Contents

1

Introduction

Computers and mobile devices are surrounding us like never before. Many people are steadily growing accustomed to being in control of what digital content they wish to consume along with when and how they wish to consume it. As a consequence of digital content being increasingly accessible, the importance of, and demand for, high-quality software has been equally accelerating. Enhanced pressure on software vendors to deliver frequent releases of high-quality software requires increased efficiency in every stage of the software development process.

A central element in this process is software testing. This plays a critical role in quality assurance and defect detection. It is important in order to ensure that the software behaves as expected and according to specifications.

1.1 Origin

The subsidiary company of the Lyse Group, Altibox, is a telecommunications service provider who offers a selection of services, including broadband, *Internet Protocol Television* (IPTV) and *Voice Over Internet Protocol*, (VOIP). Since its origin, the company has grown a great deal, and at a considerably higher pace than originally anticipated. Because the company simply could not keep up this rapid development, they neglected to apply a fair amount of well-established business practices in several different areas, including the field of software testing.

IPTV has long been one of Altibox' most prominent services, and has previously been available primarily through the use of TV decoders. In the later years, however, one of Altibox' areas of focus has been to expand the availability of their TV and

Video On Demand (VOD) content by applying the *Over-The-Top* (OTT) approach, which means that the content will be made accessible over the Internet. This has been done through the development of *TV Overalt*, which is available both as a web application for desktops and as mobile applications for portable devices. The web-based version of TV Overalt has been used as the study object in this project.

1.2 Motivation

Altibox has wished to incorporate test automation in their testing process for some time, but have failed to prioritize this in favor of other tasks. They have expressed that upon implementing test automation in their test process, the most important factors are efficiency and control. Test runs should be completed quickly, feedback should be given rapidly and the tests should be executed in a controlled environment.

This thesis presents [tool name]; a tool that will help Altibox incorporate test automation according to their needs. However, a significant objective in this project has been to create a comprehensive tool that can be applied as a testing platform for a broad range of web applications, and not just TV Overalt. A suggestion for future work described in Chapter **??** includes incorporating support for test automation of mobile applications to expand the usage of [tool name]even further.

1.3 Purpose

The aim of this project was to design and create a tool that would substantiate test automation with a distinguished ability to optimize the time of test executions as well as provide precise control and rapid feedback of test results. The tool should satisfy the following:

- Scheduling future runs of test cases for web applications should be supported.
- Tests should be able to be executed remotely in a distributed environment.
- An algorithm for finding the optimal test case schedule amongst all test cases scheduled at a specific time should be included.

- An interface for managing automated test cases should be included, in which uploading test scripts and managing test scripts and schedule future test runs should be supported. Logs from results of previous test runs should also be available here.
- Information about test scripts, test schedules and execution logs should be stored in a database.
- Upon a failed test, a defect should be automatically reported in the issue tracking system.
- A mobile application for receiving notifications upon a failed test should be created.

1.4 Outline

The remainder of this thesis is organized as follows:

Chapter 2 provides a theoretical body of knowledge by discussing some background information relevant to this thesis. Software testing, constraint programming and optimization along with some related work will be presented in this chapter.

Chapter 3 introduces the most essential tools and technology used in this project. The Python programming language, Selenium and Django will be presented here.

Chapter ?? presents the design and architecture of the [tool name].

Chapter ?? describes the implementation of the system. Some of the features will be explained in detail.

Chapter ?? discusses the results of this project.

Chapter ?? concludes this thesis and presents some suggestions for future work.

2

Background

Before moving onto the technical substance of this thesis, acquire a theoretical body of knowledge on the subject of this thesis will be beneficial. This chapter introduces software testing and explains a number of key concepts that are relevant to this thesis. Further, it presents constraint programming and optimization of test case scheduling as well as some previous work related to this thesis.

2.1 Software Testing

A fundamental understanding of software testing is advisable in order to fully benefit from reading this thesis. Consequently, this section will present an introduction to software testing. Since this is a large field, only the most relevant concepts will be presented.

