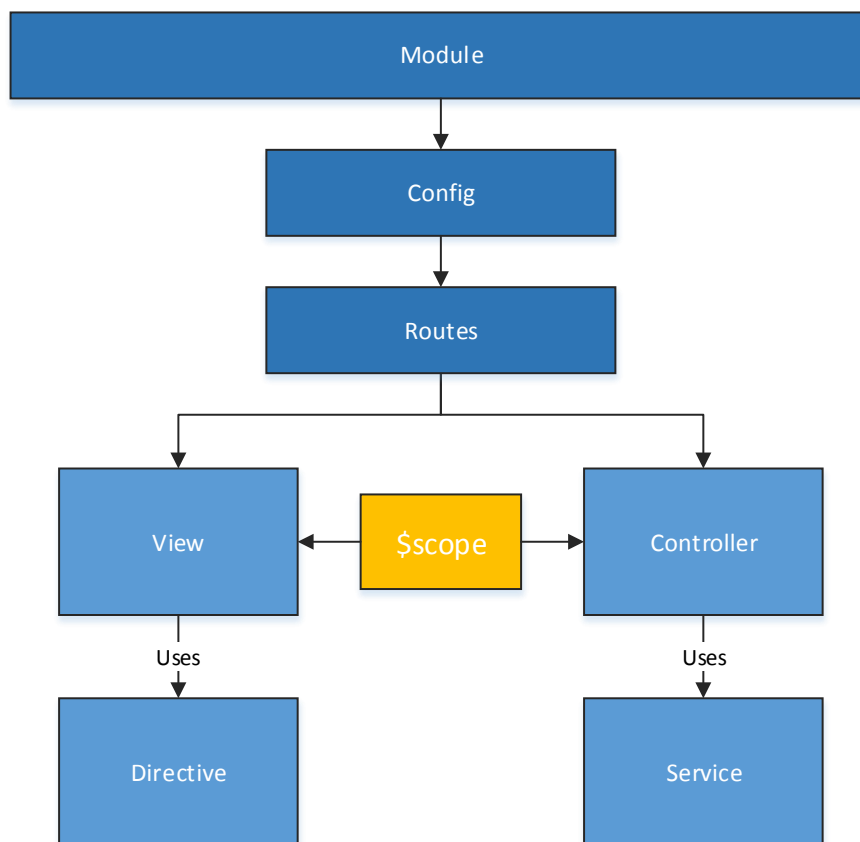


FIGURE 7.4: Overview of interaction between the AngularJS components.

7.3.1 View

The view is a one page HTML document illustrated in Figure 8.2. Filtering and searching is done through the view. DOM manipulation is mainly done in AngularJS directives and services. Behaviours in the DOM are handled with AngularJS directives, which extend the view with attributes and elements. Business logic is restricted to the controller discussed in subsection 7.3.4.

7.3.2 Model

The model is maintained in the controllers and services through the `$scope` object. The Model is a pure JavaScript object which by `."` notation refers to properties of the object. AngularJS implements two-way data-binding. Two-way data-binding changes the model in the view and in the controller. By binding a model variable to the a HTML element, the model variable is displayed and modified by the HTML element. It enables

the model to update the view and the view to update the model. It is illustrated in Figure 7.2 and in line 1 in Code listing 7.1.

LISTING 7.1: Example of two-way data-binding.

```
1 <strong>First name:</strong> {{firstName}}
2 <label>Set the first name: <input type="text" ng-model="↔
  firstName"/></label>
```

7.3.3 Directive

Directives contains business logic for behaviours in the DOM such as displaying HTML, validation, converting and arithmetic operations. Directives create behaviours by attaching a marker on the DOM. Code listing 7.2 illustrates a directive which write a text to the DOM. Line 1-6 is the directive while line 8 and 9 describes two examples of how to use the directive in the view.

LISTING 7.2: Simple directive adding a text string to the view.

```
1 app.directive('helloWorld', function() {
2   return {
3     restrict: 'AE',
4     replace: 'true',
5     template: '<h3>Hello World!!</h3>'
6   };
7
8   <hello-world/>
9   <div hello-world></div>
```

7.3.4 Controller

Attaching a controller to the view is done with the `ng-controller` directive. The AngularJS controller instantiates the `$scope` object for the model and each controller

contains a `$scope` object for its purpose. It decomposes the DOM into testable components where no DOM manipulation should be included. AngularJS controllers add behaviour to the `$scope` object. Good practise implies not referring to any specified DOM elements inside the controller, such as `div ids` or implement business logic as discussed in section 7.4.

The controllers are not big classes, they depend on service and factory components of the application to perform business logic such as change of state or performing `$http` requests.

7.3.5 Service

Managing data through `$scope` is not easy in AngularJS. Scopes can be corrupted and/or changed by controllers and directives. The model in services can be accessed through every controller which has injected the service. The component can be reused, have their own dependencies and be easily tested [40]. AngularJS dependency injection make this achievable. A service is instantiated as a singleton whenever injected, which is usable when one object need to be accessible over several components.

7.4 AngularJS principles and design patterns

AngularJS bootstraps the application and its dependencies automatically. When the application is compiled a `$rootScope` is created. An AngularJS `$scope` is a plain JavaScript object which can attach properties to it. `$rootScope` is available for every code module within the application. Every parent scope can have child scopes, which means that each child inherits from its parents scope. Every controller has a `$scope` attached to it. Code Listing 7.3 describes how the `$scope` is defined by the `ng-controller` in the view.

LISTING 7.3: Adding a scope to the View in AngularJS.

```
1 <div ng-controller="MapCtrl">
2   <div id="map" class="map"></div>
3 </div>
```

AngularJS has one and two-way data-binding which can change the model value. AngularJS removes all DOM manipulation from the view into directives. The controllers are containers for the model, they instantiate variables and perform simple operations for the current model in the DOM while directives performs the business logic. Services are code modules for reusable code throughout the application. Services does not depend on scope and can be injected to every component depending on it. Figure 7.2 illustrates the AngularJS MVC pattern with the components described in subsection 7.3.

7.4.1 AngularJS design patterns

AngularJS follow design patterns which separate the JavaScript code into business and presentation logic, independent modules which can be injected whenever needed.

Model Viewer Controller

MVC is an architectural pattern for implementing user interfaces [93]. Figure 7.2 presents the interaction between the MVC components of AngularJS. MVC separates the components concerns Model represent the scope, View is the HTML and the Controller contains the business logic [40].

Dependency injection

The code is modular by the principle of dependency injection [40]. This design pattern divide the code into modules and explicitly defines what injected modules are present. Every reusable module can easily be exchanged for another module. Dependency injection is important for unit testing, to isolate behaviour and substitute with e.g mocks. It also increases the performance by injecting only dependent modules by the specific operation.

Singleton pattern

The singleton pattern [94] is a design pattern, where a single instance of an object is created. The singleton is maintaining state and properties of the model object through the application. E.g the `Map service` instantiates a map object which is coordinated

through the application. See subsection 7.7.3 for the implementation concepts of the Map service.

7.4.2 AngularJS Service

Promises

AngularJS promises `$q` [42] is an AngularJS service for handling asynchronous functions and the return values. AngularJS `$q` library is implementing promises and deferred objects. A service which run functions asynchronously which use the returning values when the processing is done. The returning values has have a set of available functions e.g `then()`, `catch()` and `finally()`. `then()` can create chained function calls to control the handling of asynchronous function calls. Code listing 8.1 on page 94 illustrates the chained function where some asynchronous calls has to be completed before the next can proceed.

7.5 Front-end frameworks and libraries

7.5.1 AngularJS

AngularJS is a modular web framework. The modules define how the application is bootstrapped, specifies the dependencies and to declare how the wiring and bootstrapping happens. Every dependent module in the client was explicit added in the application configuration file `app.js` specified in Code listing 7.4. `ui.bootstrap` is a module containing native AngularJS directives for user interface elements used in the GUI. The client implementation used AngularJS core `ng` components. Other modules are `ngRoute` a service for linking URLs to controllers and views. Every dependencies as directives or services was included directly as described in Code Listing 7.5.

LISTING 7.4: Module dependencies in `app.js`.

```
1 angular
2   .module('masterApp', [
3     'ui.bootstrap'
4   ])
```

LISTING 7.5: Component dependency definition.

```
1 angular.module('masterApp')
2   .controller('MapCtrl', function ($scope, mapService ) {
3   // some content..
4   });
```

7.5.2 Openlayers

The old Web GIS client was written with OpenLayers 2. A new rewritten version OpenLayers 3 was released autumn 2014. In the new version the API and syntax has been changed and/or removed. OpenLayers 3 was used in this thesis to write the new client even if OpenLayers 2 would satisfy this Web GIS client requirements. OpenLayers 3 has little documentation in comparison to OpenLayers 2 which has a well documented API, books and tutorials. OpenLayers 3 is an open source project with a good community to support future development and it would become time-consuming in a later stage of development to convert from OpenLayers 2 to OpenLayers 3.

7.6 Development challenges

7.6.1 OpenLayers 3

New concepts had to be written to implement the core functionalities of the new Web GIS client. Implementing requests to GeoServer included to filter the request based on the parameter input from the users of the application. The Filter Encoding which creates and encode the filters before they are added to the request. In OpenLayers 3 the filters had to be manually merged by constructing SQL strings before adding the it to the request. Appendix C describes the development changes converting from OpenLayers 2 to OpenLayers 3.

7.6.2 Same origin Policy (SOP)

Requesting remote file (file on a different protocol, domain or port) from JavaScript code demands a connection to the server. Security issues including malicious attacks can occur, the Same origin Policy (SOP) prevents this from happen [95].

Cross-origin resource sharing (CORS), requesting data from the client code, hosted at `http://localhost:9000` to the independent server hosted at `http://localhost:8080`, is not passing the SOP. The request is not granted access to the server because the client and server are considered as different domains by most browsers except Internet Explorer [2].

Implementing cross browser requests can be done by a web proxy as suggested in [2], or using JSONP. JSONP is dynamically firing request call using the `<script>` tag which is not subject of SOP. JSONP can easily be implemented but there are some shortcomings and security. Cross Site Request Forgery (CSRF) attacks can occur, malicious pages can download and alter the data [95].

The new Web GIS client implemented a work around SOP by configuring the Chrome web browser with a CORS add-on which in a controlled development environment allowing the browser to make a request which usually should not be allowed.

7.7 Client prototype components

All components listed in Figure 7.5 on page 71 is described in this section. The components are defined with responsibility and purpose.

7.7.1 Directives

There are two date directives in the prototype implementation. **DataPickerPopup** and **DateConverter**. They are both small directives responsible of creating the date picker in the DOM and validating the date input. They are both directly used by the DOM and have no other dependencies. They are general directives which are reusable components.

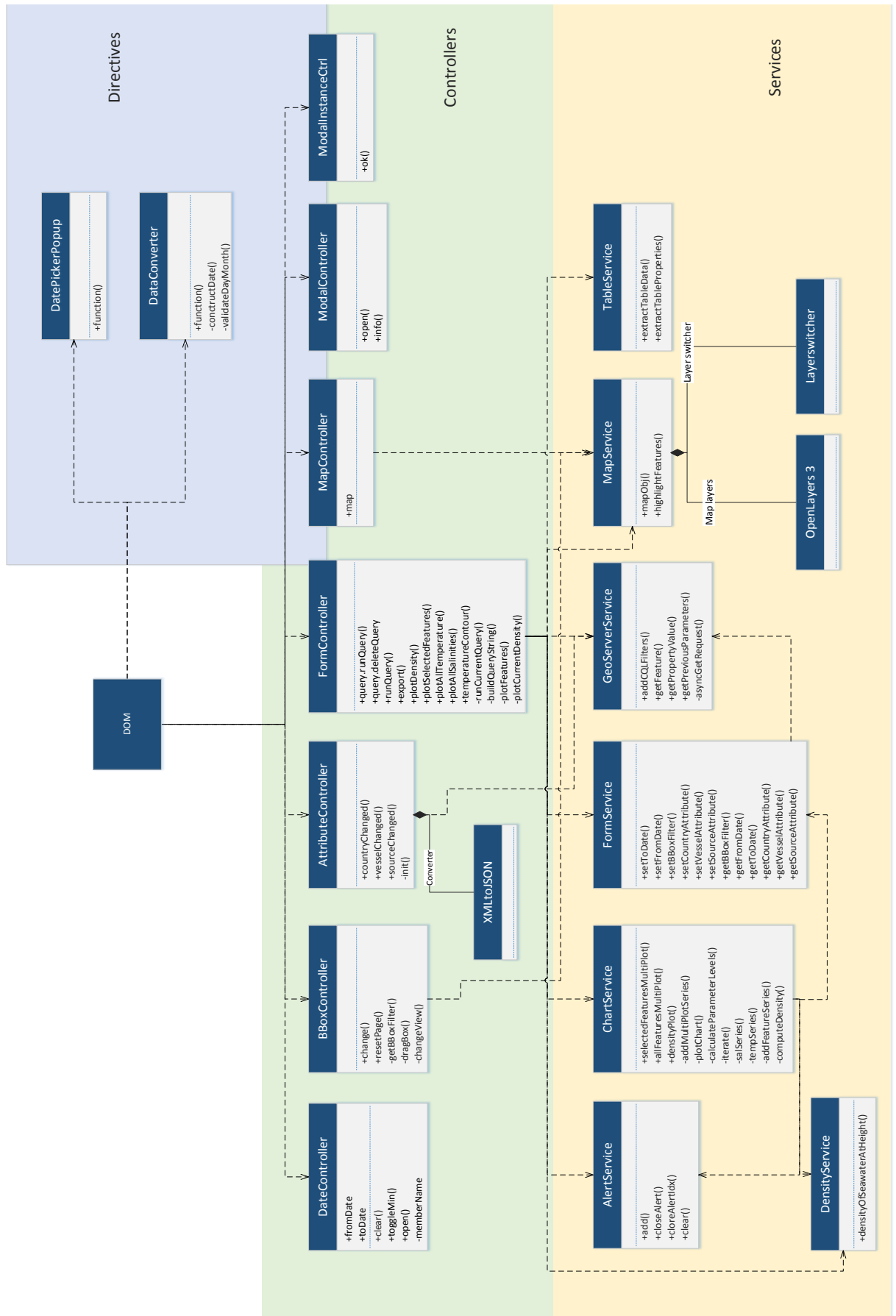


FIGURE 7.5: UML class diagram.

7.7.2 Controllers

The controllers and inheritance is illustrated in Figure 7.3 in page 63.

Form Controller

The `Form controller` connects the input from the form elements in the DOM. It is a container for handling the form variables for DOM elements such as is a element is visible or not, all captured in the `$scope` object. This controller activates every event happening in the menu and manipulates the model and updates the view. Figure 7.3 on page 63 defined the bounds of this controllers scope object.

Bounding Box Controller

The `Bounding Box controller` initialises the variables and methods inside the filter menu for bounding box in the GUI. It validates and keeps track of text input and events triggered by drawing a bounding box with the mouse.

Date Controller

The `Date controller` is initialising, validating and converting the JavaScript Date object used in the date filter, into a format which is comparable and searchable in the database. This is a generic and reusable controller.

Attribute Controller

The `Attribute controller` wrap the attribute filter in the query menu. It fetches the list of parameters from the database in an `init()` call, to dynamically update the list of parameters by the parameters in the database. It keeps track of attributes input and with two-way data-binding updates the parameter list for the current query.

ModalCtrl and ModalInstanceCtrl Controller

These controllers handles the modal widget. The modal widget is a widget used to give information about the Web GIS client. The controllers opens it and handles the content and events such as close button. These are general controllers and reusable components.

7.7.3 Services

Map Service

This singleton service creates and maintain the instance of the map object and the map controllers, zooming, layer-switcher, pointer move and data point pop-up widget. Code Listing 7.6 creates a layer by a WMS request to GeoServer, see details of parameter list in subsection 4.4.2. Code Listing 7.7 creates the map object and includes the layer created in Code Listing 7.6. The map object is a singleton which are dynamically changed by adding/deleting new layers and is updated in every controller injecting the `Map service`.

