Design and Development of an Efficient Automation Framework for Verification Cycle in Telehealth Systems

A Dissertation submitted in fulfilment of the requirements for the Degree
of

MASTER OF ENGINEERING
in
Electronic Instrumentation and Control

Submitted by

S. Srivathsan
801751005

Under the Guidance of

Dr. Sunil K. Singla
Associate Professor, EIED

Mr. Kabir Bedi
Associate Manager, SGTC

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

2019

Electrical and Instrumentation Engineering Department
Thapar Institute of Engineering & Technology, Patiala
(Declared as Deemed-to-be-University u/s 3 of the UGC Act., 1956)
Post Bag No. 32, Patiala – 147004
Punjab (India)

DECLARATION

I hereby certify that the work which is presented in dissertation entitled, "Design and Development of an Efficient Automation Framework for Verification Cycle in Telehealth Systems", in partial fulfilment of the requirements for the award of the degree of Master of Engineering in Electronics and Instrumentation Control, submitted to Electrical & Instrumentation Engineering Department of Thapar Institute of Engineering & Technology (Deemed to be University) is as authentic record of my own work carried under the supervision of Mr. Kabir Bedi and Dr. Sunil K. Singla It refers others researcher's work which are duly listed in the reference section. The matter contained in this dissertation has not been submitted, neither in part nor in full to any other degree to any other university or institute except as reported in text and references.

Place: Punjab
Date:

(S. Srivathsan)
Roll No.: 801751005

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Date: 23/8/19

(Sunil K. Singla)
Associate Professor
Electrical & Instrumentation Engineering Department
Thapar Institute of Engineering & Technology, Patiala

Date: 23 AUG 2019

(Kabir Bedi)
Associate Manager
Stryker Global Technology Corporation
Gurugram, Haryana

ACKNOWLEDGEMENT

Firstly, I would like to thank my mentor Mr Kabir Bedi Associate Manager, Stryker Global Technology Corporation for providing me with the opportunity to work on this project and for giving me enough technical support to accomplish it.

I would also like to extend my gratitude to Dr Sunil Kumar Singla, Associate Professor, Electrical & Instrumentation Engineering Department Thapar Institute of Engineering & Technology, Patiala for his continuous guidance and motivation.

Special thanks to all my colleagues and friends for their numerous insights when I was looking for answers and keeping me in good spirit during the whole course of this dissertation.

Lastly, a big thanks to my family for providing me with great moral support and to the institutions that gave me a chance to collaborate with highly esteemed people.

S. Srivathsan
(801751005)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

EHR – Electronic Health Record

EMR – Electronic Medical Record

MRI – Magnetic Resonance Imaging

TestNG – Test Next Generation

API – Application Program Interface

IDE – Integrated Development Environment

POI – Poor Obfuscation Implementation

ABSTRACT

Widespread use of telecommunication services in the healthcare industry has created a boom in "telehealth" technologies. Hospitals have adopted various technologies such as monitoring patient health remotely, transferring X-Ray and MRI scan images digitally, and interacting with patients through live video conferencing. It has prompted industries creating healthcare technologies to provide hospitals with better products at a fast pace. Before releasing the products into the market, thorough testing is essential as these products affect the lives of patients. However, each verification cycle for a product slows down the pace of product deliverance; hence, efficient methods are required to resolve this problem. The purpose of this dissertation is to enhance the testing process for the telehealth products by introducing automation techniques into the verification cycle. In this dissertation, a framework for automation testing has been built by studying the testing processes and various necessary tools required to achieve the same. The architecture is then implemented, and the results have been discussed. Furthermore, additional changes that can be made in future to improve this framework has also been described towards the end of this dissertation.

ABOUT THE COMPANY

Stryker Corporation was found by Dr Homer Stryker in 1941 and was previously known as The Orthopedic Frame Company. It started in the basement of Borgess Hospital, Kalamazoo, Michigan, when Dr Stryker began his medical practice as an orthopedist[1].

He found that most of the medical devices used in orthopaedics were less efficient than their potential, in both caregiving efficiency and meeting patient requirements. He started replaced existing non-efficient devices by his designs. One of his first designs was a mobile hospital bed called the Wedge Turning Frame, containing frame that pivoted from side to side. This turning frame became famous as the "Stryker Frame" in the industry as allowed doctors the freedom to position patients as required without moving the patients. With the success of his invention, Stryker formed the Orthopedic Frame Company(later known as Stryker Corporation), to build and sell beds.

The company continued to innovate and began to manufacture a more diverse line of medical devices such as a device to remove plaster casts, that soon become standard throughout the orthopaedics field. Until L. Lee Stryker died in 1977, Stryker was a family-owned and operated company. He was replaced by John W. Brown who immediately acquired Osteonics Corp which was a three-year-old company that made hip implants for joint replacement surgeries. Other initiatives undertaken under Brown's direction was the Stryker's entry into the arthroscope market. Brown stumbled upon some orthopaedics instruments in a medical conference that allowed surgeons to make a tiny incision in the skin and inspect soft tissue by inserting a narrow, optical tube into the knee that helped diagnose the need of surgery for repair. Earlier, surgeons required to make a six-inch diagnostic cut in order to inspect ligaments and muscles that were invisible to X-Rays. Apart from helping in the diagnosis of knee injuries, arthroscopic tools could also be used during operations to repair problems.

Brown discovered arthroscopic devices quite late when several other companies had already entered the field. Since the use case for such tools was extremely high, Stryker proceeded with its product development and worked on driving existing products to perfection in its technology. The company first created its own three-inch-long, micro camera, that transmitted images from inside the body to a display monitor in the operating theatre. It considerably increased the ease of

surgeons and also allowed operations to be videotaped to prevent malpractices. Arthroscopic equipment became very popular in sports medicine due to many athletes preferring the procedure for its ease and comfort.

Between 1970 and 1980, healthcare costs had increased tremendously in the United States due to the passage of the Medicaid and Medicare bills in the mid-1960s. It triggered a social and governmental backlash. To curb exorbitant spending under the Medicaid and Medicare plans, the federal government devised a new payment system for hospital patients on Medicare. In this system, reimbursement to the hospitals was provided based on the type of diagnosis rather than the length of stay of a patient in the hospital or the amount of care given. It was called the Prospective Payment System and was designed to reduce unnecessary treatments and hospitalisations.

The Prospective Payment System made it difficult for both hospitals and patients to claim the medical benefits since many treatments and procedures that previously needed hospital admission now became outpatient procedures. With the reduction in hospital admissions and patient stays, healthcare providers turned to various ways to reduce hospital costs. As a result of low hospital admissions, fewer hospital beds were needed, thus affecting the Stryker's primary source of revenue. To balance this slowdown, Brown encouraged to explore new product areas that were less affected by healthcare cost controls.

One such area was biotechnology. In 1985, Stryker collaborated in a research program with Creative BioMolecules Inc. with the purpose to develop an implant that utilised an osteogenic protein called OP-1. It occurred naturally in humans and helped to promote natural bone growth and healing. Stryker was able to use OP-1 and was able to prove that it was able to accelerate the formation of new bone when implanted previously not healing the bony area.

After a year, Stryker added specialised in endoscopic systems with the acquisition of Syn-Optics Inc. Endoscopic systems included: medical video cameras, light sources, powered instruments, and disposable materials used in minimally invasive surgical procedures augmenting its operations in the arthroscopic optics field. The company could now expand devices earlier used only in joint surgery to surgeries in the abdominal area, ears, chest and stomach, thus creating a widespread increase in the popularity for these tools.

By the end of 1991, as the market for endoscopic tools enjoyed a substantial boom in its usage, sales of Stryker's endoscopy division grew more than one and half times of its starting revenue. Not only in endoscopy, but Stryker also enjoyed immense success in arthroscopic tools, powered surgical instruments, hospital emergency room beds and orthopaedic implants. With many inhouse traditional and more invasive procedures becoming outpatient procedures, Endoscopic devices quickly replaced a wide range of diagnostic and surgical applications.

As the company's product became widely diverse, Brown made a firm decision to shift Stryker corporation from its traditional management style of holding all divisions as a single unit. He decentralised the company by distributing it into several fully autonomous operating divisions. By segmenting the company into various divisions, each division now became responsible for its own goals, manufacturing operations, and its sales and marketing executives. It benefitted the sales team as each section of sales was now focused only on a minor set of products rather than having to sell the whole portfolio of Stryker. Currently, Stryker works broadly in three business segments: Orthopedics, MedSurg (Medical and Surgical), and Neurotechnology & Spine.

Orthopaedics products mainly deal with the implants used in hip and knee joints replacements. This division also provides equipments to treat fractures and joint replacements for shoulders, ankle and writs.

Product portfolio under MedSurg is broadly classified into three categories: 1.) Endoscopy and communications systems[Endoscopy], 2.) Surgical equipment and surgical navigation systems [Instruments] 3.) Patient handling and emergency medical equipment [Medical].

Stryker Neurotechnology and Spine products include a variety of neurosurgical/neurovascular devices that assist in minimally invasive endovascular techniques and traditional brain and open skull base surgical procedures. Lastly, spinal implant products for cervical and thoracolumbar used in deformity and degenerative therapies are developed under Stryker.

CHAPTER - 1 INTRODUCTION

1.1 ENDOSCOPY: AN OVERVIEW

Endoscopy is a medical procedure that helps a doctor to observe the internal state of the human body. In this procedure, the doctor can get a direct visual inspection of any interior part of the body such as tissues, blood vessels or organs, by using an optical viewing instrument called endoscope which is introduced into the patient via a natural orifice or a tiny surgical incision. The word originates from two Greek words: '*Endo' and 'scope or skopos'* which means to look within. Historically, endoscopy was used to examine stomach, oesophagus and colon by inserting the endoscope through mouth or anus. With the onset of small and powerful cameras, doctors now use endoscopy to diagnose different parts of the bodies, including throat, ear, nose, urinary tract and joints. Most endoscopes consist of tube-like protrusion fitted in front of a tiny camera with a powerful light to illuminate the internal parts. Several types of endoscopes exist that vary mainly in their length and flexibility. Their usage is determined by the doctor based on which part of the body needs to be seen and the vantage point to view. For example, if a doctor wants to examine knee joint for a patient, a rigid scope would be used while a flexible scope would be used to observe the colon. Figure 1 shows a few examples of scopes used in endoscopic procedures.

Figure 1: a) Rigid scopes b) Flexible scope

Endoscopy has been used to observe various internal body parts of a human. Based on which body part is under observation, the endoscopic procedure is segregated into various categories. Table 1 shows the list of categories with their description as identified by the American Cancer Society (ACS).

