UNIVERSITY OF CALGARY

An Exploratory, Longitudinal Case Study on Testing of Web Mapping Applications

by

Abhishek Sharma

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

September, 2012

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled " An Exploratory, Longitudinal Case Study on Testing of Web Mapping Applications " submitted by Abhishek Sharma in partial fulfillment of the requirements of the degree of Master of Computer Science.

Supervisor, Dr. Frank Maurer, Department of Computer Science

Dr. Mario Costa Sousa, Department of Computer Science

Dr. Steve H.L. Liang, Department of Geomatics Engineering

Date

Abstract

Web mapping is the process of designing, implementing, generating and delivering maps on the World Wide Web. With increased ease in collecting geospatial data, web mapping applications are becoming ubiquitous. Software testing is done to ensure that applications are functioning as per expectations. Results from empirical studies showed that testing has been the least emphasized aspect of web mapping application development and little is known about it.

This thesis presents results of an exploratory, longitudinal case study on testing of web mapping applications. It includes results of a qualitative exploratory study to understand current practices and issues faced by developers. A unit testing framework for an open-source mapping library was developed and evaluated as per requirements gathered from the exploratory study. Lessons learned after a longitudinal experience in developing and testing a web mapping application are also discussed.

Acknowledgement

My odyssey through graduate school was made eventful by multiple people who encouraged me and helped me at each and every point of this journey; I would like to extend my sincere gratitude to them.

- To my parents, for standing by my side throughout my life. Thank you for all your encouragement to always strive to do the best.
- To Dr. Frank Maurer for providing me with this wonderful opportunity to come to Calgary and work under your guidance. Thanks for your limitless support, encouragement, feedback, and for being a source of inspiration during the course of this thesis.
- To Shreya, for being a great friend and an excellent mentor.
- To Teddy, Chris and Tulio, for supporting me and cheering me up. Without all the humor and interesting discussions, graduate school would never be this entertaining. Thanks for your never-ending friendship.
- To Tedd, thanks for all your support from the day I set foot in the ASE lab.
- Thanks to all my friends in ASE lab and elsewhere.

To my dear parents:

For all your love and support, all your hard work and dedication throughout my life.
Everything that I have and I will achieve in my life is because of you.

Table of Contents

List of Figures

Chapter One: Introduction

Spatial analysis of geographical data associated with an information set has always been of great interest to mankind. One of the first reported uses of spatial analysis dates back to the year 1832 by French geographer Charles Picquet, who used halftone color gradients on a map to show percentage of deaths by cholera [1]. With the advancements in technology, paper maps have been replaced with dynamic and interactive web mapping software applications. This thesis discusses testing of web mapping applications using an exploratory and longitudinal case study.

This chapter aims at providing an introduction for this thesis. Section 1.1 will provide an overview on geospatial analysis, explaining the importance of geospatial information and its use in decision making. Sections 1.2 and 1.3 will discuss the motivation behind this thesis and research questions, respectively. Sections 1.4, 1.5 and 1.6 focuses on goals, contributions of this thesis and provide an overview of thesis structure, respectively.

1.1 Geospatial Analysis Overview

Geospatial information refers to data that is tied to a set of geographic coordinates. Geospatial information is stored in coordinates and topologies and this data can be geographically mapped. Geospatial information is usually presented in association with a data set that is used to describe other variables associated with a particular location on the map, showing population density or fluoride levels in water in a state, for example.

For centuries, visual display of geospatial information in the form of maps has been of great importance in various domains. Some common examples include land planning, oil pipeline planning, and climate monitoring. Geospatial information is used as a tool for decision making.

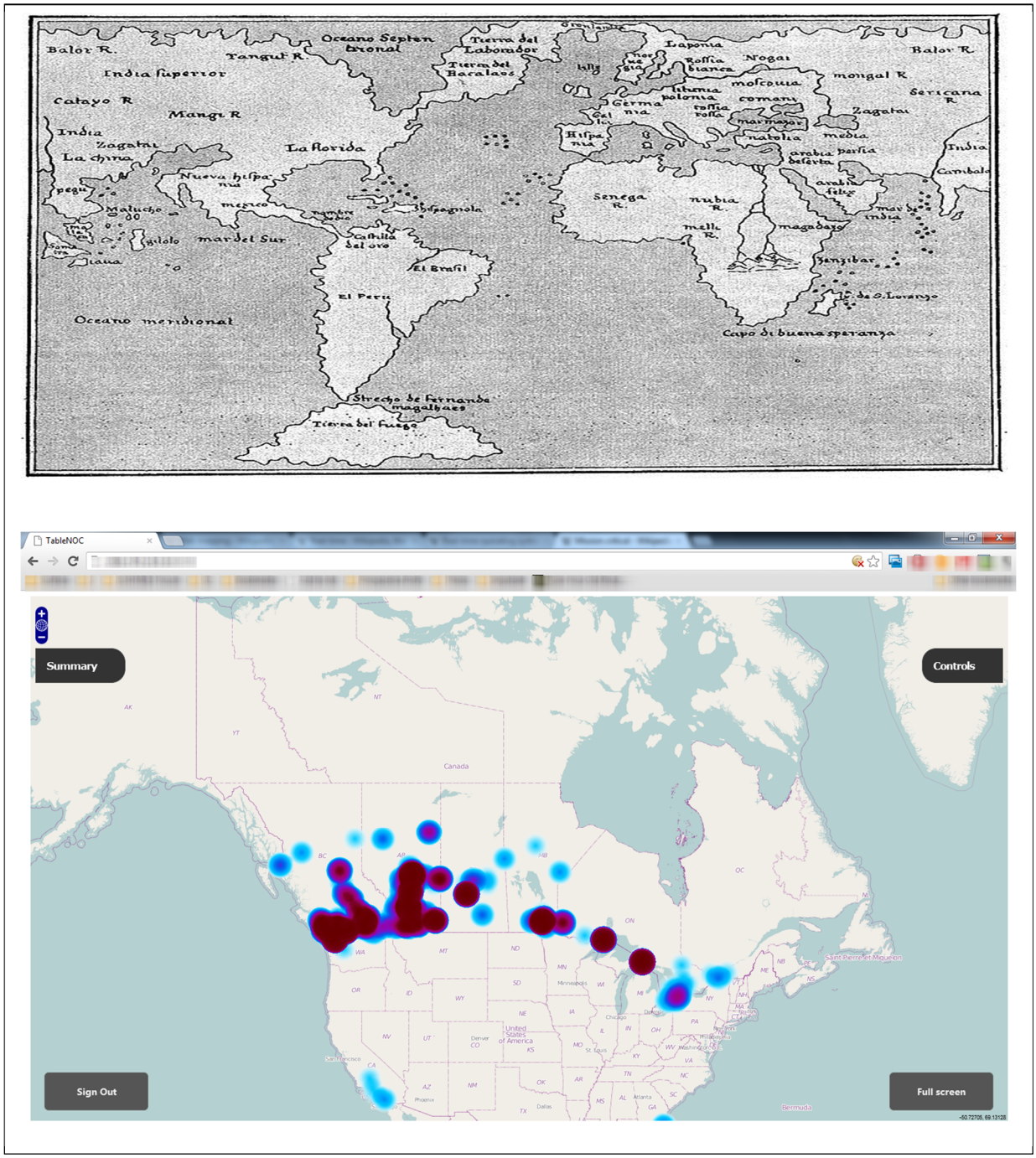

Figure 1: A paper map [95] **(upper) and A web mapping application used in NOC (lower)**

Analysis of information displayed on maps can provide important insights about how and why a decision is affecting the results of a business endeavour. For example, assessing a product's market penetration based on quantitative data across a country can help in making decisions on

focusing marketing campaigns to target desired user groups. Other similar examples may include election campaigns where displaying voting tendencies or results from pre-poll surveys on a map can turn out to be a faster and easier way to interpret the collected data. The data that is overlaid on a map can be historical data or can be collected in real-time. Examples of real-time applications include mapping applications used in telephony Network Operations Center (NOC) to visualize the location of incoming or outgoing calls on a map to monitor network load.

Methods of visualizing and analysing geospatial data have drastically changed over time. The change from paper maps to interactive real-time mapping applications has made their usage easier by providing various features in a single place instead of need for multiple printed maps. Figure 1 shows pictures of a paper map and a web mapping application that is used in a NOC to monitor network data.

1.2 Motivation

Due to their usefulness in business decision making web mapping applications are gaining popularity in industrial settings. TableNOC is a web mapping application that is being developed by the author and colleagues at the Agile Surface Engineering lab at the University of Calgary in collaboration with an industrial partner, Ivrnet Inc. The primary goal behind development of TableNOC is to provide a near real-time visualization of network call data in Network Operations Center (NOC). This web mapping application is being used for monitoring network calls and in decision making. TableNOC is being developed using open-source geospatial resources including OpenLayers [2] and MapServer [3]. At present the application is being used for decision making in two different scenarios –

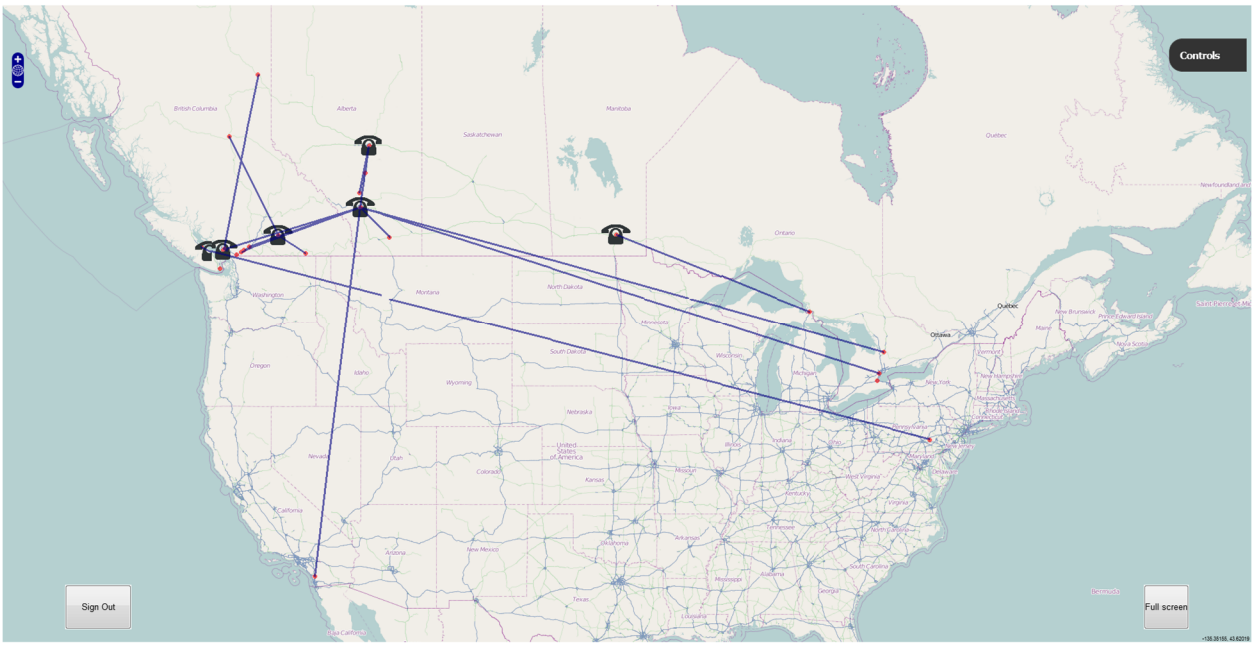

Figure 2: TableNOC - End-to-end mapping of calls

1. Network Operations Center – End-to-end mapping of calls in near real-time can be very useful in monitoring network state. This can be used for proactive decision making in cases of network congestion and fault detection. This is shown in Figure 2.
2. Sales and Marketing - TableNOC visualizes information about phone calls by linking the content of the call (sales, technical support, billing) to its location of origin on a map. This is useful in making marketing decisions based on distribution and focus of responses. Examples include election survey campaigns.

The importance of reliable functioning of TableNOC can be clearly understood on the basis of usage scenario of this application. Any failures caused due to faulty components can affect the decision making and cause financial loses as well as inconvenience to the clients, especially in cases when these failures lead to network disruption. Effective testing of an application can ensure its reliability by verifying its functioning against the expectations. After initial research to gather more information about testing of web mapping applications it was found that testing has

been the least emphasized aspect of web mapping application development and little is known about it. This became motivation to know more about testing of web mapping applications and is the primary driving force for this thesis.

1.3 Research Questions

An exploratory, longitudinal case study about testing of TableNOC forms the foundation of this thesis. Exploratory research is employed to gather insights about a field when it is in preliminary stage [4]. Exploratory studies use secondary research methods like literature reviews and qualitative studies. Exploration is a broad-ranging undertaking that starts with a generalized scope of the problems [5].

Three broad research questions were identified –

1. What is the current state of research in testing of web mapping applications?
2. How can the existing framework support for testing of web mapping applications be improved?
3. How can we effectively test a web mapping application?

1.4 Research Goals

There are three main research goals of this thesis. The first is to provide an overview of the current research in testing of web mapping applications using exploratory research methods. Data collection is done using qualitative methods like interviews and grounded theory work on forum and blogs. Results from the analysis are used to portray a picture of practices and problems encountered during testing of web mapping applications. These results are also used to draw requirements for a test framework. Chapter 3 of this thesis provides details about the exploratory study.

The second goal of this thesis is to assess existing tool support and try to improve it. This is discussed in detail in Chapter 3 and Chapter 4 of this thesis that discusses an exploratory study and OLUnitTest – the testing framework that was developed, respectively.

The third and final goal is to attempt to provide an answer to the third research question on the basis of experiences gained. These insights about various technicalities involved in development and testing of a web mapping application are gathered longitudinally along the course of development of TableNOC and are discussed in Chapter 6.

1.5 Thesis Contributions

The three major contributions of this thesis to the field of testing of web mapping applications are –

1. An up-to-date picture of current issues and practices in testing of web mapping applications based on an exploratory qualitative study.
2. Extending tool support for testing of web mapping applications by developing OLUnitTest – a unit testing framework for OpenLayers [2].
3. Recommendations for testing of web mapping applications based on longitudinal experiences gathered while developing and testing a TableNOC.

1.6 Thesis Overview

This chapter presented a background for this thesis; it included a description of web mapping applications and basics of software testing. Finally, research questions and goals were discussed in this chapter.

Chapter 2 will present the background and related work about testing of web mapping applications. Chapter 3 presents analysis of results from an exploratory qualitative study.

Chapter 4 discusses the existing tool support, explains the need of a new testing framework. A detailed description about the testing framework that is developed as a proof of concept is also presented here.

Chapter 5 presents the results of evaluation of the testing framework developed by the author. A limited user study and a self-evaluation study were conducted to evaluate OLUnitTest – the testing framework that is developed by the author.

Chapter 6 describes experiences about testing TableNOC and provides recommendations.

Finally, Chapter 7 discusses the contributions of this thesis and future work.

Chapter Two: Background and Related Work

This chapter is aimed at providing a brief contextual background for this thesis so that the contributions of this thesis can be easily understood. Section 2.1 focuses on web mapping applications, describing architecture and various components. This section explains Service-Oriented Architecture (SOA), use of spatial databases, mapping servers, discusses Open Geospatial Consortium (OGC) standards for the data, and client-side APIs for developing web mapping applications.

Section 2.2 provides an overview of testing, discussing various methods and levels of testing. The next section lists challenges of testing web mapping applications providing examples to understand the complexity of these applications. Section 2.3 lists the major challenges faced in testing of web mapping applications. Section 2.4 discusses the related work.

2.1 Web Mapping Applications

Web mapping in general is the process of designing, implementing, generating and delivering maps on the World Wide Web [6]. Web mapping and web cartography go hand in hand. Whereas web mapping deals with the technological aspect of delivering maps on the web, web cartography is related to more theoretical aspects of publishing maps. There is a very thin line between web GIS and web mapping applications. Often these terms are used synonymously, but Web GIS puts a greater emphasis on analysis, and processing of project specific data using exploratory approaches [7]. Enhanced analysis capabilities include - directional analysis, geometrical processing, map algebra, and grid models. Web mapping applications can be seen as a subset of web GIS applications.

Kraak *et al.* [8] in their book provided a classification of web mapping applications based on static or dynamic nature and further divided each category into view only or interactive maps. But due to an increase in the number of types of web maps this classification was revised to the following categories:

1. Analytic web maps – These web maps provide capabilities to analyze data. These are quite similar to web GIS, however the analysis is done on server side GIS due to limited capabilities of web browsers.
2. Animated web maps – As the name suggests these maps show changes in the map over time by animating one of the geographical or temporal variables.
3. Collaborative web maps – These maps are generated by users across the web. One of the excellent examples is OpenStreetMap, which is generated and edited by people using it over the Internet.
4. Realtime web maps – These maps show a variable in close to realtime. These maps are primarily used in monitoring and control systems.
5. Static web maps – These are the most primitive type of web maps providing no animation or interactivity. As the name suggests these are static images which are infrequently or never updated.