LISTING 7.6: Initialise and create map layers.

```
1 var mapLayers =
2     new ol.layer.Image({
3         title: 'Measurements',
4         source: new ol.source.ImageWMS({
5             url: 'http://localhost:8080/geoserver/wms',
6             params: {'LAYERS': 'opengeo:station',    VERSION: '1.1.1' ←
7         }, 'TILED': true},
8         serverType: 'geoserver',
9         visible: true,
10        opacity: 0.2
11    }),
12    renderOptions: {
13        zIndexing: true
14    }
15    });
```

LISTING 7.7: Create map object.

```
1 map = new ol.Map({
2     target: 'map',
3     renderer: 'canvas',
4     layers: [
5         new ol.layer.Group({
6             'title': 'Base maps',
```

```
7         layers: [  
8             new ol.layer.Tile({  
9                 title: 'OSM',  
10                type: 'base',  
11                visible: false,  
12                source: new ol.source.OSM()  
13            })  
14        ],  
15    },  
16    new ol.layer.Group({  
17        title: 'Overlays',  
18        layers: [mapLayers]  
19    })  
20 ],  
21 view: view,  
22 controls: ol.control.defaults().extend([  
23     new ol.control.ScaleLine(), new ol.control.ZoomSlider() ←  
    // ScaleLine in the bottom of the map  
24 ])  
25 });
```

The `Map` service has public methods supporting the map functions: `setFilter()` and `highlightFeatures()` to change and or add new layers into the map. The `Map` service is a reusable and independent component as illustrated in Figure 7.5.

GeoServer Service

`GeoServer` service is an independent AngularJS service component which main purpose is to send HTTP requests to GeoServer. It is dependent of `$http` service which communicates with the web service by `XMLHttpRequest`. The `GeoServer` component uses the AngularJS core service `$http` to build requests and send them to GeoServer.

`GeoServer` has public methods which performs different HTTP requests. Figure 7.5 displays this service and its private and public functions. This component can easily be extended to send other requests or be changed with some other component.

Code Listing 7.8 and 7.9 specifies a request to GeoServer with additional parameters. The request is a query string as seen in Code Listing C.3 in page 137, how it is handled in the implementation is described in Code Listing 7.10. It describes a response with `$http` methods `success()` and `error()` which handles the callback from GeoServer. The parameters are `data`, `status`, `statusText` in both cases.

LISTING 7.8: Get data from GeoServer.

```
1
2 // fires a GetFeature WFS request to server
3     getFeature: function (extraParameters, callback) {
4         // some default parameters that will be set ↵
5         automatically if not
6         // overridden in extraParams
7         var parameters = {
8             REQUEST : 'GetFeature',
9             SERVICE : 'WFS',
10            VERSION : '1.0.0'
11        };
12
13        // extend provided parameters onto default parameters, ↵
14        make request
15        return asyncGetRequest(angular.extend(parameters, ↵
16        extraParameters), callback);
17    }
```

LISTING 7.9: Asynchronous request.

```
1
2 // fires an async. HTTP GET request to server
3     var asyncGetRequest = function(parameters, callback) {
4         previousRequestParameters = parameters;
5
6         return $http({
7             url: WFSserver,
8             method: 'GET',
```

```
9      params: parameters ,
10      callback: callback});
11  };
```

LISTING 7.10: Use of GeoServer service.

```
1  GeoserverFactory.getFeature({
2      TYPENAME : 'station',
3      OUTPUTFORMAT : 'json',
4      SRSNAME : 'EPSG:4326',
5      CQL_FILTER : filter
6  }).success(function(data, status, statusText ) {
7      // some code....
8  }).error(function(data, status, statusText ){
9      $log.error('Error in request to server' + status + 'Status ←
      text from server ' + statusText );
10  });
```

Form Service

Form service is a model for the form elements and its properties. It is designed to be a setter/getter service to maintain access to properties across scopes in controllers.

Alert service

Alert service instantiates a global alert service for every component injecting it. Every alert is customisable by a set of parameters **type**, **msg**, **timeout**. **type** defines what kind of alert it is e.g warning or success, **msg** is the actual alert message and **timeout** is parameter for how long the alert should be displayed. The alerts are attached to the **\$rootScope** which is available for every code component and independent of defined controller scope. This service is a reusable utility component.

Density service

The **Density service** has a public API which provides the density calculation of temperature and salinity measurements. Code Listing 7.11 specifies the public function call and parameters used. This service is independent and fully reused from the old Web GIS client application [2]. Code Listing 7.12 specifies the call from **Chart service** in line 2 where the parameters for density, salinity value, temperature value and sea level are included in the calculations. Illustration and demonstration of this feature is discussed in subsection 8.6.2. Detail implementation for **Density service** is refereed reading to [2].

LISTING 7.11: Calculate density.

```
1 // Public API here
2     return {
3         densityOfSeawaterAtHeight: function (salinity, temperature↵
4         , height) {
5             return densityOfSeawater(salinity, temperature,
6                 gaugePressureInSeaAtDepth(height));
7         }
8     };
9
```

LISTING 7.12: Example of calculating density inside **Chart service** .

```
1     computedData[i].properties.level = level;
2     computedData[i].properties.value = densityFactory.↵
3     densityOfSeawaterAtHeight(salVal, tempVal, level);
4
```

Table service

The **Table service** has two public functions. Code Listing 7.13 instantiates the column properties of a data table in the view. It uses a JavaScript function to iterate the list of JSON objects and fetches the object data before returning the data. Line 5 gets each object property for the first JSON object in the data list. The other public function `extractTableData(JSONdata)` extracts data from a HTTP JSON response.

LISTING 7.13: Extract table properties.

```
1 extractTableProperties: function(JSONdata) {  
2     // Construct the table headings from the data  
3     angular.forEach(Object.keys(JSONdata[0].properties), ↵  
        function(key){  
4         // Value of each properties for the Grid headings  
5         colDefs.push({ field : key });  
6     });  
7 }
```

Chart service

Chart service uses Highcharts and constructs charts with different parameters to the view. This service has a list of dependencies which make this service unique for this application. It is not loosely coupled such as the other services and will obviously benefit from decoupling in the future. Still the **Chart service** has a public API which draw the different charts described in subsection 8.6.

7.8 Testing and quality assurance

7.8.1 W3C standard compliance

W3C standards define a quality assurance of HTML5 and CSS3 files through online validators. The validators was easy to use by appending the HTML or a CSS text in an online text field. The HTML file was validated by W3C mark up validation service [96]. The validation had current errors regarding the AngularJS **ng** directive which was altered by adding a **data** attribute in the beginning like **data-ng-app** to get the HTML validate.

The CSS file was validated by W3C CSS3 [97] validator. The CSS file validated without errors in the validator for CSS3 standard compliance. The validator assess the code to assure the quality is updated to the standard compliance.

7.8.2 Unit testing

JavaScript is a dynamical language and hard to debug. It is hard for the compiler to debug errors in asynchronous methods. A test suite is beneficial for building a maintainable and sustainable JavaScript application. The input and output of JavaScript methods must be tested to assure the code is doing what it is supposed to. The client prototype has a test suite. The test suite include sets of test cases covering a part of the code. The test cases are simple component instantiation, state initialisation and unit tests.

The unit test covering black box test between components assuring the interfaces between the components works as expected. Code Listing 7.14 specifies a test case for the `Map controller`. The test verifies if the `$scope.map` variable in the `Map controller` is initialised by calling the `highlightFeatures()` of the `Map service`. Code line 4-10 is the set up method for the test cases where dependencies are injected in the first line. Line 6-8 set the spy on and adds the `andcallFake()` method to return the string `'map'`. A Jasmine spy is a test double [98] function which mocks the `highlightfeatures()` method with specified values for testing. Line 9 initialises the controller being tested and sets the dependencies. Line 12-14 is the test method expecting the scope variable `map` being set to `'map'` when the controller is initialised.

LISTING 7.14: Example of test case.

```
1
2     var scope, MapCtrl;
3
4     beforeEach(inject(function ($controller, $rootScope, ↵
mapService) {
5         scope = $rootScope.$new();
6         spyOn(mapService, 'highlightFeatures').andCallFake(function ↵
() {
7             return 'map';
8         });
9         MapCtrl = $controller('MapCtrl', { $scope: scope, mapService↵
: mapService });
10    });
11
```


Component	License
WebStorm	Academic License
Yeoman	Non
Grunt	Non
Bower	Non
Travis	Non

TABLE 7.1: Development tools used in implementation

manipulation by focusing on document traversal, event handling, animation and AJAX interactions [99].

AngularJS and jQuery have different approaches for handling DOM manipulation. JQuery couples the logic in the controller to DOM with `div` tags, AngularJS decouples DOM and the logic by introducing directives by the use of `$scope` variables. See subsection 7.4 for more on AngularJS DOM manipulation.

The rewriting of client code resulted in a completely new implementation code and structure. AngularJS uses part of the jQuery library but has its own implementation features for building interactive web applications as described in subsection 4.5.2. AngularJS is a web framework with unique syntax. It is a framework which give guidelines how the structure and implementation code should appear in difference to jQuery which in a JavaScript library for feature rich DOM manipulation.

7.10 Front-end development environment

The Web GIS client was implemented on a Windows 7 64 bit personal computer including 4GB RAM, 2,4GHz processor. Subsection 4.5.6 and 4.5.7 describes the development environment and Figure 7.7 represent the client technologies including development environment. Development tools are displayed in Table 7.1.

A software development environment was constructed as a supporting element to iterative development and CI. The build server, Travis CI was connected to a GitHub repository for continuous testing of newly integrated code. WebStorm was managed by an automatic build tool Grunt in addition to Yeoman, a code generator and a package manager Bower. Read subsection 4.5.5 for technology description.

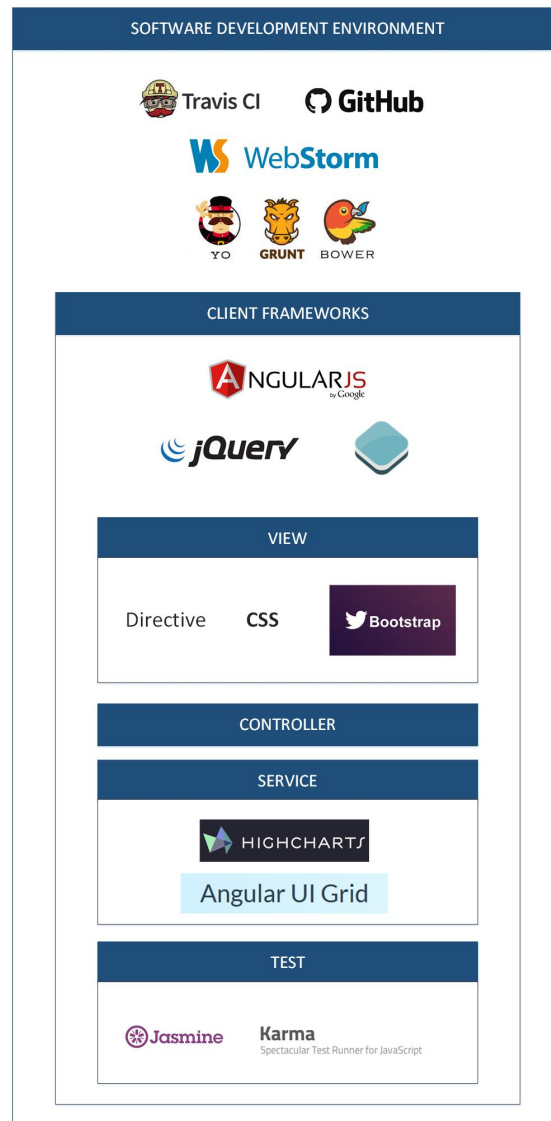


FIGURE 7.7: Software technology in the presentation layer

7.10.1 Continuous Integration

Travis CI support was successfully configured and linked with GitHub. Travis has good integration support for GitHub. GitHub was the version control for the project, all code pushed to Travis was built and all unit tests were tested. The CI and interaction between GitHub and Travis was an important factor to assure the code base was running as expected.

Chapter 8

System demonstration

This chapter describes the features of the client prototype. All core features are described by examples illustrating the GUI and specifying the interactions between the code components.

Implementation concerns such as the installation and configuration of geographic database and web server is referred to chapter 6. Design and implementation decisions are discussed in chapter 7 and the choice of software technologies are presented in chapter 5.

8.1 Web GIS client front page

The Web GIS front page displays a map with a set of data points. The data points are scoped to temperature and salinity measurements performed by Norwegians during the year of 2003. Figure 8.1 illustrates the front page GUI. In the top left side is the menu button, an accordion menu which extends when clicked. Figure 8.2 displays the extended menu with its content. The front page does not scroll other than the menu content.

8.2 Display map and its controllers

The map and its controllers are as illustrated in figure 7.3 a large part of the web page. The map controllers for zooming, switching background or data layer and mouse pointer

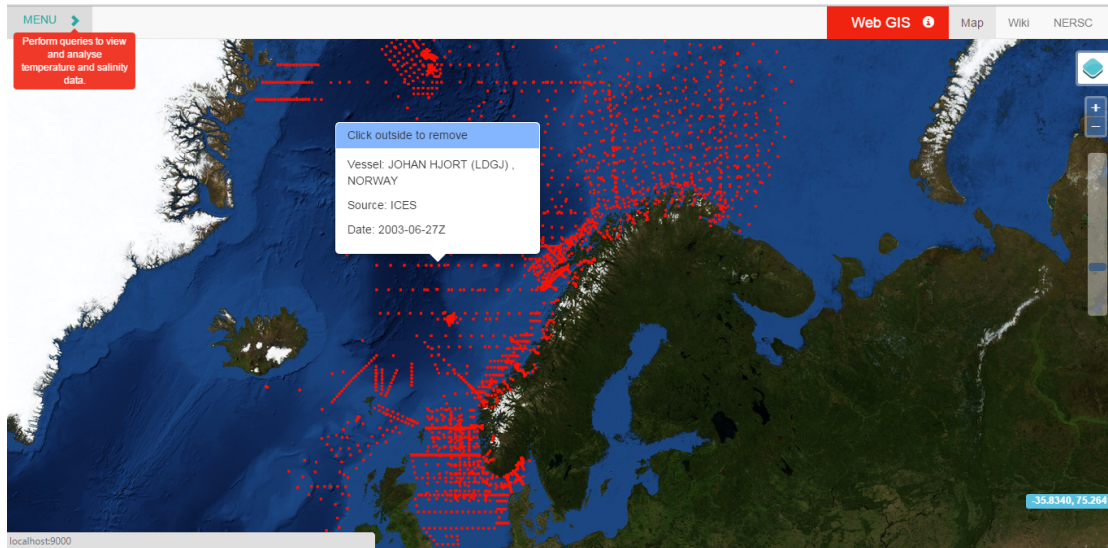


FIGURE 8.1: Map controller interface.