Software testing is the process of evaluating the quality of the application, system or component in question with the intent of revealing any defects, faults or otherwise undesirable behavior of the system under test. It may involve any action oriented toward assessing the software with the goal of determining whether it meets the required results.

Seeing as software is developed by humans, and humans can make mistakes, it is sensible to presume that software may also have mistakes. The severity of a software defect can range from small and unimportant to largely expensive or dangerous. Releasing low-quality or faulty software may have critical consequences for businesses. Software testing is necessary to identify system defects and errors. Furthermore, it helps ensuring the quality of the product, and thus gaining customer

confidence in the product as well as in the company itself.

[Something about test levels, UI/GUI testing and the value of detecting bugs as early in the process as possible?]

[8] distinguishes *functional* and *non-functional* testing the following way:

Functional testing seeks to validate that the test object behaves as specified in the functional requirements, which describes what the system must be able to do. According to [5], characteristics of functional requirements are suitability, accuracy, interoperability, and security.

Non-functional testing is aimed at testing the object for its non-functional requirements, that is to say the attributes of the functional behavior or the attributes of the system as a whole. According to [5], reliability, usability, and efficiency are characteristics of non-functional requirements.

This thesis focuses on functional testing.

Further, [8] divides software testing methods into two categories; static and dynamic testing. Static testing is the testing of a component or system at specification or implementation level, and is conducted without executing the software, through forms of analysis such as document inspection.

Contrary to static testing, dynamic testing explicitly involves the execution of the test object. An advantage to this method is that the actual program is executed either in its real environment or in an environment close to the real environment. This means that in addition to testing the design and code, elements such as the compiler, the libraries, the operating system and so on are also tested [9]. Dynamic testing is the method addressed in this project.

This project is concentrates on *automated testing*, which is the practice of utilizing tools and writing scripts that conduct tests when executed. While automated tests may run quickly and effectively, can be cost-effective and are useful for repeating tests the same way repeatedly, the natural exploration that occurs when humans perform manual testing does not happen through executing automated test scripts. Writing and maintaining test scripts are time consuming, so it is important to only use resources on automating test cases of value. Tests and tasks that would be error prone if done manually are typical subjects for automation [6]. This includes regression testing, which is the testing of a previously tested object

after changes are made in the software or its environment, and is done to detect any new defects that has been introduced after the changes [8]. Test automation should not replace manual testing as a whole; the two should complement each other through a combination of adequate amounts of both manual and automated testing.

There are some pitfalls that one should be aware of upon entering the area of test automation. It is not uncommon that companies generally have unrealistic expectations of test automation, and expect that including test automation in a software project guarantees instant success, which is not the case. Another typical mistake is believing that test automation requires less time and resources than it actually does [6].

However, the benefits of applying test automation to software projects are many. When considering the benefits of automated testing, these are compared to the duration, effort, or coverage that would have alternatively been obtained through manual testing. While simultaneously saving time, reducing effort and providing better coverage and thus lower risk, additional benefits to choosing test automation may include better predictability of test execution time. Some types of testing are hard, impractical, or even impossible to conduct manually, at least not in a meaningful way. These types of testing include performance, stress, load, and reliability testing, which are all areas that can be tested with the right automation in place, and in turn reduce the risk [6].

2.2 Constrained Optimization

[Will write this section l8r]

2.3 Related Work

Some work has previously been done on the subject of test case scheduling and automation of UI tests for web applications. This section presents the scheduling method *TC-Sched* and the automated testing platform *Sauce Labs*.