A web mapping application is comprised of different components. This often includes databases, mapping server, web application server, and web services. Figure 3 shows the architecture of a web mapping application that uses GeoServer [9]. The following sub-sections will discuss in detail the architecture of a web mapping application by explaining individual components used by a web mapping application.

2.1.1 Service-Oriented Architecture for Web Mapping applications

The Service-Oriented Architecture approach emphasizes on loosely coupled components in an application. This architecture is based on services provided by various components that can be reused or swapped in and out as and when required. This architecture seems to be a good fit for web mapping applications because of its loosely coupled nature [10]. Firstly, in contrast to standard GIS applications where normally only a small percentage of the functionalities in the software are used, web mapping applications based on SOA provide users with just the functionality they need. This makes these applications accessible from light weight clients like cell phones and other hand-held devices. Secondly, SOA prevents inconsistency in local copies since all the clients access the system from a single source [11]. Figure 3 shows that components of a web mapping application use various services provided by mapping servers, these services include Web Map Service (WMS), Web Feature Service (WFS), Web Coverage Service (WCS) and so on.

A Web Map Service (WMS) is a standard protocol for serving geo-referenced map images over the Internet that are generated by a map server using data from a geospatial database. A WMS request defines the geographic layer(s) and area of interest to be processed. The response to the request is one or more geo-registered map images (returned as JPEG, PNG, etc) that can be displayed in a browser application. The WMS interface also supports the ability to specify whether the returned images should be transparent so that layers from multiple servers can be combined or not [12].

The OGC Web Feature Service (WFS) standard [13] [6] provides an interface allowing requests for geographical features across the web using platform-independent calls. A geographical feature is the component of the Earth that is present within a region that is bound inside a map

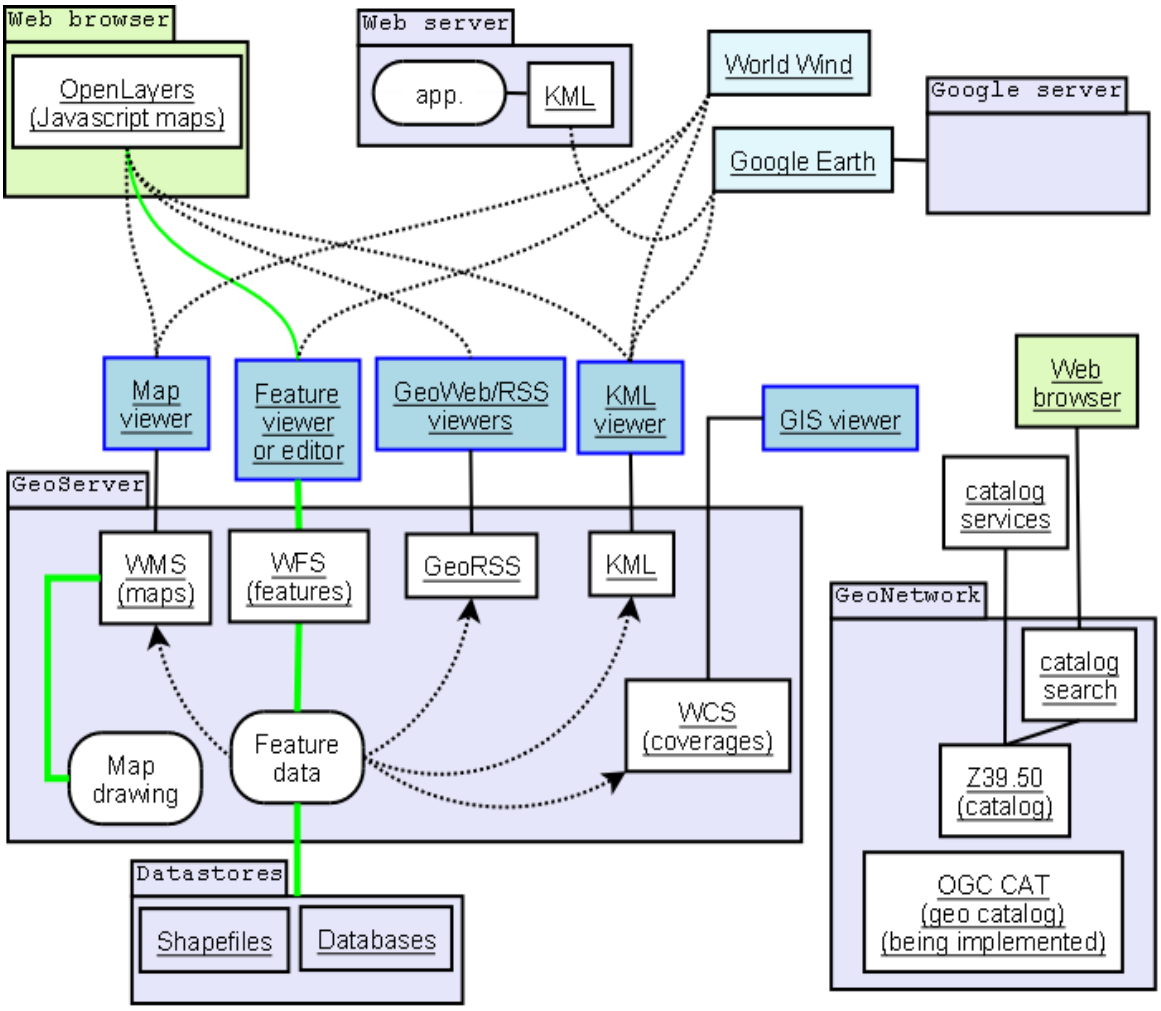

Figure 3: Architecture of a web mapping application that uses GeoServer [93]

[14]. For example, rivers, lakes, roads, houses. WFS can be used for data manipulation operations of geographic features. These operations include:

- get or query features based on spatial and non-spatial constraints
- creation of new feature instances
- delete a feature instance
- update a feature instance

The basic WFS can be used for querying and retrieval of features, whereas a transactional Web Feature Service (WFS-T) is required for creation, deletion and updating of features. The client

generates the request and posts it to a web feature server using HTTP. The web feature server then executes the request. There are two encodings that can be used for WFS operations:

- XML
- Keyword-Value pairs

The OGC Web Coverage Service (WCS) supports electronic retrieval of geospatial data as "coverages" – which is digital geospatial information representing space/time-varying phenomena. A WCS provides access to coverage data in forms that are useful for client-side rendering, as input into scientific models, and for other clients [15].

Further SOA provides clear separation between components, ease of reuse and flexibility of using light weight clients are some other benefits. But SOA does come with some challenges like lack of testing, security and interoperability.

2.1.2 Spatial Database

A spatial database is a database that is optimized to store, retrieve and query spatial data. Spatial data may include points, lines, and polygons. These databases are modified to understand specific types of features or geometry contained in a data set. The Open Geospatial Consortium (OGC) and International Organization for Standardization has created a standard ISO 19125 [16] or Simple Feature Access (SFA) to specify a common storage model for geographic data. This includes defining geographical data using well-known text [6]. Well-known text (WKT) is a text markup language for representing vector geometry objects on a map, spatial reference systems of spatial objects and transformations between spatial reference systems [6]. This standard also defines spatial predicates and operators that can be used to generate new geometries from existing geometries.

Spatial databases provide some special characteristics in addition to features provided by other database systems. These include:

1. Spatial indexing – Indexes provided by non-spatial databases cannot effectively handle indexing of data with spatial properties. These properties include distance between points and presence of points in a bounding box. Some of the common spatial indexing techniques are grid, Z-order, quadtree, octree, UB-tree, and R-tree.
2. Spatial measurements – Finding distance between points, lines and polygons.
3. Spatial functions – Methods to modify existing features to create new ones, intersection of features, for example creating a bounding box around features. Features are real world objects, whether natural or man-made, that are represented on a map [14].
4. Constructor functions – Functions to create new features from queries whenever vertices are specified that can make up lines. This is also applicable on multiple line segments that can form polygons.
5. Observer functions – Functions to retrieve specific geometrical information related to a feature, like center of a circle, intersection of line segments, and angle between lines.

All OGC compliant databases can support storing and processing of spatial data. Some common spatial databases are:

- Microsoft SQL Server 2008 onwards
- Postgre SQL with PostGIS extension
- Esri geodatabases
- MongoDB
- Oracle Spatial
- IBM DB2 with Spatial Extender
- MySQL (partial support for data type geometry)
- SpatialLite

Geospatial vector data can also be stored as Esri shapefiles. A shapefile stores nontopological geometry and attribute information for spatial features in a data set. This geometry is stored as a shape comprising a set of vector coordinates [17].

2.1.3 Mapping Servers

A mapping server is core to a web mapping application. Mapping server directly interacts with the spatial database and is used for creating and managing geospatial web services. A mapping server can be deployed on-premises within organization's service oriented architecture or can be deployed on a cloud as per requirements. As shown in Figure 3 a mapping server can provide a variety of services that adhere to OGC standards. There are both open source and proprietary options available for mapping servers like MapServer [3], GeoServer [9], Esri's ArcGIS server [18], Deegree [19]. The upcoming subsections will discuss two popular open-source mapping servers - GeoServer and MapServer, which are frequently used by web mapping application developers and are also used by the author as mapping servers in TableNOC.

2.1.3.1 GeoServer

GeoServer is an open-source mapping server written in Java that allows users to publish and modify data from any major spatial data source using open standards. GeoServer uses the Restlet [20] framework to provide REST services and has Jetty [21] as the packaged server. Figure 4 shows GeoServer's high-level architecture showing different modules that actively interact at runtime using the Spring IOC framework. GeoServer can read a variety of data formats including: PostGIS, Oracle Spatial, ArcSDE, DB2, MySQL, Shapefiles, GeoTIFF, GTOPO30, ECW, and MrSID. GeoServer essentially comprises two aspects – the configuration aspect and the data store or rendering aspect. The entire configuration is done using the web interface and XML configuration files. Data Store is used for rendering of features. Geoserver supports many

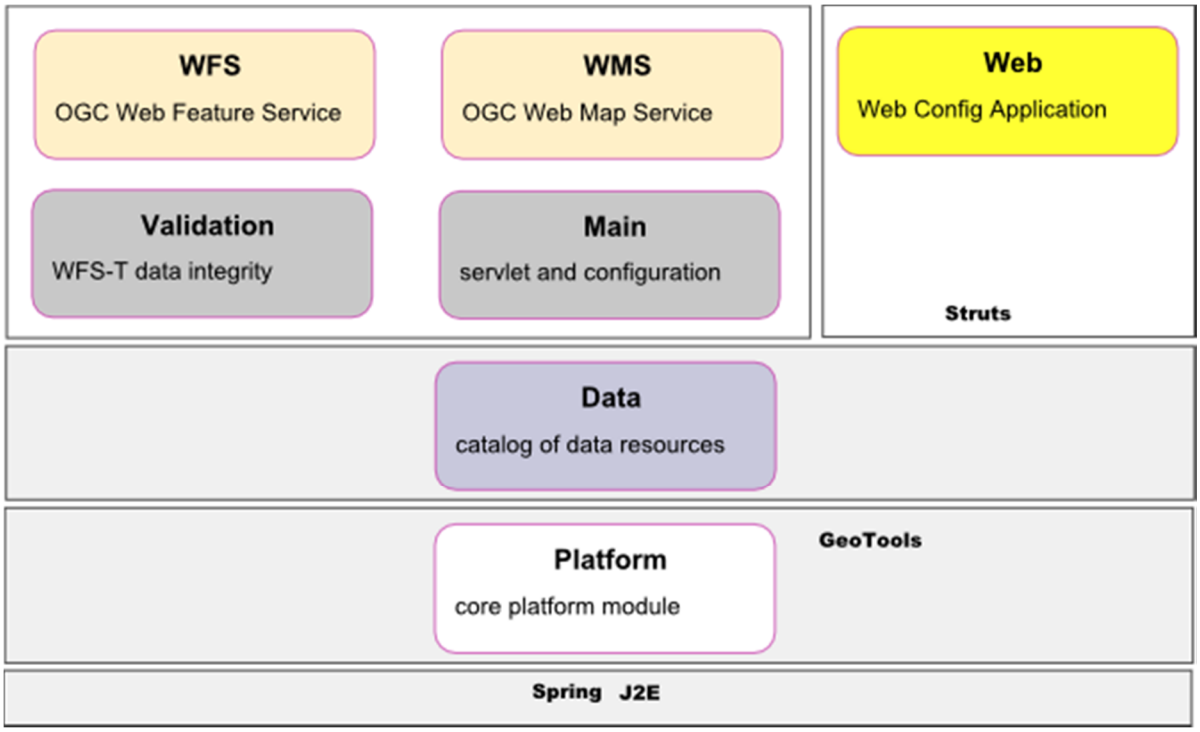

Figure 4: GeoServer - high level architecture showing different modules

different data stores including Web Feature Server [6], Property files, Shapefiles [17] and databases.

GeoServer is being used by a couple of big organizations for providing mapping services, these organizations include - Massachusetts state GIS, Ordance Survey (National Mapping Agency of UK), Food and Agricultural Organization of United Nations, World Bank.

2.1.3.2 MapServer

MapServer is an open-source project that is used to display dynamic spatial maps over the internet. MapServer can run as Common Gateway Interface (CGI) or via MapScript [22] that supports multiple programming languages. MapScript is a scripting language that can be used to interact with MapServer programmatically. MapServer has support for display and querying of hundreds of raster, vector and database formats, it is platform independent, can run on various operating systems, and supports on-the-fly map projections. A map projection is any method of

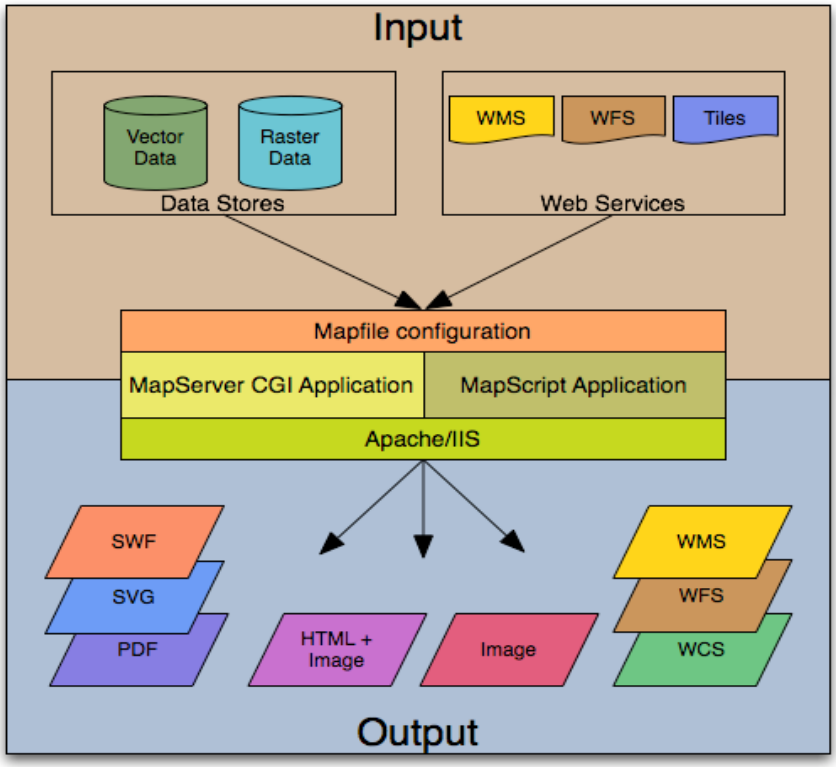

Figure 5: MapServer Architecture

representing the surface of a sphere or other three-dimensional body on a plane [23]. Figure 5 shows the architecture of basic MapServer applications. A simple map server application comprises following components:

1. Map File – a structured configuration file to describe configuration settings for the map. This has connection parameters for the database and layer properties.
2. Input data – this can be in the form of any OGC standards.
3. MapServer CGI – the binary or executable file that receives requests and returns images, data.
4. Web/HTTP Server – this hosts the MapServer CGI so that it can be provided as a service.

2.1.4 OGC Standards for Geographic Data

Open Geospatial Consortium (OGC) is an international organization that was established in 1994 for encouraging development and implementation of open standards for geospatial content and services. Abstract Specification is a set of documents that define generalized architecture for the backbone of OGC standards; other standards that are defined for specific needs of geospatial data are built atop Abstract Specifications [24]. The OGC standard baseline comprises more than 30 standards, some of them are:

- Catalog Service for the Web (CSW)
- Geography Markup Language (GML) – XML format for geographical information
- Geospatial eXtensible Access Control Markup Language (GeoXACML)
- Keyhole Markup Language (KML)
- Web Coverage Service (WCS)
- Web Feature Service (WFS)
- Web Map Service (WMS)
- Styled Layer Descriptor (SLD)

The OGC Catalog Service defines common interfaces to discover, browse, and query metadata about data, services, and other potential resources [25]. GML [26] and KML [27] are XML notations that are defined to express geographical features and serve as open interchange format for geographic transactions over the internet. KML was originally developed to work with Google Earth [28] but was later accepted by OGC as a standard since it is complementary to most of the key existing OGC standards including GML, WFS and WMS. KML is an XML language focused on geographic visualization, including annotation of maps and images. GeoXACML [29] is the geo-specific extension to eXtensible Access Control Markup Language (XACML). XACML defines a declarative access control policy language implemented in XML

and a processing model describing how to evaluate authorization requests according to the rules defined in policies [30]. A SLD is an XML schema specified by the OGC for describing the appearance of map layers. It is capable of describing the rendering of vector and raster data. A typical use of SLDs is to instruct a Web Map Service (WMS) of how to render a specific layer [31]. Details about WMS, WFS and WCS can be found under subsection 2.1.1.

