Master of Science in Telecommunication Systems
May 2019

A Search-Based Approach for Robustness Testing of Web Applications

Karthik Gurram
Maheshwar Reddy Chappidi

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Contact Information:
Author(s):
Karthik Gurram
E-mail: kagu17@student.bth.se

Maheshwar Reddy Chappidi
E-mail: mach17@student.bth.se

External advisor:
Ricardo Britto, PhD
OE Change Leader, Ericsson
E-mail: ricardo.britto@ericsson.com

Bodil Jansson
RD Manager Program Office and Operation
E-mail: bodil.jansson@ericsson.com

University advisor:
Ricardo Britto, PhD
Department of Software Engineering
E-mail: ricardo.britto@bth.se

Dept. Computer Science & Engineering	Internet	:	www.bth.se/didd
Blekinge Institute of Technology	Phone	:	+46 455 38 50 00
SE–371 79 Karlskrona, Sweden	Fax	:	+46 455 38 50 57

Abstract

Context: This thesis deals with the robustness testing of web applications on a different web browser using a Selenium WebDriver to automate the browser. To increase the efficiency of this automation testing, we are using a robustness method. Robustness method is a process of testing the behaviour of a system implementation under exceptional execution conditions to check if it still fulfils some robustness requirements. These robustness tests often apply random algorithms to select the actions to be executed on web applications. The search-based technique was used to automatically generate effective test cases, consisting of initial conditions and fault sequences. The success criteria in most cases: "if it does not crash or hang application, then it is robust".

Problem: Software testing consumes a lot of time, labour-intensive to write test cases and expensive in a software development life cycle. There was always a need for software testing to decrease the testing time. Manual testing requires a lot of effort and hard work if we measure in terms of person per month [1]. To overcome this problem, we are using a search-based approach for robustness testing of web applications which can dramatically reduce the human effort, time and the costs related to testing.

Objective: The purpose of this thesis is to develop an automated approach to carry out robustness testing of web applications focusing on revealing defects related to a sequence of events triggered by a web system. To do so, we will employ search-based techniques (e.g., NSGA-II algorithm [1]). The main focus is on Ericsson Digital BSS systems, with a special focus on robustness testing. The main purpose of this master thesis is to aim at the robustness testing to optimize the test sequence length by minimizing their length, while simultaneously maximizing the fault revelation.

Method: For this approach, a meta-heuristic search-based genetic algorithm is used to make efficiency for robustness testing of the web application. In order to evaluate the effectiveness of this proposed approach, the experimental procedure is adapted. For this, an experimental testbed is set up. The effectiveness of the proposed approach is measured by two objectives: Fault revelation, Test sequence length. The effectiveness is also measured by evaluating the feasible cost-effective output test cases.

Results: The results we collected from our approach shows that by reducing the test sequence length we can reduce the time consuming and by using the NSGA-2 algorithm we found as many faults as we can when we tested on web applications in Ericsson.

Conclusion: The attempt of testing of web applications, was partly succeeded. This kind of robustness testing in our approach was strongly depended on the algorithm we are using. We can conclude that by using these two objectives, we can reduce the cost of testing and time consuming.

Keywords: Robustness testing, Web Applications, NSGA-2 Algorithm, Selenium Web Driver.

Acknowledgement

”Process of learning, one should not forget the purpose of Learning”

Swami Vivekananda

We firstly would like to thank our parents and siblings for their strong blessings and continuous financial support to accomplish this thesis successfully. We sincerely express our gratitude towards our supervisor Ricardo Britto, who has supported us throughout our thesis with their patience and knowledge.

We would like to thank Bodil Jansson and RMCA team from Ericsson for their regular suggestions, support and feedback which greatly helped us in completing this thesis work.

We would also like to thank our professors Patrik Arlos, Markus Fieldler, Dragos Ilie, Kurt Tutschku for the support they have given us throughout the Masters. We would like to thank our professor Kurt Tutschku for his hard work and patience in examining all the thesis.

Nomenclature

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AUT	Application Under Test
BSS	Business Support Systems
CDS	Chrome Driver Service
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma-separated values
DOM	Document Object Model
FSM	Finite State Machines
GUI	Graphical User Interface
GA	Genetic Algorithm
HTTP	Hypertext Transport Protocol
IDE	Integrated Development Environment
IE	Internet Explorer
JS	JavaScript
JSON	JavaScript Object Notation
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OSS	Operations Support Systems
RC	Remote Control
RMCA	Revenue Manager Catalogue Adapter
SoS	Systems of Systems
URL	Uniform Resource Location
QA	Quality Assurance

List of Figures

List of Tables

Contents

Chapter 1
Introduction

A web application is a web-based software application that users access it using a web browser [2] over the Internet.Web browser is a software application that is used for accessing information over the world wide web. Each individual web page, image, and video is identified by a distinct Uniform Resource Locator (URL) enabling browsers to retrieve these resources from a web server and display them on a user's device. Web applications involve interactions with the users and are dynamic in nature, whereas websites are used to disseminate information and are usually static in nature. Web applications might also include web pages which are static, in addition to the dynamic pages [3].

The use of web applications has dramatically increased over the last decade. In most of the consumer-based software products, data must be shared among millions of people and are mostly cloud-based. Examples include Twitter, Facebook, Google, etc. Cloud computing is heavily relied on by companies that provide software as a service (Saas) [4]. Most of the software provided by these companies are web-based e.g. Amazon. Web applications are not restricted to desktop computers. Web apps provide a general platform for users to share their data.

Today many companies are facing challenges of providing the customers with reliable, faster and more cost-effective services that depend on developing and maintaining a consistent set of quality system requirements and measurement techniques [5]. In a competitive world, where customers search for high-quality products, the software testing activity has grown in importance, aiming to assure quality and reliability to the product under development. Due to the demand for releasing new functionalities quickly and increasing the complexity of the software, testing has become critical and increasingly difficult [6]. To maintain a balance between the quality of testing and increasing criticality and size of the software, practitioners have turned towards automation[7].