2.3.1 TC-Sched

TC-Sched is a time-aware method for minimizing the execution time of test cases within a distributed system with resource constraints. The method was proposed by Mossige, Gotlieb, Meling, and Carlsson in [7]. The authors consider the following in their paper:

- A set of test cases $\mathcal{T} = \{t_1, \ldots, t_n\}$ along with a function $d : \mathcal{T} \longrightarrow \mathbb{N}$ that gives each test case a duration d_i
- A set of global resources $\mathcal{G} = \{g_1, \ldots, g_o\}$ along with a function $g : \mathcal{T} \longrightarrow \mathcal{G}$ that describes which resources are used by each test case
- A set of machines $\mathcal{M} = \{m_1, \ldots, m_m\}$ along with a function $f : \mathcal{T} \longrightarrow \mathcal{M}$ which assigns a subset on machines to each test case on which the test case can be executed

They further define the *optimal test case scheduling* (OTS) as the optimization problem $(\mathcal{T}, \mathcal{G}, \mathcal{M}, d, g, f)$ of finding an execution ordering and assignment of all test cases to machines, such that each test case is executed once, no global resource is used by two test cases at the same time, and the overall test execution time, T_t, is minimized. T_t is the time it takes to solve the scheduling problem, T_s, plus the time it takes to execute the schedule, $C*$, $T_t = T_s + C*$.

As established in the bullet list above, TC-Sched addresses situations in which some operations may require exclusive access to one or more *global resources*, such that only one test case can access each resource at a given time. To include the aspect of global resources in this project would have been redundant since there are no such global resources involved. Apart from that, TC-Sched is a closely related scheduling method that has been of inspiration. The implementation of the scheduling algorithm used in this project will be described in Chapter **??**.

2.3.2 Sauce Labs

Cloud testing services use cloud computing environments as a means of simulating real-life user traffic. Sauce Labs is a cloud-hosted platform for automated testing of web and mobile applications. One of the founders, Jason Huggins, were also the original creator of Selenium, which will be introduced in Section 3.2.

Sauce Labs bear some similarities and some dissimilarities with the design of [tool name]. They both support automated testing of web applications using Selenium and allow remote testing in distributed environments through the use of Selenium Grid (see Subsection 3.2.2). However, whereas Sauce Labs perform test execution solely on virtual machines that are created prior to each test run, and then destroyed immediately after, [tool name]requires the use of real, physical machines or devices, or customized virtual machines or devices running on said physical machines or devices, so that the user remains in complete control at all times. Additionally, Sauce Labs executes tests in their own data centers contained in an remote location, with their own environments, while [tool name]tests the test object locally and in its intended environment. Further, Sauce Labs do not automatically create issues in the issue tracker. Sauce Labs is a paid service for which customers pay to gain access to using a limited number of concurrent virtual machines for a limited amount of time each month. The optimization algorithm in this project also distinguishes the two projects even further.

3

Technology

A number of distinctive tools and technologies have been applied throughout the work on this project. This chapter briefly introduces the most essential of these tools and technologies.

3.1 The Python Programming Language

The Python programming language has largely been used in the conduction of this project. Python has efficient high-level data structures, and is known for being simple, yet powerful. Since Python programs are executed by an interpreter, it is considered an interpreted language [3].

Building this project on Python was a natural choice for several reasons. Python is compatible with Selenium, which is introduced in Section 3.2. Other reasons include offering a large set of libraries and having high-quality documentation. Additionally, Python supports multithreading and is suitable for running a program as a service. Details of the implementation will be discussed in Chapter **??**. Version 2.7.11 of Python has been used in this project.

3.2 Selenium

Selenium is an umbrella project for automation of web browsers. Several different tools and libraries are included in the framework, all with the common goal of supporting browser automation. Selenium can be used to automate browser tasks, but is primarily used for test automation. The project is released under the Apache 2.0 license and is thus free and open-sourced. Selenium is compatible with several

different programming languages, including Python. The Selenium tools used in this project will be introduced below.

3.2.1 Selenium WebDriver

Selenium WebDriver is a tool that helps with automation of tests for web applications. It interacts directly with the browser by sending calls using the native support for automation for each browser [2]. Selenium WebDriver runs on Windows, Linux and Macintosh, and supports most conventional web browsers, including Firefox, Chrome, Internet Explorer, Opera and Safari.