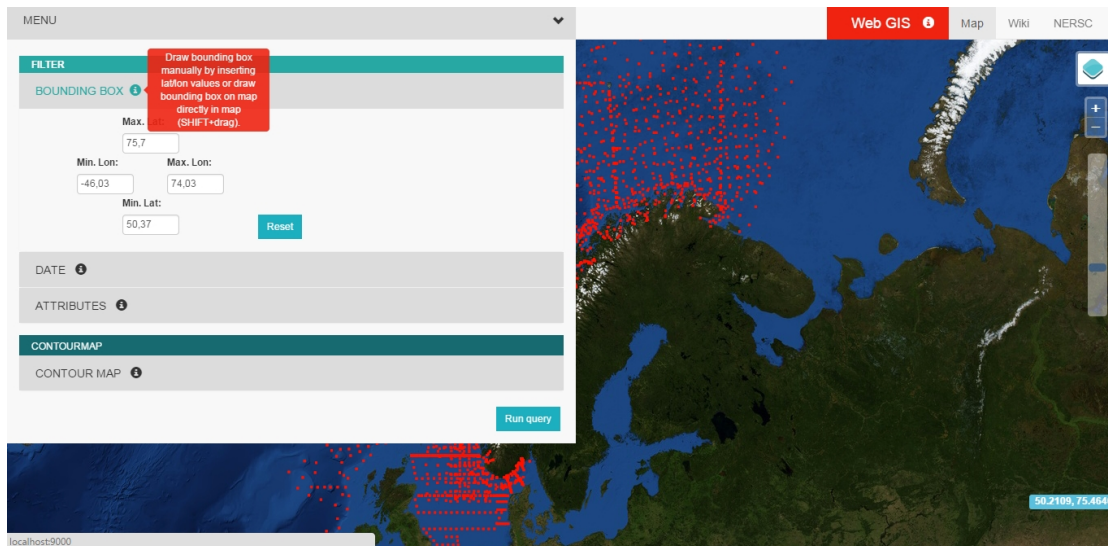


FIGURE 8.2: Map controller interface with menu.

coordinates are available whenever the map is present. Mouse events are also available for scrolling for zooming in and out and panning. Section 7.7.3 describes how the `Map service` component initialises and handles the map object during the application. This section fulfils FR04, which requires a map widget and common map controllers.

8.3 Query data, search in data

A dataset query is initialised when the **Run query** button inside the menu is clicked. The query can be customised with a set of filters as illustrated in Figure 8.5. The button

event builds a request by collecting all filter attributes from the **Form service** and builds a filter string attached as a parameter to the `getFeature()` function to the **GeoServer service** component. The response from **GeoServer service**, a callback containing a JSON object. Figure 8.3 illustrates the interaction between the client and geographic web server and database during a request procedure. This functionality fulfils FR01, Query data by a set of parameters.

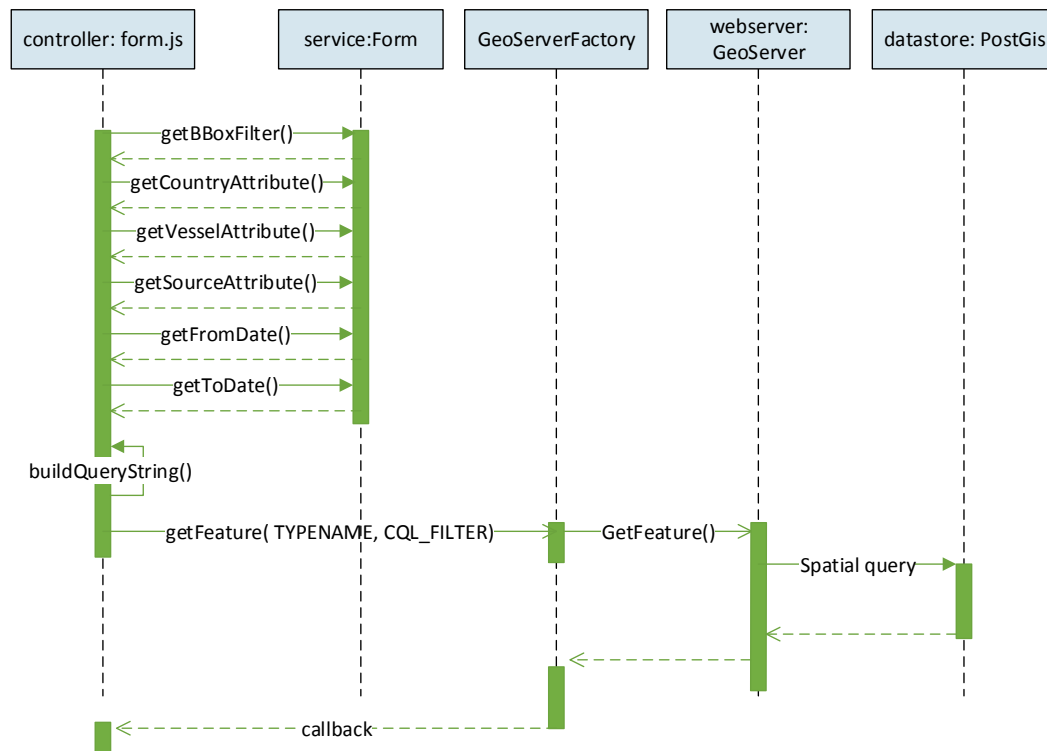


FIGURE 8.3: Sequence diagram of requesting filtered data.

The design of the query, GUI has evolved during the iterative UCD process. Figure 8.4 illustrates the first GUI design. Feedback from the users has evolved the GUI. Figure 8.5 defines the latest menu GUI designed to improve the usability for the users.

Usability testing resulted in important feedback which emerged into new functionality, as described in subsection 5.4.4. A new functionality of previewing resent queries and their filters was developed into a new user story. The users wanted to customise the query function for optimality.

User story

The image shows a web-based GUI titled "Filter" with a dropdown arrow in the top right corner. It contains three main sections: "Bounding box", "Time", and "Attributes". The "Bounding box" section has four input fields arranged in a 2x2 grid. The top row contains "Min. Longitude:" with the value "-104.564453125" and "Max. Latitude:" with the value "81.43353827677157". The bottom row contains "Min. Latitude:" with the value "27.799882937477804" and "Max. Longitude:" with the value "132.56445312500003". Below these fields is a blue "Change" button. The "Time" and "Attributes" sections are currently empty. At the bottom of the "Filter" panel is a "Query" button.

FIGURE 8.4: First GUI design of the bounding box controller interface.

The image shows a more complex web-based GUI. At the top is a "MENU" bar with a dropdown arrow. Below it is a "FILTER" section with a teal header. Under the "FILTER" section is a "BOUNDING BOX" section with an information icon. This section contains four input fields: "Max. Lat:" with the value "75,7", "Min. Lon:" with the value "-46,03", "Max. Lon:" with the value "74,03", and "Min. Lat:" with the value "50,37". A blue "Reset" button is located to the right of the "Min. Lat:" field. Below the "BOUNDING BOX" section are three sections: "DATE" with an information icon, "ATTRIBUTES" with an information icon, and "CONTOURMAP" with a teal header. Under the "CONTOURMAP" section is a "CONTOUR MAP" section with an information icon. At the bottom right of the entire interface is a blue "Run query" button.

FIGURE 8.5: Final GUI design bounding box controller.

As a user I want to rerun an old query from a list of my resent queries so I don't have to re-enter the query filters.

A list of previous queries lets the user reduce time and effort to run the queries again. This procedure is as illustrated in Figure 8.3, initialised with a list of the previous query objects containing all the filter parameters. Figure 8.6 displays the new GUI with a list of queries in the top. Each query in the list can be reran and permanently deleted.

8.4 Display query response in map

For every query to GeoServer the new layer is being redrawn to display the response dataset. The `highlightFeatures(filter)` method in the `Map service` component is called, with the current query filters as parameter. The singleton map object in `Map service` add the new map layer and draws the data points in the browser. This functionality fulfils FR02, Present queried data inside map in browser.

Every data point inside the map widget has a information label attached to it. For every data point clicked, a label containing information of the data point is opened. The data point information is requested by the `GetFeatureInfo()` function, described in subsection 4.4.2. A single click event in `Map service` activates the request and attach the response to a pop-up in the view. This functionality fulfil FR05, which requires a query by a specific point in the map. Figure 8.1 illustrates data point rendered as a WMS raster layer into the map.

8.5 Display response data in tabular form

Figure 8.6 displays the result from a query, non geographic data in a tabular form. The table can filter by parameter, add/remove columns as illustrated in the right drop down menu in Figure 8.14. It can also show information about the number of items, showing items and selected items as seen in the bottom of the table in Figure 8.6. The table enables easy selecting/deselecting one or all rows of data.

`Table service` generates the table properties by calling the method `extractTableProperties(JSONdata)`. Data in the tables are extracted from the response (callback) JSON object by calling the `extractTableData(JSONdata)` method. The Angular UI-Grid is implemented as a container for presenting the dataset.

MENU

QUERY RESULT

Date:2003-01-01|2003-12-31, Country:ICELAND, Vessel:, Source: ▶ ✕

Date:2003-01-01|2003-12-31, Country:NORWAY, Vessel:, Source: ▶ ✕

Result table **Export data** **Plot**

Id	Flag	Lat	Lon	Date	Source	Country	VesselNa...
240124	2048	61.118	-3.276	2003-10-10Z	TRACTOR	NORWAY	G.O. SAR...
240228	2048	61.215	-3.686	2003-10-10Z	TRACTOR	NORWAY	G.O. SAR...
240277	2048	61.251	-8.015	2003-10-08Z	TRACTOR	NORWAY	G.O. SAR...
240286	2048	61.269	-7.996	2003-10-08Z	TRACTOR	NORWAY	G.O. SAR...
240294	2048	61.284	-7.969	2003-10-08Z	TRACTOR	NORWAY	G.O. SAR...
240298	2048	61.3	-7.935	2003-10-08Z	TRACTOR	NORWAY	G.O. SAR...
240306	2048	61.3	-4.107	2003-10-10Z	TRACTOR	NORWAY	G.O. SAR...
240319	2048	61.328	-7.883	2003-10-08Z	TRACTOR	NORWAY	G.O. SAR...
240373	2048	61.383	-4.463	2003-10-10Z	TRACTOR	NORWAY	G.O. SAR...
240407	2048	61.46	-4.826	2003-10-10Z	TRACTOR	NORWAY	G.O. SAR...

Total Items: 1880 (Showing Items: 25)(Selected Items: 6)

FIGURE 8.6: View data result set in tabular form.

8.6 Display dataset in chart

Dataset in tabular form is meta data of current observations, and do not display values of measurements done at the observation position. To display measurements in charts one or several rows of data must be selected to request to view the measurement values in a chart. FR07 requires data represented in charts to compare multiple measurements. FR07 is fulfilled with the set of charts available in this client prototype. Figure 8.8

illustrates the density chart were the y-axis present the depth in sea level and the x-axis represent the values for the temperature, salinity and density.

To display unique temperature, salinity or density data values in a chart, the user has to select the desired datasets in the result table. SQL views to access temperature and salinity values based on a specific `station` are provided by GeoServer. Section 6.2.3 describes SQL view and their purpose. The data response from GeoServer is a parameter to `allFeaturesMultiPlot()` in the `Chart service` component. `Chart service` uses the Highcharts library to create interactive charts displaying the data. Figure 8.7 illustrates the interactivity with the chart, zooming and label of data points.

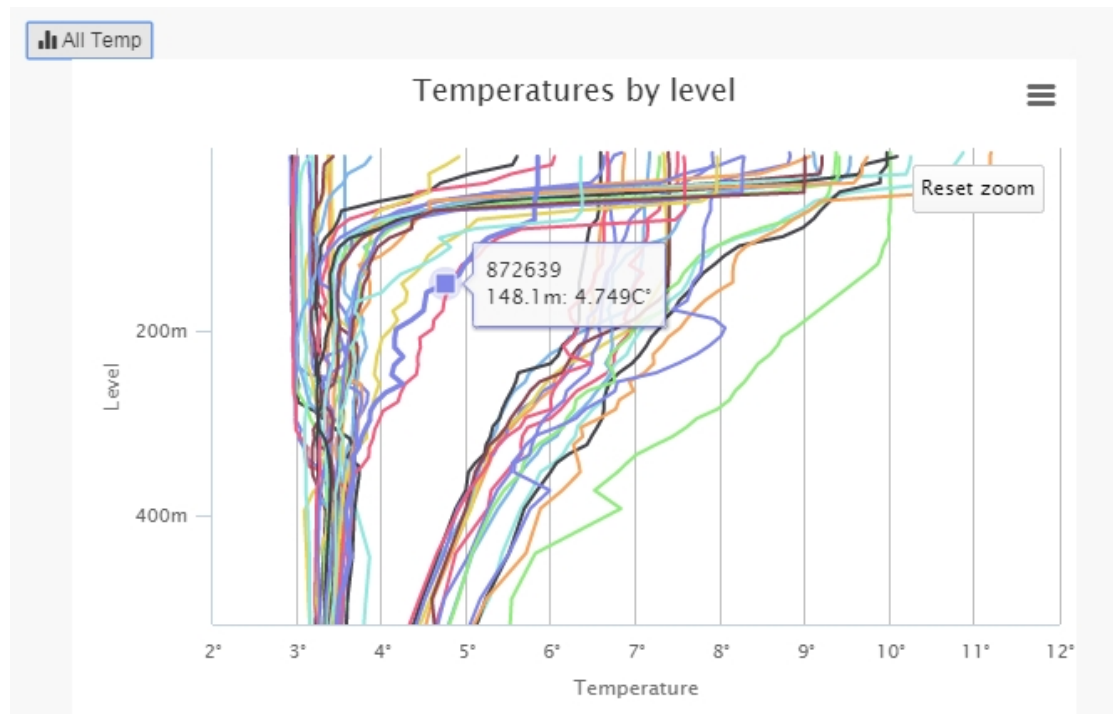


FIGURE 8.7: Zooming and data point label in temperature chart.

8.6.1 Display temperature and salinity

A temperature and salinity measurement value can be generated and displayed in a chart by selecting a row in the data table. It is displayed in Figure 8.8 without the density plot. This functionality is also applicable for displaying all available temperature or salinity measurements available within dataset.

8.6.2 Display density chart

Displaying a density chart includes calculation of one temperature and one salinity value. The result is displayed in a chart together with temperature and salinity values. The `densityPlot(selectedIDs)` method inside `Chart` service activates the `densityOfSeawaterAtHeight(salVal, tempVal, level)` of `Density` service. The `Chart` service renders the density chart in the view.

This functionality implement FR08, which request density calculation of temperature and salinity values and presented into a chart. Figure 8.8 displays a density chart containing both temperature, salinity and density graph.

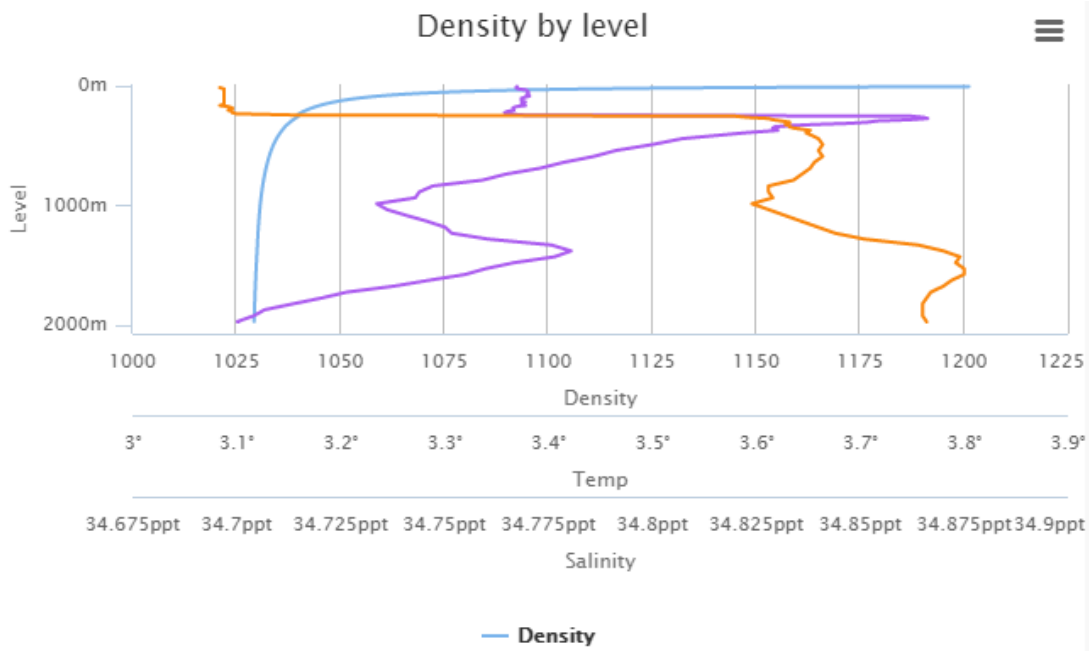


FIGURE 8.8: Density chart for a set of temperature and salinity data.