Test case generation is a fundamental part of software testing. A test case is a set of conditions under which a tester will determine whether a web application under test satisfies requirements or works correctly and are used to expose faults within a web application. Creating more test cases requires more resources. Moreover, manual test-case generation is very time consuming and does not guarantee the best test cases since it depends on the engineers writing them. Many approaches have been proposed to automate software robustness testing. Most of them are based on fault-injection, i.e., they consist in feeding the system under test with (sequences of) invalid inputs, chosen within a fault-model, and supposed to exhibit robustness failures [8][9]. However, they differ in the way these inputs are chosen, and we review below some of them we consider as the most representatives.

The main purpose of this thesis is to use robustness testing to optimize test sequence length by minimizing their length, while simultaneously maximizing the fault revelation.

1.1 Research Problem and Motivation

Business support systems are software systems (systems of systems) [10] that telecommunications service providers use to run its business operations towards customers. Together with operations support systems, they are used to support various telecommunications services. BSS deals with the taking of orders, revenues, payment issues, etc. It supports four processes: customer management, revenue management, order management and product management.

Most of the companies are opting for automated testing to achieve benefits such as less human effort, quality improvement and time to market, but various studies show that achieving this is not easy [11]. Achieving efficient automated testing is dependent on how to perform tests within a shorter time and with less effort. By having an automated testing process, it would be possible to reduce the time and manpower needed for testing significantly. Organizations are experiencing problems with maintainability and time-consuming development of automated testing tools [11][6].

Software testing is a major cost factor in software development. The cost to generate a test suite for web applications is very high and very often demands human intervention for both generation and execution [12]. Web applications can have complex interactions between the events triggered by the web applications, which makes very difficult to have good automation testing for systems of systems [13]. One of the main challenging aspects of systems to test is robustness testing. In robustness testing, the system is tested against an unreliable environment or unexpected inputs. Robustness testing is difficult because it is hard to specify robustness properties. Many different definitions of robustness have been given in the literature, ranging from recovery after unexpected inputs to the resilience of the system in an unreliable environment.

The main reason to carry out this thesis work is to reduce the cost factor in software development. Although automation helps, it is still tough to capture defects that result from a sequence of events, which are not captured by executing isolated test cases. In general, these defects are captured in later phases of the development cycle, sometimes by the customers when carrying out exploratory end-to-end testing in their test labs. This may lead to bigger costs since existing research shows that the later you are identifying a defect, the bigger the cost associated with the defect fixing [14].

So, this is one scenario which motivated us to use an automated approach that supports the identification of defects related to a sequence of events which could lead to higher customer satisfaction, shorter deployment time, and lower cost. The process of test case generation can be automated using genetic algorithms which are specialized in optimizing solutions and uses a greedy approach when creating test cases. Therefore, we will use genetic algorithms to find a cost-effective test suite that reveals faults within web application during robustness testing. Cost-effective test suite means a test suite which can identify defects and bugs in the web application with the lowest test cases as possible. As discussed, many testing regions are explored by testers and developers as they suffer from biases while developing test cases. Moreover, there will be heavily used sections of the web application and they might not robustly test it. Hence to overcome such biases, we automatically generate test cases using this approach.

For this, the methodologies that can be adapted to automate robustness testing of web application using a search-based approach is studied. To identify defects relate to the sequence of events, the robustness testing of the web applications needs to be evaluated. So, the research work focuses on the robustness testing of web applications considering the test cases (test sequence length) as one of the parameters. The other parameter includes Fault revelation (Identifying faults or bugs in web applications). We automatically generate the test cases using an evolutionary approach. From the evaluation of the parameters obtained when the test cases are performed shows that these parameters are responsible for the cost factor.

1.2 Aims and Objectives

The main purpose of this thesis is to support automation of robustness testing of web applications and to reveal a maximum number of faults using least number of test cases using search-based approach I.e. using meta-heuristic algorithms [15] like genetic algorithms to automate the process of test case generation for web applications. Using actual user interactions, we initialize our algorithm. The goal is to generate an evolved set of cost-effective test cases that will reveal faults in the web application. Our approach implements a comprehensive list of web UI events for effective and efficient testing by evolving efficient tests that optimize competing objectives of maximizing fault revelation and minimizing test sequence length.

The objectives of this master thesis include the following:

- Explore the use of the search-based approach in test case generation of web application testing.
- Automate the process of test case generation.
- Reveal faults with a feasible number of test cases.
- Develop an effective and feasible approach to perform robustness testing of web application at the case company.
- Evaluate the proposed approach by using it to test a real web application in the case company.
- Analyze the results and explore which parameters have the most impact on the results.

1.3 Research Questions

Based on the research aims and objectives, the research questions formulated for this study are reported in this section. Research question RQ1 will be answered by conducting an improvement case study at Ericsson. On the other hand, research questions, RQ2 will be answered by experimentation.

RQ1: How can robustness testing of web applications will be automated and optimized to maximize defect identification in a web application?

RQ2: How effective is the proposed approach?

1.4 Scope of the Thesis

The focus of this thesis work is to develop an effective and feasible approach to perform robustness testing of the web application at the Telecommunication case company. The performance of our approach is evaluated based on the generated output test cases. The parameters such as a number of a chromosome, population, generation have the most impact on the results are identified. To perform the experiments, an experimental test-bed is set up. These experiments are conducted to analyze the results and performance of the framework.

1.5 Expected outcomes

The thesis report is expected to reflect the knowledge gained by fulfilling the research aims and objectives by answering the research questions. This includes:

- Through the meetings with employees at Ericsson and with a thesis supervisor, the challenges and difficulties in the thesis work are identified.
- An automated approach for carrying out robustness testing of web application at the case company.
- Evaluation of the feasibility, effectiveness and efficiency of the proposed approach for robustness testing of web application using existing historic data collected at Ericsson.

Chapter 2

Background

This chapter gives a clear overview of web applications, why it is hard and why it is important to test web applications, a search-based approach using relevant algorithms and their respective parameters.