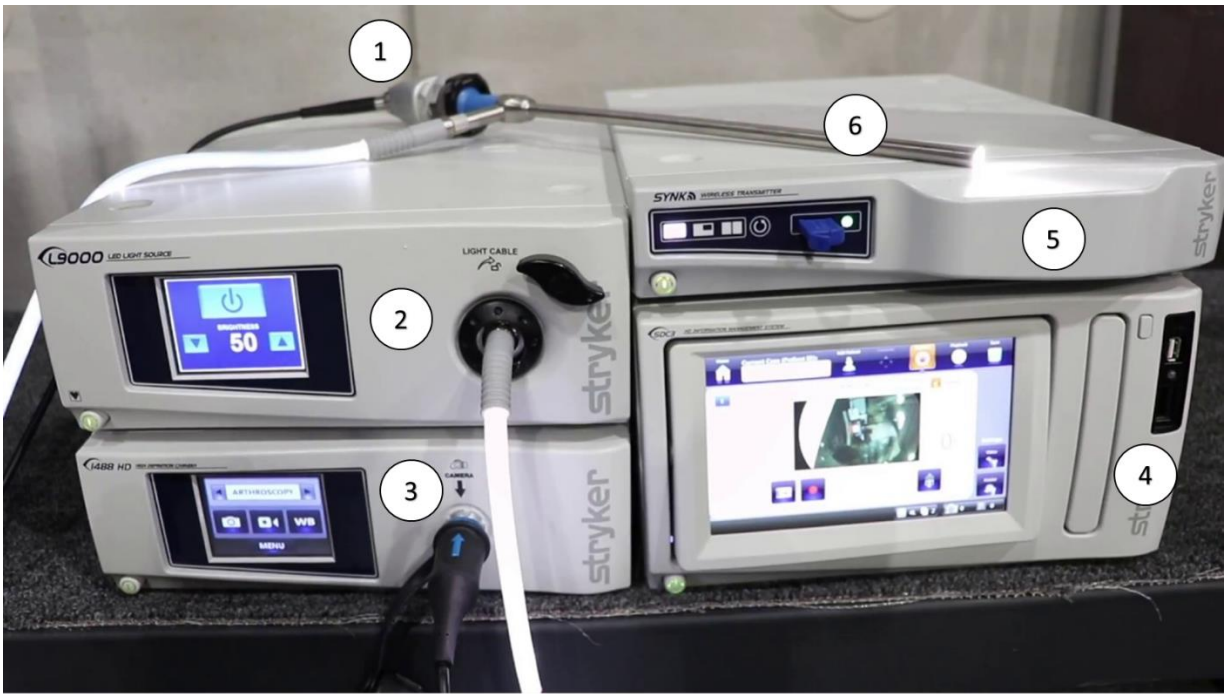
Table 1: Types of Endoscopy

Type	Body Part Examined	Where scope is inserted	Doctors
Arthroscopy	Joints	In an incision near the affected joint	Orthopaedic surgeon
Bronchoscopy	Lungs	In the nose or mouth	Thoracic surgeon
Colonoscopy	Colon	In the anus	Gastroenterologist or proctologist
Cystoscopy	Bladder	In the urethra	Urologist
Enteroscopy	Small intestine	In the mouth or anus	Gastroenterologist
Hysteroscopy	Inside of the uterus	In the vagina	Gynaecologists
Laparoscopy	Abdominal or pelvic area	In a small incision near the abdomen	Various types of surgeons
Laryngoscopy	Larynx	In the mouth or nostril	Otolaryngologist
Sigmoidoscopy	Rectum	In the anus	Gastroenterologist
Thoracoscopy	In between the lungs and the chest wall	In an incision on the chest	Thoracic surgeon
Ureteroscopy	Ureter	In the urethra	Urologist

Source: https://www.cancer.net/navigating-cancer-care/diagnosing-cancer/tests-and-procedures/types-endoscopy

Apart from helping surgeons and physicians to observe internal body parts, recent advancements in endoscopy have improved the efficiency and ease of various surgical procedures. Procedures such as cholecystectomy (removal of gall bladder), appendectomy (removal of the appendix), and hysterectomy (removal of the uterus) have become minimally invasive. Doctors now perform these procedures by cutting a tiny incision on the appropriate area of the body rather than placing a large cut revealing all the internal organs and tissues. It provides added benefits to the patients as they lose less blood during the procedure and can recover much faster than in any other standard surgery. Hospitals encourage doctors to perform surgeries assisted with endoscopes rather than the

standard procedures as they provide a healthy profit margin to the organisation. A typical setup used during an endoscopic surgery is shown in Figure 2.



1	Camera
2	Light Source
3	Camera Controller

4	Stryker Digital Capture
5	Stryker Synk
6	Camera Scope

Figure 2. Apparatus set up for endoscopic procedure

The most primary element for an endoscopic setup is the camera. These camera systems are used to produce pictures and videos in high definition of the internal human anatomy during surgical endoscopic procedures. The cameras are usually sensitive to both the visible and infrared spectrum and are connected to a camera controller also called a camera control unit (CCU) using an integral cable. The images and videos are transferred from the examination point to the camera head via a variety of rigid and flexible scopes The CCU helps adjust the camera presets according to the procedure and can control capture of video and still images in addition to the camera head. Along with the camera head and the scope, a LED light source is connected via optical fibre to illuminate the internal organs and tissues inside the body. The light source provides real-time endoscopic visibility and near-infrared fluorescence imaging. It enables surgeons to perform minimally invasive surgery using standard endoscopic visible light as well as visual assessment of vessels, blood flow, related tissue perfusion and biliary anatomy near-infrared imaging. Monitor displays are used to project the images and videos that are processed through the camera system for the

surgeon to view and navigate the procedure accordingly. These displays are integrated with the camera via a Synk platform which enables surgeons to replace wired cabling with smooth wireless connections. Lastly, the images and videos captured are managed using a Digital Capture device that helps to store the case related media files in any desired destination such as USB drive, CD-drive, printers. These digital captures prove useful when surgeons want to share the surgery details with their patients or their colleagues. By being able to store the data related to the surgery, the surgeon can discuss any complexity present and can also teach his fellow students based on the media captured. Certain digital captures are also integrated with hospital EMRs which gives them the additional capability to pull the patient information and thus minimise any errors encountered during manual insertion of patient details before the beginning of the surgical procedure.

1.2 CHOLECYSTECTOMY: AN APPLICATION OF ENDOSCOPY

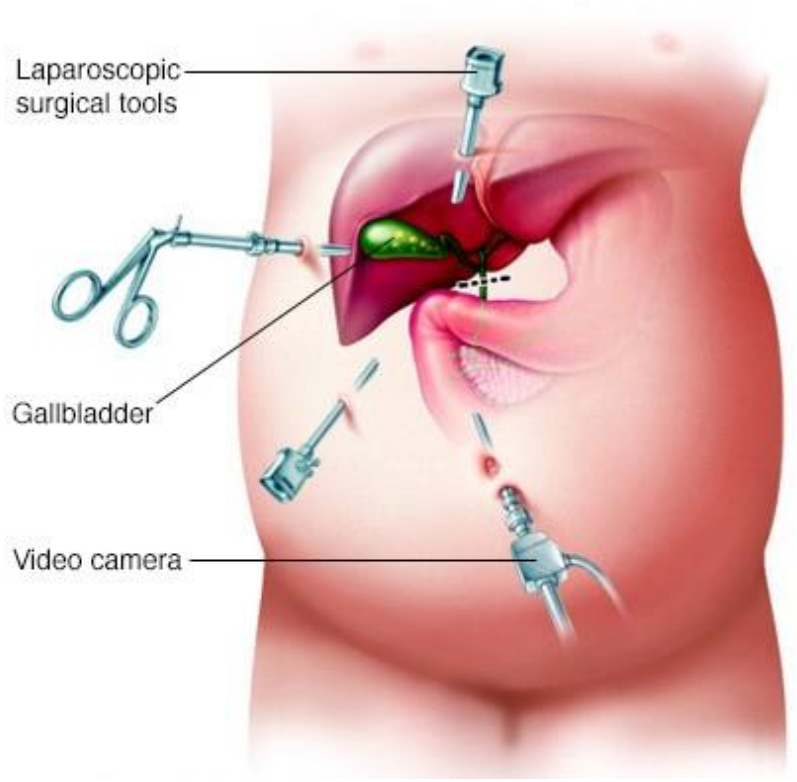


Figure 3: Minimally invasive Cholecystectomy

Cholecystectomy is a surgical procedure that is conducted when the surgeon recommends the removal of the gallbladder, which is a small pouch-like organ located below the liver which stores bile juice secreted by the liver. This bile juice is squeezed into the intestines through ducts which

are then used to digest food. Sometimes small calcified deposits called gall stones are formed inside this gall bladder that creates a blockage in the bile ducts connecting the gallbladder and the intestines. It causes extreme discomfort and interferes with digestion. Gallbladder is entirely removed to rectify the problem in many cases. Once the removal is finalised, the doctor then assesses the medical condition and overall state of health of the patient to determine if laparoscopic procedure could be performed on the patient. A laparoscope is a modified version of endoscope made especially for operating near the abdominal area. It contains a narrow tube along with a light source and a small video camera. With the laparoscope, the surgeon can operate on the patient by making one or more tiny incisions. Once the camera is inserted into the incision, the surgeon views the images and videos on display mounted near the table. It enables the surgeon to explore and examine the interior of the abdomen and presents the surgeon with greater detail and clarity than the human eye. However, it is important to understand that during the procedure, the surgical team is always prepared for an open procedure if the situation demands a more direct approach. Figure 3 illustrates the setup for cholecystectomy on the patient and Figure 4 shows the external arrangement of camera and visual systems.

Figure 4 Laparoscopic surgery set up

On the day of the operation, the patient is asked to change into a surgical gown. The patient then receives a sedative by mouth and an intravenous line before being transferred to the operating table. In the operating room, the anesthesiologist sits beside the head of the patient to administer

general anaesthesia. Antiseptic solution is applied to the skin around the area where the incisions will be done. A sterile drape is then placed around the operative site after the anaesthetic has kicked in and an incision is made above the umbilicus. In this orifice, a hollow needle is inserted through the abdominal wall, and the abdomen is inflated with carbon dioxide. Inflation of the abdomen using carbon dioxide expands the abdomen away from the internal organs, giving the surgeon a better view. A small port is created for the laparoscope, and three more incisions are made. Once in place, the laparoscope allows the surgeon to locate both the liver and gallbladder.

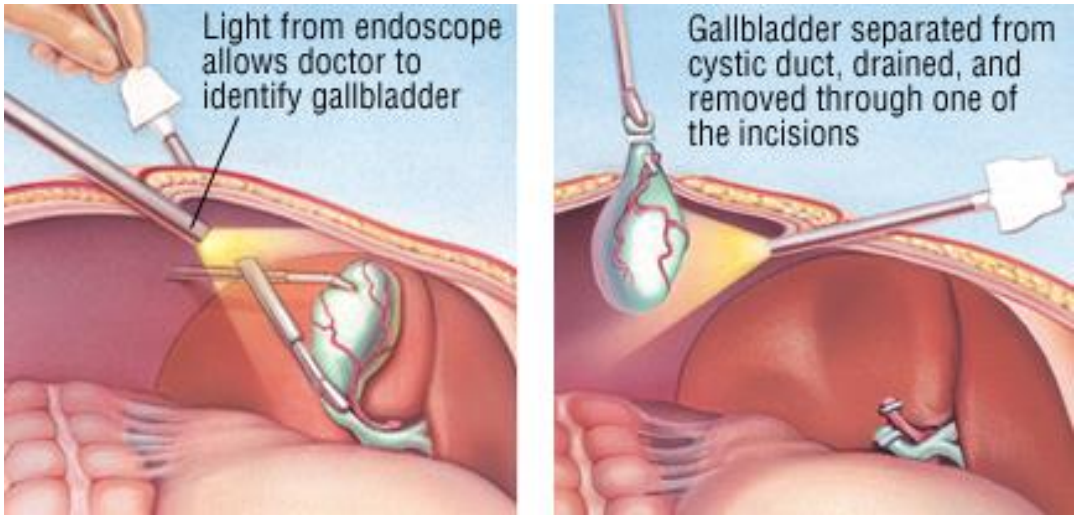


Figure 5: Final stages of cholecystectomy

The surgeon then removes the tissue between the gall bladder and liver to expose the cystic duct and artery. The surgical team clamps both the duct and the artery using clips to prepare the gallbladder for removal. Finally, remaining tissue connecting the liver and gallbladder is cut, and the gallbladder is moved into the laparoscopic working port before removing it from the body. All of the instruments are taken out, carbon dioxide is allowed to escape, and tissues are sewn together with sutures or staples to complete the surgery. Figure 5 illustrates the final stages of cholecystectomy done through laparoscopic procedure.

1.3 TELEHEALTH IN ENDOSCOPY

Endoscopic procedures have become immensely popular because of their efficiency in diagnosis and fast recovery for the patients. The diagnostics and surgeries performed using endoscopy are in general outpatient procedures where the patient can go home with a minimal recovery period at the hospital. Advancements have been made to improve and support such procedures due to their proven success. The first idea was to utilise the monitor display present to exhibit the video feeds from the endoscopic camera. Since there were multiple displays mounted near the patient table to enable all the surgeons and nurses to view the inner contents of the patient, one of the displays could be momentarily used for additional purposes. Large routers were placed within the operating rooms (OR), and these routers were connected to the hospital internal networks. It was the beginning of telehealth in endoscopy where live surgeries could be displayed from one OR to another. Surgeons could now present their operation to a nearby location in real-time. This technology was soon expanded to provide live video conferencing facilities allowing surgeons to gather intraoperative consultations and educate students in an auditorium.

