Shortening Testing Time of a Web-based Business Application in Scrum using Automated Testing

Muhammad Qasim

University of Tampere
Faculty of Natural Sciences
Software Development
M.Sc. thesis
Supervisor: Timo T. Poranen
December 2017

University of Tampere
Faculty of Natural Sciences
Software Development
Muhammad Qasim: Shortening Testing Time of a Web-based Business Application in a Scrum using Automated Testing
M.Sc. thesis, 38 pages and 12 appendix pages
December 2017

Manual testing of web-based business applications causes delay in software delivery time because it is time-consuming, slow, error prone and less reliable. Automated testing is faster as compared to manual testing because it requires less human effort and thus reduces the error and maintenance cost. Based on the research findings from literature review, it was concluded that Selenium performs better as compared to other automated testing tools for testing a Single Page Application (SPA). This thesis focused on using Selenium as an automated testing tool for testing SPA. Furthermore, best practices of automated testing were utilized which resulted in faster software testing time. Interviews were conducted to assess the time taken per release during manual testing phase. Historical data from the past three years were also collected and analyzed to measure time difference in manual testing and automated testing. A significant effect was observed in testing time with the introduction of automated testing as compared to manual testing. Findings from the thesis conclude that automated testing leads to achieve shorter testing time and increased chances of detecting errors in a SPA web application.

Keywords and terms: automated testing, Selenium, Single Page Application (SPA), testing time, automated testing best practices, Scrum

ACKNOWLEDGMENTS

بِسْمِ اللهِ الرَّحْمٰنِ الرَّحِيْمِ

In the name of Allah (God), the Most Gracious, the Most Merciful.

I would like to express my deepest gratitude to thesis supervisor Dr. Timo T. Poranen at the University of Tampere, Finland for his valuable knowledge, time and supervision throughout this thesis.

I would like to thank the software company PlanMill Oy for providing me the opportunity to conduct the research at its premises and my colleagues, especially project manager Marjukka Niinioja for her swift feedbacks and assistance.

This thesis is dedicated to my friends and family, especially my late father Mr. Muzahir Hussain for instilling in me the inspiration, confidence, and passion for achieving the highest. Moreover, I would like to thank my elder brother Dr. Raza Habib for his guidance during the thesis and their unconditional love and support.

Contents

1. Introduction

Web-based business applications make use of the state-of-the-art technology to provide internet based solutions to its users which enhances efficiency and business productivity. In today's ever changing technology in emerging competitive information markets, fast paced implementation of vital business applications helps to cut down costs and boosts market share. On one hand, competitive advantage can be achieved by being a first mover in the market as compared to being late, while on the contrary delivering a defective and erroneous software application would be catastrophic. One way to prevent the errors in a software application or a program is by testing it before delivering it to the customer.

Testing is the process of executing a software program with intent of finding an error [Myers, 2004]. An error is a behaviour that differs from the expected behaviour of a software program. A typical example of an erroneous situation is an unexpected crash of a program. An inoperative state (erroneous condition) occurs within the existing functionality due to modification or addition to the existing code while developing software. These states or conditions are called regression. In order to identify a regression, it is essential to redo testing of already tested components of a software program every time the code is altered.

Testing can be done manually or automatically. In manual testing, a test plan is manually executed step-by-step by a software tester which is time-consuming and prone to human error. In automated testing, specialized testing software executes a test plan which is fast and error-free. Automated testing is becoming more and more essential, when businesses today are trying to deliver their software as fast as possible.

This thesis is structured in five chapters. First, the literature review is conducted in Chapter 2 and Chapter 3. It includes types of software testing: manual and automated testing, bottlenecks of manual testing and need of automated testing, benefits of automated testing and comparison of popular software testing tools available in the market. In Chapter 4, methodology and details of the approach to solve the problem is explained. It includes test cases, tools and techniques adopted for building the automated testing tool to test a Single Page Application used for daily time reporting. This chapter also covers the implementation of automated testing best practices for a web-based Enterprise Resource Planning (ERP) business application to reduce its testing time in a Scrum development methodology. In Chapter 5, empirical findings of the practiced approach to automate testing on TimeApp SPA and testing time difference between manual and automated regression testing of ERP business application are discussed in detail. This chapter also concludes the thesis.

This thesis also contains appendices. Appendix A shows the code snippet written in Java for automating TimeApp SPA. Appendix B lists all the Selenium function used to

automate this SPA. Appendix C depicts the work flow, using sequence diagrams, between test program and SPA. Appendix D lists all the hardware and software required to setup test environment. Appendix E includes the guide on interview conducted at PlanMill Oy with manual testers to know the time taken to perform manual regression testing of company's ERP application and Appendix F presents, in tabular format, the actual testing time recorded by both manual and automated testing over a period of past 3 years from January 2014 to December 2016.

2. Web Applications and Development Process

Web-based applications are well known and widely used types of software programs [Dobolyi et al., 2011]. Web applications have gained popularity due to ubiquity of web browsers and also maintaining a web application requires relatively less amount of complexity. A web application can either be a small website or a multi-tiered application and can serve thousands of heterogeneous globally distributed users simultaneously.

Client-Server Architecture

In client-server architecture there are two entities, a client and a server. When a browser on the client side requests a webpage, server responds to HTTP requests from that client. After server receives a HTTP request, it analyses the URL, extracts the folder path and the document name request by the client and serves the client with requested information. Client side data is short lived and is lost when page is refreshed. However, server side data is persistent. Client-Server architecture is illustrated in Figure 1.

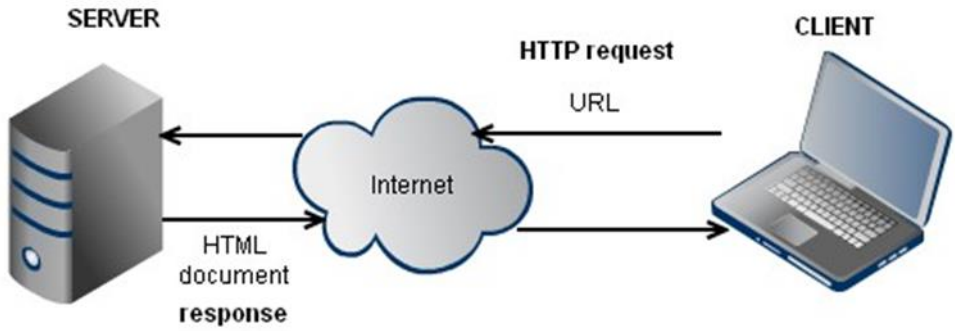


Figure 1: Web application client-server architecture [László, 2010].

Web application is a client-server application in which web browser acts as a client. When a user accesses a webpage using a web browser, server prepares the webpage containing response data. The webpage is delivered to client via HTTP requests and data is displayed in elements of that web page. Although client and server application could run on a same computer, but in standard web applications client and server execute on different computers.

A server could perform centralized or decentralized processing of incoming client requests. In a centralized processing a single host computer manages all requests whereas in distributed processing more than one decentralized computer handles client requests. These multiple host computer could either be located at the same location or could be at geographically dispersed location.

Document Object Model

The Document Object Model (DOM) describes the structure and relationship between different elements in a webpage. A webpage is a document displayed in a web browser.

The DOM of a webpage represents elements of a webpage which can be manipulated. It provides a standard API to access data from HTML and XML documents. DOM can be used to add and access elements in the document using a logical hierarchical structure. Document Object Model is illustrated in Figure 2.

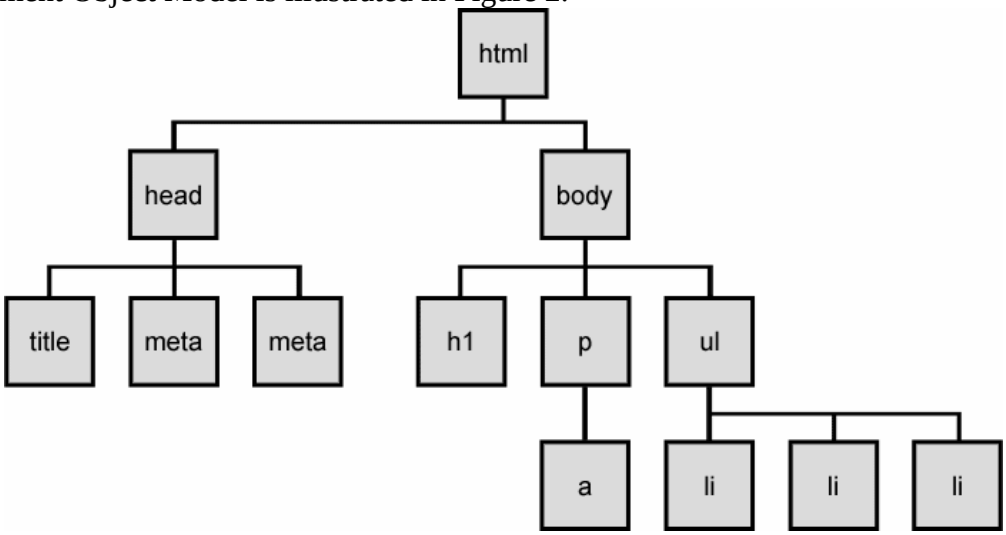


Figure 2: Document Object Model [Chow, 2016].

HTML or XML elements can be treated as tree structure, therefore, can be selected as object. Elements in a form of a webpage such as a link or an image can be accessed using hierarchical object structure of DOM. For example, to access the first link element in a form, the DOM access structure would be *document.form[0].getElementById(0).href*. There exist many different browsers freely available to navigate websites. Although, each browser may have taken a different approach in implementing DOM, they all follow the DOM conformance standard developed by World Wide Web Consortium [W3C, 1994].

AJAX

Asynchronous JavaScript and XML (AJAX) is modern method of client server communication. AJAX based webpages are dynamic and user-friendly as compared to traditional websites. Using AJAX a client can receive response from servers in formats such as JSON and XML. AJAX uses JavaScript and XML together in order to communicate with server. AJAX can be used in websites which require real time validation of user data. In a traditional website, that would not be possible for the user without him having to completely fill all data and submit the form to get validation errors.

Before AJAX was used in client-server communication, whenever a client sent a request to server, the entire web page had to refresh to display the updated received information. However, with AJAX, the communication with server is “asynchronous”. This means that the client browser does not have to wait for a response from server. The browser user interface does not show blocked or loading symbol for the time period when the request is being processed. User can continue using the website as usual and

when the server completes and sends data response to clients, only the elements responsible for displaying the updated information are refreshed without requiring refreshing the whole page. Difference between classic and AJAX web application models is depicted in Figure 3.