2.1 Web applications

Web applications are Internet-based software applications [16], where the users interact with them by sending requests and receiving a response [17]. As shown in the figure 2.1 web applications rely on a server and using a web browser. Users interact with the web application by sending requests [18]. The server accepts the requests before sending the appropriate result to the user. This returned result is rendered by the web browser to be displayed to the user in a user-friendly user interface (UI). The component which is interacted by the user is the Core component. These interactions which are communicated with a database are stored on a server and is handled by the back end logic. The back end logic determines what the returned results will be every time a user makes a request. The front-end logic handles the results which are displayed to the user.

The data and features of the web application are accessed by the users via requests, usually an HTTP request [19]. An HTTP request is made up of the following components: method of the request, resource and a query string. Figure 2.1 provides a visual description of an HTTP request. When the server receives an HTTP request from the web browser, the server processes the method and resource and takes appropriate action. For example, if the request is to retrieve a particular web page (a GET Method), then that specific web page is looked by the server by looking at the resource and the query string. Then, the server will send back a response in the form of HTTP data if such a web page exists which is rendered by a browser to display in a human-friendly format. If there is any fault in the back-end logic or if the server fails to send the resource, the server responds with an appropriate error.

The content of the web page is not static and is usually dynamic and relies on the back-end code. If the database changes or updates, the content of the web page may change as it may be derived from a database. The content may also change due to the user interactions with the web page. The state generated by one user may persist beyond the user's time interacting with the web application and hence known as the persistent state.

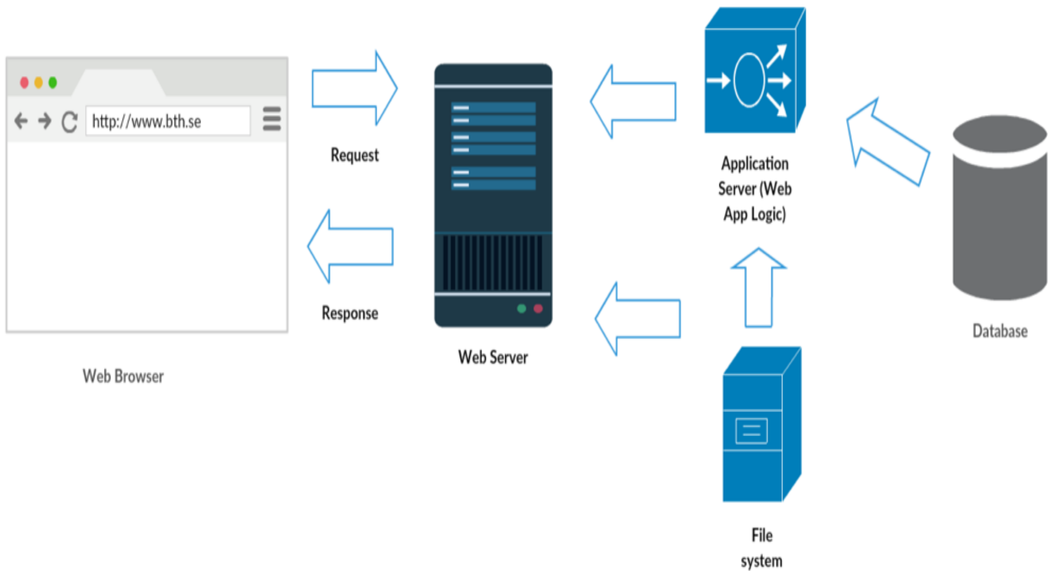


Figure 2.1: Visual description of an HTTP request

The content of the static application does not change unless explicitly changed by the creator. Compared with the faults on the static page, faults in the dynamic part of the web application are more common and expensive to fix. The dynamic web applications [20] involve complex back-end logic and databases. The entire web application may be crashed due to a fault in the heavily used feature or section of the code. Therefore, it's harder and important to test dynamic web applications [21].

2.1.1 User requests, User Sessions and Access Logs

The HTTP is generally used by the web browser to make requests to the server. A request has the following core components: what kind of the request it is dictated by the request method, a request resource which is composed of the resource name and resource path that also acts as a URL which allows the server to serve the correct response and query string that passes in the data a server might need to save in the database or to return the correct response. Figure 2.2 shows the general overview of an HTTP request.

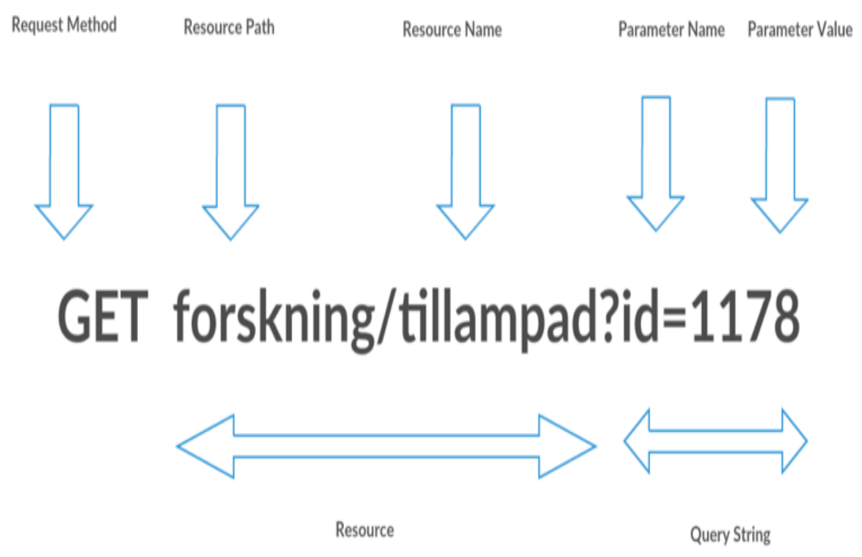


Figure 2.2: General overview of an HTTP request.

A user spends on a website for a specific amount of time period and session of activities of the user within this time is known as "User session" [22]. Even if the user takes a break and comes back to the website he visited recently; it is considered as one session. The interactions of the user with the web application can be tracked by the user session. The user sessions can be stored by web applications in the form of logs known as Access Logs. These access logs can be parsed and processed later to determine the user's behaviour. The access logs save the records of how the web application was used.