2.1.5 Client-side API

Client-side APIs are used for building rich mapping applications that can be displayed in a browser. These APIs can be used to embed dynamic maps in any web page. Client-side APIs are available both in JavaScript and Flash / Flex options. Some of the popular open-source APIs include:

1. OpenLayers – OpenLayers [2] is an open-source JavaScript library for displaying map data. It provides an API for building rich map applications similar to proprietary web mapping applications like Google Maps [32], and Bing Maps [33]. A variety of data formats are supported by Openlayers including GeoRSS, KML, GML, GeoJSON and data from mapping servers that follow OGC standards.
2. OpenScales – OpenScales [34] is an open-source mapping framework that is written using ActionScript 3 and Flex. OpenScales enables users to create rich mapping applications for both desktop and mobile environments.
3. GeoExt – GeoExt [35] is another open-source JavaScript mapping library that has features of both ExtJS [36] and OpenLayers. ExtJS allows building web applications using AJAX, DHTML and DOM scripting. ExtJS provides a large set of interactive GUI controls that can be used in web applications. All these features are also inherited by GeoExt. GeoExt is primarily used to develop powerful desktop style web.

TableNOC also uses OpenLayers and GeoExt APIs for displaying a map.

Other proprietary APIs include Google Maps API [37], Bing Maps API [38], ArcGIS JavaScript API [39]. These APIs can be used to embed Google Maps [32] and Bing Maps [33] sites into web pages. ArcGIS Mapping APIs are used to build and embed interactive maps that use Web services from ArcGIS Server [40].

2.2 Testing – An Overview

Software development comes with a basic problem that applications usually have software bugs [41]. A Software bug is a common term that represents a wide range of errors or mistakes that cause unexpected behaviour in a software system. Bugs can be seen as consequences of human factors in the task of programming. A software failure is a result of a ripple effect caused by bugs in a system that make the defect noticeable and render system unable to function [42].

Software testing is any activity aimed at evaluating an attribute and capability of a program or system and determining that it meets its required results [43]. Testing can be seen as a method which is practiced to make sure a system works as per expectations. Tests are usually used to improve quality, verify and validate the behaviour of the application and code and for estimating the reliability of the program. A test by and large consists of two parts, a set of instructions and expected result [44]. These instructions can be used to test interactions with the application, or a part of the application. A bug is reported if the obtained results are different from expected results; this provides a mechanism to check if the application under test shows any discrepancy with the desired behaviour.

A lack of proper testing has caused some horrific disasters, for example the Ariane 5 satellite launcher malfunction was caused by a faulty software exception routine resulting from a bad 64-

bit floating point to 16-bit integer conversion [45]. Further bugs in mission critical systems like software for aircraft and health systems are dangerous to human lives as well as property [46], [47]. Above mentioned examples clearly indicate the importance of testing in software systems.

This thesis will classify testing based on methods and levels of testing.

2.2.1 Methods of Testing

Traditional classification divides testing into two broader categories – The box approach, which is based on how a test engineer looks at the system, and Graphical User Interface (GUI) testing.

1. The box approach [48]–
    a. White-box testing – With this testing approach, the tester has access to the internal of the system; including data structures, algorithms and code that implements them. This provides the tester with power of testing the application using various methods like fault-injection, mutation, and statically testing the code. This testing approach focuses on correctness of internal structure of an application. White-box testing also helps in determining the coverage of test suite or in simple terms completeness of test suite depending on function and statement coverage.
    b. Black-box testing – As the name suggests this technique treats an application like an opaque black box assuming no knowledge of internal structure. Some of the common black-box testing techniques include: equivalence partitioning, model-based testing, exploratory testing and specification testing. Black-box testing comes with an inherent disadvantage of “blind exploration”, due to absence of information about internal structure there might be some parts of the application

that might remain untested. Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.

2. Graphical User Interface (GUI) Testing – For the purpose of this thesis, GUI testing refers to a system level testing technique that is used on a complete, integrated system to ensure functioning of the system as per expectations. This testing technique is primarily used to detect defects in a system as a whole [49] [50]. GUI testing can be done using both manual exploratory approach [51] [52] and automated testing tools [53]. Exploratory testing is defined as simultaneous learning, test design, and test execution; that is, the tests are not defined in advance in an established test plan, but are dynamically designed, executed, and modified [49]. A manual exploratory testing approach is used for system level testing of the web mapping application (TableNOC) developed by the author.

2.2.2 Levels of Testing

The Software Engineering Body of Knowledge (SWEBOK) classifies testing based on levels where the tests are added [49], the levels defined are – unit-, integration-, and system testing. Other levels of testing are categorized by the testing objective rather than where tests are added.

1. Test target – These levels are defined on the basis of where the testing is done during the development and maintenance process. This can be further broken down into three sub levels.
    a. Unit testing – Unit testing is used to verify if isolated pieces of a software are working as per expectations. These pieces can be small modules of code or can be tightly bound units of code. Unit tests are typically written by the developer who writes the code. The primary goal of unit testing is to isolate each part of the

program and verify that it is working correctly [54]. Unit test can be compared to a written contract which ensures that a piece of code is working as per expectations. Unit tests help in early detection of problems, refactoring of code, reduces uncertainty in units, in turn simplifying the integration of components and last but not least as a tool for gaining a better understanding of the system [55]. Unit test suites can later be used for regression testing of the system for ensuring correct functioning of units for a longer term.

b. Integration testing [56]– As the name suggests integration testing is done to verify how different components interact with each other. Usually for larger projects integration testing is done incrementally when components are added to the project. Integration testing identifies problems that occur when units are combined, because any errors discovered when combining units are likely related to the interface between units. Integration testing can be done in a variety of ways but the three common strategies that are usually followed are:

    i. The top-down approach, where the highest-level modules should be test and integrated first. This allows high-level logic and data flow to be tested early in the process.

    ii. The bottom-up approach requires the lowest-level units be tested and integrated first. These units are frequently referred to as utility modules. By using this approach, utility modules are tested early in the development process.

    iii. The umbrella approach requires testing along functional data and control-flow paths. First, the inputs for functions are integrated in the bottom-up

pattern discussed above. The outputs for each function are then integrated in the top-down manner.

c. System testing – System testing is done at the top most level; this testing technique is used to verify the behaviour of the system as a whole. Some testing techniques that fall under the category of system testing are: Graphical user interface testing, software performance testing, load testing, security testing, scalability testing.

2. Objectives of testing [49] - Testing is conducted in view of a specific objective, which is stated more or less explicitly, and with varying degrees of precision. Stating the objective in precise, quantitative terms allows control to be established over the test process. Testing can be aimed at verifying different properties. Test cases can be designed to check that the specifications are correctly implemented, which is variously referred to in the literature as conformance testing, correctness testing, or functional testing. However, several other non-functional properties may be tested as well, including performance, reliability, and usability. Some of the common testing types based on objectives of testing are: Acceptance testing, Installation testing, Conformance testing, Alpha and beta testing, Performance testing, Regression testing, Stress testing, and Usability testing.

2.2.3 Code Coverage Analysis

Code coverage is a measure used in software testing. It describes the degree to which the source code of a program has been tested [57]. Some of the prominent coverage criteria are [58]:

- Function coverage – Has each function in the program been called?
- Statement coverage – Has each statement in the program been executed?
- Decision coverage – Has every edge in the program been executed?

- Condition coverage – Has each Boolean sub-expression evaluated to both true and false?

Code coverage is usually reported on a percentage scale, for example if 9 out of 10 statements are executed, 90% statement coverage is reported. Code coverage bears a direct correlation with software reliability in general. Increase in code coverage is likely to increase reliability [59].

For the purpose of this thesis statement coverage is defined as statement execution coverage, where a statement is considered tested if it is executed at least once by the tests.

2.3 Major Challenges of Testing Web Mapping Applications

Subsections 2.1 and 2.2 provided a contextual background describing in detail about web mapping applications and testing respectively, this subsection will present the major challenges of testing web mapping applications. Web mapping applications can be considered as a special type of web application, but this category of applications has some characteristics that are specific to the geographical information domain. Web mapping applications are similar to web applications in the sense that they follow the same architecture [60] as web applications and are accessed over the Internet. Testing of Web mapping applications come with challenges of testing web systems and complexities of geospatial domain. Challenges [61] [62] [63] faced while testing web mapping applications can be broadly subdivided into two categories.

2.3.1 Complexity

Web mapping applications have some properties that add to the complexity of the application, which in turn increase the difficulty in efficiently testing a web mapping application. Some of these properties are listed below:

1. Complex architecture – Web mapping applications have a complex architecture in which different technologies are interconnected with each other. In addition to this, web

mapping applications are based on off-the-shelf components which are tied together in a dynamic fashion.

2. Heterogeneous environment – These applications can be composed of different execution environments depending on the requirements; this includes variety of operating systems, Web servers, and Web browsers.
3. SOA and third-party components – Using third-party components makes the applications prone to hidden bugs, since at some level the developer has to trust third-party components for providing correct services. Testing third-party services has always posed a challenge when following SOA, since these services are not under control of the consumers. Further, with SOA service level testing has a greater importance than system level testing.
4. Dynamic and real-time nature of web mapping applications – When web mapping applications dynamically generate data for user requests, different components participate in this process of dynamic data generation including mapping server and databases. This in turn increases the number of points of failure of a web mapping application. Further, some mapping applications map data that is being gathered in near real-time. This dynamic nature of the application increases the number of interactions possible with the application, increasing the test cases manifold. These features add to the complexity of applications under test.

2.3.2 Lack of tool support

Testing of web mapping applications requires domain specific knowledge due to the specialized nature of applications belonging to this category. This knowledge is related to the geospatial domain and is required to write test cases that are specific to geospatial errors caused by mapping

libraries and mapping servers. Lack of proper tools and testing frameworks that can be used to efficiently test web-mapping applications is another major setback in testing mapping applications. This will be discussed in detail in Chapter 3 of this thesis.

2.4 Related Works

A systematic search [64] was conducted on IEEE Xplore [65] and Scopus [66] to obtain publications. It was noted that some researchers use the terms web mapping and web GIS interchangeably. So, for our search both web mapping and web GIS applications are included to avoid negligence of any relevant literature. The following search string was used with scope limited to title and keywords.

("testing" AND ("web mapping" OR "GIS" OR "geographic information system"))

A total of 34 papers were obtained from the search when the domain was limited to computer science or/and the geospatial domain. Out of these following 3 papers were found relevant to this thesis and are discussed below.

Liu and Tang [67] discuss exploratory research in GIS software testing. Testing experiments were conducted on two spatio-temporal software systems. This paper points out how little attention the GIS community has given to software testing. Further, it reports that a greater emphasis is placed on non-functional testing techniques like security testing. A testing model that is devised for experimentation is reported in this paper. This paper points to the lack of sufficient attention towards testing of GIS software, however this claim is totally subjective on authors' opinion and is not substantiated by any empirical evidence.

Maogui and Jinfeng [68] discuss the use of an automated testing tool in GIS modeling. They have conducted an experiment using an automation tool – AutoIt for automating some simple

tasks on the GUI level. These tasks include finding answers to simple questions like “how much area of the land is suitable for farming in every county?” in an automated fashion that would have otherwise required a user to manually perform this operation. It should be noted that this paper did not provide any discussion from a testing perspective but it did explore the possibility of automating user interaction.

The GIS community puts greater emphasis on performance and availability testing. Along these lines, in a similar experiment Horak *et al.* [69] propose a way to test and measure the availability of Web Mapping Service (WMS) for end users. The primary focus of this experiment is performance testing of a web mapping applications. This study uses various different techniques for stress and availability testing of a web mapping application both for shorter and longer term durations.

From the low number of results from the search for relevant literature it can be deduced that testing of web mapping applications is still a domain to be explored or it can be an irrelevant problem. Further, none of the papers made an attempt to explore and gather more information about testing of functionality of a web mapping application. A greater emphasis can be seen on testing of non-functional requirements than testing the functionality and correctness of a web mapping application. A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors [70]. Non-functional testing includes benchmarking, performance testing, stress testing, security testing and so on. Secondly, no paper reported any qualitative or empirical studies that were done to gather more information about testing of web mapping applications. Finally, neither current tool and framework support was assessed nor any attempt to make addition to it was listed in the literature.

Chapter Three: Testing of Web Mapping Applications - An Exploratory Study

The low number of results from the search for relevant literature became a motivation for an exploratory study to gather details about testing of web mapping applications and to investigate whether this is an relevant problem or not. Exploratory research is usually conducted in an early phase where a problem is not yet clearly defined, or its scope is unclear [5]. Exploratory research also helps in determining research design and data collection methods for future studies. This type of research relies on secondary methods that include but are not limited to interviews, focus groups, pilot studies.

The aim of this study was to understand current practices and issues that are faced by developers while developing and testing of web mapping applications. This chapter is organized into four subsections where data collection, analysis, and results are discussed, respectively.

3.1 Data Collection

Data was collected from two sources. First, a series of semi-structured interviews were conducted with participants having experience with the development of web mapping and web GIS applications. The primary reason behind keeping these interviews semi-structured was to let participants freely express their opinions and experiences about testing of web mapping applications.

A total of 9 Participants were interviewed, Figure 6 shows the experience of participants in developing web mapping applications or/and GIS applications. Participants P4, P6, P7, and P8 were professionals from industry and the rest of the participants were from academia. Participants had a mixed background, where 5 of them have a computer science background and rest are from geomatics.

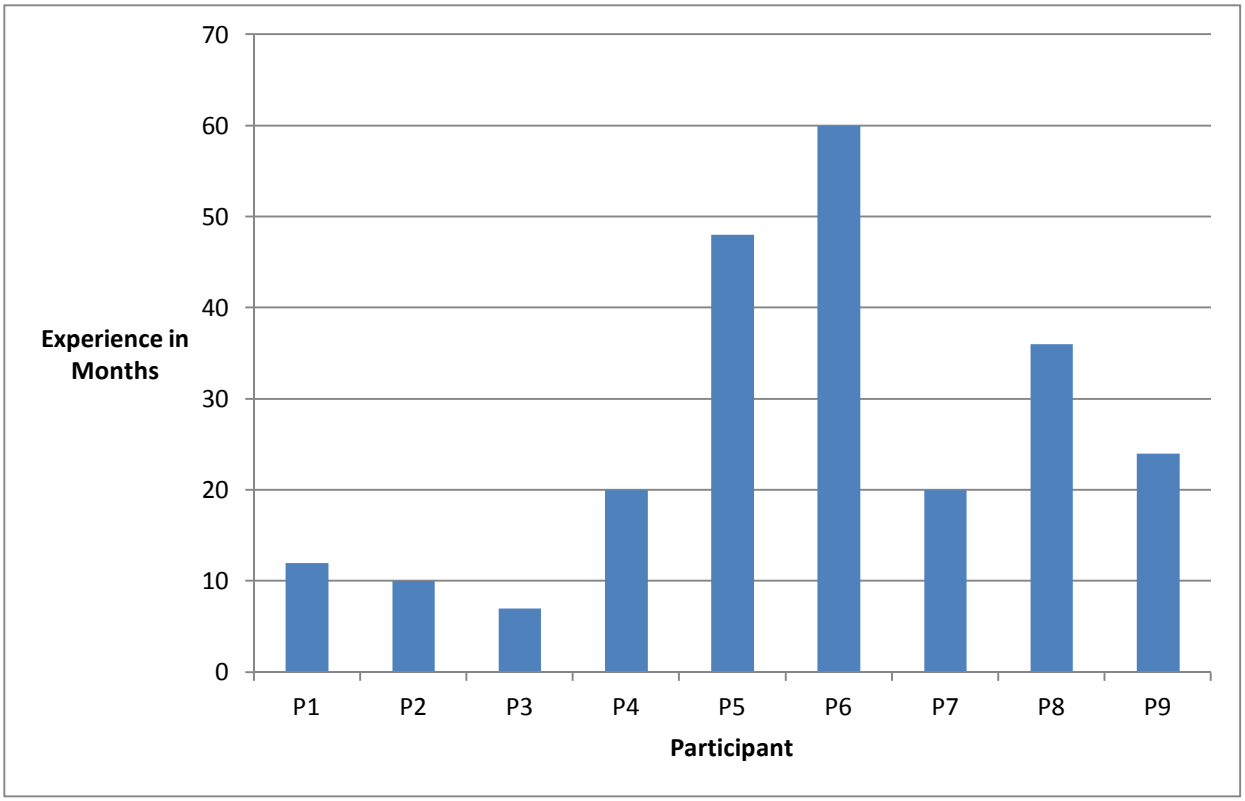

Figure 6: Participant experience in developing web mapping applications – Exploratory study

These interviews were conducted both in person and over email. Participants included both developers from academia and professionals from industry.