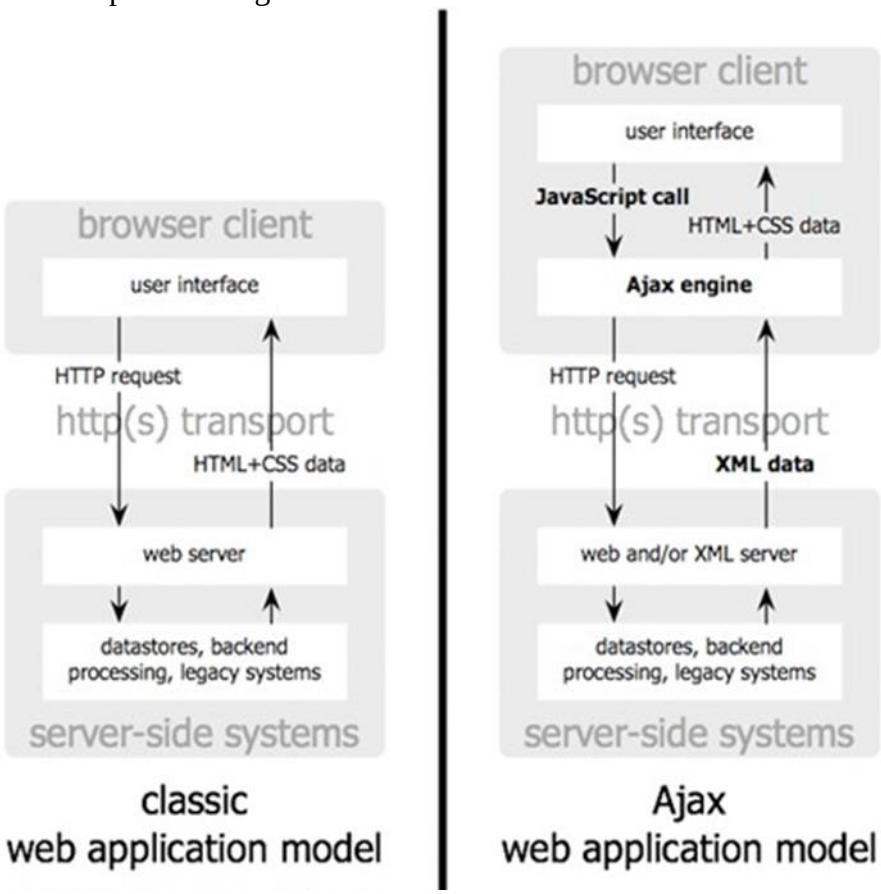


Figure 3: Classic vs AJAX web application model [Garrett, 2005].

The benefit of AJAX based client server communication is that it takes relatively less response time and network resources to serve a request as compared to traditional method of communication. This enhances user experiences and makes a user feel comfortable when he interacts with website.

2.1. Single Page Application

Single Page Application (SPA) is a web application that has single webpage inside which dynamically updates the page information based on the user input. SPA starts and stays on a single page after loading and provides a desktop like user experience. SPAs are responsive as they are built using AJAX which does not require full page reload to update the displayed web content. Server-side is stateless in a SPA. Silver [2016] states that AJAX call handles the routing at client-side instead of at the server and requires client side development.

SPA JavaScript Framework

Due to the rapid application development requirement, JavaScript frameworks can be used to develop interactive web applications. Most SPAs are built using JavaScript framework and help developers to use HTML and JavaScript fluidly. Although, traditionally JQuery [2016] has been used to build complex web interfaces, in the last couple of years JavaScript frameworks have gained popularity in developing complex user interfaces for the SPA. Using JavaScript frameworks allows developers to focus on UI development without diving into the code complexities. Some of the most popular JavaScript libraries to develop SPA are Angular [2016], React [2016] and Backbone [2016].

Oscillot [2016] explains *React* is developed by *Facebook* and helps to extract information from web elements by separating DOM into components. React creates an internal state machine for the existing DOM elements on a webpage. When DOM elements change on reloading of AJAX, the new elements are added to the state machine representation. This approach allows for updating the DOM very quickly.

2.2. Software Development Process Models

This section covers information on software development process models including incremental process model. It also discusses agile software development methodology including Scrum to develop and deliver software to the customers.

A software process is a group of related activities that produces software [Sommerville, 2010]. According to Sommerville [2010] the following four activities are, in some form, part of many software processes:

Software specification: The specification defines the functionality and limitations of the software.

Software design and implementation: The designed software is implemented and it meets the specification.

Software validation: Validation is performed on the implemented software to make sure that it matches the specification.

Software evolution: The software is evolved to cope with the changing customer requirements.

A software process model represents a basic form of a software process [Sommerville, 2010]. There are many different process models that are used in development of software. Some examples include Waterfall model, Incremental model and Spiral model. The applicability of these models depends on the requirements to develop software. In this thesis incremental process model is discussed only.

Incremental Process Model

The incremental process development model holds the notion of developing an initial implementation of software, receiving the user feedback on it and iterating it through

several increments until an acceptable system by customer is developed [Sommerville, 2010]. In this model, the first three software process activities: specification, development, and validation are interrelated, with feedback across activities.

Incremental development works the way we are accustomed to solve our problems. We seldom develop a complete solution to a problem. We work our way towards the solution by taking steps in a linear fashion, backtracking to the step when realizing a mistake is made. In a similar manner, software is developed in increments. It is cheaper and easier to make changes in software when produced incrementally.

Each increment of the software implements some of the requirements demanded by the customer. It is usually seen that the businesses include the most important or most urgently required functionality by the customer in early increments of the software. This gives customers an early access to the software to evaluate it for the needed functionality. These early increments are then modified for any missing or unnecessary functionality. The benefit of following this pattern is that only this increment has to be improved which is much easier and cheaper at this stage of development. Customers can also propose new requirements to be included in the later increments of the software.

Nowadays, incremental development is the most common approach to develop software applications. One of the approaches to develop software applications is agile software development approach which is discussed in detail in following sections.

Agile Software Development

Of the incremental development methods, agile software development method is one of them which rely on small increments and rapid feedback from customers which are made available to them every two or three weeks. Informal communications rather than formal meetings are preferred in this method to minimize the documentation.

Agile methods are well suited for projects where the software requirements change rapidly during the development process. These methods can deliver working software rapidly to the customers. This enables them to propose new and changed functionality to be included in later increments of the software.

The leading developers of agile methods agreed on a manifesto called as an agile manifesto [Agile Manifesto, 2001] to reflect agile methods which states: *"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: individuals and interactions over processes and tools, working software over comprehensive documentation, collaboration over contract negotiation, and responding to change over following a plan. That is, while there is value in the items on the right, we value the items on the left more".*

Scrum Method

Scrum [Larman, 2003] is an agile software development method that was realized by Jeff Sutherland and his development team in the early 1990s. The principles of this method are consistent with the agile manifesto. Scrum consists of three phases. The first phase also known as an outline planning phase includes forming the goals for the software project and designing the software architecture. This phase is followed by the sprint cycles which deliver the software in gradual increments to the customers.

A sprint cycle could be of two or three weeks or more. This depends on the team size and velocity of software increment. A sprint cycle starts with a sprint planning meeting. Sprint is a planning unit in which requirements to be delivered to customers are assessed and prioritized for the development and gets implemented as well. The entire team brainstorms on prioritised tasks and discusses possible solution at an abstract level. Sprint planning notes are added to the task. Near the end of the sprint there is a sprint review meeting held between the development team, the product owner and scrum master. The team then presents briefly the solution to the tasks implemented. Product owner decides whether the task at hand fully satisfies the definition of done only in which case the task will be eligible for release in the upcoming release cycle. Otherwise it would be pushed back to the backlog as a technical debt.

The final phase closes the project with the delivery of completed software to the customer, writing of the required documentation such as user manuals and a formal meeting with all the stakeholders of the project to assess the lessons learned from the project. After a sprint is completed successfully, the team gets together for sprint retrospective meeting. The team ponders and reflects on the good aspects, discuss possible improvements and identifies tasks for the next sprint cycle.

3. Software Testing, Tools and Best Practices

A software bug is a failure or flaw in a software application which leads the program to behave in unintended ways. Generally, bugs are caused by inadequate or incorrect coding logic but could be due to an error or unintended mistake. Such an erroneous code produces invalid output and hence could lead to software crash. Finding errors in the code requires a program to be tested properly.

Myers [2004] defines testing as the process of executing a program with the intent of finding errors. Whenever a new software system is developed or an addition is made to the existing software, it should undergo testing. Testing ensures that what is developed is not error prone and is not bound to fail. Software testing ensures that the software will not crash. Although it is not possible to reach absolute zero bug state for large scale software, testing ensures customer happiness by delivering a close to bug-free software. Testing make sure that the features fixed or improved in the new version of software release have not affected the functionality of other software component and all parts of the software system are working as expected.

Testing a software application requires setting up a test environment, identifying major functional units in the system that must be incorporated in writing a test plan and validating business rules and processes by taking into consideration available resources and time constraints [Al-Hossan and Al-Mudimigh, 2011]. However, a tester might have misunderstood the functionality of the program and may have a written test case that might not correctly identify and handle borderline test scenarios [Leitner et al., 2007].

Furthermore, a test plan can have many test scenarios. A test case, however, constitutes a basic component of any test plan. A test case includes pre-conditions, post-conditions and test input parameters. The testing process ensures that almost all possible valid and invalid scenarios which could occur have been considered. Testing is a step forward towards ensuring that the incremental release will be bug-free.

3.1. Types of Software Testing

There are two types of software testing: manual and automated testing.

Manual Testing

Manual testing is the testing of software manually, i.e., without using any automated testing tool or any test script. Testers position themselves as an end user and use the software features to ensure it is bug free. A written test plan is followed to perform manual step-by-step testing of the application [Kumar, 2012]. Manually testing the same parts of a software program every time the code is changed sounds a little boring and time-consuming. Other drawbacks of manual testing include that it provides less test coverage, results in a higher number of errors and requires a large number of testing

staff. Some challenges in manual testing are: Manual testing is laborious activity and depends on tester's skills set such as patience, speculativeness, creativeness and his ability to think of all possible test data input. On large software applications performing repetitive manual testing is complicated. Sharma [2014] mentions that for large applications, repetitive manual testing can prove to be tedious and extremely time-consuming effort. Maintaining test documents for traceability and software audit purposes is difficult. Manual testing requires more human involvement. With the increase in the size of the manual test pool, organizations need to hire and train more and more testers. The increase in size of the manual test pool creates issue related to re-testing, transparency and knowledge sharing between the testers. Changes in Cascading Style Sheets (CSS), JavaScript and size difference of objects in the user interface are difficult to identify in manual testing.