Other telehealth technologies such as store and forward telecommunication technique also started to grow. In this technique, a digital capture device was placed in between the camera and the monitor such that the videos and images were passing through this capture device before reaching the display monitors. Doctors now could capture videos and images during the surgery and store it in this capture device. The captured content was then transferred to an intermediate machine that would then forward to a wide variety of destinations. The destinations consisted of Hospital EMRs, picture archiving and communication system (PACS), and other media management applications. The capture device provided additional advantages by providing bidirectional communication between the Hospital EMR. This feature was particularly useful during the surgery when patient details such as allergies, age and other personal data can be directly extracted from the EMR system. Additionally, the surgeon can also review images and videos from any previous surgery conducted on the patient to take critical lifesaving decisions.

The media management application incorporated other aspects of telehealth. This application allowed surgeons to review the surgeries conducted in the past and advise the patients accordingly. In educational, medical institutions, this feature proved highly successful as surgeons could now use the stored images and videos for teaching purposes as well. By integrating this application

with mobile devices, the necessary access to a computing system reduced altogether. Figure 6 shows an example of telehealth integrated OR.

Figure 6: Telehealth integrated OR

Stryker Communications is a subdivision of Stryker Endoscopy division that is focused on optimising patient care environments by developing systems that use telehealth technologies and their integration with EHR. Within the various telehealth products, software products present an excellent prospect for automation testing since the inputs and the outputs of a software program are precise and predictable. As a result, these products can be tested without manual intervention by creating a framework that can input the required data into the software and record the corresponding output data. Once the output data is available, any member from the testing team can analyse the results obtained. By carefully implementing such a framework, the overall testing process can be significantly optimised.

Software products selected as a part of this thesis include two web-based applications that are developed under Stryker Communications. The purpose of this thesis is to identify the appropriate

tools to automate the testing process for the web-based applications and in doing so, optimise any existing architecture of given testing framework.

1.4 LITERATURE SURVEY

A famous observation by G. E. Moore in his publication [2] that "the number of transistors on integrated circuits doubles approximately every two years" has proven to be quite accurate. This trend has been consistently observed even after 50 years since it was first published. As a result, ICs have become smaller, more powerful and much cheaper, leading to rapid development in technology. This development has enabled many people to innovate and apply technology to a plethora of industries and services. Today industries ranging from manufacturing to education all extensively use some form of technology for better efficiency. Out of the many industries, the telecommunications industry has been a significant beneficiary of this development. Large bulky telephones have been replaced by small hand-held mobile phone that can be carried anywhere. With the advent of the internet, telecommunication has now become a global necessity for any industry to thrive in the modern world. People are more connected and have access to immense amounts of data making processes more straightforward and more transparent.

Medical industry has recognised the potential of telecommunications in providing better healthcare to its customers. As a result, technologies such as telemedicine and telehealth are being used in healthcare. Telemedicine is the use of telecommunications and information technology to deliver timely health care from physicians to patients present in geographically different locations to improve patient care and management [3] while telehealth is often used to combine a broader application of information technologies such as transmission of still images, videoconferencing, e-health portals for both patient and physicians, remote health monitoring, distance medical education and other applications that are used to support healthcare services [4].

Instead of telemedicine, telehealth seems more popular as a term for depicting remote delivery of healthcare services. Telemedicine signifies that remote health care delivery is entirely between physicians and patients. However, in modern-day healthcare, physicians are working more as members of broader teams of health care practitioners and much less as independent individuals. Thus, using terms that highlight the work of one undermines the importance of an overall team working together seems to be an injustice towards the whole team. As telemedicine has ventured into new-fashioned growth areas, such as home health care services and sharing health information

to patients and practitioners, health care organisations have started using the word telehealth instead of telemedicine to make it more egalitarian [5].

Taipale et al. [6] explores the state of test automation in software testing organisations by collecting data from 41 organisations that develop professional software for industries in telecommunication domains. Results showed that test automation was viewed as beneficial but was not utilised widely in the companies. Quality improvement, execution of more tests in less time and easy reuse of test architecture were some main benefits of test automation. Costs associated with developing test automation in dynamic, customised environments was a difficulty.

Di Lucca et al. [7] presents the main differences between traditional and web-based applications. The paper further describes different types of testing carried on a web-based application targeted at the functionality of the application. It formed the basis for determining which type of testing would be performed on the web-applications. Also, test protocols from Stryker helped in further narrowing down to the exact test steps that would be followed while testing.

Singh et al. [8] and Sharma et al. [9] provide a comprehensive analysis on various automation tools available. Both the papers present a tabular view of the analysis and provide insight into the capabilities of these tools. Kaur et al. [10] provides a more granular comparison between the most popular tools widely used for web-automation testing. With further inputs from Devi et al. [11], Jagannatha et al. [12] and Ranstrom et al. [13] on automation tools combined with the business requirements of the project, selenium was found to be the most suitable for automation testing.

Architecture of the testing framework is inspired by Wang et al. [14]. In this publication, authors have designed an automatic software testing framework for web applications based on the Selenium and JMeter. To avoid setting up this framework each time and to make it system independent, a build tool was introduced in Kumar et al. [15]. For extensive projects, management of element addresses becomes essential. If the element addresses are spread out on different modules, refactoring the element name or address becomes inefficient and redundant.

To avoid such chaos, Gojare et al. [16] proposed a separate component dedicated to maintaining locations and names of all objects used. A better approach for storing object elements is discussed in Loetta et al. [17] where a Page Object Model is proposed in which all the details of a webpage

are combined in a single page object. Further, the efficiency of using a page object model is supported by results from the study performed as a part of the publication.

1.5 OBJECTIVES

In the present world, development in technology is moving at a very rapid speed. Better products supersede technologies in months after their arrival. To keep up with the market needs and consumer satisfaction, regular updates and innovation to existing products has become a necessity. With each update, the product must be tested to ensure proper functioning. The importance of extensive testing increases manifolds in the healthcare industry as the products affect the life of patient, physician and nurses. Every iteration of the product update consumes precious verification time which can cost the company and delay product release. Thus, reducing verification time can undoubtedly boost the efficiency of product completion. The following objectives with their reasons have been devised to achieve the same:

I. To study the testing process undertaken for the products and identifying areas that can be replaced by automation. Each product undergoes different levels of testing, and each process cannot be automated. Extracting the right test case to automate can enhance the overall efficiency of testing.

II. To analyse various tools available for automation testing and selecting the appropriate tools according to the business needs. With various tools available to accomplish automation, the right amount of research is necessary to select the most feasible one.

III. To build a robust and maintainable testing framework. With each update to the product, additional test cases must be added to existing test cases. The framework should comfortably accommodate such changes and must not affect existing test cases.

IV. To collate the results obtained from automation testing and presenting a perusable report detailing the execution of each test step. So that the reports can be shared among various stakeholders.

CHAPTER - 2 TOOLS AND METHODOLOGY

2.1 BACKGROUND

The process of automation testing involves numerous steps besides executing the source code for the test cases. It begins with an understanding of user needs and drafting them into software-based requirements. It is followed by product development, verification and validation. This whole cycle is carried out in different ways depending upon the model of software development chosen for the project. This chapter is intended to provide an overall description of the testing process and toolsets that were used for the completion of a product release.

As mentioned in the overview of the introduction, web-based applications were selected for automation testing. Initially, working of the applications was studied, and their testing processes were understood. During this stage, it was found that since the test cases were created without considering automation, several test cases were could not be automated. These test cases involved non-web-based test steps such as executing command-line instructions which were out of scope. Hence automatable test cases were identified and segregated as a pre-requisite to creating the framework.

2.2 SOFTWARE TOOLS

2.2.1 Selenium

Selenium is the backbone of the automation framework. It is an open-source tool containing a set of three components: Selenium IDE, Selenium Grid and Selenium 3(previously known as Selenium Remote Control and WebDriver) which is used to automate web browser actions. This capability of selenium has found great usage in testing of web-based applications. Additionally, it can interact with all major browsers and can be scripted in many major programming languages.

2.2.2 Programming Language and Environment

Selenium has great support to many scripting languages, Python and Java being the most prominent. Java was chosen for programming since it provided with some additional advantages such as better base framework for automation, good build integration tools and better industry support. For assistance with code maintenance and tool integration, community edition of IntelliJ IDEA was used as an IDE.

2.2.3 TestNG

TestNG is a testing framework mainly developed for Java and is used to provide a basic layout to a testing framework. It provides various features such as annotating certain functions which can be called before or after each test case, data-driven testing, running multiple instances of the same test and many others.

2.2.4 Maven

Maven is a tool used to manage a software project. It handles major aspects related to software project like managing dependencies and building the source code. With the help of the project can be deployed on any machine without worrying about environment setup. Major advantage of using maven is the central repository which has a collection of commonly used libraries.

2.2.5 Apache POI

Excel files were used to provide data to the test cases. It ensured that any change in input data could be made without affecting the existing source code for the test case. For interacting with excel files using java code Apache POI was used. It is an interface that allows a user to read, write and modify various Microsoft documents which includes MS Word, XML and MS PowerPoint.

2.2.6 Extent Reports

General reporting provided by TestNG does not contain stepwise details for each test case. However, incorporating Extent Reports into the automation framework, we can create a test report where each test step is precisely defined. Figure 1a and 1b show visual representation of both the test reports, respectively. A clear difference is visible in the amount of details displayed in both types of reports.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Default suite						
Default test	1	1	1	127		

Class	Method	Start	Time (ms)
Default suite			
Default test — failed			
VerifyReportTest	testSimulation00	1463169177671	41
Default test — skipped			
VerifyReportTest	testSimulation000	1463169177747	0
Default test — passed			
VerifyReportTest	testSimulation0	1463169177726	9

Figure 7a: TestNG Report

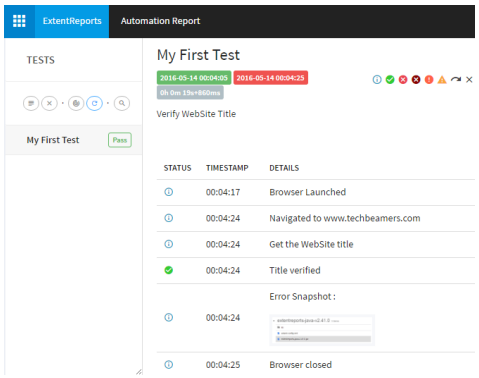


Figure 7b: Extent Report

3.3 AUXILIARY TOOLS

3.3.1 Agile PLM

It is a tool used for product life cycle management. All the documents and product-related details are stored in this tool so that every team member involved with the product can have access to any information about it. It helps in driving effective collaboration among all members by improving transparency between customers, suppliers and product team.

3.3.2 Git

While working on the script for automation, multiple developers may be involved each working on a different feature. A version control system becomes necessary to keep track of code modifications performed by each developer in the source code. Git is a tool that provides users with a distributed version control system. It helps each developer understand the changes made to the source code by others.

3.3.3 Bitbucket

Due to the involvement of multiple developers in source code creation, a central repository is required to store the basic source code. Such a central repository is made available by Bitbucket. It also supports version control systems like git, making it easier for developers to collaborate. Any changes made by the developer to source code is reviewed by other developers and then merged with the central repository.

3.3.4 JIRA

When testing is in progress, deficiency in the proper functioning of the product is a mere possibility. To track defects and to manage tasks for the product, an issue tracking system called Jira is used. Jira provides users with a comprehensive environment to describe the issue in good detail and track the progress of the issue reported.

3.2 SOFTWARE DEVELOPMENT MODELS

Two of the popular software development models were used in the projects considered for automation testing:

3.2.1 Waterfall

It is a linear model where testing of the product is started after the product is completely developed. This model requires a complete and clear understanding of the product requirements since major

changes to the product cannot be made once the development phase is completed. Once development is complete, the product is thoroughly tested, deficiency in the product's intended working is recorded and finally released.