8.6.3 Calculations for density chart

The calculation for the density chart are based on a selected temperature and salinity value for a data set. The `Density` service does the calculation; the calculation details can be studied in [2]. The density is calculated using a JavaScript method for handling asynchronous request to assure both temperature and salinity levels are available before `densityOfSeawaterAtHeight(salVal, tempVal, level)` is called. Code Listing 8.1 shows the `then()` method which calls next method when the result from the callback is finished.

LISTING 8.1: Handling asynchronous calls in AngularJS.

```

1 calculateParameterLevels(selectedIDs, '↵
    testView_station_temperature')
2     .then(function(data) {
3         // some code...
4
5         calculateParameterLevels(selectedIDs, '↵
    testView_station_salinity')
6     .then(function(data) {
7         // some code...

```

Figure 8.9 shows the interaction between the components during the density plot procedure.

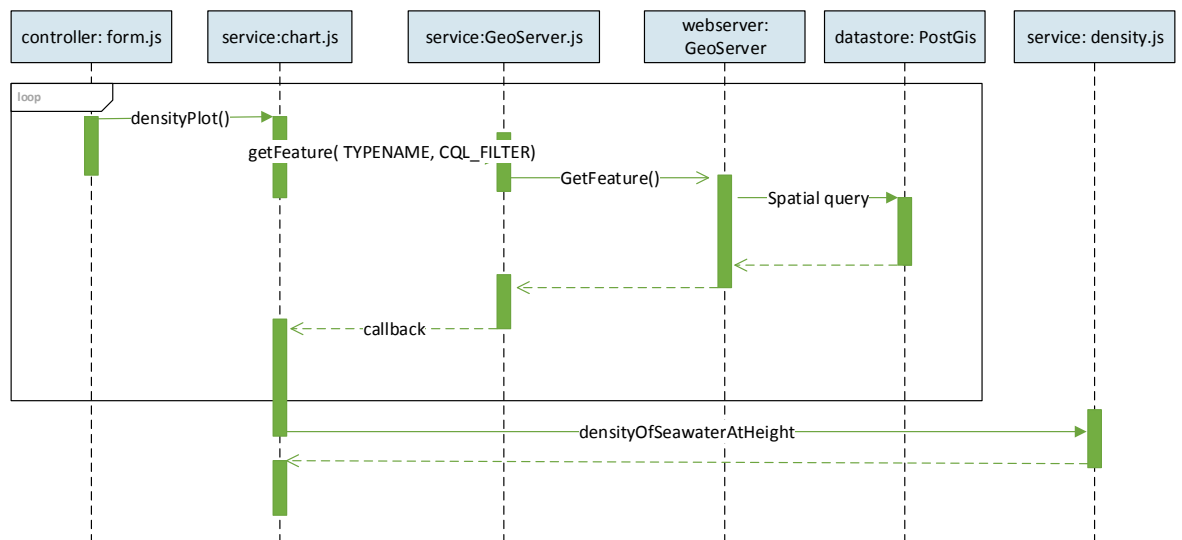


FIGURE 8.9: Sequence diagram illustrating the density calculation procedure.

8.7 Display temperature contour and contour grid

The contour menu is an independent menu from filter and query menu. The contour layer is calculated on all available datasets within the requested range of sea depths. Figure 8.11 shows the GUI inserting range of sea depths for the contour layer. This is a request which uses a SQL view layer in GeoServer to extract the temperature values.

The result is the contour layer including yellow data point where the measurement values are interpolated by. This functionality fulfils FR09, present a contour plot within a range of meters in sea depth.

8.7.1 Calculate and display contour layer

The `temperatureContoure(low, high)` method requests a contour layer with a SQL view as layer in GeoServer. A SQL view layer in GeoServer gathers proper values within the `low` and `high` parameters from the PostGIS database. The response data from the PostGIS database is styled in GeoServer with `Barnes Surface Interpolation` [100], a style layer implemented in GeoServer. The contour layer is added to the map and drawn into the browser window. Data points, geographic locations defining where the measurements were done are rendered into the browser. These data points specifies the location of the data the contour layer is interpolated from.

This feature changed during the testing sessions with the users. Figure 8.10 illustrates the first version and Figure 8.11 illustrates the latest version of the contour layer menu. The differences are not extensive but important for the users to seamlessly explore this contour feature.

User story

As a user I want to render a contour layer with a discrete set of values within a range of meters so I can read the temperature values inside the map.

In addition a color scale displaying the range of temperature in the contour layer was implemented.

8.7.2 Calculate and display contour grid

A contour grid, an alternative to the contour layer was requested by the users. Contour grid are mostly created from raster layers. The process implemented was a dynamic rendering process, where data points were interpolated by `Barnes Surface Interpolation` into a contour layer, then contour grid lines was extracted. All the functions was implemented in one style layer in GeoServer inspired by [101].

The screenshot shows a web interface for setting filters and viewing contourmaps. At the top is a 'Menu' bar with a dropdown arrow. Below it is a section titled 'Set filters to the query' containing three filter options: 'Bounding box', 'Date', and 'Attributes', each with an information icon and a dropdown arrow. Below this is a section titled 'View contourmap by ocean depth' containing a 'Contourmap' option with an information icon and a dropdown arrow. Underneath the contourmap section is a form with an 'Input depth in ocean' field containing the value '100', and two buttons: 'Temperature contour' and 'Salinity contour'. At the bottom of the interface are two blue buttons: 'Run query' and 'Display/Hide query result'.

FIGURE 8.10: Early contour menu before design change.

8.8 Export functionalities

The client has functionality for exporting datasets in text or PDF format and graphs in image format. The export functionalities fulfil FR06, Offer export functionality of requested data. Every table representing data and every chart has export functionalities.

8.8.1 Export dataset

There are several export functionalities available in the client prototype. Feedback from the usability testers requested extended functionality which led to two types of dataset export options, each of them providing different options and format for export.

Export with GeoServer

This export functionality is implemented by sending request to GeoServer by clicking the **Export** button triggering `export(format)` with desired format. A URL string is created with a filter and format for the response as specified in line 1 in Code Listing 8.2. The export is written into a new page in the browser or downloaded depending on the format. Figure 8.13 illustrates the available export formats.

MENU

FILTER

BOUNDING BOX ⓘ

DATE ⓘ

ATTRIBUTES ⓘ

CONTOURMAP

CONTOUR MAP ⓘ

Input depth in ocean (range from 0 to 3000 meters (m))

From level: To level: **Temperature contour**

Scale of temperature in celsius

0 1 2 3 4 5 6 7 8 9 10

Display/Hide query result **Run query**

QUERY RESULT

Date:2003-01-01 2003-12-31, Country:ICELAND, Vessel:, Source:	▶	✕
Date:2003-01-01 2003-12-31, Country:NORWAY, Vessel:, Source:	▶	✕
Date:2003-01-01 2003-12-31, Country:CANADA, Vessel:, Source:	▶	✕

FIGURE 8.11: Display contour layer with latest design changes.

LISTING 8.2: URL string for export data from GeoServer.

```

1 var url = 'http://localhost:8080/geoserver/wfs?CQL_FILTER=' + ↵
    urlFilter + '&OUTPUTFORMAT=' + format + '&REQUEST=GetFeature&↵
    SERVICE=WFS&TYPENAME=station&VERSION=1.0'
2     window.open(url);

```

This export function downloads all data in the table. It has no filtering feature and the usability testers argued a filtered version of export was necessary to accomplish their export as they desired. A new user story was created during the UCD process and implemented as described next.

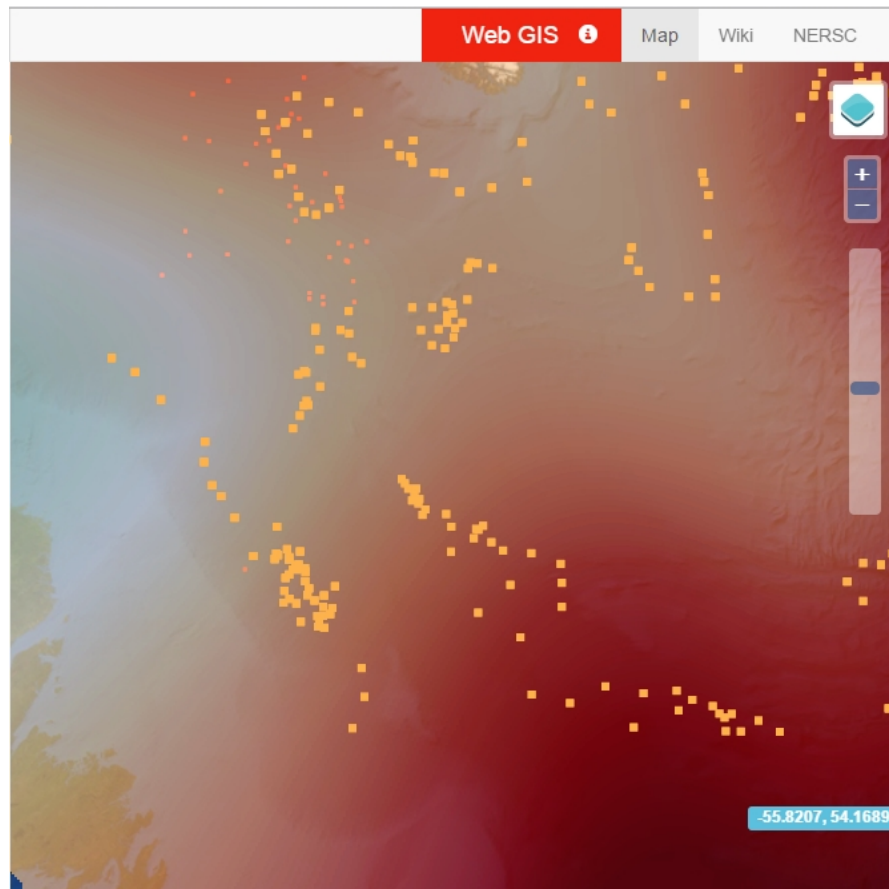


FIGURE 8.12: Contour layer with yellow data points in the map widget.

Selective export in table

The data table in the AngularUI `Grid` has integrated functionality for data export functions in PDF and CSV formats. Figure 8.14 displays the export options; Export Selected data, Export visible data and Export All data. Export is customized in the client implementation and downloaded to the browser.

8.8.2 Export graph

Every graph has an integrated menu in the upper right corner. Highcharts [65] provide export formats JPG, PNG, PDF and SVG.

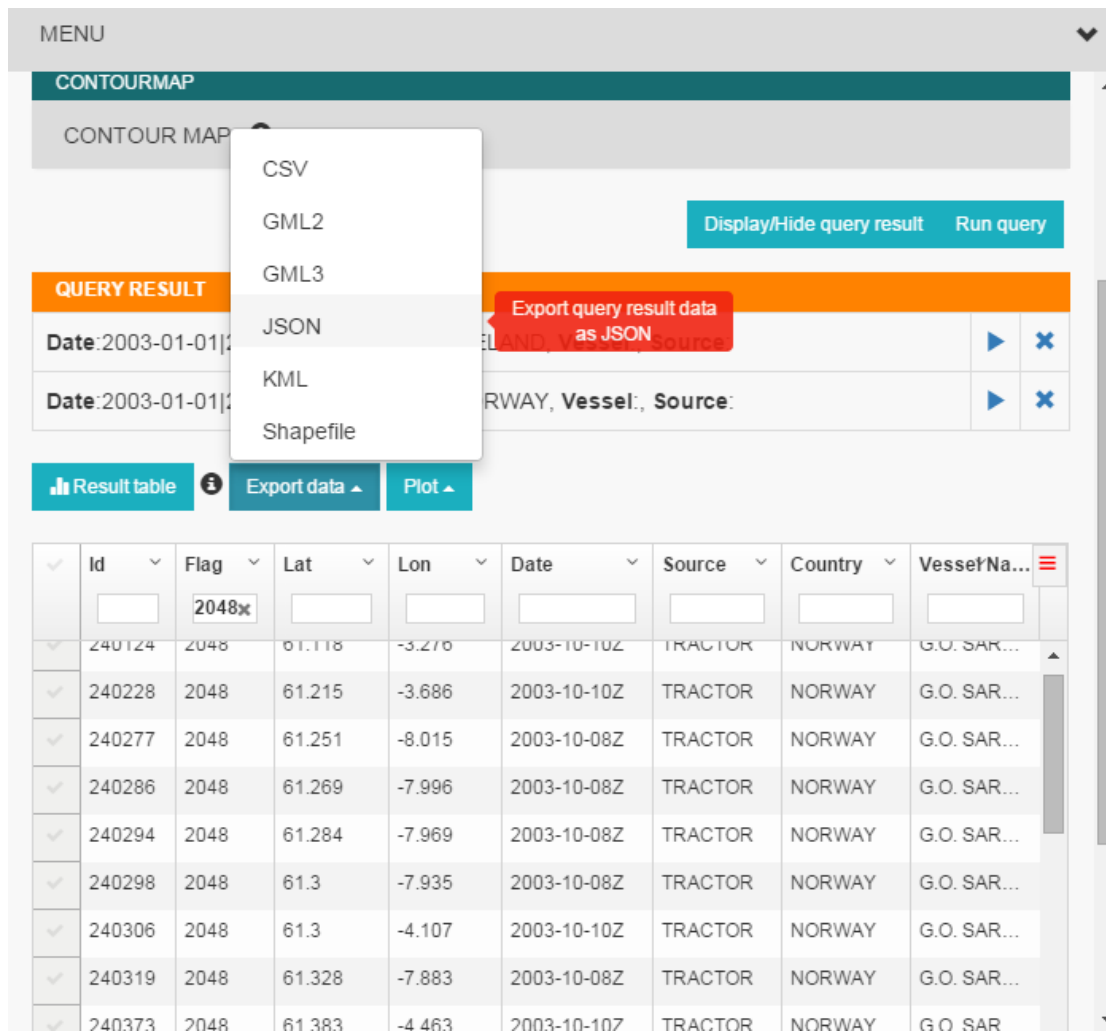


FIGURE 8.13: Export data result to text (JSON) format.

8.9 Other non functional requirements

This section demonstrates some of the implemented features to meet the non functional requirements (NFR) of the application.

8.9.1 Prevent user from making errors

The user are prevented from making errors for every input field. The input fields have validation by scoping the possible inputs. The validation is implemented by AngularJS core `ng` module directive for `input[text]`. Figure 8.15 illustrates the range of different validation functions to prevent the user from making error in the requests. The figure applies to the NFR, display meaningful error messages by defining meaningful error messages which explains what error did occur.

✓	Id	Flag	Lat	Lon	Date	Source	Country	VesselNa...	
	<input type="text"/>	<input type="text" value="2048"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>			
	240124	2048	61.118	-3.276	2003-10-10Z	TRACTOR			
✓	240228	2048	61.215	-3.686	2003-10-10Z	TRACTOR			
✓	240277	2048	61.251	-8.015	2003-10-08Z	TRACTOR			
✓	240286	2048	61.269	-7.996	2003-10-08Z	TRACTOR			
✓	240294	2048	61.284	-7.969	2003-10-08Z	TRACTOR			
✓	240298	2048	61.3	-7.935	2003-10-08Z	TRACTOR			
✓	240306	2048	61.3	-4.107	2003-10-10Z	TRACTOR			
✓	240319	2048	61.328	-7.883	2003-10-08Z	TRACTOR			
✓	240373	2048	61.383	-4.463	2003-10-10Z	TRACTOR			
✓	240407	2048	61.46	-4.826	2003-10-10Z	TRACTOR	NORWAY	G.O. SAR...	
✓									

Export all data as csv
Export visible data as csv
Export selected data as csv
Export all data as pdf
Export visible data as pdf
Export selected data as pdf
Columns:
✓ Id
✓ Flag

Total Items: 1880 (Showing Items: 25)(Selected Items: 6)

FIGURE 8.14: Export functionality in data table.