Automated Testing

Considering the shortcomings of manual testing, there is a growing need to automate the testing process. Automated testing is the use of software (under a set of test preconditions) to execute tests and verify whether the actual outcomes and the anticipated outcomes are identical.

Enterprises have failed to successfully implement automated testing due to incoherent and unworkable test automation solutions [Locke and Balaraman, 2012]. They claim that plausible methodology would be to consider the technical and monetary requirements of implementing automated testing. As testing is a continuous activity, a dedicated testing team is required to maintain the existing test automation script every software increment which requires considerable time and effort [Locke and Balaraman, 2012].

Automated testing involves using best practice and incorporating well-established tool in the software industry to automate the software testing. As the software evolves, new features and business processes are implemented. This increase in software complexity and size impacts the testing time and has financial repercussions. Therefore, there is a need to reduce testing time which could be achieved by incorporating automated testing. Automated testing aims at full computerization [Bertolino, 2007]. However, complete automation has not yet fully achieved in the software industry [Berner et al., 2005]. Malekzadeh and Ainon [2010] state that product's quality is improved by automated testing. Additionally, Karhu [2009] claims that automated testing reduce the number of defects thus improving the quality of the software product. Wissink and Amaro [2006] claim that automation reduces the overall running time of tests, i.e. the time required for testing. Similarly, du Bousquet and Zuanon [1999] mention that automation tests allow for running more tests within an allocated time period. In the case of repeated running tests, higher reliability is achieved with automated testing [Fewster and Graham, 1999]. They also assert that developers are

more confident with the software being delivered when they employ automated testing. The same set of the test cases that are frequently repeated and reusable leads to benefits [Dallal, 2007]. Polo and others [2007] mention that using test automation enhances testing tasks quality and reduces costs.

Adopting Automated Testing in Scrum

One of the most popular agile frameworks for completing large complex software programs is Scrum. In a Scrum process, there is Sprint of which the duration is two to three weeks. New functionality is added to the existing system in every Sprint cycle and the increment is released to the customer. A software release is a fully developed increment of a software product. As an example of very fast increment cycle O'reilly [2005] explains that Flickr has extremely different development model in which they deploy new builds up to every half hour.

In a running Sprint, testing of developed code starts as soon as there are items to review in a Scrum Board. Thus, testing should be performed comprehensively and rapidly in an agile working environment. Using automated testing techniques enables testers to test the developed software thoroughly. Al-Zain and others [2012] also emphasizes that in agile software development process, software testing is not a separate phase executed at the end of Sprint but instead is integrated into the development process. Regression tests are extremely necessary to be carried out in agile development in order to make sure that enhanced functionality does not break the existing system.

3.2. Methods of Software Testing

Software testing is composed of two different testing methods, black-box testing and white-box testing.

Black-Box Testing

Black-box testing does not take into consideration the internal process of the software but examines the input and output of the software. The tester will not examine the code structure, internal paths and implementation details of the software. It is entirely based on software requirements and specifications. A black-box could be an OS, a database or even a website. The steps involved in performing a black-box testing are:

- System requirements and specifications are initially analysed.
- Tester selects valid input scenarios to check whether the application being tested processes input correctly. This is filled by selecting invalid input scenario to identify if the software under test is able to detect it.
- The expected output of these scenarios is determined.
- Relevant test case with selected input is developed and executed.
- Software tester then compares the actual output with the expected output.

- Bugs and defects are identified and fixed and the system is re-tested.

White-Box Testing

White-box testing method validates the internal structure and working of the software code and examines functional requirements. It examines the working details of the each subsystem of the software application. It focuses on strengthening security, the input and output flow through the application and enhancing usability. White-box testing helps to identify bugs and poorly structured paths in the coding process, the functionality of conditional loops because each statement, object and function is tested on an individual basis. The steps involved in performing a white-box testing are:

- Understand the source code of the application.
- Write test cases and execute them.

In summary, black-box testing involves testing of an end-user type perspective, thus giving an abstraction from code. On the other hand, white-box testing is a detailed internal system test which helps detect internal errors which may lower the performance of the system. White-box testing can be quite complex depending on the application being tested.

3.3. Levels of Software Testing

There are four main levels of software testing: unit, integration, system and acceptance testing.

Unit Testing

Unit tests are developed to perform modular testing of a program's functional behaviour, in which every unit is composed of a set of functions and each function is independently tested by providing input values [Sen et al., 2005]. Unit testing is also known as component testing. Unit testing ensures that all individual components of the software program perform expected functionality. This type of testing demands software tester to be able to understand the application code in order to check if the module is developed correctly. Unit testing involves checking valid and invalid test cases. Unit tests are usually developed by software developers.

Integration Testing

Integration testing is performed after two or more software modules are integrated into a large software system. This test ensures that the developed software modules integrate successfully with the existing system and does not break any existing functionality. Data transfer and communication between different modules is tested thoroughly. In Big Bang Integration testing, tester waits for all modules to be developed but it comes with a disadvantage that it increases the project execution time. Alternatively, incremental integration testing approach can be employed in which modules are tested as and when

they are available. In such a testing scenario developers will develop a stub (not a complete implementation) to facilitate testing if their module is not yet fully developed to facilitate for testing. To increase integration testing effectiveness a top-down approach can be used in which higher level modules are tested first.

System Testing

System testing is performed when all the modules of a software product are complete and fully integrated. Completed software is interfaced with other software and hardware systems. Therefore, system testing validates the quality of the whole software system. Quality attributes such as reliability, security, and maintainability are tested in this type of testing [Luo, 2001].

Acceptance Testing

Acceptance testing is done when software is tested for compliance with the business requirements and whether it can be delivered to customer or not. In this testing, the non-functional features of the software (such as usability) are tested which establishes confidence of the customer in the software. The main testers of acceptance testing are customers and users of the software.

Regression Testing

Regression testing can be done at any of the four main levels. Regression testing means re-testing the previous working software after changing parts of the code to ensure unmodified parts of software program continue to function as expected. Therefore, such a testing ensures code changes did not introduce new errors in the existing software [Mittal and Acharya, 2003]. Furthermore, it also confirms that the software program continues to implement all the desired features as expected in the software increment and there is no grey area. This testing is performed to ensure software reliability and quality. If executed manually on each software increment release, regression testing becomes repetitive and laborious task because manually running all regression tests is bound to time constraints. Tester, therefore, chooses selective test cases and hence unnoticed defects remain part of the delivered software.

3.4. Testing Tools

Different automated web application testing tools are discussed in this section and also compared based on multiple criteria such as pricing, application and language support. Manually defining input values for performing functional testing is time-consuming and error prone and does not promise to test all aspects of the program under test [Sen et al., 2005]. Therefore, automated testing tools have to be employed to perform comprehensive testing. There are various automated testing tools such as SoapUI

[2005], HP Quick Test Pro [2001], TestComplete [1999], Coded UI [2010] and Selenium [2015] that vary in functioning, pricing and ease of use.

SoapUI is a cross-platform, open source API testing tool. It is used to perform automated functional and regression tests. HP Quick Test Pro is a commercial, licensed and user-friendly tool which works well with Web-based and Windows applications. During the execution, screenshots of each page navigated is stored [Kaur and Kumari, 2011]. TestComplete is a licensed tool which runs on different operating systems such as Windows, Web, Android, and on iOS applications. A tester can use record and play option to create attest manually which later can be played over and over again as an automated test [Dubey et al., 2014]. Coded UI is used for testing user interfaces [Nagarani et al., 2012]. Using Visual Studio, the tester will either record user actions or write test cases which can be later then played back for verification of user interactions. Selenium supports testing on multiple browser platforms [SeleniumHQ, 2015]. Along with testing web application, Selenium can be used for testing in the continuous integration environment. It supports many high-level programming languages such as Java, C# and PHP. Brief comparison of these automated testing tools is given in Table 1.

Comparison of Automated Web Testing Tools

Tools/ Criteria	Selenium	SoapUI	HP QTP/UFT	TestComplete	Coded UI
Pricing (USD)	Open source and free of cost	Open source but commercial license version for 499	Licensed and very expensive 8000	Licensed and costs 1999	999
Cross Platform	Windows, Linux, Mac, Unix	Windows XP and later	Windows only	Windows 7 and later	Windows 7 and Higher
Application Support	Web applications only	Web applications as well as Client /Server applications	Client/Server application, Mobile application	Web, Desktop and Mobile application	IE Only
Web Browser Support	Chrome, Firefox, IE and Opera	IE, Firefox, Chrome	IE-Firefox-Chrome	IE, Firefox, Opera, Chrome	
Scripting Language	Java, C#, Ruby, PHP, Python, JavaScript	Groovy, JavaScript	JavaScript	VBScript, Delphi, C++,C#, JavaScript	VB.net-C#
Technical Support	No technical support	Good technical support	Good technical support	Good technical support	Good technical support
Product Support	Open Source community	SmartBear support	HP support	SmartBear support	Microsoft

Table 1: Comparison of automated testing tools based on the listed features (Adopted from [Monier and El-mahdy, 2015]).

3.5. Selenium Testing Tool

Selenium is an automated web testing tool which contains Selenium WebDriver API. WebDriver API for Selenium provides an object-oriented application programming interface used to extract dynamic changing elements from web pages. Selenium sends direct calls to a particular browser's built-in automation features and does not inject the JavaScript code into the browser. Following are the details on three main browser drivers [Selenium WebDriver, 2012].

Firefox Driver

WebDriver interacts with Firefox browser using Firefox Driver as a plugin. Firefox Driver can be used on major platforms such as Windows, Mac and Linux which support Firefox browser. Firefox driver supports JavaScript tests in the actual Firefox browser.

Internet Explorer Driver

This driver implements wire protocol of WebDriver and runs as a server. Internet Explorer Drive supports all major versions of Windows platform such as Windows Vista, Windows 7 and 10. Just like Firefox driver, it also supports JavaScript tests in the actual Internet Explorer browser.

Chrome Driver

WebDriver utilizes Chrome Driver binary to interact with Chrome browser. Google Sites [2016] explains that WebDriver uses Chrome Driver to interact with Chrome browser. Chrome Driver can be used on major platforms such as Windows, Mac and Linux which support Chrome browser. Furthermore, Chrome Driver also supports JavaScript tests in the actual Chrome browser.