2.2 Manual testing and Automated testing

2.2.1 Manual Testing:

It is a type of testing in which test cases are executed manually (by a Person) without any support of a tool or script to find the defects [23]. In manual testing, the tester plays an important role as an end user and verify all features of the application to ensure the behaviour of the application. The Manual Testing is a very basic type of testing which helps to find the bugs in the application under test.

- Initially, Manual testing takes less time as testing is performed manually but for the period of time, we need more time and human resources, so it is time-consuming.
- It is less reliable due to human error.
- Investment is required for human resource.
- Manual Testing is only practical if test cases are run once or twice.
- Manual Testing involves human observation which may be more useful if the goal is user -friendly and customer experience.

2.2.2 Automated testing:

In automated testing, the test cases are executed by tool, script and software [24]. It is the process of automating the overall software testing, to automate the necessary and repetitive test scripts and overcome the problems of time-consuming due to repetitive and manual execution of test scripts. Test automation can help to improve software quality.

- Initially, it takes time to create the scripts and the whole set up, but at later stages, it takes comparatively less time than manual to perform regression testing.
- It is more reliable as it is performed by tools and scripts.
- Investment is required for tools and training.
- Automated testing is practical when test cases are run repeatedly over a long period of time.
- Automated testing does not guarantee user-friendliness or positive customer experience.

Automated testing is better than manual testing [25]. Automated tests are very explicit (white and black), so we have a more chance of finding a bug which was found by an automated test script by knowing what the automated test scripts execute to achieve the output. Because, the automated test scripts are explicit, they also execute consistently as they don't get tired or lazy like human-beings.

Automated testing is very much faster to run than manual testing which means we can run more test scripts in more browsers and more quickly. For example, eight browsers and four devices are tiring, but we can easily be performed with automated test scripts.

Automated test scripts also allow testing things which are not possible by manually. For example, answering a question like ‘if we have 200 accounts’, or ‘if we processed ten transactions simultaneously’ can only be solved efficiently by using automated test scripts.

2.3 Testing Web Applications

2.3.1 Software Testing:

It is a process of executing a program or the developed product with the intent of finding the bugs it possesses is called as software testing. Similarly, this testing can also be stated as the process of validating and verifying the product that works as expected. While testing, the test cases cannot be executed before the software has been developed, but it can be designed based on requirement specifications. The following attributes can explain the high quality of a test case, i.e., how good it is, According to [26],

- **Effectiveness:** It shows the detection ability of a test case to find defects or errors.
- **Exemplariness:** An exemplary test case should test more than one thing so that the total number of test cases required should be reduced. It should also pinpoint the found errors if the test tests several things.
- **Cost considerations:** How economical a test case is to perform, analyze and debug; and how evolvable it is, i.e., how much maintenance effort is required on the test case each time the software changes.

A well-designed test case requires that these attributes must be balanced one against another. For example, a high measure on the exemplariness which cost a lot to perform, analyze, debug and may require a lot of maintenance, can result in a low measure on the economic and evolvable scales [26]

2.3.2 Types of testing:

This research will involve different research areas related to software testing:

- Robustness testing.
- Search-based testing.

Robustness Testing

The term 'robust' is synonymous with strength. So, robustness is the degree to which a system or component can function correctly in the presence of stressful environmental conditions or invalid inputs [27].

Fuzzy testing is a technique that deals with robustness testing for any software application, which is a way to inject random, unexpected or wrong input values to the system to verify whether there is any memory leak or the application crashes [28]. When we provide a set of stressful environment or invalid inputs the program may crash. It becomes more significant to capture those bugs or errors and rectify it in accordance with the specified requirement. Hence to perform testing in an appropriate test environment, the suitable test cases are developed.

There are various procedures for this testing methodology. Let us understand the primary ones:
Interface robustness testing: This type of method relies on the fact that if a software product performs well, that is, it does not crash in any scenarios then it has passed the robustness criteria [29]. Example: Fuzz testing.

Fuzz Testing: Fuzz Testing or fuzzing is a Software testing technique, and it is a type of Security Testing [30]. In this type of testing, the automated or semi-automated testing techniques are used to discover coding errors or bugs and security loopholes in software, networks or operating systems by inputting a random or invalid data which is called FUZZ to the system. After the system is monitored for various exceptions, such as failing built-in code or crashing down of the system, etc.

The necessary steps for Fuzz Testing are:

1. Identifying the target system.
2. Identifying inputs.
3. Generating fuzzy data.
4. Executing the test by using fuzzy data.
5. Monitoring system behaviour.
6. Log defects.

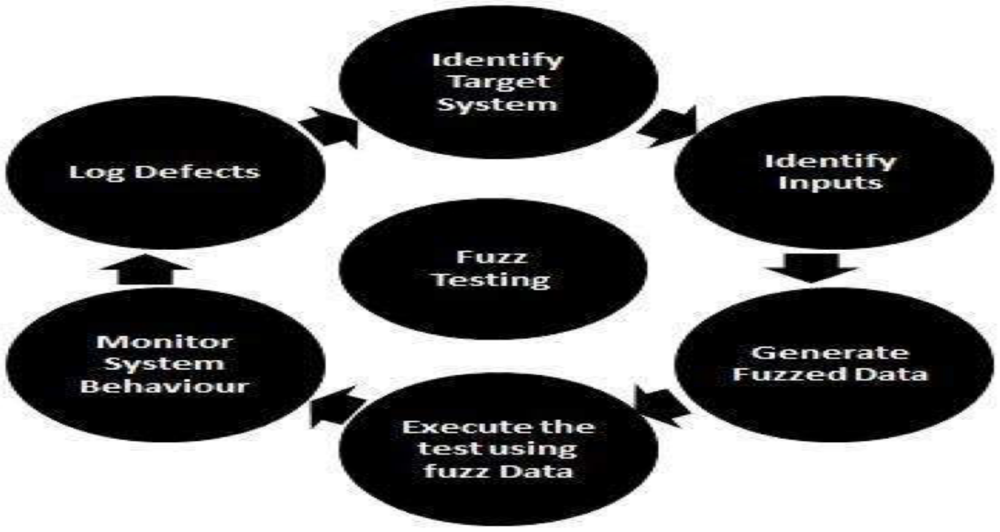


Figure 2.3: Steps for Fuzz Testing.