The WebDriver creates a new instance of the specified web browser specified. It fetches a stated web page, and then locates UI elements. This can be done in multiple ways, for instance by specifying the id, class name, link text or the XPath of each elements. When an element is located, the WebDriver can be used to perform an action on the element, such as clicking a button, selecting a radio button or populating a text field.

Source Code Listing 1: Selenium WebDriver Example

```
1  from selenium import webdriver
2
3  # Create a new instance of the Firefox WebDriver
4  driverdriver = webdriver.Firefox()
5
6  # Go to the Altibox TV Overalt start page
7  driver.get("https://tvoveraltstg.altibox.no/")
8
9  # Locate and click the Login button
10 login_button = driver.find_element_by_class_name('btn-login')
11 login_button.click()
```

Source Code Listing 1 shows a simple example of how Selenium WebDriver can be used with Python to open the Mozilla Firefox Web browser, navigate to the Altibox TV Overalt start page, locate the *Login* button, and click it. Refer to Chapter **??** for further details about implementation of test cases using the Selenium WebDriver.

3.2.2 Selenium Grid

Selenium Grid is a tool for distributing tests on multiple physical or virtual machines, and thus allowing for parallel execution. This opens up for running tests in a distributed test execution environment with support for a mixture of operating systems and browsers.

Reasons for wanting to incorporate Selenium Grid include being able to run tests against multiple browsers, browser versions and browsers running on different operating systems, and to reduce the execution time of the tests. In practice, a grid is made up of one Selenium Grid Server, or a hub, and one or more slave machines, or nodes. The nodes use Selenium WebDriver to communicate with the hub through a JSON wire protocol [**?**].

Selenium 2.0 and Selenium Grid 2.0 were used in this project.

3.3 Django

Django is a high-level web development framework that is implemented in the Python programming language. It encourages rapid development and enables efficiently maintainable web applications of high quality. The framework is a free and open-sourced project maintained by the non-profit organization *the Django Software Foundation*, who describe the framework as fast, secure and scalable [1]. Comparable to Selenium, Django is also essentially a collection of Python libraries [4]. The Django labraries can be imported and used to implement web applications. Optionally, custom HTML, CSS and JavaScript code can be applied in addition to the Python code.

Models generally play a central role in web applications built on this framework. Django models are sources of information that contain fields and behaviors of the data being stored. The models define the database layout, and each model typically maps to an individual table in the database, in which instances of the model are later stored [**?**]. Source Code Listing 2 shows a simple example of how a model is created in Django.

Source Code Listing 2: Django Model Example

```
1  from django.db import models
2
3  class Schedule(models.Model):
4      title      = models.CharField(max_length=80)
5      start_time = models.DateTimeField()
```

Aside from allowing rapid progression of development, one of the main arguments for choosing Django over building web applications from scratch or using other frameworks is its powerful administrator site. An administrator site was precisely what was required to build the dashboard of [tool name].

and an administrator site were the only reason. Another contributing factor was consistency and the ability to communicate seamlessly with the remaining parts of the system. Details concerning the implementation of the web interface created as part of this project will be explained in Chapter **??**. The Django version used in this project is version 1.9.

References

[1] *Django: The Web framework for perfectionists with Deadlines.*

[2] *Selenium WebDriver — Selenium Documentation.*

[3] *A Byte of Python.* 2004.

[4] A. Holovaty and J. Kaplan-Moss. *The Definitive Guide to Django: Web Development Done Right.* Apress, 2007.

[5] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality.* ISO/IEC, 2001.

[6] J. L. Mitchell and R. Black. *Advanced Software Testing - Vol. 3, 2nd Edition: Guide to the ISTQB Advanced Certification as an Advanced Technical Test Analyst.* Rocky Nook, 2015.

[7] M. Mossige, A. Gotlieb, H. Meling, and M. Carlsson. Optimal Test Execution Scheduling on Multiple Machines with Resource Constraints. Technical report, 2016.

[8] A. Spillner, T. Linz, and H. Schaefer. *Software Testing Foundations, 3rd Edition.* Rocky Nook, 2011.

[9] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach.* Morgan Kaufmann, 2006.