Selenium WebDriver API Commands and Operations

Fetching a Webpage

Extracting text value on a page is performed by calling the “get” method. It should be noted that WebDriver might not receive the page load complete event or control may return before the page finished loading. Therefore, it is recommended to use Explicit and Implicit Waits along with the “get” functionality.

Locating Web Elements

Web elements can be located using WebDriver object and calling “Find Element” method. An exception is thrown if an element could not be found. This method utilizes “By” query object with ID, Class name, Tag name, Name, Link Text, CSS or XPath. Few of these are explained below with examples:

By ID: Locate an element by id is the most preferred method. An example below shows how to find an element by id in Java.

```
<div id="start">...</div>
WebElement news= driver.findElement(By.id("start"));
```

By Name: Locating an element by name attribute is very common. An example below shows how to find an element by name attribute in Java.

```
<input name="news" type="text"/>
WebElement news= driver.findElement(By.name("news"));
```

By CSS: In case, a particular browser does not support style sheets then the method Sizzle is used. An example below shows how to locate an element using cssSelector in Java.

```
<div id="news">
<span class="sports">sports</span>
<span class="sports latest">latest on sports</span>
</div>
WebElement news= driver.findElement(By.cssSelector("#news span.sports.latest"));
```

By XPath: WebDriver utilizes browser's built-in XPath query functions. An example below shows how to locate an element using XPath in Java.

```
<input type="text" name="news" />
List<WebElement> inputs = driver.findElements(By.xpath("//input"));
```

Drag and Drop

Below is an example of how to perform a drag and drop operation

```
WebElement element = driver.findElement(By.name("source"));
WebElement target = driver.findElement(By.name("target"));
(new Actions(driver)).dragAndDrop(element, target).perform();
```

3.6. Concerns related to testing SPA

Silver [2016] explains that JavaScript routing instead of browser based navigation is the root cause of many issues such as handling navigation and fast back, remembering navigation and history position, handling navigation cancelling and increase performance issues.

Furthermore, SPA's have an undesirable feedback mechanism and thus badly affects user experience. In a SPA application, unlike a normal page in which browser displays loading indicator giving feedback to the user about the status of load progress, JavaScript client side routing requires developers to use JavaScript-based indicators to update load progress to UI which is not very efficient. Silver [2016] explains that although SPA's are supposed to provide a better experience but it is difficult to design them because an entire website is implemented into a single document using JavaScript. Some of the major concerns are explained in subsequent sections.

AJAX timeout issue

SPA extensively uses AJAX which implies that selenium may not be notified when the AJAX method completed. This behaviour is different from when a (real) page finishes loading. Selenium does not receive any signal when the *AJAX - Send a Request to a server* call has finished. Therefore, code optimization techniques are required for each method call in Selenium to handle timeouts of XMLHttpRequest calls. One way to achieve this in Selenium is before continuing at the next automated task the code waits for a certain amount of time to elapse. This is done by using explicit and implicit waits [WebDriver, 2016]. During testing, the automated code using Selenium is waiting for the timeout: the occurrence of which is undefined and it could take longer than normal to finish an AJAX call. This has direct implication on the test execution and as a result leads to slower automated tests.

Testing React based SPA issue

Fox [2009] explains that when writing Selenium-based tests, UI framework using which the application is developed poses distinctive challenges. Oscillot [2016] explains that React causes problems as explained below when used with Selenium for testing:

- Failure to enter text in the input element.
- Problems with reading a value of an element.
- Events not fired correctly when Selenium changes value in input fields.
- Validation failure on web-form submission.

Stateless SPA issue

SPA's UI may not necessarily reflect the correct application state if it is not maintaining user state. For example, in a case of network failure, the application loses a synchronous connection with a remote database; the UI might not reflect the current changes until the connection is restored. This adds complexity in the testing because tester must either recreate server states to emulate all possible scenarios due to statelessness of the application [Fox, 2009].

3.7. Automated Testing Best Practices

Automated testing best practices in an agile software development are a way to increase the efficiency of automated testing tool resulting in shortening the software testing time According to Stockdale [2016], automation of test cases significantly reduces software tester's manual testing time of repetitive tasks. Automated testing keeps software development processes agile and lean. Best practices of automated testing for reducing the software testing time include choosing the right test cases to automate which means test cases which consume large amount of time such as repetitive tests, test with large data sets and test cases to be run in different web browsers should be considered for automation. Automated testing should be done throughout a sprint i.e. running the test cases before, during and after the sprint. Moreover, developing automated tests that last longer i.e. the testers should focus on writing small test cases which focus on software units. Furthermore, such test cases are written which are independent of UI and thus are not affected by the UI changes.

SmartBear [2016] claims that following automated testing best practices ensure a complete software testing: decide the test cases to automate, test often and test early, choose the correct automated testing tool, create test data of good quality, and create automated tests that are not affected by changes in user interface.

Srivastava [2002] explains that testing time is reduced by implementing automation and thus affects the quick time-to-market increment of the developed software application. With no human intervention automated test cases can run unattended twenty-four hours and thus testing can be completed faster as compared to manual testing.

Motwani [2010] discusses that before starting-off with automated testing the tester has to make sure that interface to be tested has been identified, scope of automation has been defined, individual test cases to be automated have been identified, test cases have been fine-tuned., the right tool has to be decided, follows proper test scripting standards and identifying common steps and converting them into functions.

Bach [1999] recommends that distinction is to be made between the automation and the process that it automates. The test tool should be carefully selected and assure product maturity to reduce maintenance costs. Right automation tool selection depends

on the application to be tested. Polo et al. [2013] provide a list of guidelines on test automation claiming that for a particular project, scalability should be taken into consideration when selecting test automation tool. Moreover, for each test activity, entry and exit conditions should be defined and test cases should be as specific as possible for testing software features. Furthermore, he mentions that input and output for each test case should be clear. Atlassian [2016] recommends running automated tests in parallel.

4. Case Study

A case study was conducted at PlanMill Oy [PlanMill, 2016]. PlanMill Oy is a leading software company providing user-friendly web-based Enterprise Resource Planning (ERP) cloud solutions designed for the service businesses. As a vendor, the core objectives of the company are to ensure long lasting customer relationships and credibility. To ensure these accomplishments, a company has to establish itself as both dependable and reliable. It must be noted that many factors may affect the customer relationship such as late product deliveries or deliveries failing to match customer's expectations.

4.1. Background

The company incorporated the Scrum method involving sprint cycles of two to three weeks for developing and delivering software increments to the customers. Approximately three years ago, the company was using manual testing approach to test the software application for regression before delivery of the increment to the customer. This manual testing job involved four testers. Testers were manually executing the test scripts written on the document. The test scripts were composed of test cases to perform regression testing on user interface of web-based ERP application before every increment was delivered to the customer. There were total of fifty test cases. Manual testing, being time-consuming task, was taking too much time to test the application. Over the years, the company's customer base grew exponentially and to serve all the customers' requests the software should be delivered more frequently to the customers but manual testing was slowing down this frequent delivery. To achieve the frequent software delivery, the company developed an automated regression testing tool using Selenium to shorten the testing time in a Scrum.

The company also started developing the Single Page Application besides its traditional client-server application. The React JavaScript library was used to build the first SPA. It was necessary to automated regression testing of this application as well.

4.2. Hypothesis Development

As mentioned in previous Section 4.1, the company started using Selenium to perform automated regression testing; it was assumed that Selenium could also be used to execute automated regression testing on Single Page Application. Hence, following hypothesis was formulated which is as follows:

Hypothesis H1: *Selenium could be used to perform automated regression testing on ReactJS based Single Page Application.*

As also mentioned in the previous Section 4.1, realizing the need of automated testing; the company started using it to perform regression testing on the software increment before delivery to the customer. However, the company did not incorporate any best practices for automated testing while developing the automated regression testing tool. It was assumed that implementing best practices could further shorten the regression testing time. Hence, second hypothesis was formulated which is as follows:

Hypothesis H2: *Using automated testing best practices in the existing automated regression testing tool could shorten testing time.*

4.3. Research Questions

Based on the hypothesis, this thesis focused on answering following two research questions:

1. *Can Selenium testing tool is used to automate regression testing of a ReactJS based Single Page Application?*
2. *Can the application of automated testing best practices help in shortening the testing time of a web-based business application in a Scrum?*

4.4. Implementing Automated Testing Tool for SPA

This work in this section was set to demonstrate the first hypothesis H1. It involved creating an automated test tool using Selenium for testing a web-based calendar SPA called as *Time App*. This is a calendar application which was used for reporting time by employees of the company. The screenshot of landing page of *TimeApp* calendar application is shown in Figure 4. It has three views: day, week and month. We have used weekly view of *Time App* to do automated testing. It has functionality to create, update, resize, move and delete time reports. The methodology that was incorporated to implement automated testing is explained in following sections.

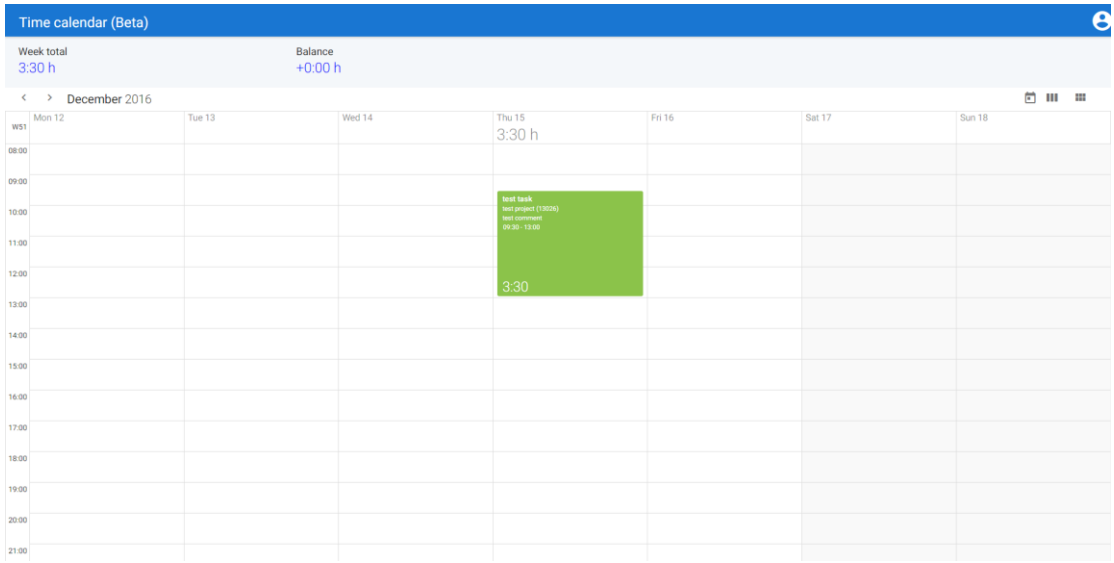


Figure 4: Screenshot of landing page of TimeApp web-based calendar application.