The simplest form of fuzz technique is sending invalid or random input to the software as an event. This technique of passing random or invalid input is very powerful to find errors in many services and applications. Many techniques are also available, but it is effortless to implement practically. We can implement these techniques by changing the existing inputs, and we can change the input by interchanging the bits of input.

Types of bugs or errors detected by using Fuzz Testing [30] [31] are:

- Memory leaks and assertion failures: This technique is widely used for large applications because the bugs which are present in the software are affecting the safety of memory, which is a critical vulnerability in the software.
- Invalid input: In Fuzz Testing, this technique was used to generate an invalid or random input which is used for handling errors in software, and this is important for the software which does not control its input.
- Correctness bugs: In Fuzz Testing, fuzzers are used to detect some types of "correctness" bugs, such as a poor search result, corrupted database, etc.

Challenges in Robustness Testing:

- The automation tool is designed for some specific purpose; hence there will be some lack of flexibility.
- Due to lack of adaptiveness of the systems, the effort to apply the principle of redundancy to structure it as per the requirement we need, but one needs to be more careful to make changes to the code being applied, as per the need of the current requirement.

- Even if the logic applied to the new piece of code is correct, complexity gets added with time as the size of a project keeps expanding.
- Maintenance.

Robustness testing can be compared to a state machine wherein providing inputs to the machine delivers a certain kind of output to us, and this output can be analyzed and assessed to ensure whether it serves the functionality along with good performance.

Search-based Testing

Automation testing helps to identify errors associated with the sequence of events. However, due to its nature, it is costly and time-consuming to carry out, because it involves human beings. Furthermore, it is in general carried out in later phases of the software development cycle, which may lead to late identification of defects and further costs. By automating this activity can lead to lower costs and shorter lead times. By usage of search-based techniques is a way to automate this type of testing.

Search-based Testing is the use of a meta-heuristic optimizing search technique [15], such as a Genetic Algorithm, to partially automate or automate a testing task; for example, the automatic generation of test data. The key to the optimization process is a problem-specific fitness function. The main role of the fitness function is to guide the search for a better solution from a potentially infinite search space, within a practical time limit [32].

Search Based Testing has many advantages including reduced efforts and improved reliability over state-of-the-art approaches to Software Testing. Search Based Testing can be implemented in various forms, including White Box and Black Box Testing. It can exploit the potential of other modern-day computing approaches including evolutionary approaches to computing such as Genetic Algorithms and Finite State Machines [33].

The approach for robustness testing of web applications which uses multi-objective search-based testing to automatically explore and optimize test sequences, minimizing length, while simultaneously maximizing fault revelation.

2.4 Gremlins.js

Gremlins.js or gremlins is a monkey testing library written in JavaScript, for Node.js and the other browsers. Gremlins are used to check the robustness of web applications by unleashing a horde of undisciplined gremlins [34].

Purpose: While developing an HTML5 application, we may anticipate different user interactions and we may manage to detect errors and patch all memory leaks. If not, then the application will break sooner or later. If n number of random actions can make an application fail, it's better to acknowledge it during testing, rather than letting users discover that application.

Gremlins simulate n number of random user actions: gremlins click anywhere in the application, move the mouse over elements that don't expect it or enter random data in text fields. The goal of gremlins is to trigger JavaScript errors or to make the application fail. If gremlins can't break an application, then the application is robust and enough to be released to real users.

This practice, also known as Monkey testing or Fuzz testing, is very common in mobile application development [34]. Now that frontend (MV*, d3.js, Backbone.js, Angular.js, etc.) and backend (Node.js) development use persistent JavaScript applications, this technique becomes valuable for web applications.

2.5 Selenium WebDriver

Selenium WebDriver is a browser automation testing tool for web applications which are used for compiling end-to-end tests [35]. It is designed in a very simpler way to provide a user-friendly API to make it easier to use. This tool does exactly what an end user would expect using a browser: the control of a browser is automated so that an end user would iterate the automated tasks. This seems like a simple problem to resolve, but behind the scenes, several tasks must be carried out before making it work. WebDriver is the name of the interface. It drives the browser either by a native user or by a remote user much more effectively. Thus, the limitations of Selenium 1.0, that affected the functional test coverage such as file uploads, downloads, pop-ups and dialogue barrier, are overcome. Selenium WebDriver has incorporated the language bindings and the implementations of the individual browser controlling code. WebDriver is just one component of selenium among many. WebDriver does not need to connect to a Remote Server to perform its task in the native machine, unlike Selenium RC.

Selenium WebDriver is completely an object-oriented API if we compare to Selenium 1.0. Let us have a look below:

Selenium 1.0 + WebDriver = Selenium 2.0 (Selenium WebDriver, 2014. Selenium WebDriver)

The implementations of WebDriver normally differ in each browser and drive the browser in order to test in a native machine with a specific browser and imitate human interactions with the web applications.

From WebDriver architecture, it is known that WebDriver directly calls the browser when tests are performed in a native machine by using the browsers built-in JavaScript support for the automation, unlike Selenium RC. In Selenium RC, the architecture works in a different way where it passes the HTTP requests to the server and the server passes it to the Selenium Core. Conversely, WebDriver does not have any proxy server in between the client and the browser. It directly gives the Selenese commands to the Selenium Core. So, The WebDriver API is a more powerful tool to speed up the execution time of tests compared to Selenium RC. Mobile browser-based tests were not possible at all before by using Selenium RC, but WebDriver can interact with the browser's rich content API and mobile applications. WebDriver supports almost all the latest versions of browsers out there in the market. It is officially stated that all future enhancements can be done only in WebDriver.