3.2.2 Agile

This model of software development involves both development and testing working together simultaneously in an iterative manner. For each iteration, either new features are developed, or existing features are modified, and testing is performed on the product. The product is demonstrated to the stakeholders after a few iterations of development and testing. During each demonstration, the functionality of the product is showcased, and feedback is gathered. Each iteration aims to provide the stakeholders with a working product with limited features until the final release. Unlike the waterfall model, changes in the product requirement can be easily incorporated into subsequent iterations. As a result, stakeholders can use the product and provide feedback and help in enhancing the product quality before the final release.

2.3 FRAMEWORK ARCHITECTURE

The framework was built using the toolsets mentioned above, and the structure was designed, as shown in Figure 8. The centre of the structure is the main script written in the Java programming language that contains the exact steps written in the test cases. The script is supported by a parameter file which has a list of all basic parameters like type of browser to be used and web address of the application. Web locators and functions related to a web page are categorised into separate classes as per the page object model. Separate classes are written for utilising the Apache POI functionality to interact with the excels files for data input and output. The execution of test cases is controlled by "testng.xml" which is a part of TestNG. Using this XML test cases can be grouped and can be run in any specific order. Reporting of the test results is done through Extent Reports, and this interface is executed using TestNG after a set of test cases are completed. This whole structure is hosted on top of a build tool maven which ensures that the dependencies to run the script are present in the local system. Maven uses "pom.xml" file to accomplish this task. This file contains the list of all dependencies required to run the script successfully. If the dependencies are not available, maven downloads it and then continues with the execution by initiating "testng.xml". Finally, the Selenium web driver is used to interact with the web application and duly execute the test cases under consideration.

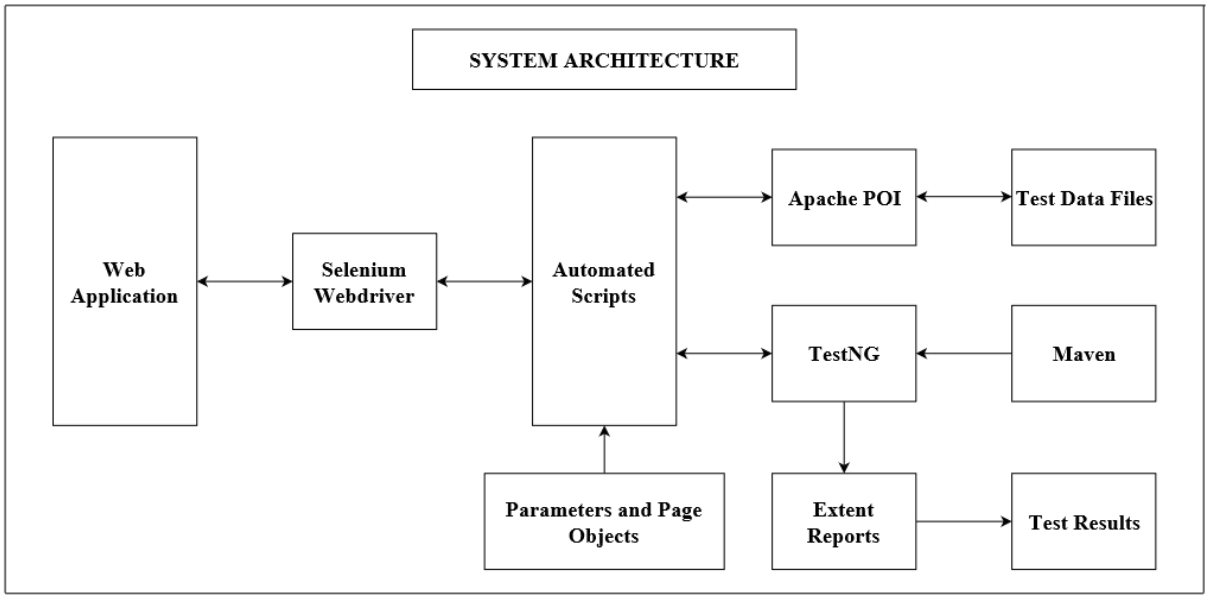


Figure 8: Architecture of the framework

2.4 FRAMEWORK WORKFLOW

Figure 9 shows a flowchart that depicts the clear workflow of the whole system. Each step in the flowchart is explained in detail below:

Step 1 – **Building and compiling the source code using Maven**

For the automation code to start execution, all the classes, libraries and JAR files used by the source code have to be collected and maintained in a known local location. These files are collectively called dependencies as the main program depends upon them for proper execution. Each dependency is recognised by a group id, artifact id and version number that are maintained online in remote repositories. Maven uses a pom.xml file to identify all the dependencies that are required. Once a dependency is declared in pom.xml file, maven searches in its local repository for its presence. If a matching dependency with the same name and version is not found, maven searches online for the exact content and downloads it into the local system. After updating the local repository, maven compiles the source code and builds it using plugins.

2.4 FRAMEWORK WORKFLOW

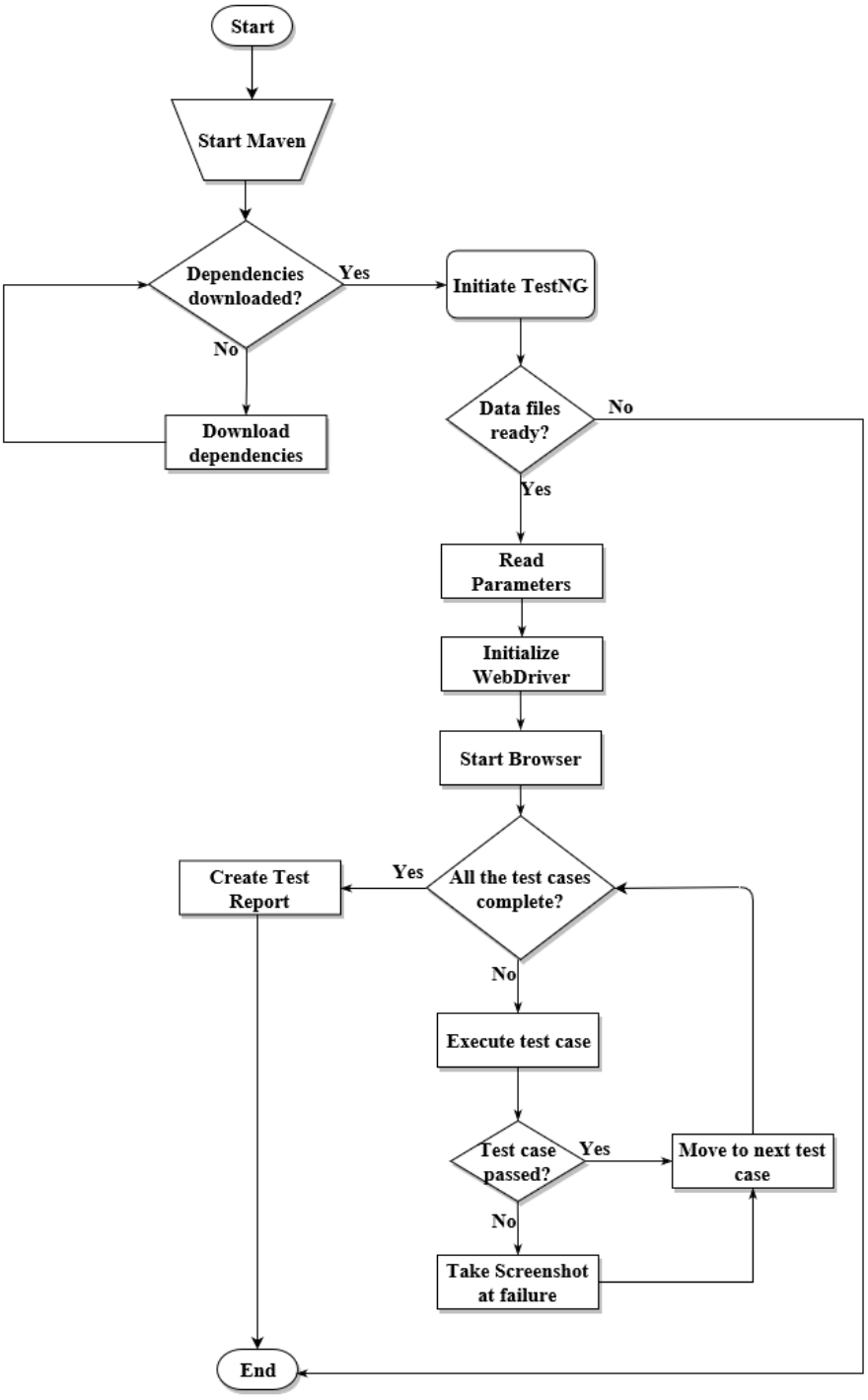


Figure 9: Workflow of the framework

Plugins form an integral part of maven that provides the user with the capability to choose their own framework through for compilation and building the source code. An example of maven dependency and plugin is shown below.

```
<dependency>
<groupId> org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>3.6.0</version>
</dependency>
```

```
<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.7.0</version>
</plugin>
```

Step 2 – **Initiation of TestNG**

Using the plugin, maven transfers the execution command to TestNG. TestNG consists of various annotations that help in organising, controlling and managing all the test cases for a test application. The execution starts with the parsing of testing.xml files that contains the list of test cases to be run in the current iteration. Each test case is represented by an individual function/method within a class that is identified by a @Test annotation All the functions that are to be executed are mentioned inside the testng.xml file within <test>…</test> tag. Only the methods mentioned within the tags are finally executed. Other annotations include @BeforeSuite, @BeforeTest, @AfterTest, @BeforeMethod, @AfterMethod and @DataProvider. With the use of these annotations, all trivial necessary checklist before and after the execution of test cases can be performed without manually calling the checklist functions each time explicitly. @DataProvider is one of the most useful annotations that is used to provide the necessary data to the current test case. By utilising @DataProvider, it is possible to test the same functions with different parameters repeatedly. In the given architecture, DataProvider gathers the necessary parameter values from the excel files.

Step 3 – **Parameter Initialization and Data Validation**

After identifying the set of test cases to be executed from the testing.xml file, the Base class is initialised. This class is responsible for reading the *data.properties* file to extract the necessary parameters essential for the execution. The properties file contains vital information such as the preferred web browser, URL for the homepage, location of web driver and other constant

parameters. The program then checks the existence of the input data using the functions provided by this Apache POI API for reading the data from the excel spreadsheets and extracts the exact parameters required for the test case. If the excel file is missing the program terminates the execution for the current test case.

Step 4 – **Starting the Web browser and Selenium Web driver**

Based on the web browser mentioned in the data.properties file, the specific web browser is opened. A web driver is associated with the opened web browser session. If a new window is to be required, then a new web driver is required to be initialised. The system uses Selenium Web Driver API to interact with Application Under Test. This API provides functions to locate the web elements on the application page and perform necessary actions with the elements.

Step 5 – **Executing the test cases**

The first action after starting the web browser is to navigate to the URL of the AUT. Test case is then executed based on the precise steps mentioned in the test protocol document. An example of a test case presented in a test protocol is shown below. In the protocol, each test case consists of eight columns. Two of the columns: Procedure Steps and Expected Results are the ones majorly needed during scripting. Each test step is verified using an '*assert'* statement that ensures that the outcome to each of the test step is accurate before moving further with the next test step. A test case is considered to be pass only if all the test steps are successfully executed. An example of a test case is shown in Table 2.