4.4.1. Designing Test Cases

Five test cases to perform automated testing on *Time App* were created based on its functionality. These five test cases are: Create Time Report, Update Time Report, Resize Time Report, Move Time Report and Delete Time Report. Sequence diagrams for these test cases also added in Appendix E.

Test Case ID: 01

Test Case Name: Create Time Report

Test Case Description: Create a time report on the specified time slot of the calendar application with input data, i.e. time report duration in hours and minutes, task name, billable or non-billable time report and comment for time report.

Preconditions: User logged in to calendar application successfully and loaded Time App with weekly view.

Postconditions: Created time report was saved in database.

Test Step #	**Test Step**	**Test Data**	**Expected Result**
01	Click on the specified time slot	Time slot day: today Time slot: 10:00-10:30	Open new time report creation dialog
02	Enter time report duration in hours and minutes text boxes	01 in hours 30 in minutes	Display entered duration
03	Select task name	task_create_time_report	Display selected task
04	Switch billable option on		Select billable option
05	Enter comment for time report	comment_time_report_1	Display entered comment
06	Click save button on dialog		Display created time report on calendar
07	Validate data on created time report	task_create_time_report comment_time_report_1 10:00 - 11:30 1:30	Valid data

Test Case ID: 02

Test Case Name: Update Time Report

Test Case Description: Update a time report on the specified time slot of the calendar application with input data, i.e. time report duration in hours and minutes and comment for time report.

Preconditions: User logged in to calendar application successfully and loaded Time App weekly view having one time report from 14:00-14:30 in today's date.

Postconditions: Updated time report was saved in database.

Test Step #	**Test Step**	**Test Data**	**Expected Result**
01	Click on the time report	Time report task name: task_update_time_report	Open edit time report dialog
02	Enter time report duration in hours and minutes text boxes	03 in hours 00 in minutes	Display entered duration
03	Enter comment for time report	comment_timereport_updated	Display updated comment
04	Click save button on dialog		Display updated time report on calendar
05	Validate data on updated time report	task_update_time_report comment_timereport_updated 14:00 - 17:00 3:00	Valid data

Test Case ID: 03

Test Case Name: Resize Time Report

Test Case Description: Resize a time report on the specified time slot of the calendar application.

Preconditions: User logged in to calendar application successfully and loaded Time App with weekly view having one time report from 12:30-13:00 in tomorrow's date.

Postconditions: Resized time report was saved in database.

Test Step #	**Test Step**	**Test Data**	**Expected Result**
01	Hold handle on time report and drag downwards to add 1 hour	Time report task name: task_resize_time_report	Display resized time report having an additional hour
02	Validate data on resized time report	task_resize_timereport comment_timereport_resized 12:30 - 14:00 1:30	Valid data

Test Case ID: 04
Test Case Name: Move Time Report
Test Case Description: Move a time report to the specified time slot on the calendar application.
Preconditions: User logged in to calendar application successfully and loaded Time App with weekly view having one time report from 18:00-18:30 in today's date.
Postconditions: Moved time report was saved in database.

Test Step #	Test Step	Test Data	Expected Result
01	Click and hold time report and move it to yesterday from today time slot	Destination time slot day: yesterday Destination time slot: 18:00-18:30 Time report task name: task_move_time_report	Display moved time report in destination time slot
02	Validate data on moved time report	task_move_time_report comment_timereport_moved 18:00 - 18:30 0:30	Valid data

Test Case ID: 05
Test Case Name: Delete Time Report
Test Case Description: Delete a time report on the specified time slot of the calendar application.
Preconditions: User logged in to calendar application successfully and loaded Time App with weekly view having one time report from 16:00-18:00 in tomorrow's date.
Postconditions: Time report was deleted from database.

Test Step #	Test Step	Test Data	Expected Result
01	Click on the time report	Time report task name: task_delete_time_report	Open edit time report dialog
02	Click delete button on dialog		Delete time report on calendar

4.4.2. Automating Test Cases

Automating test cases requires a selection of automated test tool at first place. Although, it was assumed in hypothesis H1 that Selenium could be used to automate testing of *TimeApp* SPA, there were some business conditions from the company management as well to select an automated testing tool. The criteria were that the selected test tool should be able to automate testing without difficulty, provide decent technical support from the vendor as well as from user community, support Java programming language and should be free of cost. Selenium fulfils these criteria as indicated in the review of automated testing tools in Section 3.4.

The WebDriver part of the Selenium was used to automate the test cases. A development environment was set up to build the automated testing tool. Maven was used to setup a Selenium WebDriver Java project. Maven downloaded the Java bindings (Selenium WebDriver Java client library) and all its dependencies using pom.xml configuration file and created a project ready to be imported in Eclipse IDE. Details on hardware and software can be found in Appendix D. Although the WebDriver API is rich, most of the times the test code will just find elements and interact with them using simple Selenium functions such as *sendKeys* or *click*. A complete list of Selenium functions used in the developed of automated testing tool for SPA can be found in Appendix B.

To locate web elements on the web-based calendar application, XPath expression was used by Selenium. WebElement was an important interface during development. It represented an HTML element. Generally, all webpage interactions were performed through this interface. Test cases were written in Java programming language and were mainly executed in Chrome browser.

The tests were dependent on test data to perform testing. The dependency test data for each test case was inserted directly into SQL Server [SQL Server, 2014] database using DbUnit library [DbUnit, 2002]. The command *mvn clean install* verify was executed in command line on Windows command prompt in project's directory to execute the tests and to generate the reports.

In following section, code implementation of test cases using Selenium functions is shown. Dependency test data which were inserted using DbUnit includes a project and a task assigned to a person.

Automating Create Time Report: Creating a time report requires a time slot to be specified on the calendar. This time slot was represented by Selenium WebElement and was located using XPath expression. The screenshot of create time report in TimeApp calendar application is shown in Figure 5.

```
String xpath_div = "//tr//td[@class='fc-widget-content']//div";
WebElement divClick = element(By.xpath(xpath_div));
```

As the new time report was created by performing a click on the calendar, so offsets x and y representing date and time respectively were calculated to locate the point to be clicked. These offsets combined with WebElement were passed as parameters to moveToElement() method of Selenium WebDriver where it moves to the element (and scroll it into view) and then performs a click on it.

```
new Actions(webDriver).moveToElement(divClick, xOffset,
yOffset).click().perform();
```

This click on WebElement opens a new dialog window to enter and save details of time report. Details include time report duration specified in hours and minutes, task, billable status and a comment. After time report creation, its data was validated. Again XPath was used to locate elements of time report. For example, task WebElement.

```
String xpath_task = "//div[@class='fc-content-skeleton']//td[" + dayOfWeek +
"]//div[@class='fc-task ellipsis' and text()='" + task_name + "']";
```

We have to wait explicitly for the visibility of task WebElement before validating its data like below:

```
wait.until(ExpectedConditions.visibilityOf(element(By.xpath(xpath_task))));
```

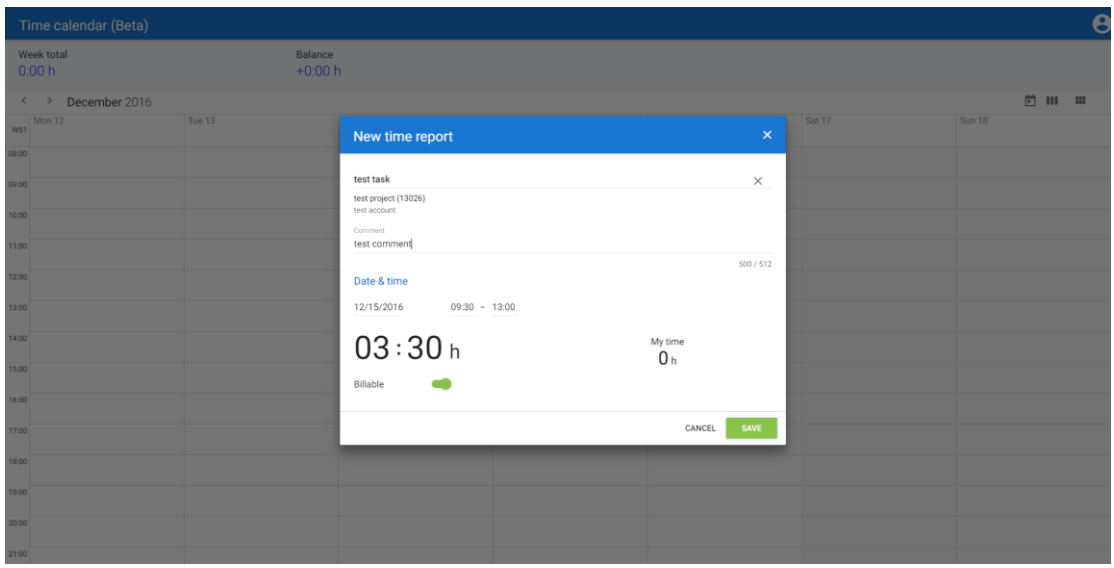


Figure 5: Screenshot of create time report in TimeApp calendar application.

Other code which was written that uses Selenium functions other than mentioned above can be found in Appendix A. Similar to automating create time report as above, other time report methods such as Update, Resize, Move and Delete were also implemented in *TimeApp* web-based calendar application.

4.5. Implementing Automated Testing Best Practices

This work was set to prove the second hypothesis H2. It involves implementing the best practices, discussed in Chapter 3, in an automated testing tool for the web-based ERP application used at the company to detect any regression before the delivery of software increment to the customer. As a first step, individual test cases to be automated were identified. This included identifying create, read, update and delete operations for each ERP module, for example, a sales order module. Test data addressing different test cases for a given operation was created carefully to ensure good quality test data. The scripts to setup and teardown test data were also created to infuse self-sufficiency in individual test cases. This test data was inserted directly into the SQL Server database using DbUnit library to save time. Otherwise, subsequent test cases had to wait for the preceding test case to finish execution and create test data. The test data independence was also essential for running automated tests in parallel.

Parallel automated testing was implemented by splitting sequential test cases into manageable self-sufficient test cases. In the non-parallel implementation of the automated testing tool, tests were executed in a hierarchical order: the child test cases were dependent on the parent test cases to create the test data. For example, to create a user in the system, an account must already be present. For this reason, the test case to create an account must run before creating a user. This also means that if the preceding test case fails, then all the subsequent test cases fail to run. This hierarchy of test cases was removed in parallel automated testing.