2.6 Genetic Algorithms

Genetic algorithms are adaptive search-based techniques. They are used to solve optimization problems. Genetic algorithm resembles the biological evolution. In biological evolution, the genes which are advantageous and has higher survival percentage are given preference. Similarly, in Genetic algorithm, predefined chromosomes and genes that have higher fitness are allowed to breed to create new genes/chromosomes of higher fitness. Genes are the basic unit of Genetic algorithm and chromosomes are made up of genes. "Genome" refers to a suite of chromosomes [36]. The chromosome or genome level is the level where reproduction can be done. Figure 2.4 shows the basic architecture of a Genetic algorithm.

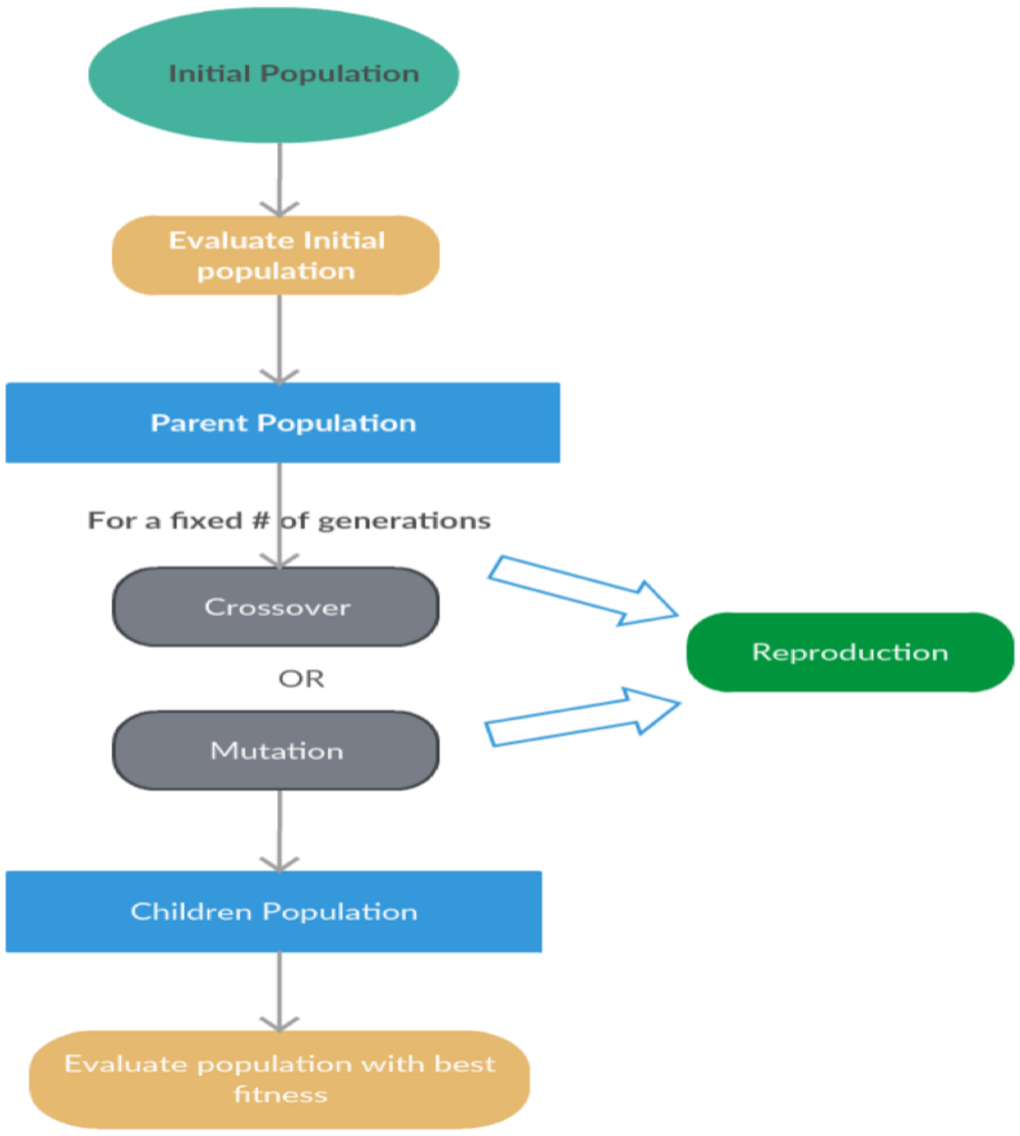

Initial Population
Evaluate Initial population
Parent Population
For a fixed # of generations
Crossover
OR
Mutation
Reproduction
Children Population
Evaluate population with best fitness

Figure 2.4: Architecture of Genetic Algorithm.

Usually, to determine how individuals are chosen for the mating purpose, a selection scheme is applied. Parent population refers to the individuals who are selected for mating and children population refers to the population which is generated by reproduction. Selection can be divided into two categories: Initial population and reproduction population. The initial population can be filtered based on the initial fitness value and are usually chosen at random. Children population are reproduced by a cycle of parent population and Hence called as a generation. For multiple generations, reproduction can be done.

Tournament selection refers to a selection scheme which is used to choose parent chromosomes for reproduction. Each individual who is selected at random from the parent population is also tested for their fitness. The one with the highest fitness is selected as a parent individual. The process is repeated for the next parent and these two parents go through the crossover to produce two child individuals. The process is further repeated to yield the child population thus producing a new generation of individuals.

Reproduction refers to the process of generating a new individual from selected individuals. This process generates a child population by considering the parent population and selecting individuals for reproduction. The child population generated by reproduction is the new generation of individuals.

Crossover refers to the two ways of reproduction. It involves the swapping of one part of a chromosome with another. I.e. swapping the first half of the chromosome with a bottom half of another. The crossover between two parents creates two child chromosomes.

Mutation usually involves random insertion or deletion of genes in a chromosome. Sometimes a gene may be minor ways. Mutation brings diversity within the population.

Chapter 3
Related Work

Regarding related work in this area, Ke Mao, Mark Harman and Yue Jia [37] performed a literature study on these areas: Sapienz approach, Evaluation and Search-based testing.