Table 2: Test case example

Test ID	SRS Id	Status (P/F)	Defect ID	Title	Procedure Steps	Expected Result	Actual Result
TC.01	SRS.01	N/A	N/A	Verify that the login functionality for valid and invalid users	**Pre-Conditions:** a). Two users exist : User A with valid credentials and User B with invalid credentials. **Steps:** 1. Navigate to Homepage 2. Click on SignIn link 3. Enter the user credentials and press on the Login button. 4. Verify that valid user is successfully logged in, and invalid user cannot log in.	1. User should be navigated to Homepage successfully 2. User should be able to click on SignIn link and should be navigated to login page 3. User should be able to enter user credentials and press on Login button successfully. 4. Valid user should be successfully logged in, and invalid user should not be logged in.	N/A

Test ID	SRS Id	Status (P/F)	Defect ID	Title	Procedure Steps	Expected Result	Actual Result
				works correctly	4 a. Verify that for User A signoff link is available 4 b. Verify that for User B Error message is displayed.	4 a. Signoff link should successfully appear for User A. 4 b. Error message should be successfully displayed for User B.	

If the expected results do not match the actual results during execution, the framework captures a screenshot of the web-browser session when the discrepancy was encountered. This screenshot is later used to analyse the reason for the failure. The failure is then recorded with detailed steps in defect management tool, and the corresponding defect id is updated in the test report.

Step 6 – **Report Generation**

A final HTML report is generated using Extent reports library after execution of all the test cases is complete. The report contains details of every test case and the overall statistics. It also records the time taken by each test case and presents a discernible view of the whole execution process. Extent report library utilises the ITestListener interface present in the TestNG framework to extract information about each test method.

CHAPTER - 3 APPLICATION OF THE FRAMEWORK

3.1 OVERVIEW

In the previous chapter, the structure of the whole automation testing framework was discussed. This architecture was applied towards the testing of web-applications that were used as a part of telehealth technologies. Due to proprietary concerns, the actual code written for testing cannot be disclosed. For a better understanding, a web-application *'AlturoJ'* is used to demonstrate the application of the framework. AlturoJ is a sample banking web application created under an Apache 2.0 license that acts as a platform for demonstrating issues related to app security. This site can be accessed using the URL: http://altoromutual.com:8080. The following sections elucidate the code structure and its working using an example test case.

3.2 CODE STRUCTURE

The code structure used can be broken down into four major parts: Fundamental classes, Utility classes, Functional classes and configuration files. Three classes are segregated into different packages while the configuration files are spread over the root directory and the resources package. Figure 10 shows a graphical representation of the structure used. Refer to Appendices A to D for detailed source code

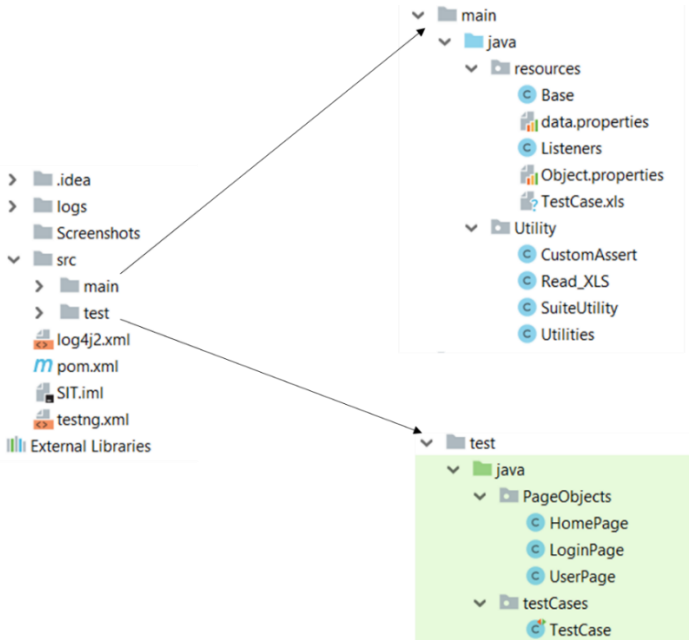


Figure 10: Graphical representation of code structure

Fundamental Classes

Base and Listeners are the two fundamental classes used. The base class contains code that would be used by all the test cases and is essential to initialise the web driver that will execute the browser actions. The listeners class inherits all properties of the base class and is used to keep track of the status of test cases and helps in generation of report. This class mainly listens to each test case and passes the results to extent report API for presentation. These two classes are stored with the resource package.

Utility Classes

These are a set of four classes that contain functions general functions which can be used by test cases as needed. SuiteUtility and Read_XLS are used together to extract data values from excel files and provide input to the DataProvider annotation. CustomAssert class contains various types of assertions that are used to verify the test steps of the test case. If an assertion fails for a particular test step, then the whole test case is marked as a failure in the final report. Lastly, the Utilities class contains methods that represent a minor action to be performed such as clicking on a web link in a web browser and pausing execution of the script for a defined period.

Functional Classes

These are classes that are specific to a web application. Each class in the PageObjects package represent a different webpage of the application. They contain the necessary user interacting functions related to that particular webpage. For example, HomePage class has *clickOnSignIn()* that is used to click on Sign In link present on the homepage of the website. The actual test cases are written inside testCases package. Each class within this package may contain multiple test cases based on the similarity or reduction in complexity.

Configuration files

The configuration files include *pom.xml, testing.xml* and two java properties files, namely *data* and *objects*. The *pom.xml* contains a list of dependencies and plugins required for code execution and *testing.xml* dictates that which test cases are to be executed. The *Object.properties* stores all the web-element locators to be used in the project and *data.properties* file defines some basic parameters that include web-browser name, path location of web-drivers and URL of the website under test. Refer Appendix D for details on the configuration files.

3.3 TEST CASE EXECUTION

Section 1: Declaration

```
public class TestCase extends Listeners

private CustomAssert CustomAssert = new CustomAssert();
Read_XLS TestCase = new Read_XLS(System.getProperty("user.dir")+
"\\src\\main\\java\\resources\\TestCase.xls");
Read_XLS FilePath = TestCase;
public HomePage Homepage = new HomePage();
public LoginPage Loginpage = new LoginPage();
public UserPage Userpage = new UserPage();
```

The execution of the test case begins with the declaration of variables and class objects. Any function to be performed on a webpage is carried out using the methods defined in the corresponding page's class.

Section 2: The Test Case

```
@Test(dataProvider = "verifyTestLogin", priority = 1)
 public void verifyTestLogin(String TestName, String TestDescription, String UserID,
                             String Password, String UserName,String ExpectedResult)
 {
     try
     {
         //Pre Condition
         //Two users exist: User A with valid credentials and User B with invalid
         credentials
         test.log(Status.INFO,"Testcase : "+TestName+" started : "+TestDescription);
         log.info("Testcase : "+TestName+" started: "+TestDescription);

         //1. Navigate to Homepage
         driver.get(prop.getProperty("url"));
         CustomAssert.assertTrue(Homepage.isOnlineBankingLinkPresent(), "Navigated to
         Home Page");

         //2. Click on SignIn link
         Homepage.clickOnSignIn();
         CustomAssert.assertTrue(Loginpage.isLoginButtonPresent(), "Navigated to login
         page ");

         //3. Enter the user credentials and press on Login button
         Loginpage.setLogin(UserID);
         Loginpage.setPassword(Password);
         Loginpage.clickOnLoginButton();

         //4. Verify that valid user is successfully logged in and invalid user cannot
         login
         //a. Verify that For User A signoff link is available
         //b. Verify that For User B Error message is displayed
         if (ExpectedResult.contains("Success"))
         {
             CustomAssert.assertContains(Userpage.getUserName(),UserName, "Is Login
             successful for active user
             " + UserName+"-");
             Userpage.clickOnSignOff();
             CustomAssert.assertTrue(Homepage.isOnlineBankingLinkPresent(), "Logout is
```

```
                successful for user
                "+UserName+"-");
            }
            else
            {
            CustomAssert.assertTrue(Loginpage.getErrorMessage().contains(ExpectedResult),
            "Is Login unsuccessful for inactive user " + UserName+"-");
                test.log(Status.PASS, "Skipping Logout as Login was unsuccessful");
                log.info("Skipping Logout as Login was unsuccessful");
            }
            test.log(Status.INFO,"Testcase : "+TestName+" completed : "+TestDescription);
            log.info("Testcase : "+TestName+" completed: "+TestDescription);
        }catch (Exception e)
        {
            test.log(Status.ERROR, "Exception  " + e);
            throw e;
        }

    }
```

After declaration, the control passes over to the @Test that gathers the parameter values from @DataProvider and calls *verifyTestLogin.* Each step of the test case is executed according to steps mentioned in Table 2 and then verified by using CustomAssert class. CustomAssert class utilises the functions in java Assert class and adds some description to each assertion present. For example, in step 2, the SignIn link is clicked. It is then verified using CustomAssert by looking at the presence of Login Button. It is based upon the logic that if the Sign In link was successfully clicked then the web page would be navigated to Sign In page and the login button would be available. Try and catch statements are used to tackle unexpected exceptions.

Section 3: Auxiliary Functions

```
    @DataProvider(name = "verifyTestLogin")
    public Object[][] verifyTestLogin() {
        return SuiteUtility.GetTestDataUtility(FilePath, "verifyTestLogin");
    }

    @BeforeTest
    public void BeforeTest()
    {
        driver = initializeDriver();
        test.log(Status.INFO,"Driver Initialized");
        log.info("Driver Initialized");

    }
    @AfterTest
    public void AfterTest()
    {
        driver.quit();
        driver = null;
        test.log(Status.INFO, "Driver killed");
        log.info("Driver Killed");
    }
    @AfterMethod
    public void AfterMethod()
```

```
{
    if(Userpage.isSignOffPresent())
    {
        Userpage.clickOnSignOff();
    }
    driver.get(prop.getProperty("url"));
}
```

These auxiliary functions assist in providing parameter values for test functions and creating a uniform web environment for each test case. As an example, consider the BeforeTest function. This function is executed before every test tag in testing.xml is executed. Refer to Appendix A for testing.xml. It used to ensure that the web driver is initialised and the browser is open in the right configuration. Likewise, AfterTest ensures that the browser is closed once the test is completed. It ensures that each test case gets the same environment before it is executed.

3.3 TEST REPORT

Test reports form the essential part of automation. The presentation and clarity of details mentioned in the report determine the ease of understanding and analysis. Use of extent reports library provides with the ability to detail every step of the test case with great details. Based on the test case described in Table 2, the variation in the test report is illustrated.

Scenario 1: Test case success

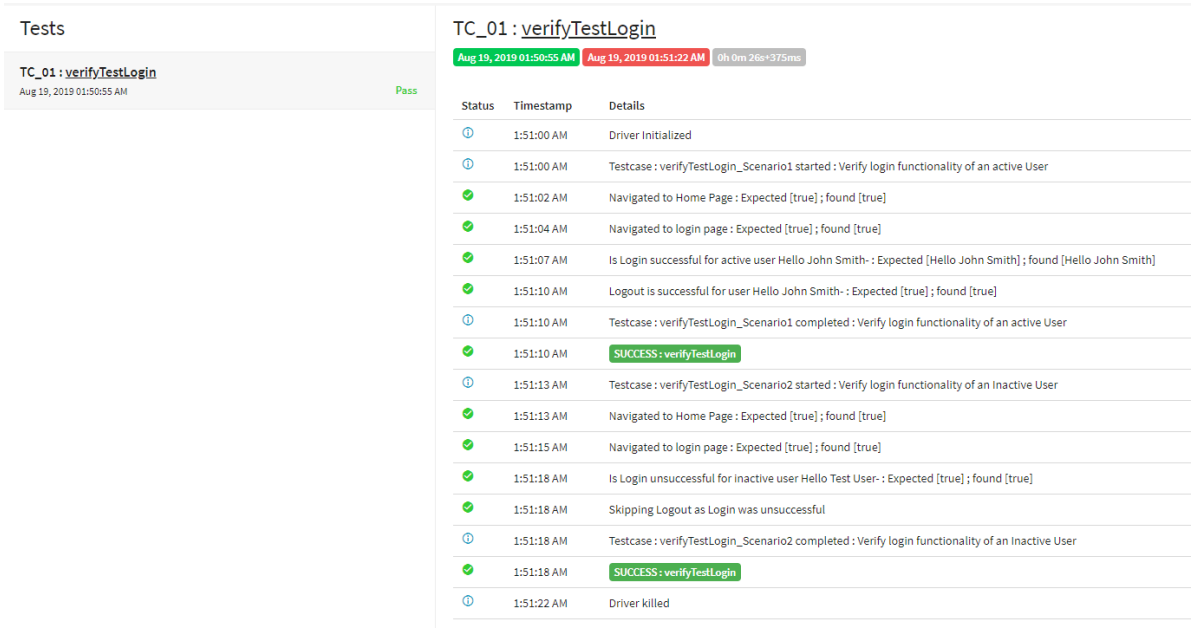


Figure 11:Report of a passed test case

Figure 11 shows the final report of a successfully passed test case. In the figure, there are two columns. On the left, there is a list of all the test cases that are executed while on the right, there is an elaborate step-by-step description of a selected test case. Each test step passed is marked with a green checkmark label indicating that the particular test step passed. There are further details of the time taken to execute the complete test case.