The second source of data involved a systematic search on forums and blogs. The OSGeo forum [71] was searched exhaustively for posts related to testing of web mapping applications. Search was conducted using following search string: *("testing" AND ("web mapping" OR "GIS" OR "geographic information system")*. The search results were limited to first 15 pages in the order of relevance, which approximately included about 3300 user replies on various threads in the forum. Google Blog search [72] was used to search for relevant blog posts with results limited to first 9 pages of the search results. Only 3 blog posts were qualified for further analysis.

3.2 Analysis

Keeping in consideration the exploratory nature of this study, the Grounded Theory method by Glaser and Strauss [73] was used to analyze the data collected from the interviews and from forums and blogs. The grounded theory approach was chosen for the following reasons:

1. The grounded theory method operates in a reverse fashion than traditional methods, rather than beginning with a hypothesis, the first step is data collection. This was one of the critical factors for preserving the exploratory nature of this study and to avoid any preconceived notions about testing of web mapping applications.
2. This method aims at systematic generation of theory from data, in other words formulate hypotheses based on conceptual ideas. A lack of any substantial work done in direction of this thesis left us with absence of any theory and hence generation of theory from the data seemed to be the most appropriate step.
3. Another goal of a grounded theory study is to discover the participants' main concern and how they continually try to resolve it.
4. This method aims at conceptualizing what is going on by using empirical research rather than aiming for accurate descriptions. Grounded theory is about concepts rather than being a descriptive method.

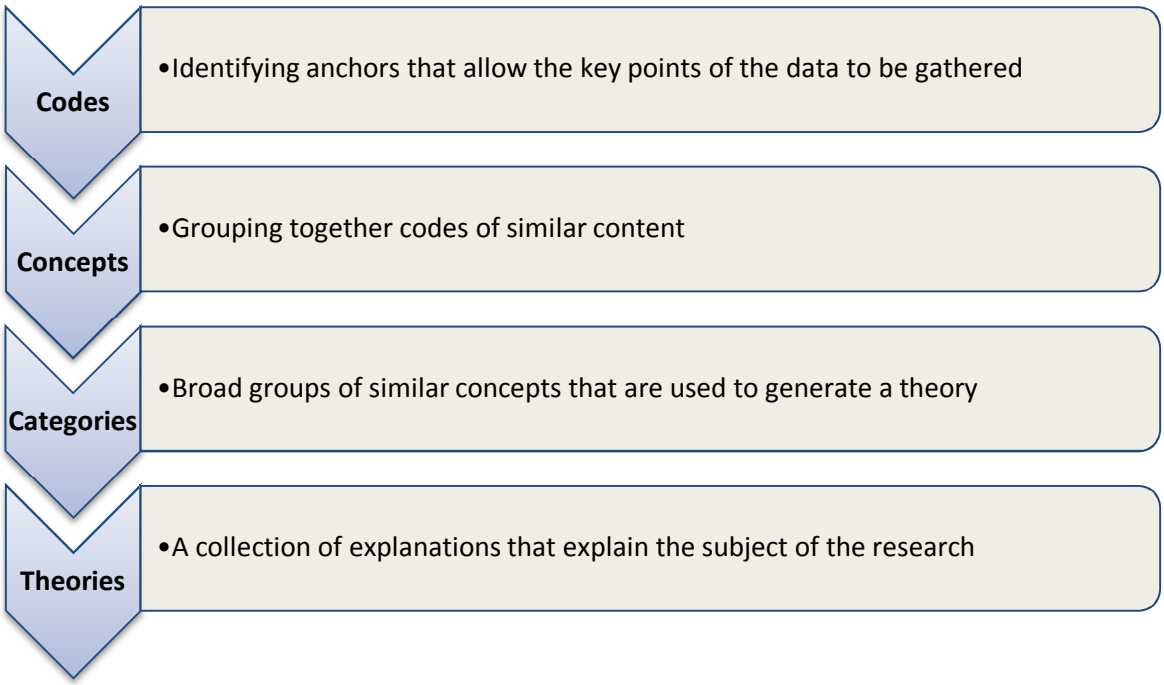

Figure 7: Grounded Theory Method - Four Stage Analysis [73]

Glaser & Strauss in their book – The Discovery of Grounded Theory [73] suggested a four stage analysis process, the flowchart in Figure 7 shows this process.

The first stage of this process involved open coding where everything that was collected was coded. This is can be seen as the first level of abstraction where line by line coding is done. For example the following statement by a participant –

"*...usually we don't do automated testing [of functionalities] because it takes too much time to set that up, just looking at the system can give you an idea if it is [working] fine or not...*".

was assigned these codes – less automated testing, time consuming, visual feedback, manual testing.

This was followed by selective coding to group together codes to form concepts. Selective coding is the process of choosing one code to be the core, and relating all other codes to that code. This helps in wrapping up similar codes together. For example codes like – *not updating to*

latest version, used one stable version, swapping old version for new requires retesting, can be combined together and represented by one concept – *stick to one version*.

The next step involved sorting these concepts into categories like *stick to one version* and *manual testing takes time* can be put together into a category *resistance towards change*. In the above example two concepts are tied together and are placed in the category, it can be interpreted as –Since entire system must be retested whenever a component is updated and *manual testing takes time* so developers tend to *stick to one version* therefore this becomes a reason for *resistance towards change.* These categories can later be used to generate explanations similar to practices and issues that are described below.

3.3 Results

After the four stage analysis process, high level theories were formulated. These are presented below in two categories – Practices and Issues related to testing of web mapping applications.

3.3.1 Practices

Practices correspond to activities related to testing of web mapping applications that are being followed by developers.

1. *Dependency on visual feedback* – It was mentioned by almost all the participants that while developing web mapping applications they usually depend on visual feedback rather than automated checks. One of the participants stated, "… *just looking at the system can give you an idea if it is [working] fine or not…".* Similarly, participant P1 said this when asked about how did he recently detect a problem – *"...it was not displaying any data when I logged into the application…after zooming in and zooming out a couple of times the issue seemed to be with the bounding box function which was*

silently failing…". This is not surprising since maps in general are used for visualization of data that can be tied up with geographical locations. It was indicated that bug discovery, bug reporting and debugging are greatly dependent on visual feedback when working with web mapping applications. Participant P5 stated –"*...after few months finding issues and solving them becomes a bit easy…obviously if you come across a problem frequently, you look at it and you know what is wrong…".* Similarly when asked about the debugging approach P3 said – "*...for this I use hit and trial process, make a guess what could be wrong, change it, load the application and see if it is fixed…sometimes I just get lucky when one change fixes multiple problems…".*

2. *No or minimal attention towards automated testing of functionalities* – It turned out to be a fact that minimal attention is paid towards automated testing in general. Features and functionalities are rarely tested in an automated fashion. For example, a statement by participant P1 where he was discussing how he fixes issues when they are discovered either by him or some user – "*...looking into every feature to see if it is working is not possible, things do break sometimes but how would I [automatically] know if something is broken…we don't have tests written for that."* Though automated non-functional testing techniques like load testing and stress testing were used more often. A quote from an interview serves as a good example, "*...I do test the time required by the map to load and also how much time does the mapping server takes to generate tiles…".* Benchmarking performance seems to be an important practice for applications that are being developed in industry.
3. *Manual testing vs. Automated testing* – It was noted that developers rely highly on manual testing and reliance on automated testing was close to none. Manual exploratory

testing is a technique that is used by developers for detecting bugs on the system level. Only one instance reported use of an automated regression test suite to see if a set of functionality was working as per expectations or not.

4. *Testing of third-party components and integration testing* – the GIS community relies heavily on the use of third-party components, usually mapping servers. A trust factor was noted when using third-party components. Benchmarking of third-party components was commonly observed but integration testing for errors is not commonly practiced.
5. *Resistance towards change* – An interesting fact that was uncovered in the last stage of grounded theory method: resistance towards change. The majority of web mapping applications follow a service-oriented architecture where individual components provide services that are consumed by clients, these components need to be updated due to various reasons like release of a newer version. On the contrary to the belief that service-oriented architecture provides loose coupling so switching components for newer versions can be done frequently, it was found that developers try to avoid this and stick with the working version of the system. From the analysis it can be deduced that primary reason behind this is time consumed to manually re-test all the features after the update is applied. Participant P4 stated – "*...well... we don't have [automated regression] test suites to check if anything is broken and [manually] testing it again and again after some library or server is moved to a different version is too much hassle."* Participant P9 stated – *"...I tried to do that [update components to newer versions] for a small company and they wanted it to be thoroughly tested... it was sheer pain... [manual] testing killed a lot of time."* These statements seem to be in agreement with the results that were obtained from the analysis.

3.3.2 Issues

Some of the prominent issues that were reported are:

1. *Lack of specialized tool support for automated testing* – This was the most common issue that was reported by developers when testing web mapping applications. During interviews almost all the participants mentioned this at least once and maybe more on different occasions. Participant P6 stated – "*…few months ago I tried to use Selenium, just as an experiment to see if it can be used for automating some actions …but I was not impressed by that tool…at least for me it seemed impossible to write tests for some of the features…[when asked to provide an example]…well…how would you write a test for a simple use case of zooming or panning the map to verify [geographic] features…it is very complicated to do so with tools that are not designed for specifically testing* [web] *mapping applications…"*. Selenium [74] is a browser automation portable testing framework that provides a record/playback tool for authoring tests without learning a test scripting language.

   Participant P8's opinion about unit testing framework was – "*…I have tried it only twice and it is not impossible to write tests for OpenLayers code with JsUnit or Test Another Way, it is just annoying…you will have to write same lines again and again…and testing properties related to geometry of features is not supported…with WFS I use geometry class quite often and this is the first thing I look into any testing framework… "*. These statements can be used as an argument to support the case for a need of testing tools that are designed keeping in consideration domain specific information. Further, this seems to affect the usability of traditional testing tools and frameworks that are used for testing web applications. On one occasion a developer

reported - *"...amount of time required to write test code for [web] mapping applications using current frameworks discourages us to write test code."* The same issue was encountered when collecting data from forums and blog posts, developers were trying all available options to find a usable solution, but in the end decided to give up on automated testing of web mapping application. When inquired about unit testing P1 stated –"*...I tried to do that for a course project – a mapping application, but it was very limited even though we were following Agile and should have written unit tests for everything… Test Another Way for OpenLayers is a confusing choice since you will have to go back and forth to see how did you write previous test case, also it didn't seem user friendly when it comes testing geometries…"*.

2. *Manual testing and Domain specific knowledge* – As mentioned earlier that lack of specialized tool support forces developers to either abandon testing of web mapping applications or move towards manual testing. But even manual testing seems to be a challenging task due to domain knowledge that is required for applications belonging to this category. A simple example is debugging of an application where points are showing up at a different location on the map than expected. An experienced tester can easily attribute the bug to a mismatch of geographic coordinate systems [75] used by the geographical data and the base map, but a novice developer might end up spending a great amount of time on understanding the reason behind the bug.
3. *Absence of adequate documentation related to resources* – It was noted that a large number of issues related to testing that were reported on forums were related to absence of proper documentation. This issue was centered on open-source resources but on few instances this issue was encountered for proprietary tools also.

4. *Absence of reported experiences or research explorations* – Before trying anything new, in general everyone seeks details about prior experiences, the same is the case with testing of web mapping applications. The analysis of collected data indicated a need for experience reports. On multiple occasions it was mentioned that developers have no clue how to proceed to find a solution to a problem. P4 stated – *"…Googling doesn't help much at times, you end up on pages which will just point you to generic web application testing tools…I was surprised to find nothing that actually talk about testing functionality of web maps when I searched ACM…though I only searched first 4 or 5"*. P9 stated – *"…they should compile a guide for this [testing of web mapping applications]…"*. This clearly indicates a need for active research exploration about testing of web mapping applications.

3.4 Conclusion

This chapter presented a qualitative analysis of current practices and issues that are faced by developers. Apparently, testing tools and frameworks that are developed keeping in consideration common web-based applications tend to be less usable when it comes to testing of web mapping applications. This can be attributed to design issues and target audience. These tools are not designed to be specifically used for testing web mapping applications which in turn makes writing tests for even simple tasks a tricky process (comments by Participant P6 and P8). Secondly, none of these tools are developed to accommodate users who might not have a computer science background or knowledge about testing, and are looking out for easy approaches for using automated testing. This was confirmed by a comment where a participant gave up on using a testing framework after some time because it took too much time to write tests even for simple tasks.

From results of this study it can be inferred that in general testing tools and framework support for web mapping applications should be improved. A similar need was felt by the author while developing TableNOC – a web mapping application that is being developed using OpenLayers, details about this can be found in subsections 1.2 and 5.2. Tools and frameworks that are specifically developed keeping in consideration the testing of web mapping applications on a system level, integration level and unit level should be developed. But for the purpose of this thesis and due to scope and time constraints developing a generic testing tool that can fulfill all the requirements is not possible. Therefore, a decision was made to focus on unit testing by developing a specialized unit testing framework for OpenLayers. Results of this study also served as high level requirements for the unit testing framework that will be discussed in the upcoming chapter.

Chapter Four: OLUnitTest

In Chapter 3, one of the issues that were prominently mentioned by developers was lack of specialized tool support. A generic need for tools and frameworks that are especially designed to test web mapping applications was expressed. The author felt a similar need based on personal experiences while developing TableNOC. To support the argument and to improve the existing tool support, the author decided to develop a specialized unit testing framework. Requirements for tool support were drawn from the exploratory research discussed in Chapter 3 and personal experiences. The decision to focus only on unit testing was made because of generic nature of requirements and scope constraints. Therefore, to validate this idea of a domain specific testing framework and minimize the risk involved in this process, a solution to only a part of the whole problem of lack of specialized testing tools was developed.

Due to plug-in based architecture of the developed unit testing framework it can be easily migrated to test web mapping applications developed using various client-side libraries including OpenLayers, GeoExt, ArcGIS JavaScript API. A plug-in is a set of components that adds specific abilities to a larger software application [76]. However, OpenLayers was chosen as the target library for developing a unit testing framework because it is being used for the development of TableNOC and this presents an opportunity to validate the efficacy of the developed solution by using it in the practical context of testing TableNOC.

OpenLayers [2] is an open-source JavaScript mapping library that is used for developing web mapping applications. Commonly available JavaScript unit testing frameworks are not designed to account for the specialized features of OpenLayers. These features include methods and properties that are specific to maps and layers, for example properties like projection, bounding box, visibility, extent, units, scales, zoom, geometry and methods for modifying these properties

like getExtent, getVisibility, getZoomFromExtent, getArea, transform, intersects, and so on. These features are specific to maps and require some understanding of geospatial domain.

The lack of inclusion of domain specific knowledge in an assertion library becomes a major usability issue for developers and testers who use these frameworks to write unit tests for applications developed using OpenLayers. Domain specific assertions like isMapProjection, isBaseLayer, isCurrentZoom will likely reduce the number of lines of test code that are required to test functionalities, in turn reducing both time and effort required for testing. Further assertions pertaining to geometry of geographic features like isGeometry, isPolygonPresent, isCurveLength, isCurvePresent are not provided by any of the existing testing frameworks and hence test cases for verifying properties related to geometry of geographic features cannot be written easily, these assertions also help in increasing code coverage leading to increased reliability [59].

One of the example testing framework is Test Another Way [77], a JavaScript unit testing framework which is recommended by OpenLayers community for unit testing. Test Another Way suffers from following problems that can be broadly divided into two categories:

1. Issues related to lack of domain specific knowledge - Lack of assertions related to geospatial functions and properties, including map projection, bounding box, geometry and many more. The absence of assertions for testing of feature geometries was also raised by participants during interviews.
2. Issues related to testing in general – These issues were gathered based on experience of using this framework for testing web mapping applications.

a. Based on the commit history at GitHub, Test Another Way does not appear to be in active development [78]. This turns out to be a problem because the documentation has not been updated and support from the testing community is close to none.
b. Convoluted and confusing from user perspective – Based on the personal experience and experience of users in the exploratory study, the interface for displaying whether a test has passed or failed is not intuitive. A user has to choose the tests from a list, which in some browsers show up in a hidden panel creating unnecessary confusion.
c. Requires a browser to run tests, and lacks support for some browser-OS combinations, especially the new pairs like Chrome-OS X.
d. Test execution cannot be randomized – This increases the risk of bugs in our test code, especially when our tests are sequence dependent. This is also problematic where tests are written under the assumption that they are self-contained, when in reality they rely on an external factor that is congruent with the order in which the tests were run. An ideal test case should pass independent of other test cases that are present in the test suite, but sometimes test cases form a dependency on each other and when the execution is randomized they seem to fail. To avoid this dependency problem test case execution should be randomized.
e. Does not provide test suites. Instead, test files must simply co-exist in a list

These issues became the motivation for developing a new testing framework. Requirements for this testing framework were derived from both general testing and from geospatial system properties. To support testing of web mapping applications, both requirement sets should be met.