They used previous studies related to these areas and explained what has been done previously regarding the above respective areas. They also developed a tool to classify the different aspects of automated testing. For this approach authors proposed few definitions, using these definitions a tool was developed which can be used to compare, classify or elaborate on automated testing. Later, the validity of the tool is checked by comparing with different existing tools and techniques. In this paper, the main point is that we use the same classification used by authors to categorize the aspect of the purpose of automation for the Robustness testing of web applications.

This research papers will explore the benefits and challenges of an approach of Robustness testing of web applications using search-based techniques. This automated approach will support the identification of defects related to a sequence of events, which lead to higher customer satisfaction, shorter deployment time, and lower cost.

Much existing work on test automation has focused mostly on traditional programming languages which target desktop applications [38][39]. Although there have been many automated test generation techniques for web applications, these techniques are not suitable for testing from User Interface level as complex interaction events are required in order to cover the state space of the application.

The existing state of practice for automated JavaScript test data generation relies on random test data generation. The most popular tool available on GitHub for fully automated JavaScript testing is Gremlins [9]. It has over 8,146 stars/-likes and 394 forks on GitHub (at the time of writing), which provides random testing for JavaScript. However, random test data generation is known to be sub-optimal, compared to more intelligent computational search techniques [38] and is widely regarded as merely a baseline sanity check against which such more intelligent techniques should be compared.

There are two more intelligent test data generation techniques for automated testing of JavaScript web applications, that represent the current state of the art. Artemis a JavaScript testing tool based on feedback-directed testing was introduced by [40], and remains an active project on GitHub4, spanning over 4 years of development, and with over 1,458 commits. Feedback from the system's behaviour on test inputs generated is used to guide the test data generation approach. JSEFT (JavaScript Event and Function Testing), were introduced by [41]. JSEFT generates Document Object Model (DOM) event-based test cases as well as test cases for individual JavaScript functions, augmenting the test inputs with automatically generated mutation-based oracles.

Although both Artemis and JSEFT potentially advance the state of practice, due to their more intelligent test data generation techniques, neither addresses the problem of UI test sequence length. A fault-revealing test sequence will tend to be more valuable if it is shorter because a shorter test sequence will be easier and quicker for developers to investigate. Shorter sequences would also tend to reduce debugging time compared to longer sequences for the same fault, all else being equal. However, for JavaScript, there is no existing testing system that seeks to reduce test sequence length, while maximizing fault revelation. Furthermore, a very limited set of UI events (such as click and type events but not scroll, touch and other arbitrary gestures). Finally, the existing state of the art implementation does not produce a separate test suite that can be collected and reproduced. The first issue (test sequence length) is a fundamental scientific limitation compared to latter constraints.

Chapter 4
Methodology

This section of the document briefly describes the research methodology adopted in carrying out the thesis work, the setup used to perform tests on web applications and also working of the setup that is required to evaluate the proposed approach.

4.1 Research Methodology

The systematic approach which involves procedures, rules and set of practices for carrying out the research work is known as Research Method. Generally, to carry out the research work the research method may depend on the nature of the research questions and complexity. In general, there are two approaches to conduct research work [42]. They are:

- Qualitative approach.
- Quantitative approach.

The qualitative approach to conduct the research deals with various subjective assessments of attitudes, behaviours and opinions. These subjective assessments are done to carry out the research work. Some common techniques such as Group Interviews are used to carry out the research work using this approach.

The quantitative approach deals with the generation of data in the quantitative form. This approach can be subjected to quantitative analysis and is further divided into three sub approaches as follows:

- Inferential approach.
- Simulation approach.
- Experimental approach.

The research methodology adopted to carry out the thesis work involves both qualitative and quantitative approach.

Hence the careful analysis of the web application tested is performed and suitable meetings are planned and conducted with the employees at Ericsson. To study the important components of Ericsson BSS systems, the information was gathered from the developer team. Apart from these close observations are made from the browser while accessing the web application to identify and detect some of the components in web application.

The RQ 1 can be answered by identifying a suitable technology that can be implemented for robustness testing of web applications. The methodologies that can be adapted to implement the same includes both qualitative and quantitative approaches. From the qualitative approach, subjective assessments can be considered. User experience as discussed in the previous question can be considered as a factor here. But robustness testing involves the generation of the test data and helps in re-configuring and enhancing the performance of the web applications. Hence the suitable methodology that can be adapted to implement the robustness testing is Quantitative approach. In the inferential approach, relationships are established based on the database formed through methods such as surveys. The related information from an artificially created environment can be obtained using a simulation approach. The database formed in the inferential approach may become invalid over a period with the change in the generated test data created from the robustness testing strategy. The telecom company must ensure that the web application is Robust enough for the release. Hence simulation method cannot be considered as a suitable approach. In the experimental approach, the required variables are manipulated to observe the behaviour of the experimental variable [21] and can be considered as a suitable approach as it provides the variety of automated test cases. The cost-effective output test suites are considered as the experimental variable in this approach.

The robustness testing for the subject web application can be performed by interacting with the web application. This interaction with the web application is done by a real user, can be collected and analyzed later to further enhance the web application testing. To perform the robust tests on the subject application, an open source monkey testing library can be used. Gremlins.js [34] simulates random user actions: gremlins click anywhere in the window, enter random data in forms, or move the mouse over elements that don't expect it. Their goal: triggering JavaScript errors, or making the application fail. But to implement this in run time, it is necessary to automate the testing of the web application to perform robustness tests. So, selenium [35] web driver which is a collection of open source APIs which are used to automate the testing of a web application is considered as the best option and ideal technology to be used. To optimize and maximize defect identification within the web applications this technology needs to be extended. For this meta-heuristic search-based algorithms like NSGA2 [1] can be considered. As discussed in 4.2, the interactions with the web applications can be given as inputs to the algorithm and perform the tests.

The RQ2 can be answered by relating the obtained output test suites. The output test suites can be obtained through the experimental procedure described in section 3.2 of the document. The parameters contributing to degrading the performance of the framework and the algorithm are identified based on each test case length and the time taken by the framework on every run. The result is an indication that the corresponding parameters involved in the degradation of efficiency and effectiveness.