Scenario 2: Test case failure

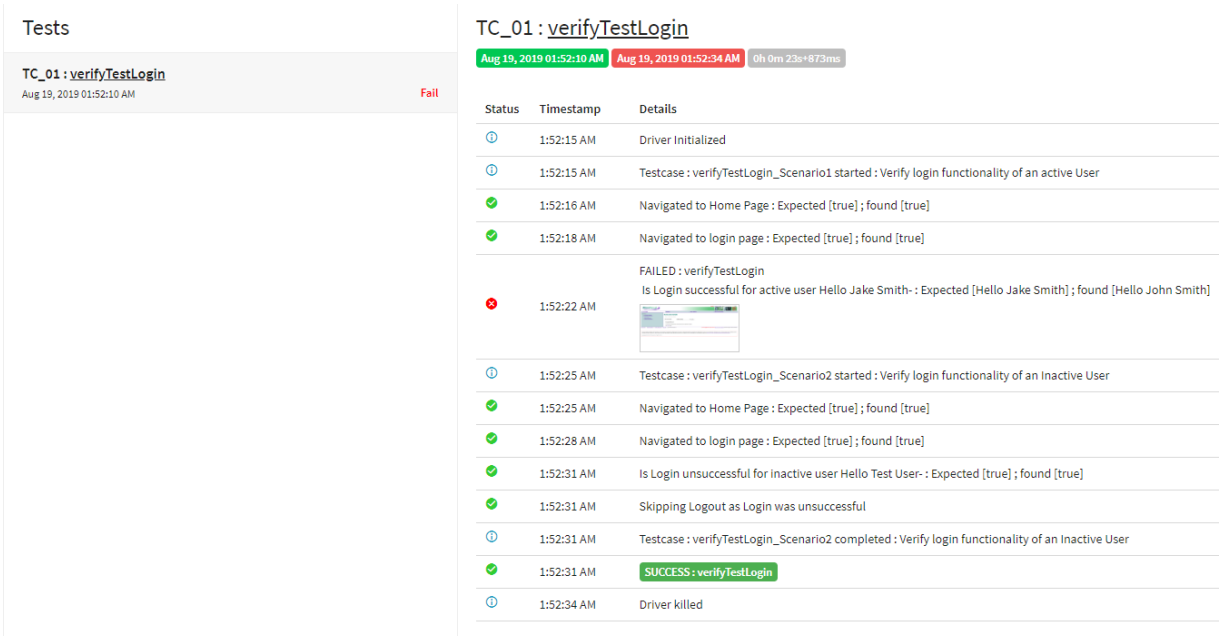


Figure 12: Report for a failed test case

Unlike the previous scenario, figure 12 shows the final report of a failed test case. It can be observed that a failed test step is indicated by a red cross mark which provides a visual cue for the error. The failed test step also provides a description and thumbnail of the screenshot at the time failure occurrence. The screenshot can be enlarged in by using a mouse click on the thumbnail provided. An enlarged screenshot is shown in figure 13a. The reason for failure is highlighted in figure 13b.

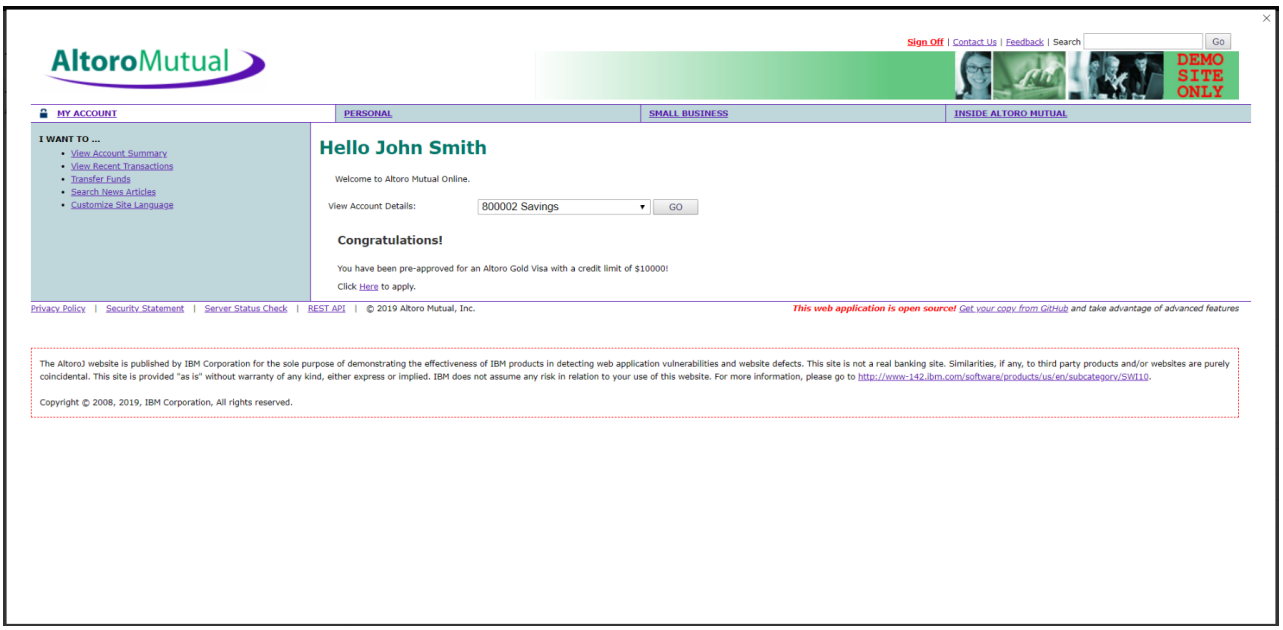


Figure 13a: Screenshot of the failure

Figure 13b: Reason for failure

CHAPTER - 4 RESULTS AND DISCUSSIONS

4.1 TEST CASE COVERAGE

As mentioned earlier, two of the web-based applications were chosen for automation testing purpose. Both the applications had a different set of test cases that were based on their functioning. From the pie charts shown below in Figure 14, it can be seen that around 47% of total test cases were automated for both the web applications. As a result, these automated test cases were run independently and on multiple web browsers without requiring a resource from the verification team. Moreover, the scripts were run multiple times in succession to test the stability of web application's new features added and were also used to regressively tested to check for any malfunction in existing functionality.

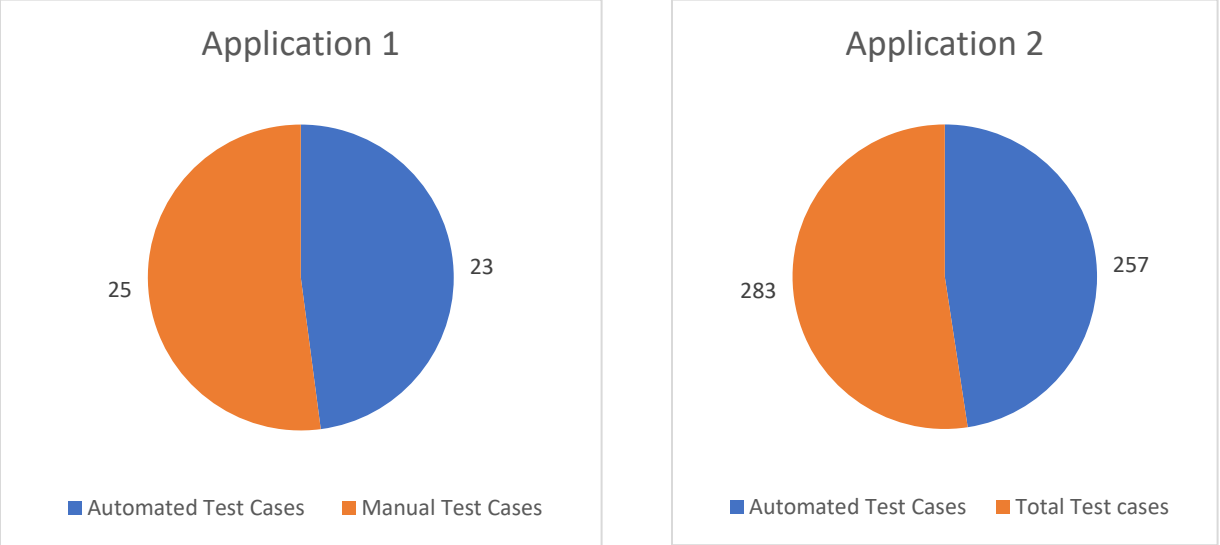


Figure 14: Automated Test cases Vs Manual Test cases for Web Applications

4.2 REDUCTION IN VERIFICATION TIME

Execution of test cases consumes a major part of the verification cycle apart from documentation and bug filing. So, automating test case save precious amount of time for the verification team members and improves the execution time since scripted web-based actions such as click, filling form details can be done at high speed when compared to manual execution. In theory, automating test cases must improve the overall execution time of verification. Figure 15 supports the effect of automating test cases on the reduction in execution time. It is visible that if a large number of test cases are automated there is a significant reduction in overall testing time for a given product.

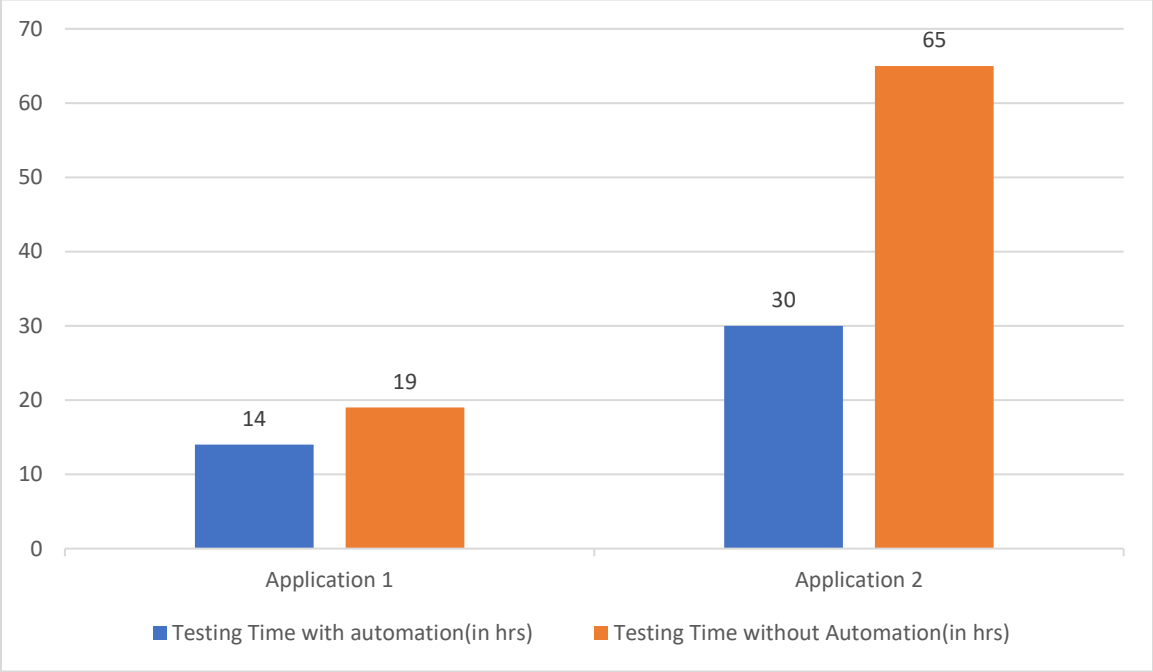


Figure 15: Total Time consumption- Automated Test cases Vs Manual Test cases

$$Time\ saved\ (\%) = \frac{Time\ taken\ without\ automation\ -\ Time\ taken\ with\ automation}{Total\ time\ taken\ without\ automation} * 100$$

$$Time\ saved\ Application\ 1\ (\%) = \frac{19 - 14}{19} * 100 = \ 26.31\%$$

$$Time\ saved\ Application\ 2\ (\%) = \frac{65 - 30}{65} * 100 = 53.84\%$$

CHAPTER - 5 CONCLUSIONS AND FUTURE WORKS

The dissertation was aimed at recognising the potential of involving automation testing into the verification cycle of a product. The testing process used for the products has been studied, and relevant test cases have been selected to accomplish the same. After selecting pertinent test cases, various tools required to accomplish automation have been researched. Tools such as Selenium, TestNG, maven and Apache POI were found prominent and were used to create a framework for automation testing. The structure and methodology of the framework were discussed, and the changes made to the architecture was described. As part of improving user understanding for the outputs from the automation script, an external reporting library called Extent Reports was used. This library provided a vibrant and readable report with a stepwise description of the test case. This framework was then implemented successfully to testing of two web applications and a considerable reduction in verification time was observed. The results were quantitatively presented showing that there is a reduction in the workforce on manual testing if the test cases were automated and the time consumed to execute an automated test case was much less when compared with its manual execution. 26.31% and 53.84% of staff-hours were saved by using the automation architecture for relevant test cases.