OLUnitTest – a unit testing framework for OpenLayers was developed by the author to primarily address the issue of absence of assertions related to geospatial properties and methods. OLUnitTest can be downloaded from here (https://github.com/absharma/OLUnitTest).

Some prominent features of OLUnitTest are:

1. In addition to regular assertions, OLUnitTest has a specialized set of assertions that cover the majority of geospatial functions and properties used by OpenLayers. Some of these assertions include isMapProjection, isBaseLayer, isCurrentZoom, isGeometry, isPolygonPresent, isCurveLength, isCurvePresent. These assertions provide the ability to test properties related to maps and geographic features effectively. Sample code from the assertion library can be found in Appendix 4.
2. OLUnitTest follows a plug-in [76] based architecture where assertion libraries can be added and removed as per need; this feature was included as per a requirement that was drawn from user feedback collected from the pilot study conducted in the early phase of development. Details about this pilot study can be found in Chapter 5.
3. JavaScript unit tests can be executed directly in the test framework of choice (MSTest, NUnit, xUnit, etc) without the need for a web browser.
4. Hassle free addition of test suites to Continuous Integration [79] server since OLUnitTest provides you with a freedom to choose from a variety of host frameworks.
5. OLUnitTest uses a straight forward and easy to remember assertion style that does not require unnecessary chaining of objects using the ‘dot’ operator.

At present browser-less execution of unit tests is only supported on Windows based machines; nevertheless the plug-in architecture provides the freedom to integrate the assertion libraries with

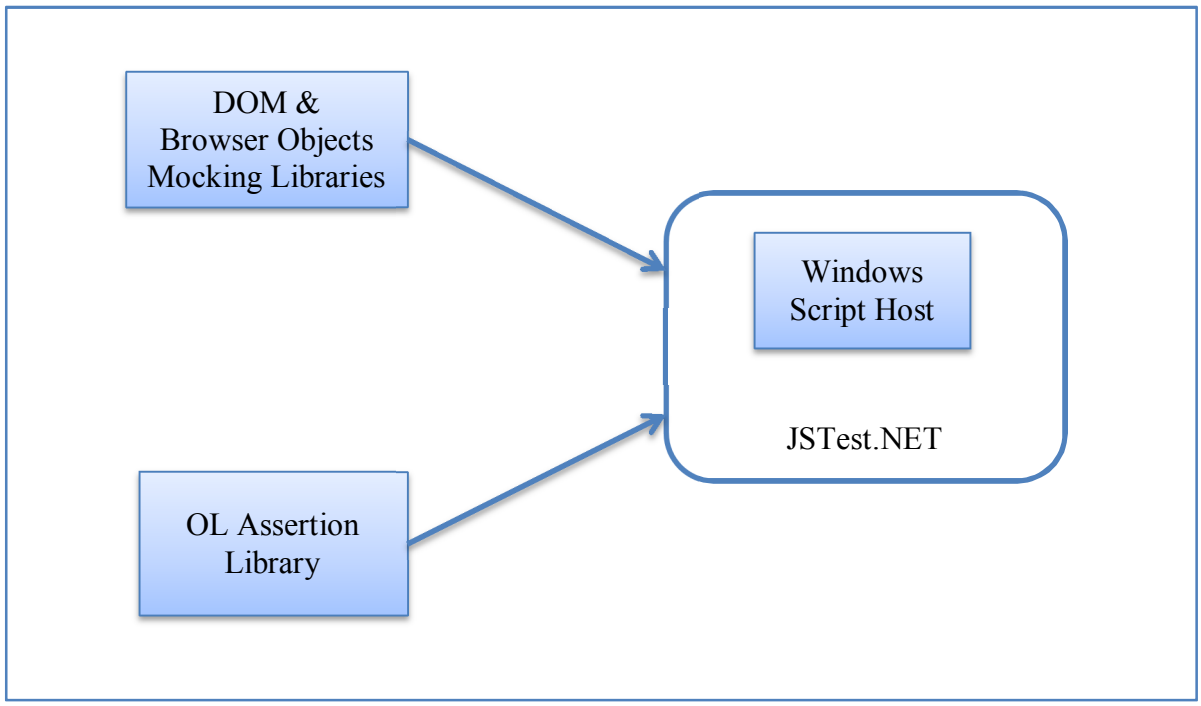

Figure 8: OLUnitTest - High-level Architecture

other unit testing frameworks that use similar assertion style for use on a variety of operating systems.

4.1 Implementation

OLUnitTest was developed while following an incremental approach where user feedback was taken and changes were made accordingly. The overall development process is comprised of three iterations, where each iteration was about 6 weeks in duration. A pilot study was conducted after the first iteration and a limited user study followed the third iteration.

OLUnitTest uses JSTest.NET [80] at its core. Figure 8 shows a high-level architecture of OLUnitTest. JSTest.NET is a light-weight managed wrapper around Windows Script Host [81]. Windows Script Host is an automation technology by Microsoft, and is language independent in the sense that it can make use of different active scripting language engines. Windows Script

Host can support and interpret plain text JavaScript by default. Since Windows Script Host is native to every machine that runs a Windows based operating system no external installations are required. This was one of the primary reasons for choosing a core that relies on Windows Script Host rather than JVM based JavaScript execution hosts

4.2 Technical Challenges during Implementation

4.2.1 Windows Script Host Bug

While developing OLUnitTest the author discovered that Windows Script Host suffers from an undocumented bug; it cannot load scripts that are greater than 512 Kilobytes. This became an issue since even the minimized version of OpenLayers is greater than 512 KB. This problem was solved by dynamically dividing the OpenLayers JavaScript file into smaller chunks and then loading those one at a time.

4.2.2 DOM and Browser Object mocking

Most of the JavaScript unit testing frameworks use browser instances to run unit tests, requiring starting a browser instance for every automated build. This becomes a problem when tests are timed and latency caused by instantiating a browser breaks the build. Therefore, Windows Script Host was chosen as a core so tests can be executed without instantiating a browser. But mocking libraries are required to replicate browser behavior since OpenLayers uses both browser objects and DOM objects. This issue was partially solved by using external mocking libraries that can be used as plug-ins where variables are instantiated before the tests are loaded. For some methods related to rendering an actual browser must be instantiated since emulating it produces erratic results.

4.3 Conclusion

This section discussed problems with existing testing frameworks using a Test Another Way as an example, motivated developing a new testing framework, and provided design details about the implemented solution. The next chapter will focus on evaluation of the developed solution in a two-step process – a limited user study and self-evaluation using a case study.

Chapter Five: Evaluation

Evaluation is always considered as the most important part of any research activity, and is used to validate the final results. Evaluation can be formally defined as - A systematic, rigorous, and meticulous application of scientific methods to assess the design, implementation, improvement or outcomes of a program [82]. The previous chapter focused on details about implementation of OLUnitTest including discussions about various design decisions. This section will present details about evaluations of the developed testing framework.

Evaluation of OLUnitTest is done by two methods: a limited user study and as a part of a longitudinal case study on testing of a web mapping application (TableNOC).

Limited user studies were conducted to evaluate whether the second research goal of this thesis, which is aimed at improving the existing tool support was accomplished or not. This was done by using objectives-based [83] evaluation methods to assess the usability of OLUnitTest. Objectives-based studies essentially involve specifying operational objectives and collecting and analyzing pertinent information to determine how well each objective was achieved.

The decision to perform the second part of evaluation using a case study was taken to assess the usability of OLUnitTest in a broader context and over a longer period of time. Further, this evaluation method is in direct alignment with the third research goal of this thesis – gaining detailed insights about testing of web mapping applications.

This chapter is broadly divided into two sub-sections: Section 5.1 presents details about the user studies, which includes a pilot study and a limited user study. Section 5.2 discusses self-evaluation of OLUnitTest that was performed by the author.

5.1 User Studies

Experiments are used to test an assumption [84]. User studies can be considered a type of controlled experiment where the usability of a tool or framework is evaluated [85]. These studies were divided into a pilot study and a limited user study. A pilot study was conducted during the development of the testing framework and a limited user study was conducted after the development was completed.

5.1.1 Pilot Study

Pilot studies are usually done before a full scale study in an attempt to avoid waste of time and resources on a inadequately designed project. A pilot study played an important role in gathering insights about the usability of the testing framework. For this evaluation, the pilot study served both as preliminary study as well as a means of gathering feature requests from the users. OLUnitTest was developed in an incremental fashion. Features were added as per requirements or suggestions from participants.

This study was conducted in early phase of development after the first iteration was complete. The primary aim behind this study was to verify if the requirements that were deduced from exploratory study discussed in Chapter 2 were in accordance with user opinion or not. The secondary aim for this study was to gather feedback about the initial implementation.

This study was conducted with 2 participants both of them belonging to academia with web mapping application development experience of 12 and 7 months, respectively. The study involved writing unit tests for a code snippet, followed by a semi-structured interview in which their feedback was collected. The code snippet instantiates a new map object and adds widgets to it, this code snippet can be found in Appendix II. The participants were asked to write unit

tests for this snippet which included testing addition of controls, layers, and centering of the map. During the study participants had access to resources and tutorials about unit testing. Both the participants successfully completed the task which was decided on the basis of whether a 50% of statement coverage is achieved or not. Participants also provided feedback on additional features that would be desired by them.

The feedback on inclusion of geospatial assertions was positive since participants reported it intuitive. After the feedback from participant P1 who expressed interest in trying the same assertion library with browser based testing frameworks like JsUnit [86], a decision on moving to plug-in based architecture where the library is used as an external component was made. The initial implementation was bundled as one entity and the assertion library could not be used separately.

5.1.2 Limited User Study

A limited user study was conducted after the implementation was finished. This study was conducted with 8 participants, out of which 6 belonged to industry and 2 were from academia. Figure 9 shows the participant experience in months. Participants P2, P4, and P6 have a geomatics background and rest are from computer science.

All the participants in this study were located remotely. Participants were approached on the basis of their involvement with the open-source web mapping community including OpenLayers, GeoServer and MapServer.

This study was similar to the pilot study in which participants were asked to write unit tests for a code snippet. This snippet for was taken from an example where a geographic feature was being dynamically resized.

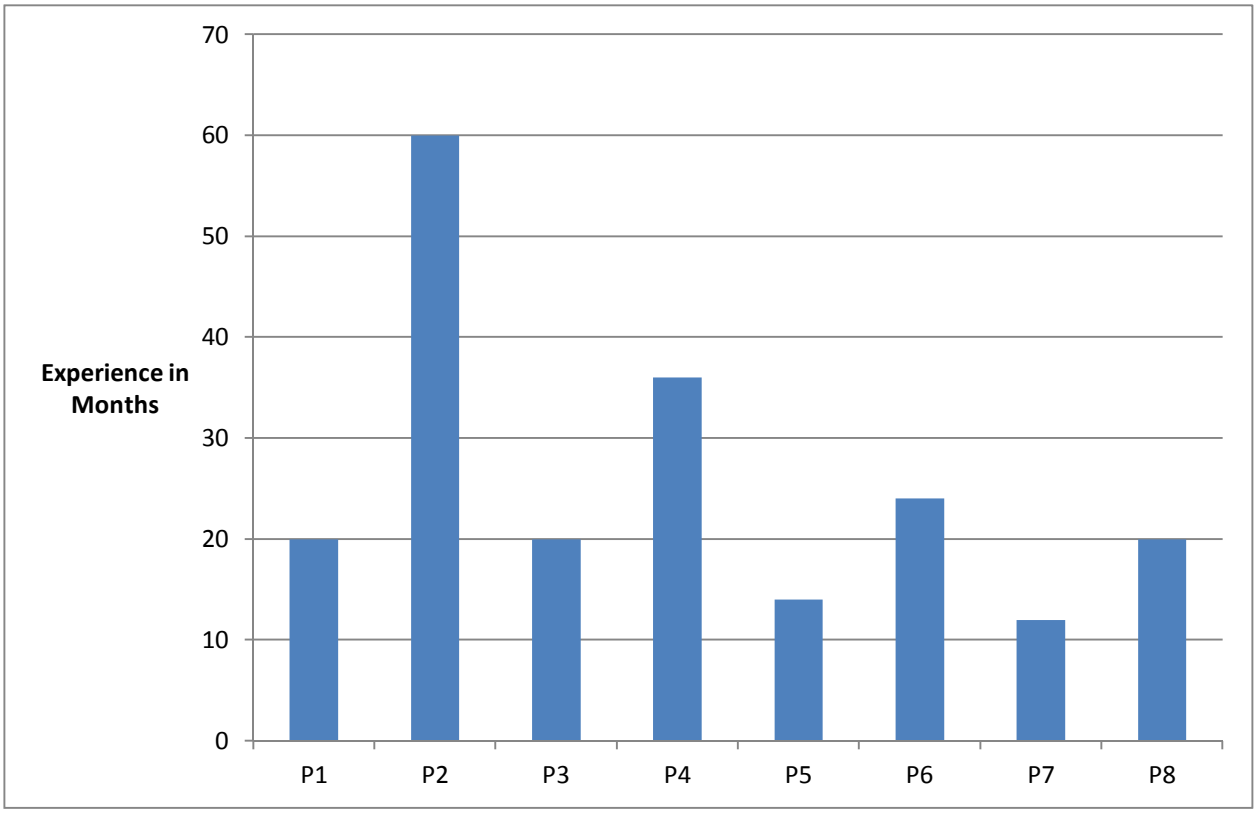

Figure 9: Participant experience in developing web mapping applications –Evaluation study

This example was specifically chosen since it was using geometry and vector related properties and methods. This was done to make sure that geospatial assertions related to geometry and vectors that are absent in other testing frameworks can be used. The code snippet can be found in Appendix II.

The data collected from the participants was analyzed to calculate the statement coverage metric. For the purpose of this evaluation the statement coverage is defined in terms of execution of the line (or statement). If a statement is executed at least once by a test it is considered to be covered by the test case. For example participant P1's unit tests executed 21 statements out of a total of 29, and hence the statement coverage is 72%. It was decided that statement coverage metric [57] below 50% will be considered as an unsuccessful attempt in completion of the task, and accordingly 6 participants completed the study. The results are shown in Figure 10.

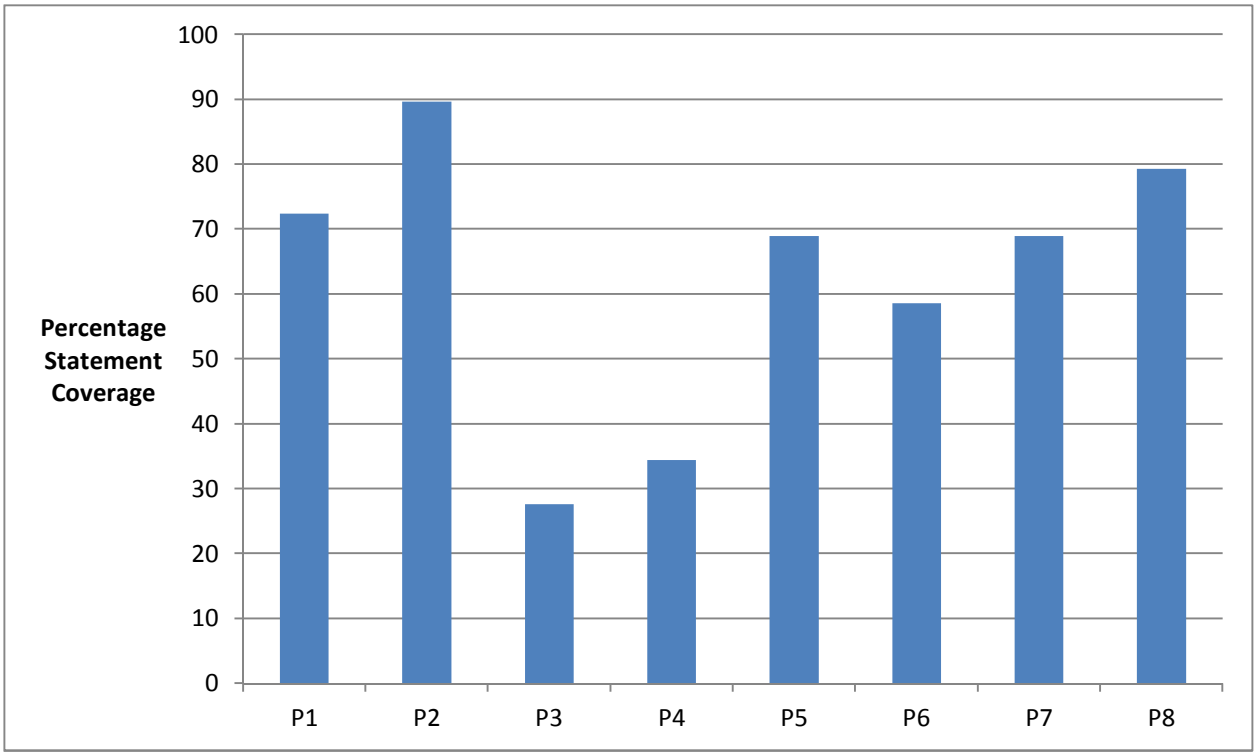

Figure 10: Results of the user study - Percentage Statement Coverage

After the studies participants were asked to fill out a questionnaire where they rated the framework on a 5 point Likert scale [87] shown in Table 1. This questionnaire can be found in Appendix III. For the purpose of this thesis it is assumed that ratings are separated by an interval level [88], where each rating is at an interval of 1 from the next. This was done for the ease of plotting of the data on a chart and for statistical analysis. The results of this study are shown in Figure 11.