Measuring Testing Time Difference

The testing time spent on doing regression testing on web-based ERP application can be divided into three phases based on the use of manual and automated testing types. Every phase spans over a period of one year. First phase includes when there were no automated tests and testers at the company were executing the regression tests completely manually. It is dated from January 2014 to December 2014. Second phase includes the introduction of automated testing tool without the implementation of automated testing best practices. It dates from Jan 2015 to December 2015. The third phase starts when the automated testing best practices were implemented in existing automated regression testing tool. It spans from January 2016 to December 2016.

The time required for manual testing was recorded by conducting an interview with employees of the company who were involved with manual regression testing in phase 1. The time spent on manual testing was recalled by the interviewees and answered in the interviews. The interviewees were also part of the decision-making team and have good insight into the transition phase from manual testing to automated testing. Structured interview questions were asked with basic interview guideline to follow [Kvale, 1996]. Interviewees responded to interview guide (referred in Appendix E)

which was sent in advance to them via email. The language of the interviews was English.

The time expended on automated regression testing in second and third phases was collected from the Jenkins [Jenkins, 2011] server. Jenkins was used to execute automated regression tests at the end of every sprint cycle and it maintained the history of the all the executed automated tests with the record of consumed testing time over the period of past five years.

5. Results and Conclusion

This chapter discusses the results and answers the research questions brought forward in Chapter 4. It also concludes the thesis.

5.1. Selenium tool for testing SPA

According to hypothesis H1, it was assumed that Selenium could be used to automate testing of ReactJS based Single Page Application. In the literature review, it was mentioned that SPA uses JavaScript / AJAX calls to handle the routing at client-side instead of at the server and this is the root cause of many issues such as handling navigation, remembering navigation and history position, handling navigation cancellation and increasing performance issues. It is considered difficult to test SPA purely because unlike a normal page in which browser displays loading indicator, giving feedback to the user about the status of load progress, JavaScript client side routing require developers to use JavaScript-based indicators to update the load progress of a webpage on UI which is not very efficient. As discussed in Section 4.4.2, Selenium was able to automate testing of *TimeApp*, a ReactJS based Single Page Application. All of the above mentioned issues were handled successfully by Selenium testing tool.

5.2. Shorter testing time with automated testing

According to hypothesis H2, it was assumed that regression testing time of a web-based ERP application can further be shortened by incorporating best practices of automated testing in existing automated regression testing tool used in the company. The implementation of the best practices was discussed in Section 4.5.

In order to discuss the advantages of implementing automated testing best practices, the expended regression testing time to test a software increment in a sprint cycle before and after introducing automated testing was measured as mentioned in Section 4.5. The collected data is presented in Appendix F. The following Table 2 presents and compares the regression testing time measured in all the three phases discussed in Section 4.5.

Phase 1 **(Jan 2014 - Dec 2014)**	**Phase 2** **(Jan 2015 - Dec 2015)**	**Phase 3** **(Jan 2016 - Dec 2016)**
Complete Manual Testing	Automated Testing without Best Practices	Automated Testing with Best Practices
Number of testers doing testing: 4	Number of testers doing testing: 1	Number of testers doing testing: 1
Number of test cases: 50	Number of test cases: 50	Number of test cases: 50
Avg. testing time: 12 hours	Avg. testing time: 1.16 hours	Avg. testing time: 0.58 hours

Table 2: Testing time before and after automated testing.

In phase 1, four testers were engaged in performing regression testing. Automated testing was not introduced at this point which led to a substantial amount of time being consumed to perform manual testing. The number of test cases to test was fifty. The average of all the manual testing time spent in testing of individual sprints was twelve hours. In phase 2, automated testing tool was introduced the company. The significant change as compared to previous phase was in the reduction of number of testers from four to one and decreases in average amount of testing time. The testing time was reduced twelvefold. In phase 3, as assumed in hypothesis H2, the implementation of automated testing best practices further reduced the amount of testing time as compared to testing time in phase 2. The testing time was reduced to half of previous time.

In conclusion, manual testing causes a delay in application delivery time because manual testing is time-consuming. Automated testing, however, requires less human effort and less amount of testing testing. Automating the testing of *TimeApp* indicated that, among other testing tools, Selenium is definitely a good choice for testing SPA. Selenium is considered a great automated testing tool. It is well-suited for agile projects and is also effective as a web automated testing tool on ReactJS based Single Page Application. Findings from the thesis also conclude that use of automated testing best practices leads to achieving reduced regression testing time in a Scrum.

References

[Agile Manifesto, 2001] Agile Manifesto. http://agilemanifesto.org, 2016.

[Al-Hossan and Al-Mudimigh, 2011] Amel Al-Hossan and Abdullah S. Al-Mudimigh, Practical guidelines for successful ERP testing. *Journal of Theoretical & Applied Information Technology*. **27**(1), 2011, 11-18.

[Allamaraju et al., 2000] Subrahmanyam Allamaraju, *Professional Java Server Programming J2EE Edition.* Peer Information Inc., 2000.

[Angular, 2016] AngularJS. https://angularjs.org, 2016.

[Atlassian, 2016] Atlassian Software. https://www.atlassian.com, 2016.

[Al-Zain et al., 2012] Samer Al-Zain, Derar Eleyan and Joy Garfield, Automated User Interface Testing for Web Applications and TestComplete. In: *Proc. of the CUBE International Information Technology Conference*, 350-354.

[Bach, 1999] James Bach, Test Automation Snake Oil. In: *Proc. of the 14th International Conference and Exposition on Testing Computer Software*.

[Backbone, 2016] BackboneJS. https://backbonejs.org, 2016.

[Berner et al., 2005] Stefan Berner, Roland Weber, and Rudolf K. Keller, Observations and Lessons Learned from Automated Testing. In: *Proc. of the 27th International Conference on Software Engineering*, 571-579.

[Bertolino, 2007] Antonia Bertolino, Software testing research: Achievements, Challenges, Dreams. In: *Proc. of the Workshop on the Future of Software Engineering*, 85-103.

[Coded UI, 2010] Coded UI. https://msdn.microsoft.com/en-us/library/jj620891.aspx, 2016.

[Chow, 2016] Ming Chow. Web programming. Document Object Model (DOM). https://tuftsdev.github.io/WebProgramming/notes/dom.html, 2016.

[Chrome, 2008] Google Chrome, https://www.google.com/chrome/browser, 2016.

[ChromeDriver, 2013] Google ChromDriver, https://sites.google.com/a/chromium.org/chromedriver/downloads, 2016.

[Dallal, 2007] Jehad Al Dallal, Automation of object-oriented framework application testing, In: *Proc. of 5th IEEE GCC Conference and Exhibition*, 425-434.

[DbUnit, 2002] DbUnit, http://dbunit.sourceforge.net, 2016.

[Dobolyi et al., 2011] Kinga Dobolyi, Elizabeth Soechting, Westley Weimer. Automating regression testing using web-based application similarities. *International Journal on Software Tools for Technology Transfer*, **13** (2), 111-129.

[du Bousquet and Zuanon, 1999] L. du Bousquet and N. Zuanon, An Overview of Lutess: A Specification-based Tool for Testing Synchronous Software. In: *Proc. of the 14th Conference on Automated Software Engineering*, 208-215.

[Ebert, 2012] Christof Ebert, *Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing*, John Wiley & Sons, 2012.

[Fewster and Graham, 1999] Mark Fewster, Dorothy Graham, *Software Test Automation: Effective Use of Test Execution Tools*. Harlow: Addison-Wesley, 1999.

[Fox, 2009] Brian Fox, Testing Nexus with Selenium: A lesson in complex UI testing (Part 1). http://blog.sonatype.com/2009/09/testing-nexus-with-selenium-a-lesson-in-complex-ui-testing-part-1, 2016.

[HP Quick Test Pro, 2001] HP Quick Test Pro. https://saas.hpe.com/en-us/software/uft, 2016.

[Jenkins, 2011] Jenkins Continuous Integration Server. https://jenkins.io, 2016.

[Garrett, 2005] Jesse James Garret, Ajax: A New Approach to Web Applications. http://adaptivepath.org/ideas/ajax-new-approach-web-applications, 2016

[JQuery, 2016] JQuery, https://jquery.com, 2016.

[Karhu, 2009] Katja Karhu, Tiina Repo, Ossi Taipale, and Kari Smolander, Empirical Observations on Software Testing Automation. In: *Proc. of 2nd International Conference on Software Testing Verification and Validation*, 201-209.

[Kumar, 2012] Vivek Kumar, Comparison of Manual and Automation Testing. *International Journal of Research in Science and Technology*, **1**(5), 2012.

[Kvale, 1996] Steinar Kvale, *InterViews: An introduction to qualitative research writing*. Thousand Oaks: Sage Publications, 1996.

[Kaur and Kumari, 2011] Manjit Kaur and Raj Kumari. Comparative Study of Automated Testing Tools: TestComplete and QuickTest Pro. *International Journal of Computer Applications*, **24** (1), 2011, 1-3.

[Larman, 2003] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2003.

[László, 2010] Kottyán László. Application Development in Web Mapping. http://www.tankonyvtar.hu/en/tartalom/tamop425/0027_ADW1/ch01s02.html, 2016

[Leitner et al., 2007] Andreas Leitner, Ilinca Ciupa, Bertrand Meyer, and Mark Howard, Reconciling Manual and Automated Testing: the AutoTest Experience. In: *Proc. of the 40th Hawaii International Conference on Systems Science*, 261.

[Locke and Balaraman, 2012] Gareth Locke and Reghunath Balaraman. Test Automation at the Enterprise Level. https://www.infosys.com/IT-services/independent-validation-testing-services/white-papers/Documents/test-automation-enterprise-level.pdf, 2016.

[Luna, 2014] Eclipse Luna. https://projects.eclipse.org/releases/luna, 2016.

[Malekzadeh and Ainon, 2010] Mehdi Malekzadeh and Raja Noor Ainon, An Automatic Test Case Generator for Testing Safety-Critical Software Systems. In: *Proc. of the 2nd International Conference on Computer and Automation Engineering*, 136-167.

[Maven, 2004] Apache Maven. https://maven.apache.org, 2016.

[Mittal and Acharya, 2003] Naina Mittal and Ira Acharya, An Open Framework for Managed Regression Testing. In *Testing of Communicating Systems*, 265-278.

[Monier and El-mahdy, 2015] Mohamed Monier and Mahmoud Mohamed El-mahdy, Evaluation of automated web testing tools, *International Journal of Computer Applications Technology and Research.* **4**(5), 405-408.