Due to lack of time, some of the areas were left unexplored. The following ideas can be implemented in order to supplement the given architecture and to drive further the efficiency

1. In the current architecture, the automation script is manually triggered, and a testing resource must initiate the script. In future, triggering of the script could also be automated by introducing Continuous Integration platforms like Jenkins that would build and deploy the testing script by itself without any manual intervention.
2. Another area of improvement would involve saving the automation report into a pdf along with the original Html format and emailing the reports to a configured authority once the execution of test cases is completed.
3. Lastly, extending the web-based automation testing to other types of applications such as mobile and desktop could prove very valuable as various tools like Appium, Testim, AutoIT and Winium exist in the market for both mobile and Windows-based application testing.

REFERENCES

1. "Stryker Corporation." International Directory of Company Histories. Encyclopedia.com. (August 20, June 2019). https://www.encyclopedia.com/books/politics-and-business-magazines/stryker-corporation
2. Moore, Gordon E. "Cramming more components onto integrated circuits." (1965): 114-117.
3. Bashhur, R.L., 1995, Telemedicine effects: Cost Quality and Access, Jr. of Medical Systems, 1981.
4. American Telemedicine Association. "Telemedicine, telehealth, and health information technology." An ATA Issue Paper. Consultado em 3 (2006).
5. Telemedicine and Telehealth: Principles, Policies, Performance and Pitfalls by Adam W. Darkins and Margaret A. Cary. Free Association Books, London, 2000. No. of pages 316. ISBN 1-853-43518-X.
6. Taipale, O., Kasurinen, J., Karhu, K., & Smolander, K. (2011). Trade-off between automated and manual software testing. International Journal of System Assurance Engineering and Management, 2(2), 114-125.
7. Di Lucca, Giuseppe A., Anna Rita Fasolino, Francesco Faralli, and Ugo De Carlini. "Testing web applications." In International Conference on Software Maintenance, 2002. Proceedings., pp. 310-319. IEEE, 2002.
8. Singh Jagdish, Monika Sharma, "A Comprehensive review of web-based Automation Testing Tools", International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE), Volume 3, issue 10, October 2015.
9. Sharma, Monika, and Rigzin Angmo. "Web-based automation testing and tools." International Journal of Computer Science and Information Technologies 5, no. 1 (2014): 908-912.
10. Kaur, Harpreet, and Gagan Gupta. "Comparative study of automated testing tools: selenium, quick test professional and testcomplete." Int. Journal of Engineering Research and Applications 3, no. 5 (2013): 1739-1743.
11. Jagannatha, S., M. Niranjanamurthy, S. P. Manushree, and G. S. Chaitra. "Comparative Study on Automation Testing using Selenium Testing Framework and QTP." vol 3 (2014): 258-267.
12. Devi, Jyoti, Kirti Bhatia, and Rohini Sharma. "A Relative Analysis of Programmed Web Testing Tools." International Research Journal of Engineering and Technology (IRJET) 4, no. 5 (2017): 386-389.
13. Ranstrom, Regina, Cynthia Nikolai, and Greg Madey. "Automated Web Software Testing With Selenium." REU Project Paper, Notre Dame (2010).
14. Wang, Fei, and Wencai Du. "A test automation framework based on WEB." In 2012 IEEE/ACIS 11th International Conference on Computer and Information Science, pp. 683-687. IEEE, 2012.
15. Kumar, Ajeet, and Sajal Saxena. "Data driven testing framework using selenium WebDriver." International Journal of Computer Applications 118, no. 18, (2015).
16. Gojare, Satish, Rahul Joshi, and Dhanashree Gaigaware. "Analysis and design of selenium webdriver automation testing framework." Procedia Computer Science 50 (2015): 341-346.
17. Leotta, Maurizio, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. "Improving test suites maintainability with the page object pattern: An industrial case study." In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, pp. 108-113. IEEE, 2013.

APPENDIX - A

FUNDAMENTAL CLASSES

Base.java

```
package resources;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.openqa.selenium.*;
import java.io.File;
import java.io.FileInputStream;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class Base
{
    public static WebDriver driver = null;
    public static Properties prop;
    public static Properties object = null;
    private final static Logger log = LogManager.getLogger("File");
    public static String timeStamp ="";
    public static int waitTime = 15;
    public static String reportName = "HtmlTestReport";

    public Base()
    {
        try
        {
            prop = new Properties();
            File resourcesDirectory = new
File("src/main/java/resources/data.properties");
            FileInputStream fis = new
FileInputStream(resourcesDirectory.getAbsolutePath());
            prop.load(fis);

            object = new Properties();
            File objectDirectory = new
File("src/main/java/resources/Object.properties");
            FileInputStream f = new
FileInputStream(objectDirectory.getAbsolutePath());
            object.load(f);

            waitTime=Integer.parseInt(prop.getProperty("waitTime"));
            reportName = prop.getProperty("reportName");
        }
        catch (Exception e)
        {
            log.error(" Error in initializing Base ",e.fillInStackTrace());
        }
    }
    public WebDriver initializeDriver()
    {
        try
        {
            String browserName = prop.getProperty("browser");
            String chromeDriverPath = prop.getProperty("chromeDriverPath");
            String firefoxDriverPath = prop.getProperty("firefoxDriverPath");
            String ieDriverPath = prop.getProperty("ieDriverPath");
```

```
            if (browserName.toLowerCase().equals("chrome"))
            {
                System.setProperty("webdriver.chrome.driver", chromeDriverPath);
                driver = new ChromeDriver();
            }
            else if (browserName.toLowerCase().equals("firefox"))
            {
                System.setProperty("webdriver.gecko.driver", firefoxDriverPath);
                driver = new FirefoxDriver();
            }
            else if (browserName.toLowerCase().equals("ie"))
            {
                System.setProperty("webdriver.ie.driver", ieDriverPath);
                driver = new InternetExplorerDriver();
            }

            driver.manage().window().maximize();
            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        }
        catch (Exception e)
        {
            log.error(" Error in initializing webdriver ",e.fillInStackTrace());
        }
        return driver;
    }
    public static String getScreenShotPath()
    {
        return prop.getProperty("screenshotPath");
    }


    public void getScreenshot(String folder, String method)
    {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("_yyyy-MM-dd__HH-mm-ss");
        LocalDateTime now = LocalDateTime.now();
        timeStamp = dtf.format(now);
        try
        {
            File screenshot
=((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
            String screenshotPath=getScreenShotPath();
            new File(screenshotPath + "\\" + folder).mkdirs();
            String failedCaseScreenshot = screenshotPath + "\\" + folder + "\\" +
method + timeStamp + ".png";
            FileHandler.copy(screenshot, new File(failedCaseScreenshot));
        }
        catch (Exception e)
        {
            log.error(" Error in taking screenshot ",e.fillInStackTrace());
        }
    }

}
```

Listeners.java

```
package resources;

import com.aventstack.extentreports.*;
import com.aventstack.extentreports.markuputils.ExtentColor;
```

```
import com.aventstack.extentreports.markuputils.MarkupHelper;
import com.aventstack.extentreports.reporter.ExtentHtmlReporter;
import com.aventstack.extentreports.reporter.configuration.Theme;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.testng.*;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Listeners extends Base implements ISuiteListener, ITestListener
{
    public final static Logger log = LogManager.getLogger("File");
    private ExtentHtmlReporter htmlReports;
    private ExtentReports extent;
    public static ExtentTest test;

    @Override
    public void onStart(ISuite suite)
    {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("_yyyy-MM-dd_hh-mm-ss");
        LocalDateTime today = LocalDateTime.now();

        String fullReportName =
System.getProperty("user.dir")+"\\"+reportName+dtf.format(today)+".html";

        htmlReports = new ExtentHtmlReporter(fullReportName);
        extent = new ExtentReports();
        extent.attachReporter(htmlReports);
        htmlReports.config().setDocumentTitle("Automation Testing Report");
        htmlReports.config().setReportName("Automated TestCase Example");
        htmlReports.config().setTheme(Theme.STANDARD);
    }

    @Override
    public void onFinish(ISuite suite)
    {

        extent.setSystemInfo("OS","Windows 10") ;
        extent.setSystemInfo("Browser",prop.getProperty("browser")) ;
        extent.setSystemInfo("Author","S.Srivathsan") ;
        extent.flush();

    }

    @Override
    public void onTestStart(ITestResult results)
    {

    }

    @Override
    public void onTestSuccess(ITestResult results)
    {
        test.log(Status.PASS,MarkupHelper.createLabel("SUCCESS : "+results.getName(),
ExtentColor.GREEN));
        log.info("SUCCESS : "+results.getName());
    }

    @Override
    public void onTestFailure(ITestResult results)
    {
        try
```

```
        {
            getScreenshot(results.getInstanceName(),results.getName());
            String folder =results.getInstanceName();
            String method = results.getName();
            String screenshotPath=Base.getScreenShotPath();
            String failedCaseScreenshot = screenshotPath + "\\" + folder + "\\" +
method + timeStamp + ".png";

            MediaEntityModelProvider p = null;
            try
            {
                p =
MediaEntityBuilder.createScreenCaptureFromPath(failedCaseScreenshot).build();
            } catch (IOException e)
            {
                log.error(" Exception in MediaEntityBuilder ",e.fillInStackTrace());
            }

            test.log(Status.FAIL," FAILED : "+results.getName()+ "   <br />
 "+results.getThrowable().getMessage(),p);
            test.addScreenCaptureFromPath(failedCaseScreenshot,(method+timeStamp));
            log.fatal(results.getName()+" Failed due to : ",results.getThrowable());
        }
        catch (Exception e)
        {
            log.error(" Exception in Listeners ",e.fillInStackTrace());
        }
    }

    @Override
    public void onTestSkipped(ITestResult results)
    {
        test.log(Status.SKIP,MarkupHelper.createLabel(" SKIPPED : "+results.getName()+
"   <br />

 "+results.getThrowable().getMessage(),ExtentColor.ORANGE));
        log.info(results.getName()+" Skipped due to : ",results.getThrowable());
    }

    @Override
    public void onTestFailedButWithinSuccessPercentage(ITestResult iTestResult) {

    }

    @Override
    public void onStart(ITestContext results)
    {
        test=extent.createTest(results.getName());
    }

    @Override
    public void onFinish(ITestContext iTestContext)
    {
        extent.flush();
    }

}
```