BOUNディング BOX ⓘ

Max. Lat:

The value must be in range -90 to 90!

Min. Lon:
Required!

Max. Lon:
Not valid number!

FIGURE 8.15: Example of input validation.

8.9.2 Information and guidance

This NFR is accessible throughout the client indicated by the information symbol. Clicking the information symbol displays information about the current feature. Figure 8.16 illustrates a system guidance, giving information about the features of the Web GIS client.

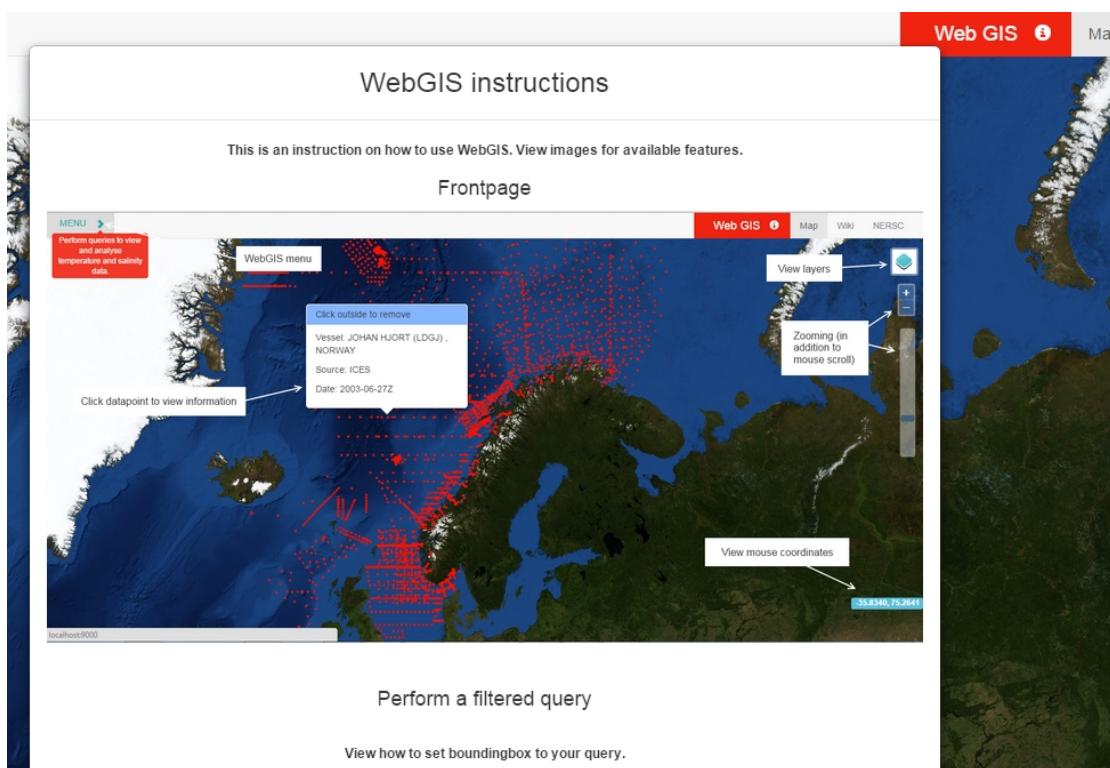


FIGURE 8.16: Give system information and guidance.

Chapter 9

Evaluation

This chapter evaluates the sub-goals stated in subsection 1.2.1 and the requirements stated in section 3.2. Each sub-goal, functional- and non functional requirements are evaluated and the achievement of each one of them are discussed. In the end of this chapter an evaluation summary evaluates the state of the sub-goals and requirements completeness of this thesis.

9.1 Sub-goals evaluation

The sub-goals are the guideline towards the fulfilment of the overall goals of this thesis. Each sub-goal is justified with background information to evaluate if it is completely fulfilled or not.

9.1.1 Conduct usability research in GIS and non GIS applications to determine use of usability methods and principles

GIS usability research and UE articles have been studied during the work on this thesis. The field of usability research in non GIS applications are much more comprehensive in comparison with GIS applications. Many articles stated lack of proper methods to evaluate task based applications in the geovisualization domain. These findings built knowledge about performing usability studies in the GIS domain. Among others, that it is important to get to know the user, their goals and how they work with the geographic

data inside the application. Geographic data can be represented, visualized, explored and analysed in different ways. With this in mind, articles and books on general usability engineering methods, practises and principles were researched and evaluated to find the best usability methods for the Web GIS client development.

Usability standards and practises

There are many usability papers on different models embracing the usability of software development. The usability standard ISO/IEC 25010 for product quality was used during the UE. The usability attribute in ISO 25010 [79] in addition to a selection of Quensberrys E5s [81] usability attributes was the base target working towards usability. It resulted in usability attributes for developing an easy to learn, error tolerant and engaging Web GIS to improve the UX. The use of heuristics, resulted in a guideline for developing GUI elements with focus on their usability.

Usability methods

The usability methods; heuristics, empirical usability testing, thinking aloud and questionnaires were applicable to achieve usability in the UE. As stated in section 5.4 these methods were suitable for three testers.

Empirical testing resulted in feedback on how the testers perceived the client, their body language, objective and subjective opinions. Thinking aloud was successful to understand the users cognitive thinking. The communication and direct feedback from the users gave valuable information to understand how the users were thinking and feeling while interacting with the client.

Measurement

Research on how to measure usability, by quantitative and qualitative methods resulted in a tool set where both methods were used. Empirical testing, thinking aloud, usability testing tasks and questionnaires completes the tool set retrieving important feedback from the users. Successfully measuring the outcome of usability studies was a key factor to improve and evolve the usability in the client.

Summary

The overall research resulted in a set of usability methods which worked toward the goal of usability features in the Web GIS client successfully. The selections of usability methods worked well together and capture both quantitative and qualitative data.

The tool set of usability methods can be implemented as a framework for Web GIS system development, which is further discussed in future work in chapter 10. The chosen usability methods proved success in a small sized Web GIS application with a set of three usability testers. In further development when the application is growing larger the methods should be revisited and analysed to find if the methods still apply.

The usability test results compiled together with an evaluation of the development process, concluded in a Web GIS client implemented with an user-centred approach. The UCD process evaluation is described next.

9.1.2 Develop an iterative user-centred design approach to achieve software quality in terms of usability and UX

Based on the research presented in the previous subsection a user-centred design process was developed and used in developing the Web GIS client. The process is illustrated in Figure 5.4. The UCD process worked iteratively with the users to address usability issues and in developing functionality to achieve the user tasks with satisfaction. The process activities were building blocks performed one by one assuring a user-centred developing process.

The outcome of the development based on the activities was rewarded. Every activity outcome was important to the next step in the process. Picking a collection of valuable UE methods was a key factor for this success.

The usability study activities, performing usability testing and the test process were successful. Feedback from the testers were positive and the users were satisfied.

The work with multidisciplinary tasks worked with success, for a one person team. Converting this process to work with larger teams would benefit from using a project management tool which integrates the multidisciplinary work tasks. The designer, UX

specialist, software architect, designer, developer, tester and others must be able to cooperate within the development cycle artifacts.

9.1.3 Perform analysis and selections for software stack which best complies to GIS technology and interactive web applications

The open source software (OSS) technologies were installed and configured with success. The OSS criteria, requiring active development communities and proper documentation were met. Criteria important for the successful implementation and further maintenance and development.

PostGIS and GeoServer ran as expected, they served the client with geographic and non geographic data and data layers. OpenLayers 3, the mapping library implemented most features and functions regarding the mapping functionality. OpenLayers 3 had some shortcomings in the API which are discussed in [Appendix C](#).

The AngularJS framework introduced structure in code, readable code and modularity in the code base. AngularJS is an extensive client framework which benefits the client in present and future development.

The additional libraries created recognisable interface design elements (Bootstrap), interactive graphs to display the data (Highcharts) and displaying the data tables (AngularJS UI Grid).

The criteria and standard compliance for the software stack were defined in [chapter 4](#). The outcome was a set of technologies which has served the Web GIS client well. The software components was configured and worked together without major problems, a successful choice of software technologies for the client requirements fulfils this sub-goal and basis for further development.

9.1.4 Install, configure and populate GIS backend technologies

The installation and configuration of PostGIS data store and GeoServer data assessor was successful. PostGIS was successfully populated with a dataset, measurements of temperature and salinity of the year of 2003. The population of PostGIS was done step by step instructed in the old Web GIS thesis rapport [\[2\]](#) with exceptions of minor

adjustments changing modifying the test scripts and SQL view syntax. The steps for this new Web GIS client is described in chapter 6.

GeoServer was populated with data from the PostGIS data base through the PostGIS data store and served the client data through the WMS and WFS protocol. Standards for compliance of GIS functionality described in chapter 4.2 was successfully available and accessible to the client application. This sub-goal was fully achieved.

9.1.5 Develop, design and implement the Web GIS client

The client was implemented using OSS without third party software, a requirement in subsection 3.2.1. The HTML and CSS files was validated to meet the W3C standards of HTML5 and CSS3 compliance as described in subsection 7.8.1.

The completion of this goal can be assessed by addressing each of the FR listed in Table 3.1. Each of the FR is listed bellow including the assessment of completion.

FR01: Query data by a set of parameters

This requirement is discussed in section 8.3 and illustrated in Figure 8.5. FR01 is fully implemented.

FR02: Present queried data inside map in browser

Present data in map is fully implemented, explained in section 8.4 and displayed in section 8.1.

FR03: Present queried data features in tabular form

Section 8.5 describe the completion of this requirement. Figure 8.6 illustrate the table containing the response data and confirms the completion of this functionality.

FR04: Present a map widget with basic map functionality

This FR is demonstrated in section 8.2. Figure 8.1 illustrates the client front page map where zooming, panning, layer-switcher and identification of mouse pointer in map are

all fully implemented.

FR05 Do query based on a specific point on the map

This requirement is implemented to the extent of query a data point for its feature data. The implementation supports a pop-up widget containing data point information is illustrated in Figure 8.1. It is fully implemented for every data point in layers resulted from a filtered query. Query based on data points in a contour data point layer is not implemented.

FR06: Offer export functionality of requested data

The export functionality is fully implemented for a range of formats. Export formats are available in the data table, filtered, selected or all data in table. See description of export functionalities in subsection 8.8.1 and illustrated in Figure 8.14. Data export based on GeoServer is also described in subsection 8.8.1 and illustrated in Figure 8.13. Export of chart data is also fully implemented by Highchart implementation as described in subsection 8.8.2.

FR07: Present graphs to illustrate and compare multiple data parameter values

Presentation of data in interactive charts are supported for all available data. The charts implement zooming, interactive data points and export chart to image file. See Figure 8.7 and Figure 8.8 for examples, and section 8.6 for implementation details. This requirement is fully implemented.

FR08: Calculate and present a chart of density for temperature and salinity data values

This requirement is fully implemented by a calculation component which is described as the `Density` service in section 7.7.3. The density chart is illustrated in Figure 8.8.

FR09: Present a contour plot within a range of meters in sea depth

Subsection 8.7.1 describes this functionality on a high level. The requirement is partly

implemented. The contour layer is integrated by an range of sea depth as desired, but the feedback from the users confirm the partly fulfilment of this requirement. The contour layer does not display as desired. The contouring include lands in addition to the sea. This can be modified in further development using land mask to remove lands from the contouring.

Other new requirements

These are requirements requested from usability tests during the test sessions. Several requirements for modification were requested developing the Web GIS client. Some of them discussed next.

FR10 defines a list of previous queries including the filters. The queries can be easily added or deleted from the history list. Subsection 8.3 describes this user story and Figure 8.6 illustrates the GUI. This requirement was fully implemented.

FR11, a request from the users to view dynamic contour grid lines which has partly been implemented but with no success. In comparison with contour layer does the grid lines not render as desired. The dynamic process described in subsection 8.7.2 did not resolve this FR when implemented.

9.1.6 Evaluate the usability engineering and usability outcome of the new Web GIS client.

The list of non functional requirements (NFR) in Table 3.2 is evaluated in this subsection. The list includes requirements to address the usability attributes described in subsection 5.1.1. These requirements are not as measurable as the functional requirements listed in the previous subsection because it is harder to evaluate if the requirements are fulfilled or not. NRF01 to NFR09 are fully implemented, the measurement of fully completion is evaluated in the usability evaluation later in this subsection. Section 8.9 demonstrates some of the NFRs in the Web GIS client GUI. NF10 open source software components is fully implemented.

The evaluation of the complete UE and usability outcome boils down to the usability test evaluations. A summary of the questionnaires and thinking aloud session during

the testing is illustrated in Figure 9.1, Figure 9.2 and Figure 9.3. These figures displays subjective feedback given directly after test session. The bars defines the percent of feedback in the range of strongly disagree to strongly agree. The feedback are subjective and based on the questions listed in Table 9.1. These figures illustrate strengths and weaknesses of the user interface and the change made during the iterations.

The bars of question 9 and 10 had negative progress from iteration I to II. There are external factors that may have an impact. This was the first test session and for a first time user the questions generated better feedback than the returning user, which may had become more critical than prior. The nature of the questionnaire could also affect the outcome. Many questions to answer and hard to give an concrete answer. Bar 7 had a constant increase which can be explained by the users knowledge of the test procedure and the prior experience with the Web GIS client.

Figure 9.4 compare the mean rating from test iteration 1-3 in the range from strongly disagree=1 to strongly agree=5. Figure 9.4 skip numbers 2 and 3 in the x-axis because the questions was changed from iteration I to II and III. This figure reveals a clearer impression on the main trends in the usability evaluation. The development from each iteration is easy to read underpins the point made for the other figures above.

Bar 6 and bar 7 in Figure 9.4 are the ones making the most progress. Bar 6, the website was engaging, is indicating an increased satisfaction from the users and a desire to investigate more. This result may indicate an increased knowledge about the client user interface and its features, and is a great contribute to the overall evaluation. Bar 7 is emphasised above.

Bar 1 and 4 are related, question 1 and 4 in Table 9.1, the similar answers underpins the creditability of the answers. The bars are identical through the iterations until the latest bar in question 1 increases. This results reveals the satisfaction and level of learning the client from the testers perspective, and how easy they think new users will experience the client. The users experience the client as easy to use after using the client in several occasions.

The conversations and feedback from the users were an important part of the outcome of this sub-goal. The UCD process resulted in knowledge about the test users which were essential for the usability outcome.

1	Thought Website was easy to use
2	The website had god error detection for preventing errors
3	Found it difficult to keep track of where they were in website (I)
3	The website was effective and gave an accurate and successful outcome (II and III)
4	Thought most people would learn to use website quickly
5	Website provided information whenever needed
6	Website was engaging, would like to come back and explore more
7	There was no unclear situations during the testing
8	Website is well organized
9	Website reminds me of a GIS application user interface
10	The workflow was easy to understand and learn

TABLE 9.1: Questionnaire asked during test sessions.