[Motwani, 2010] Vivek Motwani, The When & How of Test Automation. https://www.infosys.com/IT-services/validation-solutions/white-papers/Documents/when-how-test-automation.pdf, 2016.

[Myers, 2004] Glenford J. Myers, *The Art of Software Testing*. Wiley, 2004.

[Nagarani et al., 2012] P. Nagarani and R. Venkata Ramana Chary, A tool based approach for automation of GUI applications. In: *Proc. of Third International Conference on Computing, Communication and Networking Technologies*, 201-202

[Optimus, 2015] Optimus Information. Best Practices for Test Automation. http://www.optimusinfo.com/best-practices-for-test-automation, 2016.

[O'reilly, 2005] Tim O'reilly, What is Web 2.0: Design patterns and business models for the next generation of software, http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html, 2016.

[Oscillot, 2016] Oscillot, Testing against React.js. https://www.reddit.com/r/selenium/comments/3py4u7/anybody_testing_against_reactjs_we_are_having, 2016.

[PlanMill, 2016] PlanMill Oy. https://www.planmill.com, 2016.

[Polo et al., 2007] Macario Polo, Sergio Tendero, and Mario Paittini, Integrating Techniques and Tools for Testing Automation, *Software Testing, Verification and Reliability*, **17**(1), 3-39.

[Polo et al., 2013] Macario Polo, Pedro Reales, Mario Piattini, and Christof Ebert, Test Automation. *IEEE software*, **30**(1), 84-89.

[React, 2016] React. https://facebook.github.io/react, 2016.

[Sand, 2005] Sand, What's my testing ROI?, http://robertvbinder.com/wp-content/uploads/sites/4/2011/06/Whats-My-Testing-ROI1.pdf, 2016.

[Selenium, 2004] Selenium, http://www.seleniumhq.org, 2016.

[SeleniumHQ, 2015] SeleniumHQ, http://docs.seleniumhq.org/docs/01_introducing_selenium.jsp, 2016.

[Selenium WebDriver, 2012]. Selenium WebDriver, http://www.seleniumhq.org/docs/03_webdriver.jsp#introducing-webdriver, 2016.

[Sharma, 2014] R. M. Sharma, Quantitative Analysis of Automation and Manual Testing. *International Journal of Engineering and Innovative Technology*, **4**(1).

[Silver, 2016] Adam Silver, The disadvantages of Single Page Application. http://adamsilver.io/articles/the-disadvantages-of-single-page-applications, 2016.

[Sites, 2016] Google Sites. Chrome Driver - WebDriver for Chrome, https://sites.google.com/a/chromium.org/chromedriver/getting-started, 2016.

[SmartBear, 2016] SmarBear Software, Automated Testing Best Practices, https://support.smartbear.com/articles/testcomplete/automated-testing-best-practices, 2016.

[SoapUI, 2005] SoapUI. https://www.soapui.org, 2016.

[Sommerville, 2010] Ian Sommerville, *Software Engineering*. Addison-Wesley, 2010.

[SQL Server, 2014] SQL Server. https://www.microsoft.com/en-US/download/details.aspx?id=42299, 2016.

[Schwaber and Beedle, 2001] Ken Schwaber and Mike Beedle, Agile Software Development with Scrum. Addison-Wesley, 2001.

[Stockdale, 2016] Dayana Stockdale. How to Develop an Automated Testing Strategy. https://testlio.com/blog/how-to-develop-an-automated-testing-strategy, 2016.

[Techopedia, 2016] Software Bug, https://www.techopedia.com/definition/24864/software-bug-, 2016.

[TestComplete, 1999] TestComplete. https://smartbear.com/product/testcomplete/overview, 2016.

[Tortoise, 2002] Tortoise SVN, https://tortoisesvn.net, 2016.

[WebDriver, 2016] WebDriver: Advanced Usage, http://docs.seleniumhq.org/docs/04_webdriver_advanced.jsp, 2016.

[WebFinance, 2016] WebFinance. Application Software, http://www.businessdictionary.com/definition/application-software.html#ixzz48LF9M17C, 2016.

[Windows 7, 2009] Microsoft Windows 7, https://www.microsoft.com/en-us/software-download/windows7, 2016.

[Wissink and Amaro, 2006] Tom Winssink and Carlos Amaro, Successful Test Automation for Software Maintenance. In: *Proc. of 22nd IEEE International Conference on Software Maintenance*, 265-266.

[W3C, 1994] World Wide Web Consortium (W3C). https://www.w3.org/Consortium, 2016.

Appendices

Appendix A: Code Snippet for Automating TimeApp

```
/**
 * Checks existence of a time report dialog form
 *
 * @param name Title of a time report dialog form
 */
public void timereport_dialog_form(String name) {
   String xpath_time_report_dialog = "//div[@class='time-report-form mui-dialog mui-dialog-window mui-is-shown']//h3[@class='mui-dialog-title' and text()='"
         + name + "']";

   if (isElementPresent(By.xpath(xpath_time_report_dialog))) {
      element(By.xpath(xpath_time_report_dialog));
   }
   else {
      Assert.fail("TIME APP: Time report dialog form doesn't exists!!");
   }
}

/**
 * Enters duration of a time report in hours and minutes fields of a time report dialog form
 *
 * @param duration Duration of a time report e.g. 2:00
 */
public void enter_duration(String duration) {
   String hours = duration.substring(0, duration.indexOf(":"));
   String minutes = duration.substring(duration.indexOf(":") + 1, duration.length());

   if (isElementPresent(By.id("hour-input"))) {
      WebElement hour_input = webDriver.findElement(By.id("hour-input"));
      $(hour_input).type(hours);
   }
   else {
```

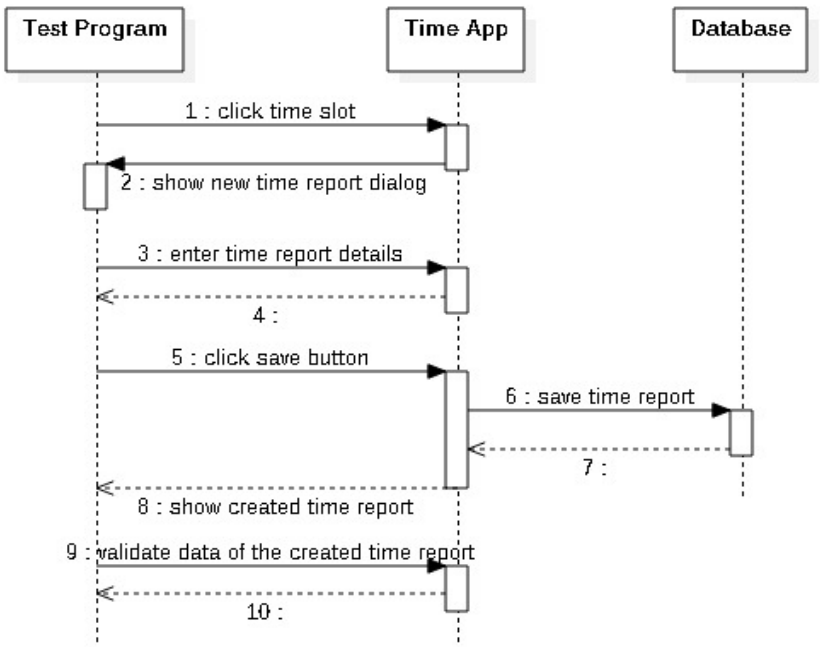
```
            Assert.fail("TIME APP: Hour duration textbox on time report dialog form doesn't exists!!");
        }

        if (isElementPresent(By.id("minute-input"))) {
            WebElement minute_input = webDriver.findElement(By.id("minute-input"));
            $(minute_input).type(minutes);
        }
        else {
            Assert.fail("TIME APP: Minute duration textbox on time report dialog form doesn't exists!!");
        }
    }

    /**
     * Clicks a button on a time report dialog form
     *
     * @param button_id Button to be clicked
     */
    public void click_button(String button_id) {
        if (button_id.equals("x")) {
            String xpath_button = "//button[@class='close-button md-close md-lg mui-icon-button mui-enhanced-button']";

            if (isElementPresent(By.xpath(xpath_button))) {
                element(By.xpath(xpath_button)).click();
            }
            else {
                Assert.fail("TIME APP: Button on time report dialog form doesn't exists!!");
            }
        }
        else {
            if (isElementPresent(By.id(button_id))) {
                webDriver.findElement(By.id(button_id)).click();
            }
        }
    }
```

Appendix B: Selenium function used to automate TimeApp

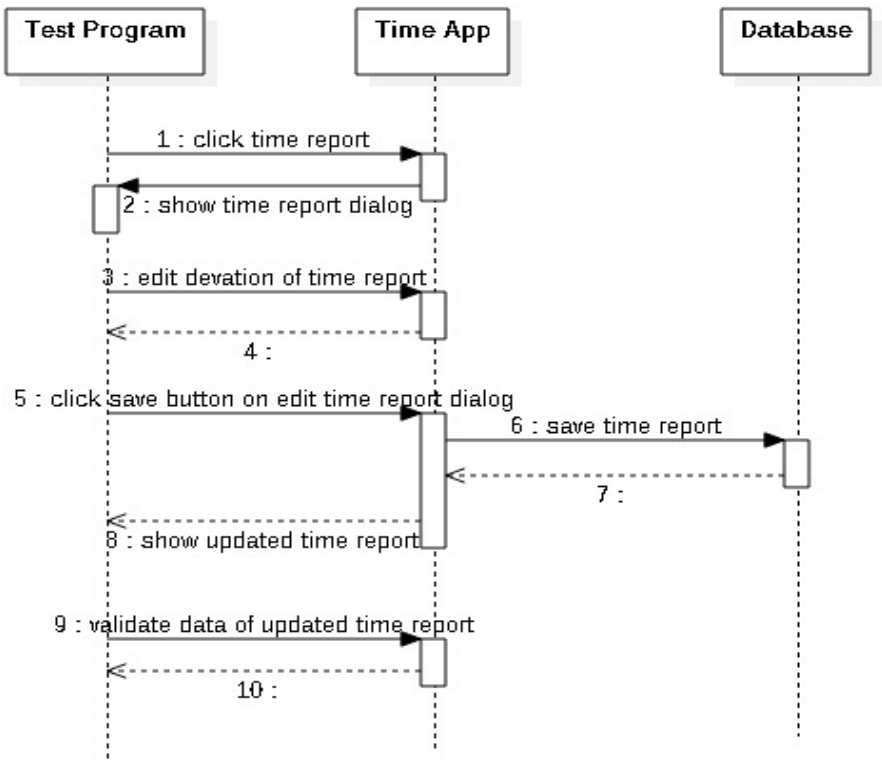
- click()
- clickAndHold()
- moveByOffset()
- findElement()
- findElements()
- moveToElement()
- sendKeys()
- getText()
- wait.until()
- release()
- perform()
- switchTo()
- frame()
- defaultContent()
- ExpectedConditions.visibilityOf()
- ExpectedConditions.presenceOfElementLocated()
- ExpectedConditions.elementToBeClickable()
- ExpectedConditions.visibilityOfElementLocated()
- ExpectedConditions.visibilityOfElementLocated()