APPENDIX - B

UTILITY CLASSES

CustomAssert.java

```
package Utility;

import com.aventstack.extentreports.Status;
import org.testng.Assert;
import resources.Listeners;

public class CustomAssert extends Listeners
{


    public void assertTrue(boolean condition,String message)
    {
        String printMessage = message+" : Expected [true] ; found ["+condition+"] ";
        try {
            try {
                Assert.assertTrue(condition);
                test.log(Status.PASS, printMessage);
                log.info(printMessage);

            } catch (AssertionError e)
            {
                Assert.fail(printMessage);
            }
        } catch (Exception e) {
            log.error(e.fillInStackTrace());
            e.printStackTrace();
            Assert.fail(" Exception error in assertion ");

        }
    }


    public void assertContains(String actual, String expected, String message)
    {
        String printMessage = message+" : Expected ["+expected+"] ; found ["+actual+"]
";
        try {
            try {
                Assert.assertTrue(actual.contains(expected),expected);
                test.log(Status.PASS, printMessage);
                log.info(printMessage);

            }
            catch (AssertionError e)
            {
                Assert.fail(printMessage);
            }
        }
        catch (Exception e)
        {
            log.error(e.fillInStackTrace());
            e.printStackTrace();
            Assert.fail(" Exception error in assertion ");

        }
```

```
    }
}
```

Utilities.java

```
package Utility;

import com.aventstack.extentreports.Status;
import org.openqa.selenium.*;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import resources.Listeners;

public class Utilities extends Listeners
{

    public void pauseScript(int timeInSeconds)
    {
        try
        {
            Thread.sleep((timeInSeconds*1000));
            //test.log(Status.DEBUG," Pausing coding for "+timeInSeconds+"s");
            log.debug(" Pausing coding for "+timeInSeconds+"s");
        }
        catch (Exception e)
        {
            test.log(Status.ERROR," Pausing coding for "+timeInSeconds+"s failed");
            log.error(" Pausing coding for "+timeInSeconds+"s
failed",e.fillInStackTrace());
        }
    }

    public WebElement getWebElement(String ObjectName)
    {
        try {
            WebDriverWait wait = new WebDriverWait(driver, waitTime);
            String elementXpath = object.getProperty(ObjectName);
            log.info("Returning the "+ObjectName+" requested");
            return
wait.until(ExpectedConditions.elementToBeClickable(driver.findElement(By.xpath(element
Xpath))));

        }catch (Exception e)
        {
            test.log(Status.ERROR,"Error in returning the "+ObjectName+" requested");
            log.error("Error in returning the "+ObjectName+"
requested",e.fillInStackTrace().getMessage());
            return null;
        }
    }

}
```

SuiteUtility.java

```
package Utility;

public class SuiteUtility {
```

```
    public static Object[][] GetTestDataUtility(Read_XLS xls, String sheetName){
        return xls.retrieveTestData(sheetName);
    }

}
```

Read_XLS.java

```
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.Cell;

import java.io.FileInputStream;
import java.io.FileOutputStream;


public class Read_XLS {
    public  String filelocation;
    public  FileInputStream ipstr = null;
    public  FileOutputStream opstr =null;
    private HSSFWorkbook wb = null;
    private HSSFSheet ws = null;

// Constructor
    public Read_XLS(String filelocation) {
        this.filelocation=filelocation;
        try {
            ipstr = new FileInputStream(filelocation);
            wb = new HSSFWorkbook(ipstr);
            ws = wb.getSheetAt(0);
            ipstr.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    public int retrieveNoOfRows(String wsName){
        int sheetIndex=wb.getSheetIndex(wsName);
        if(sheetIndex==-1)
            return 0;
        else{
            ws = wb.getSheetAt(sheetIndex);
            int rowCount=ws.getLastRowNum()+1;
            return rowCount;
        }
    }
    public int retrieveNoOfCols(String wsName){
        int sheetIndex=wb.getSheetIndex(wsName);
        if(sheetIndex==-1)
            return 0;
        else{
            ws = wb.getSheetAt(sheetIndex);
            int colCount=ws.getRow(0).getLastCellNum();
            return colCount;
        }
    }

public Object[][] retrieveTestData(String wsName){
    int sheetIndex=wb.getSheetIndex(wsName);
```

```
    if(sheetIndex==-1)
        return null;
    else{
        int rowNum = retrieveNoOfRows(wsName);
        int colNum = retrieveNoOfCols(wsName);

        Object data[][] = new Object[rowNum-1][colNum-2];

        for (int i=0; i<rowNum-1; i++){
            HSSFRow row = ws.getRow(i+1);
            for(int j=0; j< colNum-2; j++){
                if(row==null){
                    data[i][j] = "";
                }
                else{
                    HSSFCell cell = row.getCell(j);

                    if(cell==null){
                        data[i][j] = "";
                    }
                    else{
                        cell.setCellType(Cell.CELL_TYPE_STRING);
                        String value = cellToString(cell);
                        data[i][j] = value;
                    }
                }
            }
        }
        return data;
    }

}
```

APPENDIX - C

FUNCTIONAL CLASSES

HomePage.java

```
package PageObjects;

import Utility.Utilities;
import resources.Listeners;

public class HomePage extends Listeners
{
    public Utilities Utilities= new Utilities();


    public void clickOnSignIn()
    {
        Utilities.getWebElement("SignIn").click();
        Utilities.pauseScript(2);
    }

    public boolean isOnlineBankingLinkPresent()
    {
        return Utilities.getWebElement("OnlineBankingLink").isDisplayed();
    }

    public void clickOnContactUs()
    {
        Utilities.getWebElement("ContactUs").click();
    }

    public void clickOnFeedback()
    {
        Utilities.getWebElement("Feedback").click();
    }
}
```

LoginPage.java

```
package PageObjects;

import Utility.Utilities;
import resources.Listeners;

public class LoginPage extends Listeners
{
    public Utility.Utilities Utilities= new Utilities();


    public void setLogin(String UserId)
    {
        Utilities.getWebElement("UserName").sendKeys(UserId);
    }

    public void setPassword(String Password)
    {
        Utilities.getWebElement("Password").sendKeys(Password);
```

```
    }

    public void clickOnLoginButton()
    {
        Utilities.getWebElement("LoginButton").click();
        Utilities.pauseScript(2);
    }

   public boolean isLoginButtonPresent()
    {
        return Utilities.getWebElement("LoginButton").isDisplayed();
    }

    public String getErrorMessage()
    {
        return Utilities.getWebElement("ErrorMessage").getText();
    }

    public String getLogin()
    {
        return Utilities.getWebElement("UserName").getAttribute("value");
    }

    public String getPassword()
    {
        return Utilities.getWebElement("UserName").getAttribute("value");
    }
}
```

UserPage.java

```
package PageObjects;

import Utility.Utilities;
import resources.Listeners;

public class UserPage extends Listeners
{
    public Utility.Utilities Utilities= new Utilities();


    public String getUserName()
    {
        return Utilities.getWebElement("HelloUser").getText();
    }

    public void clickOnSignOff()
    {
        Utilities.getWebElement("SignOffButton").click();
        Utilities.pauseScript(2);
    }
    public boolean isSignOffPresent()
    {
        return Utilities.getWebElement("SignOffButton").isDisplayed();
    }
}
```

APPENDIX - D

CONFIGURATION FILES

data.properties

```
browser=chrome
url=https://demo.testfire.net/index.jsp
chromeDriverPath=C:\\D_Drive\\Thesis\\Drivers\\chromedriver.exe
firefoxDriverPath=C:\\D_Drive\\Thesis\\Drivers\\geckodriver.exe
ieDriverPath=C:\\D_Drive\\Thesis\\Drivers\\IEDriverServer.exe
screenshotPath=C:\\D_Drive\\Thesis\\Screenshots
waitTime=15
reportName=ExtentReport
```

Object.properties

```
#Home Page
OnlineBankingLink = .//a[@id='AccountLink']
SignIn = .//a[@id='LoginLink']
ContactUs = .//a[@id='HyperLink3']
Feedback = .//a[@id='HyperLink4']

#Login Page
UserName = .//input[@id='uid']
Password = .//input[@id='passw']
LoginButton = .//input[@name='btnSubmit']
ErrorMessage = .//span[@id='_ctl0__ctl0_Content_Main_message']

#User Page
HelloUser = .//div//h1
SignOffButton=.//a[@id='LoginLink']
```

Log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
       <Properties>
       <Property name="basePath">./logs</Property>
   </Properties>
  <Appenders>
       <RollingFile name="File" fileName="${basePath}/prints.log"
filePattern="${basePath}/prints-%d{yyyy-MM-dd}.log">
         <PatternLayout pattern="%d{yyyy-MMM-dd HH:mm:ss.SSS} [%t] %-5level %C -
%msg%n"/>
           <SizeBasedTriggeringPolicy size="1 MB" />
       </RollingFile>
     <Console name="Console" target="SYSTEM_OUT">
       <PatternLayout pattern="%d{yyyy-MMM-dd HH:mm:ss.SSS} [%t] %-5level %C -
%msg%n"/>
     </Console>
  </Appenders>
  <Loggers>
     <Root level="INFO">
       <AppenderRef ref="File"/>
         <AppenderRef ref="Console"/>
     </Root>
  </Loggers>
</Configuration>
```

Testing.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Demo test" verbose="1">

    <listeners>
        <listener class-name="resources.Listeners"/>
    </listeners>

    <!--Demo test-->

    <test enabled="true" name="TC_01 : &lt;ins&gt;verifyTestLogin&lt;/ins&gt;" >
        <classes>
            <class name="testCases.TestCase"/>
            <methods>
                <include name = "verifyTestLogin" />
            </methods>
        </classes>
    </test>

</suite>
```

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>ExampleProgram</groupId>
    <artifactId>ExampleProgram</artifactId>
    <version>1.0-SNAPSHOT</version>

    <name>Test</name>
    <!-- FIXME change it to the project's website -->
    <url>http://www.example.com</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.7</maven.compiler.source>
        <maven.compiler.target>1.7</maven.compiler.target>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>3.6.0</version>
        </dependency>
        <dependency>
            <groupId>com.aventstack</groupId>
            <artifactId>extentreports</artifactId>
            <version>3.0.6</version>
        </dependency>
        <dependency>
            <groupId>com.relevantcodes</groupId>
            <artifactId>extentreports</artifactId>
            <version>2.41.2</version>
```

```
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.8.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.8.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.directory.studio</groupId>
        <artifactId>org.apache.commons.io</artifactId>
        <version>2.4</version>
    </dependency>

    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.0.0-beta1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>3.9</version>
    </dependency>
</dependencies>

<build>

    <resources>
        <resource>
            <directory>src/main/java/resources</directory>
            <filtering>true</filtering>
        </resource>
    </resources>

    <pluginManagement
        <plugins>
            <plugin>
                <artifactId>maven-clean-plugin</artifactId>
                <version>3.0.0</version>
            </plugin>
            <plugin>
                <artifactId>maven-resources-plugin</artifactId>
                <version>3.0.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
            </plugin>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.21.0</version>
                <configuration>
                    <suiteXmlFiles>
                        <suiteXmlFile>testng.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-jar-plugin</artifactId>
```

```
                <version>3.0.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-install-plugin</artifactId>
                <version>2.5.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-deploy-plugin</artifactId>
                <version>2.8.2</version>
            </plugin>
        </plugins>
    </pluginManagement>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
  </build>
</project>
```

thesis1

ORIGINALITY REPORT

19%	16%	5%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	forumsqa.com Internet Source	1%
2	leehao.me Internet Source	1%
3	www.software-testing-tutorials-automation.com Internet Source	1%
4	Submitted to University of West London Student Paper	1%
5	howtodoinjava.com Internet Source	1%
6	www.preop.com Internet Source	1%
7	memo.polypia.net Internet Source	1%
8	www.irjet.net Internet Source	1%
9	www.stahuj.cz-filmy-zdarma.cz Internet Source	<1%