Rating	Value
very good	5
good	4
fair	3
poor	2
very poor	1

Table 1: Likert scale used for rating OLUnitTest

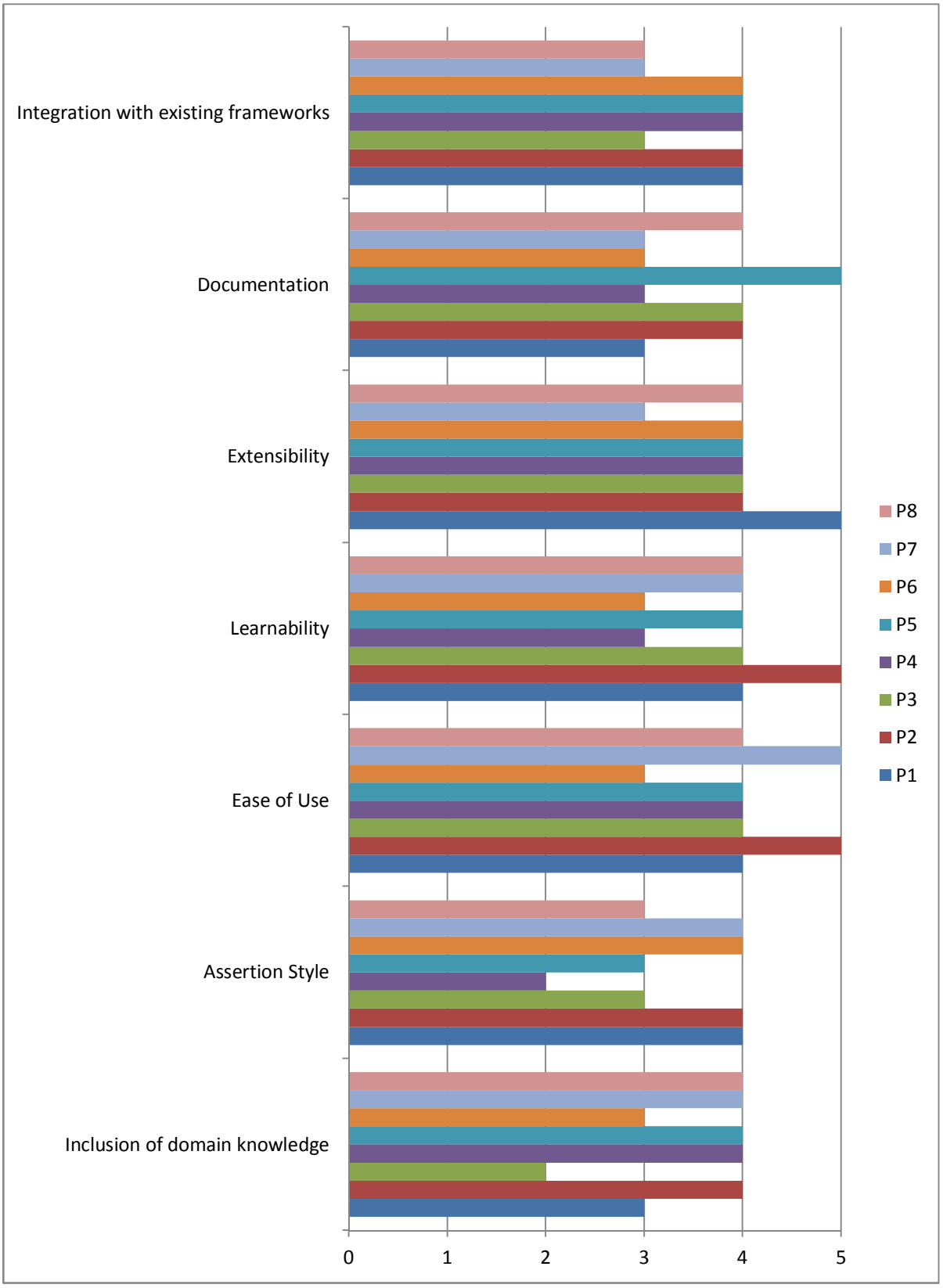

Figure 11: Results from the user study -

Participants of this user study were asked to rate OLUnitTest on various criteria. These criteria were chosen to accommodate evaluation of fulfillment of requirements and general usability metrics. The testing framework was evaluated on the following criteria:

1. Inclusion of domain knowledge – One of the goals for OLUnitTest was that it should abstract the domain specific knowledge related to OpenLayers and provide developers with extra support for testing methods and properties that are specific to mapping applications. On the basis of the modal value of ratings of OLUnitTest on inclusion of domain specific knowledge it can be claimed that this goal is accomplished.
2. Assertion Style – One of the important features of any testing framework relates to how assertions are written. As mentioned by a participant – *"...I [have] tried something called Chaining Assertion and now I'm pretty sure that I do not like chaining. Assertions should always follow simple styling...like you have used for OLUnitTest"*. On the basic of the modal value of ratings of OLUnitTest on this criterion it can be claimed that OLUnitTest follows a good assertion style and users are satisfied by it. These modal values are shown in Figure 12.
3. Ease of use – Ease of use is a broader criterion that is based on multiple factors including time taken to accomplish a task and user satisfaction. From the user ratings it can be seen that my framework 7 out of 8 participants rated this as good or higher on the scale.
4. Learnability – This criterion was introduced to get a rough estimate of how easily a new user can get acquainted with OLUnitTest. Learnability was one of the central design criteria during the pilot studies and a similar positive reaction was recorded during this user study. A comment from a participant verifies this, *"...if you know how to use*

OpenLayers then using this testing framework is a piece of cake…I might have looked at documentation only twice or thrice…".

5. Extensibility – OLUnitTest is designed keeping in mind extensibility, since it follows plug-in based architecture where user can modify and replace libraries as and when required. No specific task was included to specifically assess OLUnitTest on the basis of this criterion, but participants were informed about how to extend the libraries by adding new assertions. One of the comments provided by a participant [P1] – *"...I looked into your library and it seems you are missing some assertions, but I was excited when I found out that adding new assertions was so easy. This will be quite handy when new version of OpenLayers will be released…".* Based on the ratings and this comment it can be inferred that participants in this study seemed to be quite excited about the possibility of extending this as per need.
6. Documentation – Documentation is important part of any software system, it helps users to get started with the product. Even though participants suggested a few improvements in the testing framework documentation, the overall response from the participants was positive.
7. Integration with existing frameworks – OLUnitTest seamlessly integrates with existing testing frameworks including NUnit, xUnit and MSTest. This claim can be bolstered by a participant's comment – *"...I can use this from NUnit, this was the first thing I would have asked if it was not already included…".*

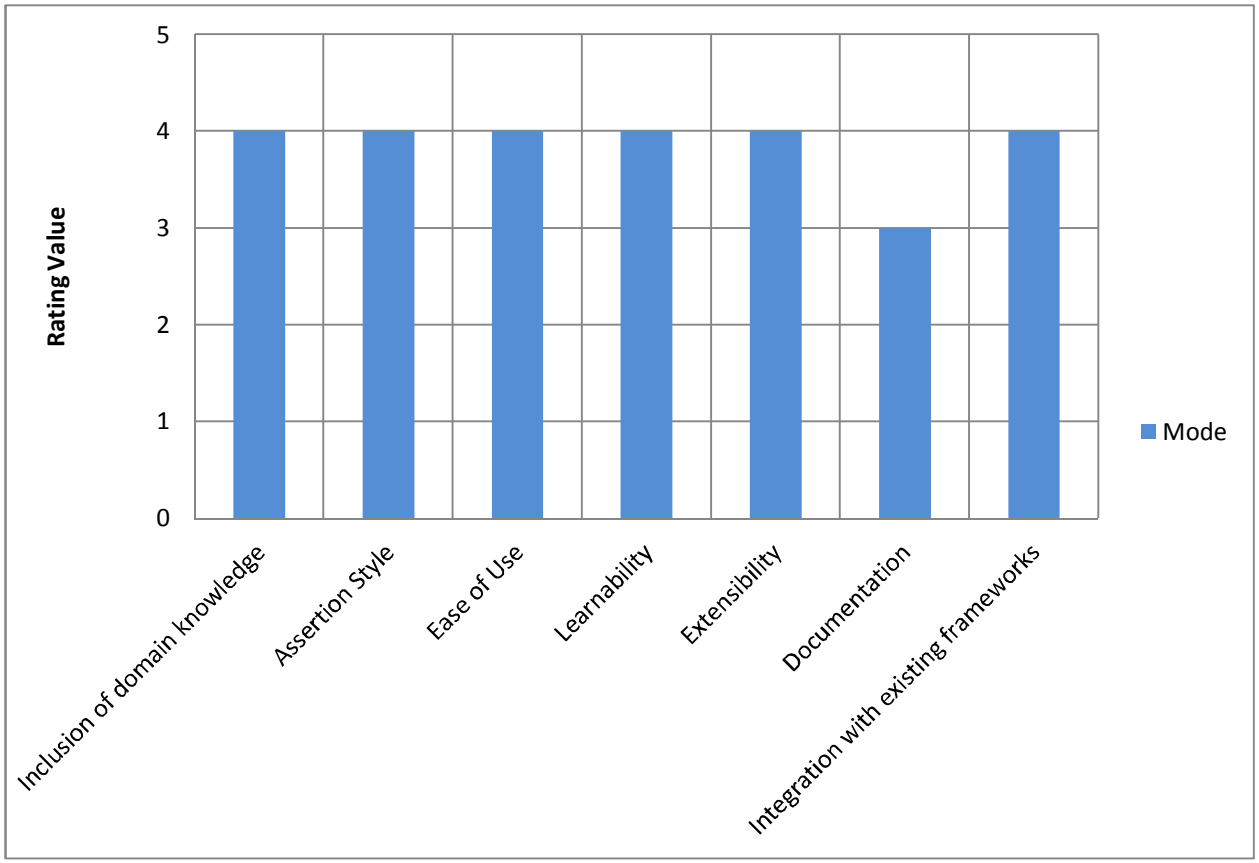

Figure 12 : Modal values of participant ratings – Evaluation Study

On the basis of percentage statement coverage shown in Figure 10 and modal value of participant ratings on different criteria, shown in Figure 12 it can be claimed that OLUnitTest can provide effective support for unit testing of OpenLayers based web mapping applications.

5.2 Self-evaluation of OLUnitTest with TableNOC

The second stage of the evaluation was conducted as a part of a case study on testing of a web mapping application named TableNOC. This was a self-evaluation which primarily focused on gathering qualitative [89] insights about OLUnitTest in the context of a larger project. Details about these insights can be found under results.

This part of evaluation was done as a case study on testing of TableNOC. Unit testing is an important step in testing any software system. A self-assessment study of OLUnitTest by using it for writing unit tests for TableNOC was conducted and notes were taken based on the experience. A comparative study was of OLUnitTest, Test Another Way [77] and JsUnit [86] was performed by the author. The design of this study is inspired from recommendations about qualitative comparative analysis given by Ragin [90]. Qualitative comparative analysis [90] is used to solve problems where the sample size is too small to use a statistical analysis technique. In this comparative study all three frameworks were used to write unit tests and notes were taken during the process.

5.2.1 Results

The tasks for the comparative study required writing unit tests for code snippets that were chosen from TableNOC. The results obtained from the studies were analyzed on the basis of two criteria –statement coverage achieved using different testing frameworks and time taken to finish the tasks. The results are shown in Figure 13 and Figure 14.

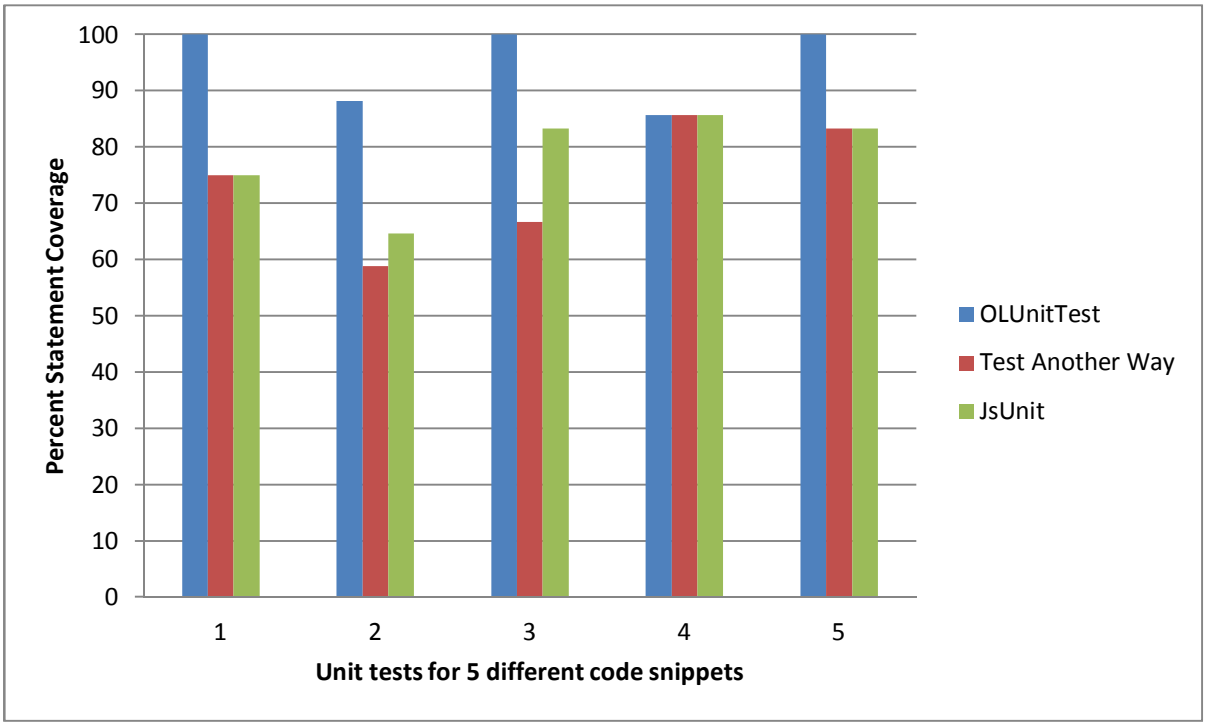

Figure 13: Percentage statement coverage of in various tasks

The findings on the basis of the comparative study and testing of TableNOC are:

1. It was found that OLUnitTest ensures highest statement coverage when used for unit testing applications that are developed using OpenLayers; this was also confirmed from the results of the comparative study. Higher statement coverage can be attributed to geospatial assertions that are being used in OLUnitTest and are absent in the other two frameworks. This can be seen in Figure 13.
2. Time is an important variable in assessing the effectiveness of a tool or framework. It was observed that on an average time taken to write unit tests was lowest with OLUnitTest and highest with Test Another Way. JsUnit and OLUnitTest were close terms of time measure; this might be because of xUnit style used by JsUnit. Results are shown in Figure 14.