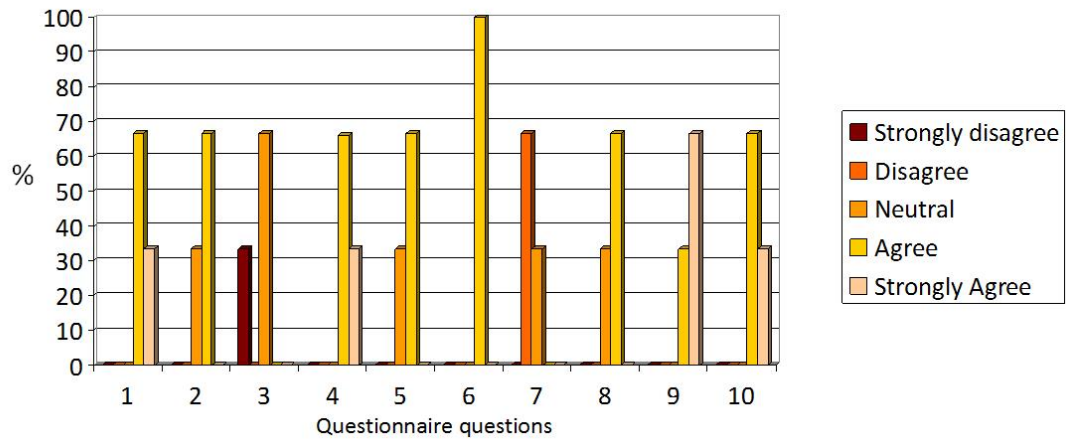


FIGURE 9.1: Usability evaluation statistics Iteration I.

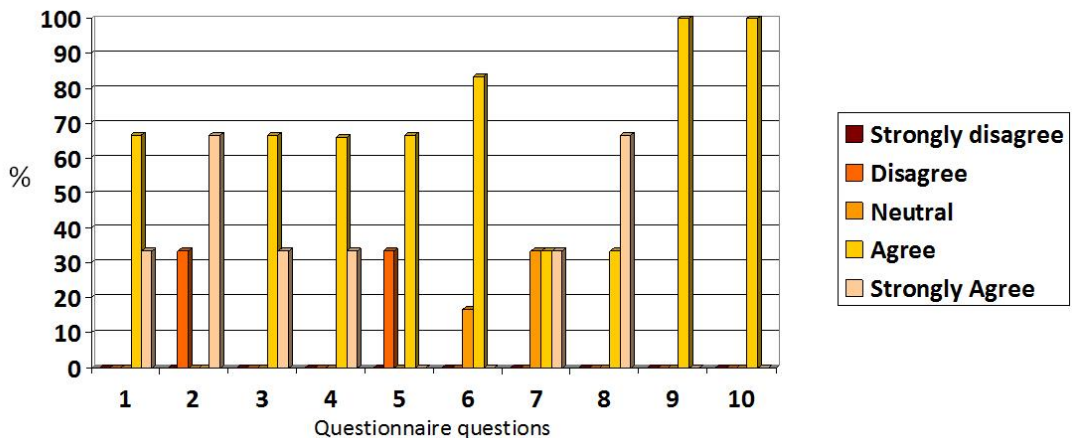


FIGURE 9.2: Usability evaluation statistics iteration II.

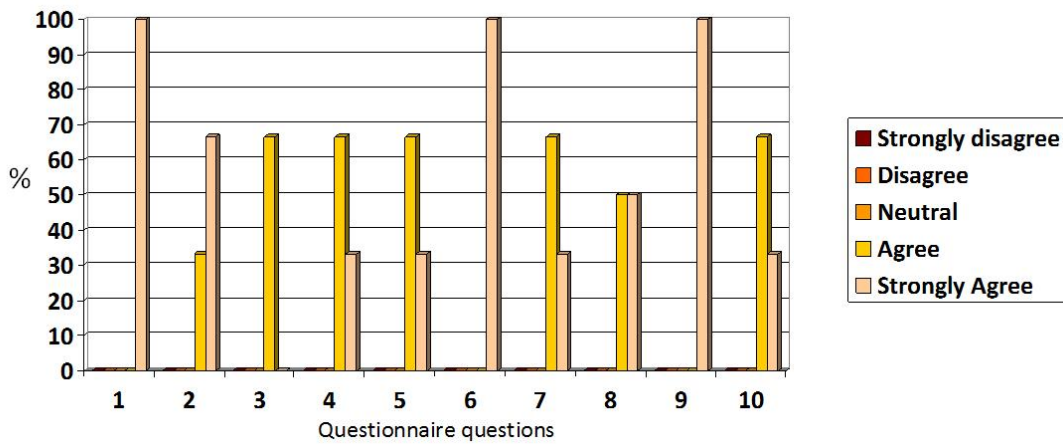


FIGURE 9.3: Usability evaluation statistics iteration III.

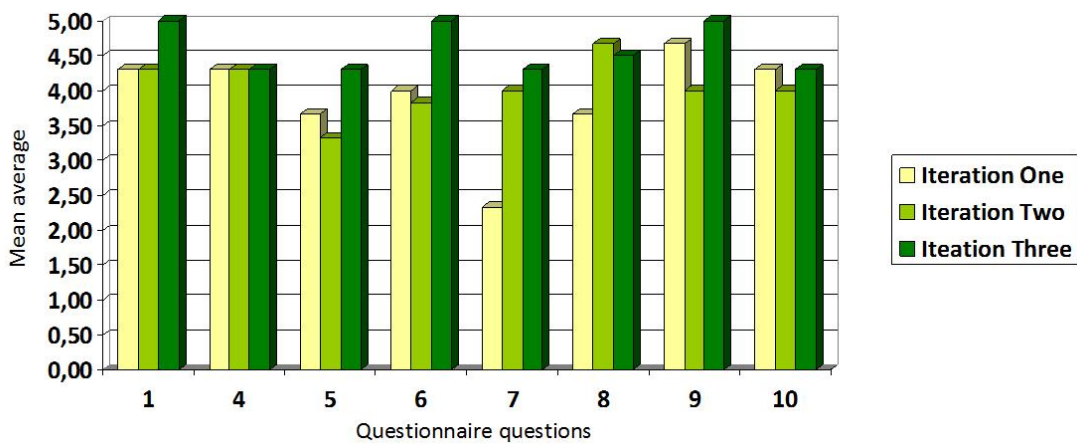


FIGURE 9.4: Usability evaluation statistics, mean rating for user testing.

Summary results

It is important when reading the results to think about the setting the testing was done. The test environment, the time scope and a specific set of testing tasks may have effected the results. Even knowing the purpose of the thesis may affected the test users' decision making. User evaluation should be conducted with a broader set of factors than only observing task completing in a laboratory [86]. In the context of human factors like emotions, circumstances such as time of day and external factors like office environment and test equipment.

Figure 9.1, Figure 9.2 and Figure 9.3 reveals an evolution towards satisfaction during the usability evaluations. The results indicates a positive development during the three iterations. This reflects the fulfilment of this sub-goal.

The charts are a quantitative proof of concept of the UE and UCD process developed during this thesis. The qualitative responses from the users were just as valuable during the testing sessions. The responses contributed to develop the prototype to what it has become today. Feedback such as feelings of frustration, excitement were also addressed during the test session. The testers communicated their feelings, the qualitative data reflected the user experience in an emotional level. Together the qualitative and quantitative results points to a positive increase of usability and UX in the Web GIS client. This sub-goal is fulfilled.

9.2 Overall objective

The conducted research in the domain of developing a user-friendly Web GIS resulted in a UCD process. The UCD process successfully improve the usability and UX of the implemented Web GIS client.

The implementation, using OSS and standardised formats for geographic data management was successfully conducted. The client has features to perform Web GIS operations achieving user goals with satisfaction.

9.3 Research questions

The evaluation on the sub-goals above counts as the measurement for answering this research questions.

Is it feasible to develop a user-centred Web GIS client by converging user interface design and UX using open source software?

The Web GIS client is a proof of concept to answer this research question. All open source software implemented a client by a UCD process influenced by similar applications and using user interface design principles such as keep it simple stupid, consistency in placement of controls, helpful information and guidance when needed and more. Testing sessions during the UCD process was valuable for qualitative and quantitative feedback for further development. By improving the design of user interface elements and features

for the users. The client outcome proved to be satisfactory in use while providing UX in the domain of Web GIS application.

Is it feasible to develop a user interface providing UX and still offer the functionality of a Web GIS?

Subsection 9.1.5 defines the implemented GIS features and according to the results in subsection 9.1.6, the results indicates successful user satisfaction interacting with the Web GIS client. The client is highly functional in serving geographical data, charts and interactive maps to the pressure of the users. This client application is a Web GIS providing a positive UX.

9.4 Evaluation summary

The Web GIS client was successfully implemented. The set of functionality was small and evolved with the UCD process. The Web GIS application is a working application even if not all FRs were fully implemented. The FRs not fully implemented will be further discussed in the next chapter.

The process of researching and developing UE methods and a UCD process was implemented into the SDLC with success. The usability quality attributes and methods of measurement presented the state of usability and UX in the Web GIS client. The overall usability sub-goals resulting in the research questions above was performed with success.

Chapter 10

Discussion and conclusion

10.1 Outcome

The outcome of this thesis is a Web GIS client developed using a UCD process. The client is a prototype developed as a proof of concept for developing Web GIS client using UE methods and practises. The Web GIS system is based on a master thesis [2], but with a extended set of features. The client prototype has been a total rewrite with new client software technologies and libraries. It has usability attributes providing positive UX for the users.

During the UE research a UCD process was developed. The UCD process was used for three iterations performing usability studies with three highly qualified usability testers. The testers had domain- and technical expertise and different levels of Web GIS experiences. The users involvement in the UCD process, their feedback from the usability test sessions have positively affected the outcome of this thesis.

The usability testing captured quantitative and qualitative data influencing the UI design, usability features and features of the Web GIS client. The testing results implied that when the users got to know the testing procedure in addition to experience with the client, more user satisfaction was achieved. This signify the importance and effect of conduction several test iterations. The findings affected the outcome of the evaluation and influenced the development of the Web GIS client during this thesis.

The implementation using AngularJS and other JavaScript libraries resulted in a dynamic and interactive Web GIS client. Several components are reusable even if the client is developed for a specific data set.

It is important to look at the big picture when analysing the results. Single results may be effected by external influence (stress, time of day and more) and/or type of answers in the questionnaire schema. The average results from the usability evaluation concludes in an increase in the users satisfaction.

10.2 Further work

The Web GIS client has a small set of features implemented by JavaScript software technologies. The client is working and provides valuable functionality. With future work the client has potential to increase the overall quality and satisfaction of the users. This section is separated into two subsections, future work in UE and implementation.

10.2.1 UE guideline for further work

The outcome of the UE work during the thesis resulted in a UCD process. It was important to define a process to evaluate and improve quality of the system [24]. The UCD model is initial and should be revised, modified and customized during further project development. The UE techniques and methods was picked based on the requirements, number of test users, scope of time and more. In further development other variables may be introduced causing the UCD to change.

Web GIS quality framework

A system quality framework managing the UE for the development has been proposed in this thesis. The research and findings in this thesis should be further extended. More quality attributes from ISO/IEC 25010 could be incorporated in the future to increase the product quality and quality of use of the Web GIS client.

The following extensions to the Web GIS quality framework are proposed:

- New test users should be introduced to assure users with a fresh view on the client give feedback and confirm the usability progress in the client application. The current usability testers would preferable continue to contribute, they are valuable assets with experience using the client. New test users would help expand the perspective of the client to assure the user interface design are continuing to be applicable for new users.
- Specialists in performing usability testing, UX experts, cartography expert should be introduced to test the client to address usability issues regular users don't find. UX experts may discover usability issues about the position of GUI elements or workflow issue. Cartography experts have expertise in colors works best in maps, position and type of elements in the map to increase the exploration and analysing of the geographic data. Usability issues the current testers may oversee during test sessions.
- Evolve the use of Heuristics by including more of the heuristics to achieve a wider set of usability in the user interface design. Heuristic, user control and freedom [80] would support undo and redo and effect the users efficiency and satisfaction. Match between system and the real world [80] would benefit the users with descriptions in words and concepts familiar to the users.
- Implement a guideline of standard GUI elements, color schemes, type of buttons and menus and position of elements to assure a standardized GUI. The implementation should be based on cooperation of UX experts and feedback from the usability testers to assemble the best possible attributes into a guideline.

10.2.2 Design, implementation guideline for further development

All the functional requirements were not fulfilled completely and are elements for further development. Further work proposal has been made based on the experiences of developing the client application. The proposals are given to apply long term improvement for enhancing the product quality of the Web GIS.

Filter encoding

Appendix C describes filtering in OpenLayers 2 versus OpenLayers 3. The implementation described in Code Listing C.2 need improvements because the filters are made by simple SQL strings appended to the query which is not a good solution. A preferred solution would be a filter object assuring the filters are appended smoothly and assuring easier maintenance of the client code. Making a request to WMS and WFS by creating a Filter Encoding object is not trivial. There exist JSONIX [102] a library which have a XML-JSON mapping for OGC schemas such as Filter Encoding [103]. JSONIX would improve the data requests to GeoServer including the export functionality.

Test coverage

The test suite does not offer a complete test coverage. The code would have better quality assurance by increasing test coverage. The test suite should be improved to assure higher test coverage covering the main operations of the application. There are several prioritised test cases which should be considered. Testing to verifying the requests sent to GeoServer to assert the filter attributes are correct. Assert that the response from server is the correct response due to the request sent. Conversion of response data sets such as convert XML to JSON and extract data from JSON object into table data.

The client would also benefit from more types of tests such as integrations tests, acceptance tests and UI tests.

Data quality

Data quality should be addressed. The quality of large GIS datasets and data parameters is a critical part of the task flow. Working with data set with missing parameters and no specific quality assurance are confusing for the users, it leads to credibility issues and the users loses their trust in the application. Trust is a quality attribute for ISO/IEC 25010 Quality in use, which implies that lack of trust will decrease the UX of the Web GIS client. A data quality model framework is proposed established. A framework should work towards fulfil the data quality of the ODB database by introducing a data set criteria to assure the data are complete and correct. Geographical data has challenges which are important requirements for creating a Web GIS with accessibility and usability [104]. The ISO/IEC 25012 Data quality model propose quality attributes

to define quality in data [105].

10.3 Lessons learned

The project work from beginning to end has been an experience of freedom, initiative and responsibility. The freedom to pick a part of the old Web GIS application to expand and evolve. The initiative to angling the project work towards the users of the application, and the responsibility given to conduct the assignment was embraced during the work and greatly appreciated.

The work on this thesis has been a multidisciplinary experience. Working with UE required initial research in order to develop a skill-set of usability methods and practises. The skill-set laid the foundation for the work towards the satisfaction of the users, the UX.

The experience valued the most was to work with the users. Involving the users at an early stage in the development process resulted in high profit when converging the GIS domain with modern web page design. A transaction which would be harder to adapt to when a larger set of features were implemented. All of the usability testers were responsive and enthusiastic during the test sessions. The development of specified user task to satisfy the user needs could not been done without the test users.

The UCD process did not use any specific GIS domain usability methods such as cartography or geographical visualisation. There are no UE knowledge developed specifically for GIS [13] and general UE methods was therefore applied developing this client application. The UCD process with all of its UE methods was based on the general domain of UE.

It was expected the client implementation would be a more pleasant experience than it has been. Due to the client application was totally rewritten, to implement new software technologies for web and web mapping development including the expressive JavaScript language for an interactive Web GIS client. The experience working with open source technologies was positive still having issues with new versions. New versions of open source libraries is often in lack of documentation and knowledge in communities, but it

still important to use the new version to assure further development of the Web GIS client.

The GIS domain and geographic data management consumed valuable implementation time, but it was well spent time to acquire important experience. Working with a full software stack was interesting and clarified how the geographic data operation's between the software layers.

10.4 Conclusion

A user-friendly Web GIS client has been built using a UCD process. The developed UCD process was valuable involving the users developing an application that works with a high level of satisfaction for the users. Usability could not be added into the end of the Web GIS client development, it had to be developed from the beginning, creating requirements of usability along with functional requirements.

No unique formula for developing usable Web GIS applications has been proposed, but a experiment has been done. Regular UE methods with an iterative user-centred approach has proven to give more satisfied users. Cooperation with the users gave valuable insight in their cognitive thinking while performing their defined task which contributed to the successful outcome.