Appendix C: Sequence Diagram of TimeApp Test Cases

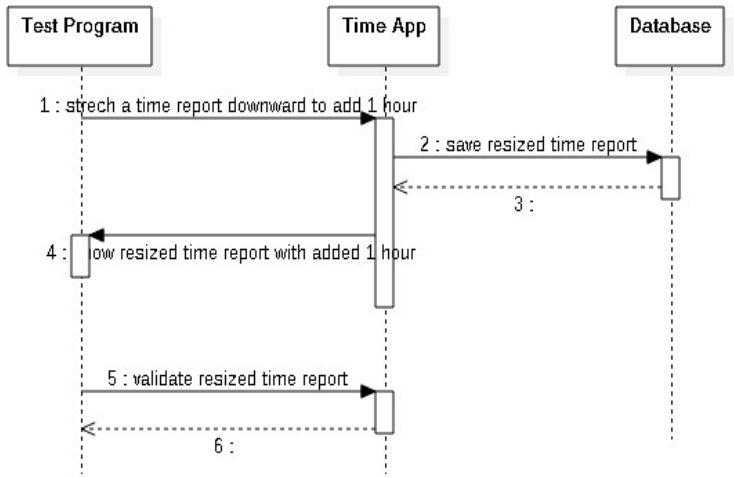
Create Time Report:



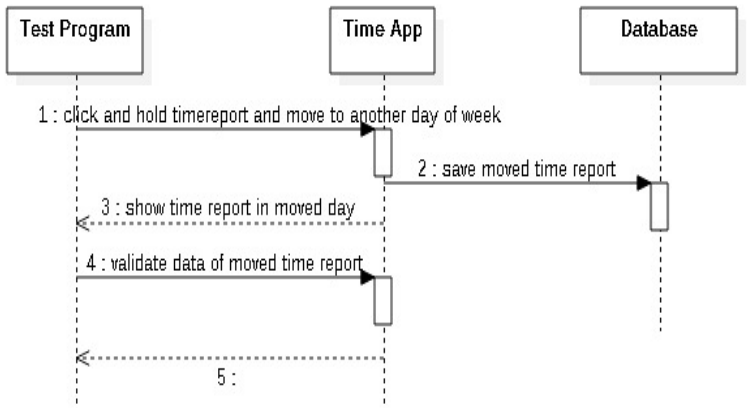
Update Time Report:



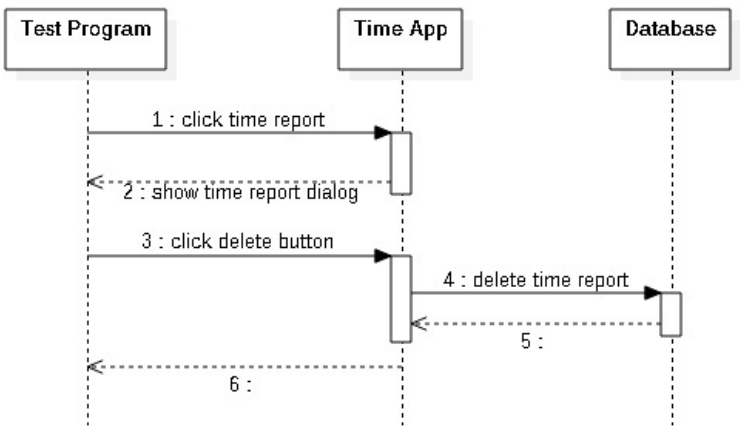
Resize Time Report:



Move Time Report:



Delete Time Report:



Appendix D: Hardware and Software Used

Hardware

An x64-based computer with Intel Core i5 CPU @ 2.40 GHz and 1GB of RAM was used for developing the automated testing tool.

Software

- **Windows OS:** Microsoft Windows [Windows 7, 2009] version 7 for 64-bit computers was running as an operating system on automated testing tool development computer.
- **Java:** Java Enterprise Edition [Allamaraju et al., 2000] version 7 was used to develop the automated testing tool.
- **Eclipse:** Luna version of Eclipse [Luna, 2014] was used as an Integrated Development Environment for developing automated testing tool.
- **Maven:** Apache Maven [Maven, 2004] version 3.3.3 was used as a build management tool to manage libraries used in the development. Maven build plugins (clean, compiler, surefire, failsafe) were also used to configure automated testing tool development project. The pom.xml file is the core of a Java project's configuration in Maven.
- **Selenium:** Selenium [Selenium, 2004] was used to automate testing of the web application.
- **Chrome Browser:** Automated tests were run in Google Chrome [Chrome, 2008] version 44 browser application.
- **ChromeDriver - WebDriver for Chrome:** ChromeDriver [ChromeDriver, 2013] version 2.19 was used to drive Chrome browser to run automated tests.
- **SQL Server:** Microsoft SQL Server [SQL Server, 2014] version 2014 was used as a database to manage and store test data.
- **DbUnit:** DbUnit [DbUnit, 2002] version 2.5.1 was used to inject dependency test data directly into the database. Using DbUnit saves time by directly injecting dependency data.
- **Serenity BDD:** Automated web tests support is strongly provided by Serenity [Serenity, 2011] using Selenium WebDriver. Serenity BDD was also used to produce illustrated and narrative reports of automated test results.
- **Jenkins:** Jenkins [Jenkins, 2011] was used to run automated tests.
- **Tortoise SVN:** Tortoise SVN [Tortoise, 2002] was used for versioning of software.

Appendix E: Interview Guide

This interview is about calculating the time taken to do manual regression testing as part of sprint testing of a web-based business application. This time was used to compare the amount of time spent on doing manual testing and automated testing.

The interview was conducted with three participants who used to do manual regression testing of the application in sprint cycles. The interview was held at the company premises.

Interview Questions:

Q1: How much time (in hours) did you spend in a sprint cycle to do regression testing manually before automated testing was introduced?

Q2: Did you have to repeat the manual regression tests? If yes, how often per sprint cycle do you have to redo regression tests manually?

Q3: What were the reasons to repeat manual regression tests?

Answers by Interviewees:

Interviewee A (Senior Software Engineer)

Q1: How much time (in hours) did you spend in a sprint cycle to do regression testing manually before automated testing was introduced?

A1: About 1 - 3 days (7 - 22.5 hour) per release depending on lately released new features that have to be taken into account in next regression tests.

Q2: Did you have to repeat the manual regression tests? If yes, how often per sprint cycle do you have to redo regression tests manually?

A2: Yes, quite often. 0 - 3 times depending on if previous new features that had to be taken into account in regression tests or not.

Q3: What were the reasons to repeat manual regression tests?

A3: Changes / additions to existing test plans – regression tests didn’t include the latest default features; then test plans needed to update first and then repeat the tests after updating.

Interviewee B (Senior Software Engineer)

Q1: How much time (in hours) did you spend in a sprint cycle to do regression testing manually before automated testing was introduced?

A1: If I remember correctly it took about 3-4 hours per person per release.

Q2: Did you have to repeat the manual regression tests? If yes, how often per sprint cycle do you have to redo regression tests manually?

A2: Sometimes, quite rarery. I can't remember exactly but I think most of the time regression tests were done only once.

Q3: What were the reasons to repeat manual regression tests?

A3: Some serious bug that was found (by regression test or otherwise) and fixed.

Interviewee C (Project Manager)

Q1: How much time (in hours) did you spend in a sprint cycle to do regression testing manually before automated testing was introduced?

A1: It's hard to tell an exact figure and it varies depending on release complexity and the evolution of our release process. Before we had Jenkins for releasing to production (4 years ago), we released changes every 3 months and we had around 5 testers, testing manually for about 5-6 hours the whole system + re-testing after fixes if bugs were found. After Jenkins was introduced, release cycles were shortened to every 3 weeks and we had around 3 testers doing testing for about 2-3 hours.

Q2: Did you have to repeat the manual regression tests? If yes, how often per sprint cycle do you have to redo regression tests manually?

A2: No, when bugs were detected, coders usually fixed it immediately and we only had to repeat the area of the fix.

Q3: What were the reasons to repeat manual regression tests?

A3: Only when bug fixes or major fixes were introduced.

Appendix F: Time taken to test Web-based ERP application

Before introducing automation testing – Year 2014

Testing weeks	**Testing time in hours**
Week 2	12.25
Week 5	11.75
Week 8	12.50
Week 10	11.25
Week 13	13
Week 15	11.75
Week 18	11.75
Week 20	12.75
Week 24	10.75
Week 26	11.25
Week 30	12.25
Week 32	13
Week 35	11.75
Week 37	12.50
Week 39	12.50
Week 42	11.75
Week 44	12
Week 46	12.25
Week 49	10.75
Week 52	11.75
Average testing time (round off)	**11.98 (12 hours)**

After introducing automated testing – Year 2015

Testing weeks	**Testing time in hours**
Week 3	1.17
Week 6	1.23
Week 8	1.22
Week 10	1.05
Week 12	1.17
Week 15	1.25
Week 18	1.22
Week 21	1.17
Week 23	1.12
Week 25	1.18
Week 28	1.13
Week 31	1.22
Week 33	1.15
Week 35	1.10
Week 37	1.17
Week 40	1.20
Week 42	1.12
Week 44	1.07
Week 46	1.17
Week 49	1.15
Week 51	1.25
Average testing time	**1.16**

After introducing automated testing – Year 2016

Testing weeks	**Testing time in hours**
Week 1	0.55
Week 3	0.62
Week 5	0.57
Week 7	0.58
Week 9	0.60
Week 11	0.55
Week 14	0.62
Week 16	0.53
Week 18	0.58
Week 21	0.57
Week 23	0.60
Week 26	0.55
Week 28	0.57
Week 30	0.60
Week 33	0.62
Week 35	0.55
Week 37	0.58
Week 40	0.53
Week 42	0.52
Week 45	0.60
Week 48	0.58
Week 50	0.53
Week 52	0.62

Average testing time	**0.57 hours**