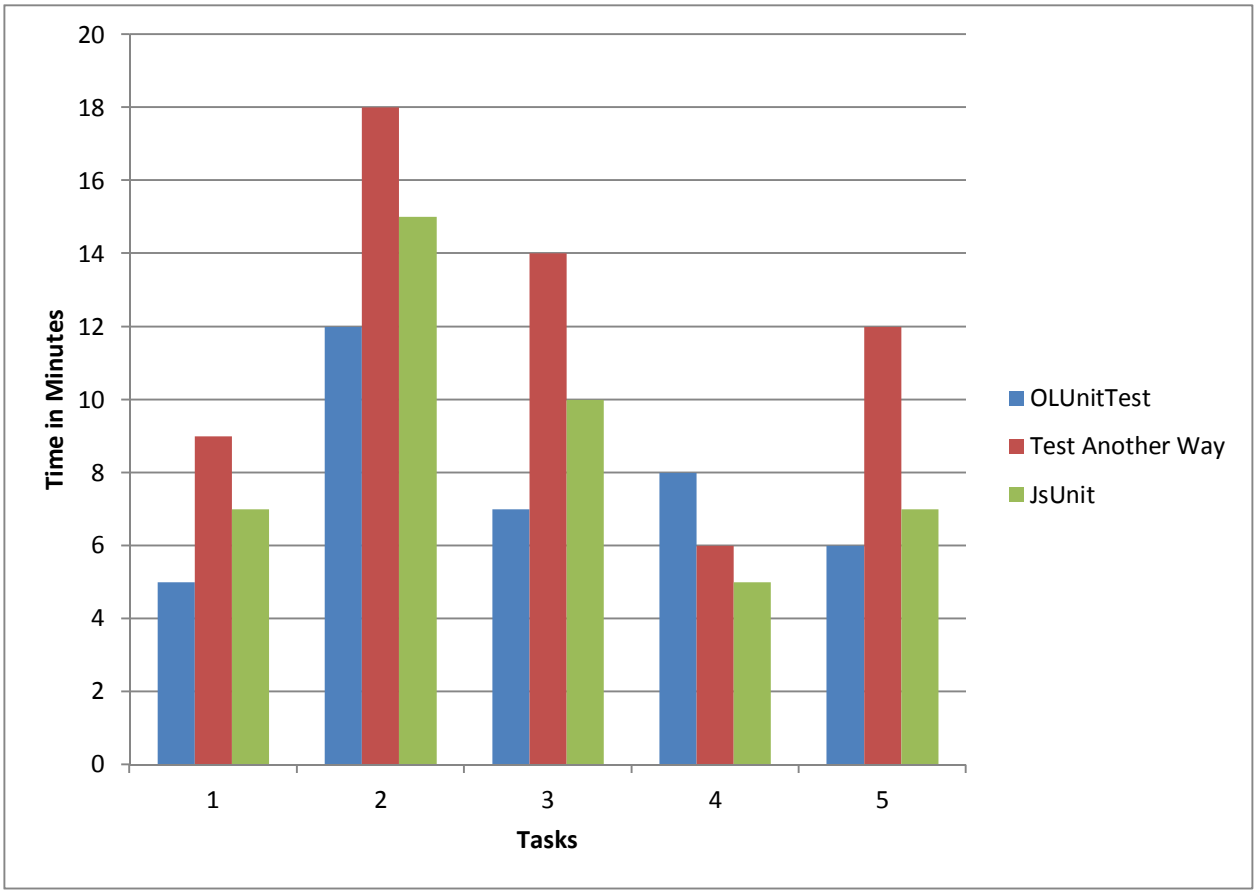

Figure 14: Time required for completing the tasks

3. For the purpose of this thesis, efficiency describes the extent to which time or effort is well used for the intended task or purpose. Efficiency was another noticeable difference that was observed between the three testing frameworks, with OLUnitTest being the most effective among the three. With unit tests written using OLUnitTest highest code coverage was achieved in the least amount of time required to write the test.
4. Based on personal experience of using OLUnitTest it was found that assertions are intuitive and can be easily related to geospatial properties.

5.3 Threats to the Validity of Evaluation

Evaluations of OLUnitTest might suffer from following threats to validity:

1. The statistical validity of this evaluation is threatened by small number of participants for the user studies.
2. Likert scale was assumed to be interval level for the ease of analysis; this might be different from perception of participants [91].
3. Selection bias while selecting participants as well as code snippets might affect the internal validity of the evaluation since the decision on selection was subjective.
4. Participants were recruited on the basis of their experience with OpenLayers and testing in general, which might be different from expertise in writing unit tests. This might have affected the results.
5. The validity of results from the self-evaluation is threatened by the fact that author is familiar with the test framework under evaluation and is the lead developer for both the system under test and the testing framework that is being evaluated. This might have reduced the time taken to write unit tests and created a positive bias.

5.4 Conclusion

In this section, evaluation of OLUnitTest was undertaken and the implications of the results were discussed. The evaluation was divided into two parts – user study and a qualitative case study. The results of the study indicated that OLUnitTest accomplishes its goal of inclusion of domain specific knowledge within a testing framework and fits easily when used in conjunction with existing frameworks like MSTest, xUnit and NUnit. The second part of the evaluation focused on using OLUnitTest to test a system that is under development. The author also conducted a comparative study to contrast OLUnitTest against two other frameworks.

On the basis of the results of this limited evaluation it can be inferred that OLUnitTest has an edge over existing frameworks when unit testing web mapping applications that use OpenLayers. This can be considered as an improvement in the existing tool support for testing of web mapping applications, which is also the second research goal of this thesis.

The next chapter brings in light the experience gained during development and testing of TableNOC and provides a set of recommendations for efficient testing of web mapping applications.

Chapter Six: Development and Testing of TableNOC – Lessons Learned

This chapter will discuss the experience gained by the author during development and testing of TableNOC, especially from a tester's perspective. Although experiences gained by the author are distributed in this thesis and are used in making various decisions related to implementation and evaluation, this chapter aims at discussing a focused set of experiences based on testing of TableNOC in general. Chapter 1 provided a contextual background about TableNOC – the web mapping application that was developed by the author and colleagues.

Based on self-experiences, the following recommendations can be provided so that testing of web mapping application can be performed easily and effectively.

- Use of testing frameworks that encapsulate domain specific knowledge related to spatial mapping should be preferred over generalized unit testing frameworks. Testing of web mapping applications draws requirements from both geospatial domain and testing in general. Assertions related to geographic feature properties reduce the time required for unit testing and increase the effectiveness of the test suite by increasing code coverage, which in turn increases the chances of bug detection.
- Unit tests are helpful both in catching bugs as well as flattening the learning curve if the developer has no or little experience in developing web mapping applications. It was found that unit testing helps in improving the understanding about the functionalities of the system. Further, unit tests using assertions that are related to domain specific properties were found to be good examples on the usage of those properties and related methods.
- System level testing techniques like GUI testing whether automated or exploratory should be used to ensure that the system is working as per expectations when considered

the whole. Unit testing does not ensure the behavior of the system as a whole and it was found that web mapping applications due to presence of a large number of points of failure should be tested for correct behavior on the system level.

- Manual exploratory testing when performed by a tester having knowledge about the geospatial domain takes lesser amount of time when compared to a tester with no domain knowledge, is required for bug detection and for finding the cause of the bug. Based on exploratory test sessions for testing of TableNOC it was found that quick uncovering the real cause of the bug usually requires some understanding of the domain knowledge.
- Automated GUI tests can save time and effort but suffer from fragility due to higher dependency on third party services for map layers including base maps. A web mapping application is usually comprised of maps that are being used as a service from third parties like Google, Bing, and Esri. Automated GUI tests tend to fail when the third party has decided to update the service for example by adding of new geographic features on a Google Map which is being used as a base map for an application will break the existing GUI tests due to changes in the background.

Chapter Seven: Conclusions

This thesis presents an innovative approach for testing of web mapping applications. The focus of the thesis is on unit testing to verify the correctness at the unit level. Most of the research done towards this thesis is exploratory in nature and provides preliminary insights about testing of web mapping applications. Chapter 1 described the motivation behind this research and provided the context for the research by providing details about TableNOC. In Chapter 2 an overview of web mapping applications and testing was presented to provide the background for better understanding of challenges of this field. In Chapter 3, an up-to-date picture of current practices and issues regarding testing of web mapping applications was presented using results of a qualitative study. Based on requirements from the exploratory a unit testing framework - OLUnitTest for testing of applications that use OpenLayers was developed which is discussed in Chapter 4. This was done to improve framework support for unit testing of web mapping applications. User studies and self-evaluations produced positive results about usability of OLUnitTest, details are presented in Chapter 5. In Chapter 6 author shared his experiences that were gained during testing of a web mapping application.

7.1 Thesis Contributions

The first contribution of this thesis is providing an overview of current issues and practices in testing of web mapping applications. This was done by first conducting a literature review to search for relevant research articles and then extend into an exploratory qualitative study. For the qualitative study, data was collected from online forums and interviews with practitioners. This helped in understanding what problems developers face and what practices related to testing of web mapping applications are followed. Some of the issues include lack of specialized tool support, poor documentation, and lack of recorded experiences related to testing of web mapping

applications. Results also indicated disregard for the importance of testing, details can be found in Chapter 3. This also answered our first research question aimed at understanding the current state of research in testing of web mapping applications.

The second research question focused on how the existing testing frameworks for unit testing of web mapping applications can be improved. In alignment with this, OLUnitTest was developed, which is also the second contribution of this thesis. OLUnitTest is developed to address the lack of specialized tool support that encapsulates domain specific knowledge, one of the major issues that were raised by developers. OLUnitTest was developed based on requirements gathered from the exploratory study. Structure and features of OLUnitTest are discussed in detail in Chapter 4. OLUnitTest was evaluated by user experiments and self-evaluation by using it for testing TableNOC. Evaluations were based on factors including ease of use, documentation, assertion style, learnability and extensibility. Details about evaluation are discussed in Chapter 5. OLUnitTest also fulfills the second research goal of this thesis – extending tool support for unit testing of web mapping applications. OLUnitTest is developed to emphasize the importance of domain specific tools for applications belonging to specialized categories.

Third contribution of this thesis are experiences based on testing of TableNOC. Experiences from development and testing of TableNOC formed the basis of requirements, evaluations and are also discussed in Chapter 6.

Based on the limited number of results from systematic searches that were conducted for finding related work it can be inferred that this thesis is one of the earliest documented works in the field of testing of web mapping applications and this justifies the exploratory nature of this thesis. The

final contribution of this thesis is that it provides a foundation for future research activities in this field.

7.2 Future Work

Testing of web mapping applications is still a new field from a research perspective and there are three directions in which it can be extended. These directions could not be explored within the scope of this thesis, so these are proposed as future work.

The first direction this research can be extended is with extensive exploratory research involving interview studies. In this thesis, studies were done with a lower number of participants due to scope limitations, but future studies should aim at providing statistically significant results with a higher number of participants. Researchers could also look into employing action research techniques to observe and record the practices followed by developers. This should be done by closely working with industrial teams that develop web mapping applications.

Results from exploratory studies should be used to propose requirements for tools, which is the second direction for future research. The next enhancement to OLUnitTest is to improve the present assertion library for OpenLayers [2] and develop assertion libraries for other client side mapping libraries like GeoExt [35], Leaflet [92] and Esri client side APIs.

Third, more case studies on development and testing of web mapping applications should be conducted. On one hand this will improve the understanding about the use and development of applications belonging to this category and secondly, validate the experiences from this thesis.

References

[1] Charles Picquet, *Rapport sur la marche et les effets du choléra-morbus dans Paris et les communes rurales du département de la Seine, par la commission nommée*. Paris, Frankce: Imprimerie Royale, 1834.

[2] OpenLayers. [Online]. http://openlayers.org/ (last accessed July 2012)

[3] MapServer. [Online]. http://mapserver.org/ (last accessed July 2012)

[4] Earl Babbie, *The Practice of Social Research*, 5th ed. Belmont, CA: Wadsworth, 1989.

[5] Robert A. Stebbins, "Exploratory Research in the Social Sciences," in *Paper Series on Qualitative Research Methods*, Robert A. Stebbins, Ed.: Sage University, 2001, ch. Volume 48.

[6] Shashi Shekhar and Hui Xiong, *Encyclopedia of GIS*.: Springer, 2007.

[7] Pinde Fu and Jiulin Sun, *Web GIS: Principles and Applications*. Redlands, CA, USA: ESRI Press, 2010.

[8] Kraak, Menno Jan, and A. Brown, *Web cartography - developments and prospects*. London and New York: Taylor and Francis, 2001.

[9] OpenGeo. GeoServer. [Online]. http://geoserver.org/display/GEOS/Welcome (last accessed July 2012)

[10] Deane Kensok and Andrea Rosso. (2006) ESRI. [Online]. http://proceedings.esri.com/library/userconf/devsummit06/papers/soa_casestudy.pdf (last accessed July 2012)

[11] K. Sahin and M. U. Gumusay, "Service Oriented Architecture (SOA) based Web Services for Geographic Information Systems," *ISPRS - International Archives of the*

Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXVII, no. B2, pp. 625-630, 2008.

[12] OGC. OpenGIS Web Map Service (WMS) Implementation Specification. [Online]. http://www.opengeospatial.org/standards/wms (last accessed July 2012)

[13] OGC. OpenGIS Web Feature Service (WFS) Implementation Specification. [Online]. http://www.opengeospatial.org/standards/wfs (last accessed July 2012)

[14] U.S. Department of the Interior. USGS. [Online]. http://webgis.wr.usgs.gov/globalgis/tutorials/overview/geographic.htm (last accessed July 2012)

[15] OpenGIS. OGC® WCS 2.0 Interface Standard - Core. [Online]. http://www.opengeospatial.org/standards/wcs/ (last accessed July 2012)

[16] ISO. (2004) ISO 19125-2:2004. [Online]. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40115 (last accessed July 2012)

[17] Esri. (1998, July) ESRI Shapefile Technical Description. Electronic. [Online]. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf (last accessed July 2012)

[18] Esri. ArcGIS - Mapping and Spatial Analysis for Understanding Our World. [Online]. http://www.esri.com/software/arcgis (last accessed July 2012)

[19] OSGeo. deegree. [Online]. http://www.deegree.org/ (last accessed July 2012)

[20] Restlet. Restlet. [Online]. http://www.restlet.org/ (last accessed July 2012)

[21] Mort Bay Consulting. Jetty. [Online]. http://jetty.codehaus.org/jetty/ (last accessed July 2012)

[22] University of Minnesota. MapScript - MapServer. [Online]. http://mapserver.org/mapscript/index.html (last accessed July 2012)

[23] Melita Kennedy and Steve Kopp, *Understanding map projections: GIS by ESRI*.: ESRI, 2000.

[24] Open Geospatial Consortium. OGC Standards. [Online]. http://www.opengeospatial.org/standards (last accessed July 2012)

[25] OGC. OGC Catalog Service for the web. [Online]. http://www.opengeospatial.org/standards/cat (last accessed July 2012)

[26] OGC. Geography Markup Language. [Online]. http://www.opengeospatial.org/standards/gml/ (last accessed July 2012)

[27] OGC. KeyHole Markup Language. [Online]. http://www.opengeospatial.org/standards/kml/ (last accessed July 2012)

[28] Google Inc. Google Earth. [Online]. http://www.google.com/earth/index.html (last accessed July 2012)

[29] OGC. GeoXACML Standard. [Online]. http://www.opengeospatial.org/standards/geoxacml (last accessed July 2012)

[30] Oasis. XACML - Cover Pages. [Online]. http://xml.coverpages.org/xacml.html (last accessed July 2012)

[31] OGC. Styled Layer Descritor - OpenGIS Standard. [Online]. http://www.opengeospatial.org/standards/sld/ (last accessed July 2012)

[32] Google. Google Maps. [Online]. http://maps.google.com (last accessed July 2012)

[33] Microsoft. Bing Maps. [Online]. http://www.bing.com/maps/ (last accessed July 2012)

[34] OpenScales. [Online]. http://openscales.org/ (last accessed July 2012)

[35] GeoExt. [Online]. http://geoext.org/ (last accessed July 2012)

[36] ExtJS- Sencha. [Online]. http://www.sencha.com/products/extjs (last accessed July 2012)

[37] Google Inc. Google Maps API. [Online]. https://developers.google.com/maps/ (last accessed July 2012)

[38] Microsoft. Bing Maps - Developer Resources. [Online]. https://www.microsoft.com/maps/developers/web.aspx (last accessed July 2012)

[39] Esri. ArcGIS Web Mapping. [Online]. http://www.esri.com/software/arcgis/web-mapping (last accessed July 2012)

[40] Esri. ArcGIS Server. [Online]. http://www.esri.com/software/arcgis/arcgisserver (last accessed July 2012)

[41] Jiantao Pan. (1999) Software Testing. [Online]. http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/ (last accessed July 2012)

[42] Mitch Allen, "Bug Tracking Basics: A beginner's guide to reporting and tracking defects," *The Software Testing & Quality Engineering Magazine*, vol. 4, no. 3, pp. 20-24, May/Jun 2002.

[43] William Hetzel, *The Complete Guide to Software testing*.: Wellesley, 1988.

[44] Lee Copeland, *A Practitioner's Guide to Software Test Design*.: Artech House, 2004.

[45] [Online]. http://www.cs.tau.ac.il/~nachumd/verify/horror.html (last accessed July 2012)

[46] P. Naur and B. Randell, "Software Engineering: Report on a Conference Sponsored by the NATO Science Committee," NATO Scientific Affairs Division, Brusells, 1968.

[47] B.W. Boehm, "Software and its Impact: A Quantitative Assessment," Datamation, 1973.

[48] Ron Patton, *Software Testing*, 2nd ed.: Sams, 2005.

[49] IEEE. Guide to the Software Engineering Body of Knowledge. [Online]. http://www.computer.org/portal/web/swebok/html/ch5 (last accessed July 2012)

[50] Emil Borjesson and Robert Feldt, "Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry," in *IEEE Fifth International Conference on Software Testing, Verification and Validation*, Montreal, Quebec Canada, 2012, pp. 350-359.

[51] Theodore D. Hellmann and Frank Maurer, "Rule-Based Exploratory Testing of Graphical User Interfaces," in *Agile Methods in Software Development (Agile 2011)*, Salt Lake City, Utah, USA, 2011, pp. 107-116.