Web GIS applications has matured into mainstream web applications and can therefore benefit from using the same design principles as modern web pages to achieve usability and UX. The new Web GIS client is not ready for deployment to a production environment. The Filter encoding in subsection [10.2.2](#) should be considered implemented and tested before deploying the application to a production environment. Filtering is a important part of the application feature and should be properly implemented and tested.

Appendix A

Personas

This appendix illustrated two of the three personas developed during the conceptual development phase. One is illustrated in Figure 5.5 and the other two bellow.


	<p>Personal Ms Natalia Geyer 34 years old PhD Student, University of Edinburgh</p> <p>Expertise Novice in Web GIS applications. Natali is part of a project team studying the environmental changes in the north Arctic sea. She work remote from her project team and does not have anyone with expertise of the application at her campus. She uses the application for the first time. She has not much experience with GIS technology other that discovery, search and analysis of online maps applications. Natali is on her second year of her PhD exploring the environmental evolution in the north Atlantic Gulf stream.</p>
---	---

FIGURE A.1: Example of a persona, Natalia.

	<p>Personal Dr. Mikka Heikenen Researcher, oceanography 43 years old Finnish Institute of Marine Research</p> <p>Expertise Regular Web GIS user. Works regularly with oceanography projects where similar GIS applications occur. Used to work with commercial and non-commercial GIS desktop applications and Web GIS's. Do not have any special expertise to a Web GIS, or a preferred type of GIS application. Mikka encounter the Web GIS application while studying results given by other project team members and look at the result inside the Web GIS application. Mikka spend much of his time bicycling in the mountains while enjoying his weekend with his girlfriend Vilma at their mountain cottage.</p>
--	---

FIGURE A.2: Example of a persona, Mikka.

Appendix B

Usability test schema

This appendix displays the complete usability test schema used by the usability testers in every test session. One schema customised for each iteration.

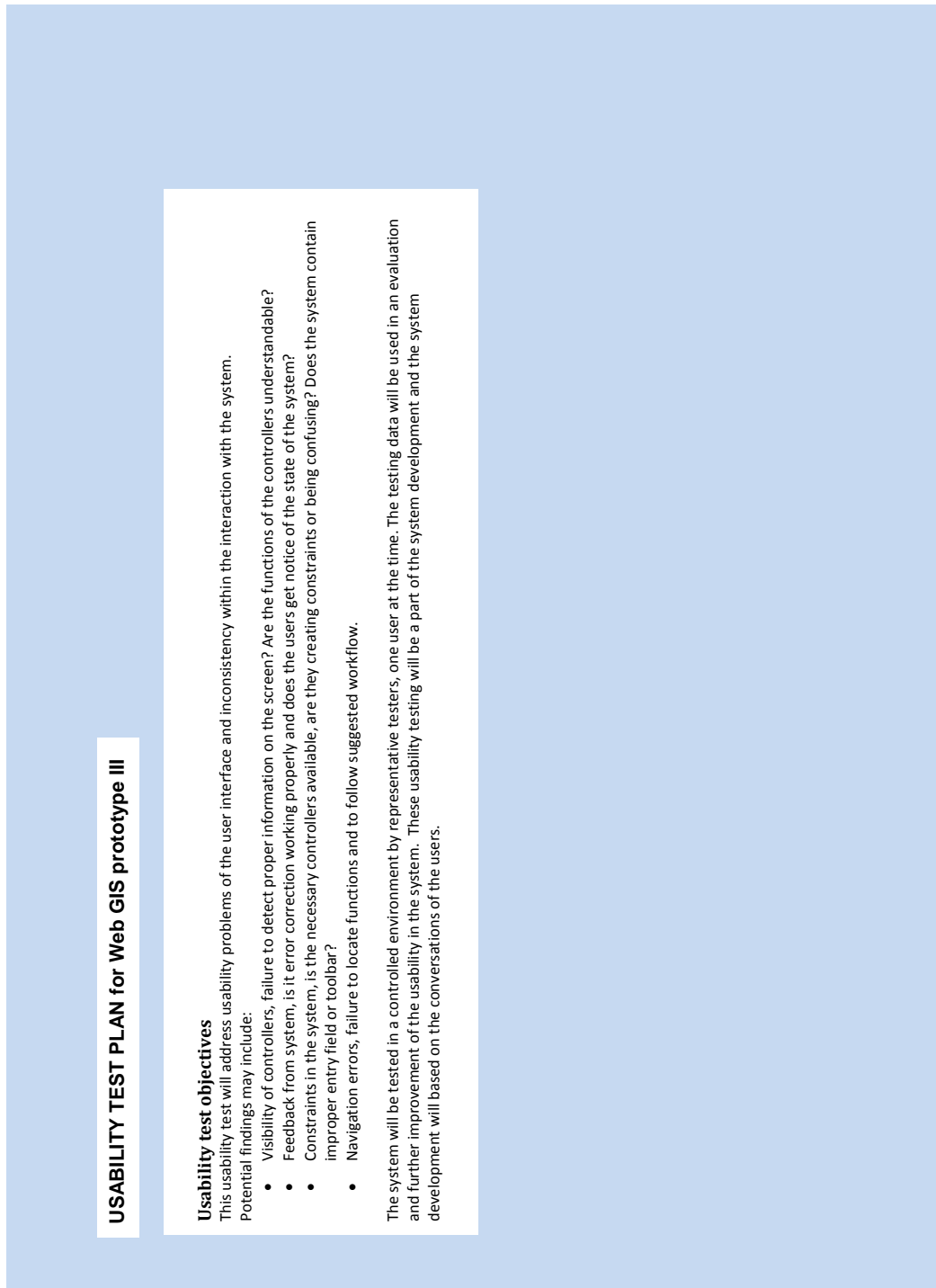


FIGURE B.1: Example of usability introduction schema.

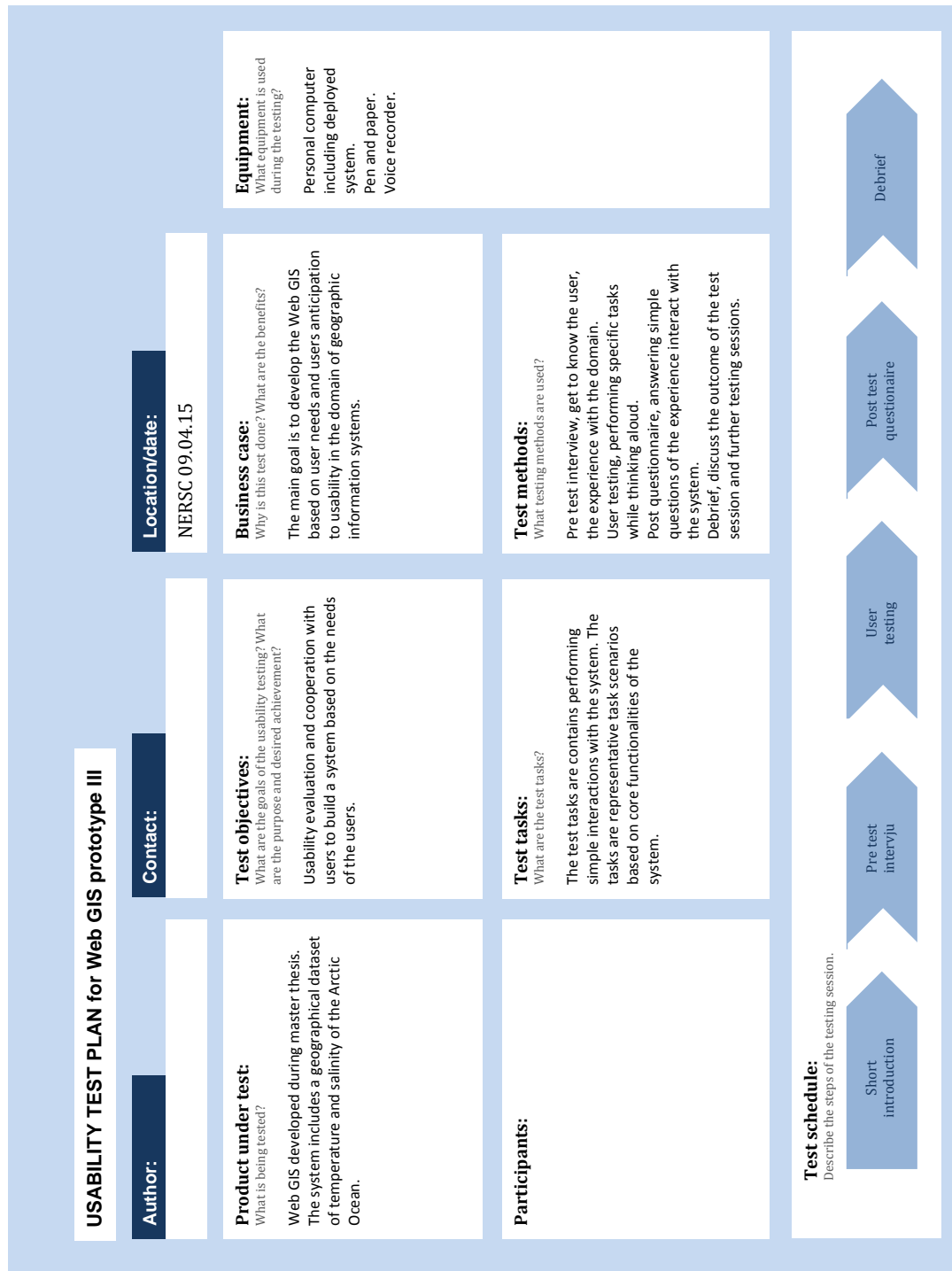


FIGURE B.2: Example of usability introduction schema.

USABILITY TEST OBSERVATIONS for Web GIS prototype III

Author:

Participant

Location/date:
NERSC 09.04.15

CRITERIA: What is the purpose of the test?	USER OBSERVATIONS: Users feedback specified by criteria.	Suggestions from author: Did a author come up with any ideas during the testing? What recommendation is a result of user observations?
Engaging: Is the interface attractive and engaging? Get to know the interface. 1. Play around with the user interface for some minutes. Investigate the user interface and functions.		
Effective: Are the users goal met effectively, Accurate and successfully. Is desired functionality easily to locate? 2. Perform a query, (set from date = 2003/01/15 and to date = 2003/12/15 and attribute country = Canada , review result in tabular form. Sort by latitude 66 and export data as pdf. 3. Use same result set as above. Select 5 first dataset in result table and export as pdf and save document to desktop.		
Efficient: Task complement, how fast is it completed? Is the navigation and workflow efficient? (In terms of clicks, shortcuts etc.) 4. Perform a query, (set bounding box around Iceland automatically by the mouse, change from date to 2003/01/15 and country to Iceland). Review result in result table.		

FIGURE B.3: Example of usability test task schema.

CRITERIA: What is the purpose of the test?	USER OBSERVATIONS: Users feedback specified by criteria.	Suggestions from author: Did author come up with any ideas during the testing? What recommendation is a result of user observations?
Efficient (continues) 5. Perform a query, (set bounding box around Norway automatically by the mouse, change from date to 2003/01/15 and country to Norway). Review result in result table. Export result data in JSON 6. View list of queries and rerun query with the same filters as assignment 4.		
Error tolerant Does the system have a correct error detection system? Does the user get confirmation of valid/non valid actions?		
7. Draw a contour map in a range of 5 to 0 meters.		
Easy to learn: Is the interface easy to learn? 8. Draw and compare two contour maps. One at 1-5 meters and the other at 10-15 meters.		

FIGURE B.4: Example of usability test task schema.

USABILITY POST TEST QUESTIONNAIRE for Web GIS prototype III

Source of tables: <http://www.usability.gov/how-to-and-tools/resources/templates/report-template-usability-test.html>

Author:

Participant:

Location/date:

NERSC 09.04.15

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Mean Rating	Percent Agree
Thought Website was easy to use							
The website had good error detection for preventing errors							
The website was effective and gave an accurate and successful outcome							
Thought most people would learn to use website quickly							
Website provided information whenever needed							
Website was engaging, would like to come back and explore more							
There was no unclear situations during the testing							
Website is well organized							
Website reminds a GIS application user interface							
The workflow was easy to understand and learn							

Comment	Severity	
	High	Low

FIGURE B.5: Example of questionnaires schema.

Appendix C

Filtering in OpenLayers 2 verses OpenLayers 3

The rewriting of OpenLayers 2 into OpenLayers 3 resulted in a complete rewriting of the implementation. OpenLayers 3 does not have Filter Encoding which was used in the old Web GIS [\[2\]](#) client implementation. Filter Encoding was used to create filter objects and combining them to use in the request to GeoServer. More implementation details can be read in [\[2\]](#). This client implementation specified the filters as common strings with the SQL syntax `AND` as described in line 2 in Code Listing [C.2](#). The filter parameter in the `getFeature()` in line 18 in Code Listing [C.1](#) was changed to use the CQL filter as specified in line 12 in Code Listing [C.2](#). Read more about CQL filter in subsection [4.4.4](#). Both type of filter specifies the parameters to GeoServer.

LISTING C.1: OpenLayers 2 syntax for handling filters.

```
1  // create filter object
2  var dateFilter = new OL.Filter.Comparison({
3      type : OL.Filter.Comparison.BETWEEN,
4      property : "stddate",
5      lowerBoundary : date.fromDate,
6      upperBoundary : date.toDate
7  });
8
9      dateTimeFilterArray.push(dateFilter);
10
11      return combineFilters(dateTimeFilterArray);
12
13
14  // combine array of filters to single filter
15  function combineFilters(filtersToCombine) {
16      return new OL.Filter.Logical({
17          type : OL.Filter.Logical.AND,
18          filters : filtersToCombine
19      });
20  }
21  // call to GeoServer
22  getFeature({
23      TYPENAME : "floats",
24      FILTER : filter
25  }, displayFeatures);
```

LISTING C.2: OpenLayers 3 syntax for habdling filters.

```
1 // create filter object
2 var dateFilterString = 'stdate BETWEEN ' + formFactory.getFromDate() + ' AND ' + formFactory.getDate();
3 filter_list.push(dateFilterString);
4 // joint filters by SQL syntax AND
5 return filter_list.join(' AND ');
6
7 // call to GeoServer
8 GeoserverFactory.getFeature({
9     TYPENAME : 'station',
10    OUTPUTFORMAT : 'json',
11    SRSNAME : 'EPSG:4326',
12    CQL_FILTER : filter
13 })
```

LISTING C.3: A request URL including CQL filter string.

```
1 http://localhost:8080/geoserver/wfs?CQL_FILTER=stdate+BETWEEN
  +2003-01-15+AND+2003-12-15+AND+stcountryname+LIKE+%27%25
  CANADA%25%27+AND+stvesselname+LIKE+%27%2541900%25%27+AND+
  stsource+LIKE+%27%25ARGO%25%27&OUTPUTFORMAT=json&REQUEST=
  GetFeature&SERVICE=WFS&SRSNAME=EPSG:4326&TYPENAME=station&
  VERSION=1.0.0
```

Bibliography

- [1] V. Volkov R. Gerdes A. Korablev V. Melsehko V. Maderich V. Denisov Johannessen O.M., L.H Pettersson and G. Shapiro. 2007: The Nordic Seas in the global climate. Final report to INTAS grant 03-51-4620. July, 2007.
- [2] Torgeir Mossige Grønning. *Data Structure, Access and Presentation in Web-GIS for marine research*. 2013.
- [3] Jakob Nielsen & Don Norman. The definition of user experience, 2014. URL <http://www.nngroup.com/articles/definition-user-experience/>.
- [4] Philip Lew, Li Zhang, and Luis Olsina. Usability and user experience as key drivers for evaluating GIS application quality. In *2010 18th International Conference on Geoinformatics, Geoinformatics 2010*, number 60773155, 2010. ISBN 9781424473021. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5567803.
- [5] Postgis spatial and geographic objects for postgresql, . URL <http://postgis.net/>. Last visited 2015-04-29.
- [6] Geoserver. URL <http://geoserver.org/>. Last visited 2014-08-19.
- [7] Suzana Dragičević. The potential of Web-based GIS. *Journal of Geographical Systems*, 6(2):79–81, June 2004. ISSN 1435-5930. URL <http://link.springer.com/10.1007/s10109-004-0133-4>.
- [8] map.geo.admin.ch. URL <https://map.geo.admin.ch/?X=190000.00&Y=660000.00&zoom=1&lang=de&topic=ech&bgLayer=ch.swisstopo.pixelkarte-farbe>. Last visited 2014-01-12.
- [9] Surging seas: Sea level rise analysis by climate central. URL <http://sealevel.climatecentral.org/>. Last visited 2014-01-12.