[52] James Bach. (2003, April) Exploratry Testing Explained. [Online]. http://www.satisfice.com/articles/et-article.pdf (last accessed July 2012)

[53] A.M. Memon, A Comprehensive Framework for Testing Graphical, 2001, PhD Thesis.

[54] Dorota Huizinga and Adam Kolawa, *Automated Defect Prevention: Best Practices in Software Management*.: John Wiley & Sons, 2007.

[55] Seyed Mehdi Nasehi and Frank Maurer, "Unit Tests as API Usage Examples," in *International Conference on Software Maintenance* , Timisoara, 2010.

[56] Microsoft. Integration Testing - MSDN. [Online]. http://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx (last accessed July 2012)

[57] Srinisvasan Desikan and Gopalaswamy Ramesh, *Software Testing*.: Pearson Education India, 2009.

[58] Glenford J. Myers, *The Art of Software Testing*, 2nd ed.: John Wiley & Sons, 2004.

[59] F. Frate, P. Garg, A.P. Mathur, and A. Pasquini, "On the correlation between code coverage and software reliability," in *International Symposium of Software Reliability Engineering*, 1995, pp. 124-132.

[60] Grady Booch. (2001, June) The Architecture of Web Applications. Document. [Online]. http://www.ibm.com/developerworks/ibm/library/it-booch_web/ (last accessed July 2012)

[61] Ye Wu, Jeff Offutt, and Xiaochen Du, "Modeling and Testing Web-based Applications," Department of Computer Science, George Mason University, Technical Report 2004.

[62] Nathan Hamiel, Gregory Fleischer, Seth Law, and Justin Engler. Challenges in the Automated Testing of Modern Web Applications. Document. [Online]. http://media.blackhat.com/bh-us-11/Hamiel/BH_US_11_Hamiel_Smartfuzzing_Web_WP.pdf (last accessed July 2012)

[63] Filippo Ricca and Paolo Tonella, "Analysis and testing of Web applications," in *International Conference on Software Engineering*, Toronto, 2001, pp. 25-34.

[64] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson, "Systematic Mapping Studies in Software Engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering*, 2008, pp. 71-80.

[65] IEEE Xplore. [Online]. http://ieeexplore.ieee.org/Xplore/guesthome.jsp (last accessed July 2012)

[66] Scopus. [Online]. http://www.scopus.com/home.url (last accessed July 2012)

[67] Yaolin Liu and Xinming Tang, "Exploratory Research On GIS Software Testing," in *International Symposium on Spatial Analysis, Spatial-Temporal Data Modeling, and Data*

Mining, Wuhan, China, 2009.

[68] Hu Maogui and Wang Jinfeng, "Application of Automated Testing Tool in GIS Modeling," in *World Congress on Software Engineering*, 2009.

[69] Jiri Horak, Jiri Ardielli, and Bronislava Horakova, "Testing of Web Map Services," *International Journal of Spatial Data Infrastructures Research*, 2009.

[70] Andreq Stellman and Jennifer Greene, *Applied Software Project Management*.: O'Reilly Media, 2005.

[71] OSGeo. [Online]. http://osgeo-org.1560.n6.nabble.com/ (last accessed July 2012)

[72] Google Inc. Google Blog Search. [Online]. http://www.google.com/blogsearch (last accessed July 2012)

[73] Barney G. Glaser and Anselm L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*.: Transaction Publishers, 1967.

[74] OpenQA. Selenium. [Online]. http://seleniumhq.org/ (last accessed July 2012)

[75] Tor Bernhardsen, *Geographic Information Systems: An Introduction*.: John Wiley and Sons, 2002, vol. 3.

[76] Vladimir Silva, *Practical Eclipse Rich Client Platform Projects*.: Springer, 2009.

[77] Artem Khodush. Test Another Way. [Online]. http://openjsan.org/doc/a/ar/artemkhodush/Test/AnotherWay/0.52/lib/Test/AnotherWay.html (last accessed July 2012)

[78] geekq. Test Another Way - GitHub. [Online]. https://github.com/geekq/Test.AnotherWay/commits/master (last accessed July 2012)

[79] Paul M. Duvall, Steve Matyas, and Andrew Glover, *Continuous Integration: Improving*

Software Quality and Reducing Risk.: Peason Education, September 2007. [Online]. http://www.martinfowler.com/articles/continuousIntegration.html (last accessed July 2012)

[80] CBaxter. (2012, January) JSTest.NET. [Online]. http://jstest.codeplex.com/ (last accessed July 2012)

[81] Microsoft. Windows Script Host Overview. [Online]. http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/wsh_overview.mspx?mfr=true (last accessed July 2012)

[82] Peter Henry Rossi, Mark W. Lipsey, and Howard E. Freeman, *Evaluation: A Systematic Approach*, 7th ed.: SAGE, 2004.

[83] Daniel L. Stufflebeam and Anthony J. Shinkfield, *Evaluation Theory, Models, and Applications*.: John Wiley & Sons, 2007.

[84] W. Thomas Griffith, *The Physics of Everyday Phenomena: A Conceptual Introduction to Physics*. New York: McGraw-Hill Higher Education, 2001.

[85] Jakob Nielsen, *Usability Engineering*, 2nd ed.: Morgan Kaufmann, 1994.

[86] JsUnit. [Online]. http://www.jsunit.net/ (last accessed July 2012)

[87] R. Likert, "A technique for the measurement of attitudes," *Archives of Psychology*, vol. 22, no. 140, pp. 1-55, 1932.

[88] S.S. Stevens, "On the Theory of Scales of Meaurement," *Science*, vol. 103, no. 2684, pp. 677-680, June 1946.

[89] James C. McDavid, Irene Huse, and Laura R.L. Hawthorn, "Applying Qualitative Evaluation Methods," in *Program Evaluation and Performance Measurement - An Introduction to Practice*. Victoria, Canada: Sage Publications, 2006, ch. 5, pp. 165-200.

[90] Charles C. Ragin, *The Comparative Method - Moving Beyond Qualitative and Quantitative Strategies*. Berkely, Los Angeles, London: University of California Press, 1987.

[91] Geoff Norman, "Likert scales, levels of measurement and the “laws” of statistics," *Advances in Health Science Education*, vol. 15, no. 5, pp. 625-632, 2010.

[92] CloudMade. Leaflet. [Online]. http://leaflet.cloudmade.com/ (last accessed July 2012)

[93] SEWilco. (2007, December) GeoServer GeoNetwork with web app. [Online]. http://en.wikipedia.org/wiki/File:GeoServer_GeoNetwork_with_web_app.png (last accessed July 2012)

[94] John Gerring, "What is a Case Study and What is it Good for?," *American Political Science Review*, vol. 98, no. 2, pp. 341-354, May 2004.

[95] Giacomo Gastaldi, Map of the World, 1548.

[96] GIS Lounge - Glossary. [Online]. http://gislounge.com/geospatial-glossary-a-through-g/ (last accessed July 2012)

Appendix I: Exploratory Study Interview Questions

The interviews of the exploratory were semi-structured in nature and were started with following questions; the subsequent questions were decided on the basis of answers to these questions.

- Can you tell me about your background in terms of education and experience?
- How much experience do you have in developing web mapping applications?
- Can you tell me details about the projects you worked on?
- How did you ensure the reliability of these projects?
    - Was testing done for these projects?
    - What kind of testing was done?
    - Which tools were used?

Appendix II: Code snippets used in user studies

1. Code Snippet used in pilot study – Hyperlinks to mapping servers are anonymized for privacy reasons.

```
var map;
function init(){
   map = new OpenLayers.Map('map', {
      controls: [
         new OpenLayers.Control.Navigation(),
         new OpenLayers.Control.PanZoomBar(),
         new OpenLayers.Control.LayerSwitcher({'ascending':false}),
         new OpenLayers.Control.Permalink(),
         new OpenLayers.Control.ScaleLine(),
         new OpenLayers.Control.Permalink('permalink'),
         new OpenLayers.Control.MousePosition(),
         new OpenLayers.Control.OverviewMap(),
         new OpenLayers.Control.KeyboardDefaults()
      ],
      numZoomLevels: 6

   });

   var ol_wms = new OpenLayers.Layer.WMS(
      "WMS",
      "http://xyz",
      {layers: 'basic'}
   );

   var gwm = new OpenLayers.Layer.WMS(
      "GI",
      "http://abc ",
      {layers: "bluemarble"},
      {tileOrigin: new OpenLayers.LonLat(-180, -90)}
   );

   map.addLayers([ol_wms, gwm,]);

   if (!map.getCenter()) {
      map.zoomToMaxExtent();
   }
}
```

2. Code snippet used in limited user study

```
var map, vectorLayer, pointFeature, lineFeature, polygonFeature;

function init(){
   map = new OpenLayers.Map('map');
   var layer = new OpenLayers.Layer.WMS( "WMS",
         "http://xyz", {layers: 'basic'} );
   map.addLayer(layer);

   var style_blue = OpenLayers.Util.extend({}, OpenLayers.Feature.Vector.style['default']);
   style_blue.strokeColor = "blue";
   style_blue.fillColor = "blue";
   var style_green = {
      strokeColor: "#339933",
      strokeOpacity: 1,
      strokeWidth: 3,
      pointRadius: 6,
      pointerEvents: "visiblePainted"
   };

   vectorLayer = new OpenLayers.Layer.Vector("Simple Geometry");

   // create a point feature
   var point = new OpenLayers.Geometry.Point(-110, 45);
   pointFeature = new OpenLayers.Feature.Vector(point, null, style_blue);

   // create a line feature from a list of points
   var pointList = [];
   var newPoint = point;
   for(var p=0; p<5; ++p) {
      newPoint = new OpenLayers.Geometry.Point(newPoint.x + Math.random(1),
                                    newPoint.y + Math.random(1));
      pointList.push(newPoint);
   }
   lineFeature = new OpenLayers.Feature.Vector(
      new OpenLayers.Geometry.LineString(pointList),null,style_green);

   // create a polygon feature from a linear ring of points
   var pointList = [];
   for(var p=0; p<6; ++p) {
      var a = p * (2 * Math.PI) / 7;
      var r = Math.random(1) + 1;
      var newPoint = new OpenLayers.Geometry.Point(point.x + (r * Math.cos(a)),
                                       point.y + (r * Math.sin(a)));
      pointList.push(newPoint);
   }
```

```
    pointList.push(pointList[0]);

    var linearRing = new OpenLayers.Geometry.LinearRing(pointList);
    polygonFeature = new OpenLayers.Feature.Vector(
        new OpenLayers.Geometry.Polygon([linearRing]));


    map.addLayer(vectorLayer);
    map.setCenter(new OpenLayers.LonLat(point.x, point.y), 5);
    vectorLayer.addFeatures([pointFeature, lineFeature, polygonFeature]);

}

var origin = new OpenLayers.Geometry.Point(-111.04, 45.68);
function resizeFeatures(scale) {
    pointFeature.geometry.resize(scale, origin);
    lineFeature.geometry.resize(scale, origin);
    polygonFeature.geometry.resize(scale, origin);
    vectorLayer.redraw();
}
```

Appendix III: Post Study Questionnaire – User Study

Based on your recent experience with OLUnitTest for unit testing please evaluate the framework. Feel free to add comments in the given space.

- Inclusion of domain knowledge – How did you find the idea of assertions related to geospatial properties like projections, feature geometry etc.?

  Very Poor Poor Fair Good Very Good

- How did you find the assertion styling?

  Very Poor Poor Fair Good Very Good

- What rating would you provide to the documentation in terms of quality?

  Very Poor Poor Fair Good Very Good

- What would you rate this framework on the basis of ease of use?

  Very Poor Poor Fair Good Very Good

- Learnability is the capability of a software product to enable the user to learn how to use it. How did you find the learnability of OLUnitTest?

  Very Poor Poor Fair Good Very Good

- Based on your experience with executing tests from different frameworks, what would you rate the integration of OLUnitTest with other framework?

  Very Poor Poor Fair Good Very Good

- Based on the documentation and your experience in adding new assertions to the existing set, what would you rate the framework design in terms of extensibility?

  Very Poor Poor Fair Good Very Good

Appendix IV: Example Assertions from OLUnitTest

```
//Map Assertions
   assert.isMapProjection = function (expectedProjection, mapObj, message)
   {
      if (expectedProjection.projCode != mapObj.projection.projCode)
      {
          throw message || "assert.isMapProjection failed.";
      }
   };

   assert.isDisplayProjection = function (expectedProjection, mapObj, message)
    {
      if (expectedProjection.projCode != mapObj.displayProjection.projCode)
      {
          throw message || "assert.isDisplayProjection failed.";
      }
    };
   assert.controlIsPresent = function (expectedControl, mapObj, message)
   {
      var flag = new Boolean("false");
      for(var i =0; i< mapObj.controls.length; i++ )
      {
         if(mapObj.controls[i].displayClass.toLowerCase()                    ===
expectedControl.displayClass.toLowerCase())
          {
           flag = true;
          }
      }
      if (flag==false)
      {
      throw message || "assert.controlIsPresent failed.";
      }
   };

   assert.layerIsPresent = function (expectedLayer, mapObj, message)
   {
       var flag = new Boolean(false);
       for(var i =0; i< mapObj.layers.length; i++ )
       {
          if(mapObj.layers[i].name.toLowerCase() === expectedLayer.name.toLowerCase())
          {
             flag = true;
          }
       }
       if (flag==false)
       {
```

```
        throw message || "assert.layerIsPresent failed.";
    }
};

assert.isBaseLayer = function (expectedLayer, mapObj, message)
{
   if(mapObj.baselayer.name.toLowerCase() != expectedLayer.name.toLowerCase())
   {
      throw message || "assert.isBaseLayer failed";
   }
};

//expectedCenter must be in the same projection as the map center
assert.isCenter = function (expectedCenter, mapObj, message)
{
   if((expectedCenter.lon != mapObj.center.lon)|| (expectedCenter.lat != mapObj.center.lat))
   {
      throw message || "assert.isCenter failed";
   }
};

assert.isCurrentZoom = function (expectedZoom, mapObj, message)
{
   if(expectedZoom != mapObj.zoom)
   {
      throw message || "assert.isCurrentZoom failed";
   }
};

//will only work if the popup ID is known, ID can be passed through the constructor
assert.popupIsPresent = function (expectedPopupID, mapObj, message)
{
   var flag = new Boolean(false);
   for(var i =0; i< mapObj.popups.length; i++ )
   {
      if(mapObj.popups[i].id === expectedPopupID)
      {
         flag = true;
      }
   }
   if (flag==false)
   {
      throw message || "assert.popupIsPresent failed.";
   }
};
```

```
//Layer Assertions

//Common Assertions between Layer and Map
  assert.isUnit = function (units, Obj, message)
  {
      if(Obj.units != units)
      {
         throw message || "assert.isUnits failed";
      }
  };

  // Possible values are ‘degrees’ (or ‘dd’), ‘m’, ‘ft’, ‘km’, ‘mi’, ‘inches’.  Normally taken from the projection.
  // Only required if both map and layers do not define a projection, or if they define a projection which does not define units

  assert.isCurrentResolution = function (expectedResolution, Obj, message)
  {
      if(Obj.resolution != expectedResolution)
      {
         throw message || "assert.isCurrentResolution failed";
      }
  };

  assert.isMaxResolution = function (expectedResolution, Obj, message)
  {
      if(Obj.maxResolution != expectedResolution)
      {
         throw message || "assert.isMaxResolution failed";
      }
  };

  assert.isMaxExtent = function (expectedMaxExtentBoundsObj, Obj, message)
  {
      if((expectedMaxExtentBoundsObj.bottom != Obj.maxExtent.bottom) ||
(expectedMaxExtentBoundsObj.left != Obj.maxExtent.left) ||
(expectedMaxExtentBoundsObj.right != Obj.maxExtent.right) ||
(expectedMaxExtentBoundsObj.top != Obj.maxExtent.top))
      {
         throw message || "assert.isMaxExtent failed";
      }
  };

  assert.areNumZoomLevels = function (expectedNumZoomLevels, Obj, message)
  {
```

```
    if(expectedNumZoomLevels != Obj.numZoomLevels)
    {
       throw message || "assert.areNumZoomLevels failed";
    }
};
```

Appendix V: Raw Data

Following tables show the raw data that was used to generate charts for calculating percentage statement coverage for limited user study and comparative study respectively.

Participant	Lines of code executed by tests written using OLUnitTest	Total Statements
P1	21	29
P2	26	29
P3	8	29
P4	10	29
P5	20	29
P6	17	29
P7	20	29
P8	23	29

Table 2: Raw data used to generate Figure 10

Code Snippet	Lines of Code	Lines of code executed by tests written using OLUnitTest	Lines of code executed by tests written using Test Another Way	Lines of code executed by tests written using JsUnit
1	8	8	6	6
2	17	15	10	11
3	12	12	8	10
4	14	12	12	12
5	6	6	5	5

Table 3: Raw data used to generate Figure 13