- [10] Ocean data — marinexplore. URL <http://marinexplore.org/explore/#explore/2d>. Last visited 2014-01-12.
- [11] Esri - gis mapping software, solutions, services, map apps, and data, . URL <http://www.esri.com/>. Last visited 2014-01-12.
- [12] Esri blog, . URL <http://resources.arcgis.com/search/?do=search&collection=blogs&filter=0&lg=en&q=usability&submit=>. Last visited 2014-01-12.
- [13] Mordechai (Muki) Haklay and Antigoni Zafiri. Usability Engineering for GIS: Learning from a Screenshot. *The Cartographic Journal*, 45(2):87–97, May 2008. ISSN 0008-7041. URL <http://www.maneyonline.com/doi/abs/10.1179/174327708X305085>.
- [14] Mordechai Muki, Haklay Corresponding, and Carolina Tobón. Usability Evaluation and PPGIS : towards a user- centred design approach Usability Evaluation and PPGIS : towards a user- centred design approach Abstract. (July):21–23, 2002.
- [15] Usability 101: Introduction to usability. URL <http://www.nngroup.com/articles/usability-101-introduction-to-usability/>. Last visited 2015-01-07.
- [16] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 0125184050.
- [17] Donald A. Norman. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA, 2002. ISBN 9780465067107.
- [18] Steve Krug. *Don'T Make Me Think: A Common Sense Approach to the Web (2Nd Edition)*. New Riders Publishing, Thousand Oaks, CA, USA, 2005. ISBN 0321344758.
- [19] N Andrienko and Gennady Andrienko. The complexity challenge to creating useful and usable geovisualization tools. *GIScience 4th International ...*, 2006. URL <http://geoanalytics.net/and/papers/giscience06.pdf>.
- [20] Alan M. MacEachren and Menno-Jan Kraak. Research Challenges in Geovisualization. *Cartography and Geographic Information Science*, 28(1):3–12, January

2001. ISSN 1523-0406. URL <http://www.tandfonline.com/doi/abs/10.1559/152304001782173970>.
- [21] David Schobesberger. Towards principles for usability evaluation in web mapping -usability research for cartographic information systems. 2009.
- [22] Annu-maaria Nivala. *Usability perspectives for the design of Interactive Maps*. PhD thesis, 2007.
- [23] Mike Cohn. *User Stories Applied*. Addison-Wesley, 2004. ISBN 0321413091.
- [24] S. Wagner. *Software Product Quality Control*. Springer Berlin Heidelberg, 2013. ISBN 9783642385704.
- [25] Furps - wikipedia, the free encyclopedia. URL <http://en.wikipedia.org/wiki/FURPS>. Last visited 2014-02-19.
- [26] Nigel Bevan. Extending quality in use to provide a framework for usability measurement. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5619 LNCS(1991): 13–22, 2009. ISSN 03029743.
- [27] Marc Hassenzahl. Marc Hassenzahl Chapter 3. *Funology*, 3:31–42, 2003. URL <http://www.springerlink.com/index/N667WUL417708T71.pdf>.
- [28] Empirical research - wikipedia, the free encyclopedia. URL http://en.wikipedia.org/wiki/Empirical_research. Last visited 2014-02-02.
- [29] Manifesto for agile software development. URL <http://agilemanifesto.org/>. Last visited 2015-04-29.
- [30] Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003. ISBN 0135974445.
- [31] Boundless, formerly opengeo, . URL <http://boundlessgeo.com/>. Last visited 2014-08-19.
- [32] Open geospatial consortium. URL <http://www.opengeospatial.org/>. Last visited 2015-04-29.

- [33] Wolfgang Kresse and David M. Danko. *Springer Handbook of Geographic Information*. Springer Publishing Company, Incorporated, 2012. ISBN 3540726780, 9783540726784.
- [34] Web map server implementation specification. URL https://earthdata.nasa.gov/sites/default/files/field/document/06-042_opengis_web_map_service_wms_implementation_specification.pdf. Last visited 2015-04-29.
- [35] Wfs reference. URL <http://docs.geoserver.org/stable/en/user/services/wfs/reference.html>. Last visited 2015-04-29.
- [36] Filter function reference, . URL http://docs.geoserver.org/stable/en/user/filter/function_reference.html#filter-function-reference. Last visited 2015-04-29.
- [37] Html tutorial. URL <http://www.w3schools.com/html/default.asp>. Last visited 2015-05-10.
- [38] Css introduction. URL http://www.w3schools.com/css/css3_intro.asp. Last visited 2015-05-10.
- [39] What is open source. URL <http://opensource.com/resources/what-open-source>. Last visited 2015-05-10.
- [40] Shyam Seshadri Brad Green. *AngularJS*. O'Reilly Media, 2013.
- [41] Angular, ember, and backbone: Which javascript framework is right for you? URL <http://readwrite.com/2014/02/06/angular-backbone-ember-best-javascript-framework-for-you>. Last visited 2015-05-10.
- [42] Angular community. Angularjs: Superherioc javascript mvw framework, . URL <https://angularjs.org/>. Last visited 2014-06-20.
- [43] Github: Octoverse 2013, . URL <https://octoverse.github.com/>. Last visited 2014-06-19.
- [44] Backbone community. Backbone.js, . URL <http://backbonejs.org/>. Last visited 2014-06-19.
- [45] Addy Osmani. *Developing Backbone.js Applications*. O'Reilly Media, 2013.

- [46] Joachim Haagen Skeie. *Ember.js in Action*. Manning Publications Co, 2014.
- [47] Wikipedia. Convention over configuration. URL http://en.wikipedia.org/wiki/Convention_over_configuration. Last visited 2014-06-20.
- [48] Ember community. Ember.js - blog, . URL <http://emberjs.com/blog/>. Last visited 2014-06-20.
- [49] emberjs/ember.js - github, . URL <https://github.com/emberjs/ember.js/>. Last visited 2014-06-20.
- [50] Friedel Ziegelmayer. Karma - spectacular test runner for javascript. URL <http://karma-runner.github.io/0.12/index.html>. Last visited 2014-08-14.
- [51] The jQuery Foundation. Qunit, . URL <http://qunitjs.com/>. Last visited 2014-08-13.
- [52] The jQuery Foundation. License — jquery foundation, . URL <https://jquery.org/license/>. Last visited 2014-08-13.
- [53] The jQuery Foundation. Lcookbook — qunit, . URL <http://qunitjs.com/cookbook/>. Last visited 2014-08-13.
- [54] TJ Holowaychuk. Mocha - the fun, simple, flexible javascript test framework. URL <http://visionmedia.github.io/mocha/>. Last visited 2014-08-14.
- [55] Behavior-driven development. URL http://en.wikipedia.org/wiki/Behavior-driven_development. Last visited 2015-05-10.
- [56] Maximilian Antoni. Home - visionmedia/mocha wiki - github. URL <https://github.com/visionmedia/mocha/wiki>. Last visited 2014-08-14.
- [57] Pivotal Labs. pivotal/jasmine git hub, . URL <https://github.com/pivotal/jasmine>. Last visited 2014-08-13.
- [58] Pivotal Labs. pivotal/jsunit git hub, . URL <https://github.com/pivotal/jsunit>. Last visited 2014-08-13.
- [59] Leaflet - a javascript library for mobile-friendly maps. URL <http://leafletjs.com/>. Last visited 2014-08-15.
- [60] Openlayers : Home. URL <http://openlayers.org/>. Last visited 2014-08-15.

- [61] Opendeo suite - boundless, . URL <http://boundlessgeo.com/solutions/opendeo-suite/>. Last visited 2014-08-15.
- [62] Openlayers - boundless. URL <http://boundlessgeo.com/solutions/solutions-software/openlayers/>. Last visited 2014-08-15.
- [63] Bootstrap the world's most popular mobile-first and responsive front-end framework., . URL <http://getbootstrap.com/>. Last visited 2015-02-09.
- [64] Ui grid. URL <http://ui-grid.info/>. Last visited 2015-05-12.
- [65] Interactive javascript charts for your webpage. URL <http://www.highcharts.com/>. Last visited 2015-02-09.
- [66] Webstorm - the smartest javascript ide. URL <https://www.jetbrains.com/webstorm/>. Last visited 2015-04-30.
- [67] Ide comparison for html 5, css 3 + javascript. URL <http://www.oio.de/public/opensource/comparison-IDE-for-HTML5-CSS3-JavaScript-shootout.htm>. Last visited 2015-04-30.
- [68] Welcome to netbeans. URL <https://netbeans.org/>. Last visited 2015-04-30.
- [69] Aptana. URL <http://www.aptana.com/>. Last visited 2015-04-30.
- [70] The web's scaffolding tool for modern webapps — yeoman. URL <http://yeoman.io/>. Last visited 2015-04-27.
- [71] Grunt the javascript task runner. URL <http://gruntjs.com/>. Last visited 2015-04-27.
- [72] Bower. URL <http://bower.io/>. Last visited 2015-04-27.
- [73] Jeffrey Rubin. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley & Sons, Inc., New York, NY, USA, 1994. ISBN 0471594032.
- [74] Terry a. Slocum, Connie Blok, Bin Jiang, Alexandra Koussoulakou, Daniel R. Montello, Sven Fuhrmann, and Nicholas R. Hedley. Cognitive and Usability Issues in Geovisualization. *Cartography and Geographic Information Science*, 28(1):61–75, January 2001. ISSN 1523-0406. URL <http://www.tandfonline.com/doi/abs/10.1559/152304001782173998>.

- [75] Terry a. Slocum, Daniel C. Cliburn, Johannes J. Feddema, and James R. Miller. Evaluating the Usability of a Tool for Visualizing the Uncertainty of the Future Global Water Balance. *Cartography and Geographic Information Science*, 30(4): 299–317, January 2003. ISSN 1523-0406. URL <http://www.tandfonline.com/doi/abs/10.1559/152304003322606210>.
- [76] AC Robinson and Jin Chen. Combining usability techniques to design geovisualization tools for epidemiology. *Cartography and ...*, 32(4):243–255, 2005. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2786201/pdf/nihms68027.pdf><http://www.tandfonline.com/doi/abs/10.1559/152304005775194700>.
- [77] Sven Fuhrmann and William Pike. User-centered design of collaborative geovisualization tools. *Exploring ...*, (November 2004):0–8, 2005. URL <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:User-centered+Design+of+Collaborative+Geovisualization+Tools#0>.
- [78] Gilbert Cockton. Putting Value into E-valuation. *Maturing Usability*, pages 287–317, 2008. URL http://link.springer.com/chapter/10.1007/978-1-84628-941-5_13.
- [79] Iso/iec 25010:2011(en), . URL <https://www.iso.org/obp/ui#iso:std:iso-iec:25010:ed-1:v1:en>. Last visited 2014-02-22.
- [80] 10 usability heuristics for user interface design. URL <http://www.nngroup.com/articles/ten-usability-heuristics/>. Last visited 2015-05-04.
- [81] Whitney Quesenbery. Dimensions of Usability: Defining the Conversation, Driving the Process. *Proceedings of the Usability Professional’s Association (UPA) conference on Ubiquitous Usability*, 2003. URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.8658&rep=rep1&type=pdf>.
- [82] Balancing the 5es. URL <http://www.wqusability.com/articles/5es-citj0204.pdf>. Last visited 2015-05-06.
- [83] Iso 9241-210:2010, . URL http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=52075. Last visited 2015-05-04.

- [84] WilliamE. Cartwright and GaryJ. Hunter. Towards a methodology for the evaluation of multimedia geographical information products. *GeoInformatica*, 5(3): 291–315, 2001. ISSN 1384-6175.
- [85] Sven Fuhrmann, Paula Ahonen-Rainio, Robert M. Edsall, Sara I. Fabrikant, Etien L. Koua, Carolina Tobón, Colin Ware, and Stephanie Wilson. Chapter 28 - making useful and useable geovisualization: Design and evaluation issues. In Jason DykesAlan M. MacEachrenMenno-Jan Kraak, editor, *Exploring Geovisualization*, International Cartographic Association, pages 551 – 566. Elsevier, Oxford, 2005. ISBN 978-0-08-044531-1. URL <http://www.sciencedirect.com/science/article/pii/B9780080445311504462>.
- [86] Virpi Roto, Marianna Obrist, and K Väänänen-Vainio-Mattila. User experience evaluation methods in academic and industrial contexts. ... *User Experience Evaluation* ..., 2009. URL <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:User+Experience+Evaluation+Methods+in+Academic+and+Industrial+Contexts#0>.
- [87] Why you only need to test with 5 users. URL <http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>. Last visited 2015-05-13.
- [88] Creating a spatial database, . URL <http://revenant.ca/www/postgis/workshop/creatingdb.html>. Last visited 2015-05-02.
- [89] Chapter 4. using postgis: Data management and queries, . URL <http://postgis.refractory.net/documentation/manual-1.4/ch04.html>. Last visited 2015-05-02.
- [90] Epsg projection 4326 - wgs 84. URL <http://spatialreference.org/ref/epsg/4326/>. Last visited 2015-05-02.
- [91] World geodetic system 1984. URL http://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf. Last visited 2015-05-02.
- [92] Angular directives for bootstrap, . URL <https://angular-ui.github.io/bootstrap/>. Last visited 2015-04-27.

- [93] Model–view–controller. URL <http://en.wikipedia.org/wiki/Model-view-controller>. Last visited 2015-05-11.
- [94] Singleton pattern. URL http://en.wikipedia.org/wiki/Singleton_pattern. Last visited 2015-05-30.
- [95] Shaumik Daityari. Working with and around the same-origin policy - sitepoint. URL <http://www.sitepoint.com/working-around-origin-policy/>. Last visited 2015-03-10.
- [96] The css markup validation service, . URL <http://jigsaw.w3.org/css-validator/validator.html.en>. Last visited 2015-05-25.
- [97] The css validation service, . URL <http://jigsaw.w3.org/css-validator/validator.html.en>. Last visited 2015-05-25.
- [98] Testdouble. URL <http://www.martinfowler.com/bliki/TestDouble.html>. Last visited 2015-05-25.
- [99] jQuery community. jquery. URL <http://jquery.com/>. Last visited 2014-06-19.
- [100] Barnes surface. URL <http://suite.opengeo.org/4.1/cartography/rt/barnes.html>. Last visited 2015-05-12.
- [101] Rendering transformations. URL <http://docs.geoserver.org/latest/en/user/styling/sld-extensions/rendering-transform.html>. Last visited 2015-05-14.
- [102] Jsonix. URL <http://confluence.highsource.org/display/JSNX/Jsonix>. Last visited 2015-05-19.
- [103] ogc-schemas, . URL <https://github.com/highsource/ogc-schemas/tree/master/scripts/lib>. Last visited 2015-05-19.
- [104] Elisa Bertino, Andrea Maurino, and Monica Scannapieco. Guest editors' introduction: Data quality in the internet era. *IEEE Internet Computing*, 14(4):11–13, 2010. URL <http://dblp.uni-trier.de/db/journals/internet/internet14.html#BertinoMS10>.
- [105] Iso/iec 25012:2008, . URL <https://www.iso.org/obp/ui/#iso:std:iso-iec:25012:ed-1:v1:en>. Last visited 2015-05-19.