4.2 Experimental Setup

This section of the document deals with the test setup. The tests are to be performed simultaneously on multiple browsers to evaluate different web applications. Test scripts are generated automatically by the Test automation framework. Since the web applications being tested are complex in nature, the user needs to control the automation framework on all the web application being tested. To run the tests suitable machine-like ubuntu is used. This machine allows the user to stop or run the tests. The results of the tests are stored locally. Each output test result consists of the number of test sequences used and the revealed errors. Apart from this, the performance of the algorithm is also evaluated.
The important components involved in the experimental setup are:

- Test Machine.
- Test Automation Framework.
- Web Browser.

4.2.1 Test Machine

Test machine is used to execute the test scripts. We conduct all experiments on a machine and the machine specifications are shown in Table 4.1 The launching of the tests on the test machine is controlled by the test automation framework.

4.2.2 Test Automation Framework

Test automation framework is used to generate the test scripts to run on the test machine. The test scripts generated by the test automation framework are used to perform the suitable actions by the algorithm.

System Component	Description
OS	Ubuntu 16.04 LTS
CPU	Intel Core i7
RAM Size	8GB
Memory Type	DDR3
Hard Drive Size	500GB
CPU(s)	4
Processor Brand	Intel
Processor Speed	1.7 Ghz

Table 4.1: Test Machine Specifications.

4.2.3 Web Browser

Web browser is a software application to access information over the world wide web. The subject web application to be tested can be accessed using a web browser.

4.3 Experimental Procedure

4.3.1 Model

As discussed in section 3.1 we represent web application usage as user sessions which consists of the web app and the requests made to it by the users. To apply a search-based approach using genetic algorithms to the web application testing we modelled web application usage as genes, chromosomes and genomes. When a user interacts with the web application, for example, clicks, scroll down, touch, form fill submit, an HTTP request is made to load the content on the front end of the web application. A gene consists of a request (event on the web app), A chromosome consists of a sequence of requests and a genome is a sequence of chromosomes. Figure 4.1 illustrates a gene, chromosomes and genome.

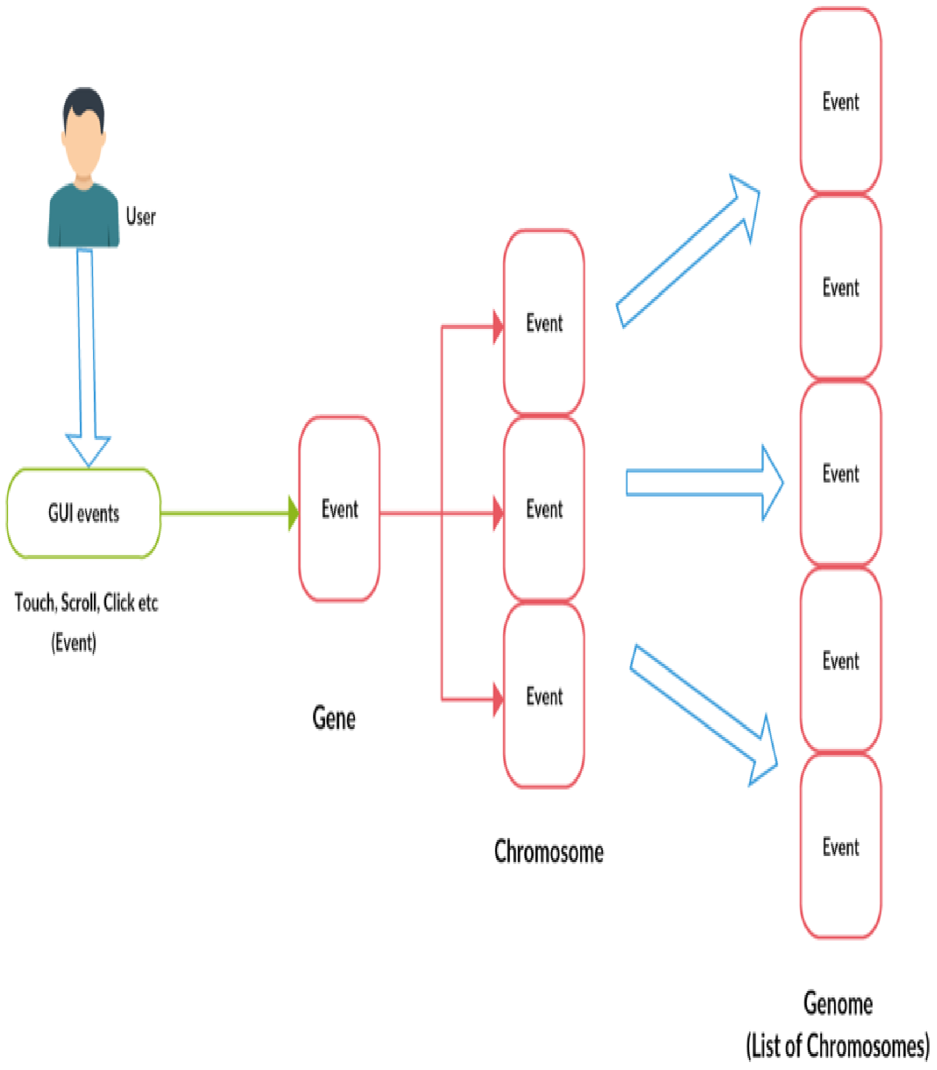


Figure 4.1: Illustrates a gene, chromosomes and genome.

The output is a sequence of chromosomes that will form our test suite since the sequence is important; our test suite consists most cost-effective sequence of test cases given the persistent states of the web application. Moreover, the output should reflect the usage of several chromosomes to mimic the real-world usage of the web application. Our approach can be broken down into three parts: converting chromosomes into inputs for the algorithm, creating an initial population from these inputs and running the algorithm. The input for the whole process logs from the web browser and the output will be the most cost-effective genome which will be our test suite. Figure 4.2 gives an overview of our approach.

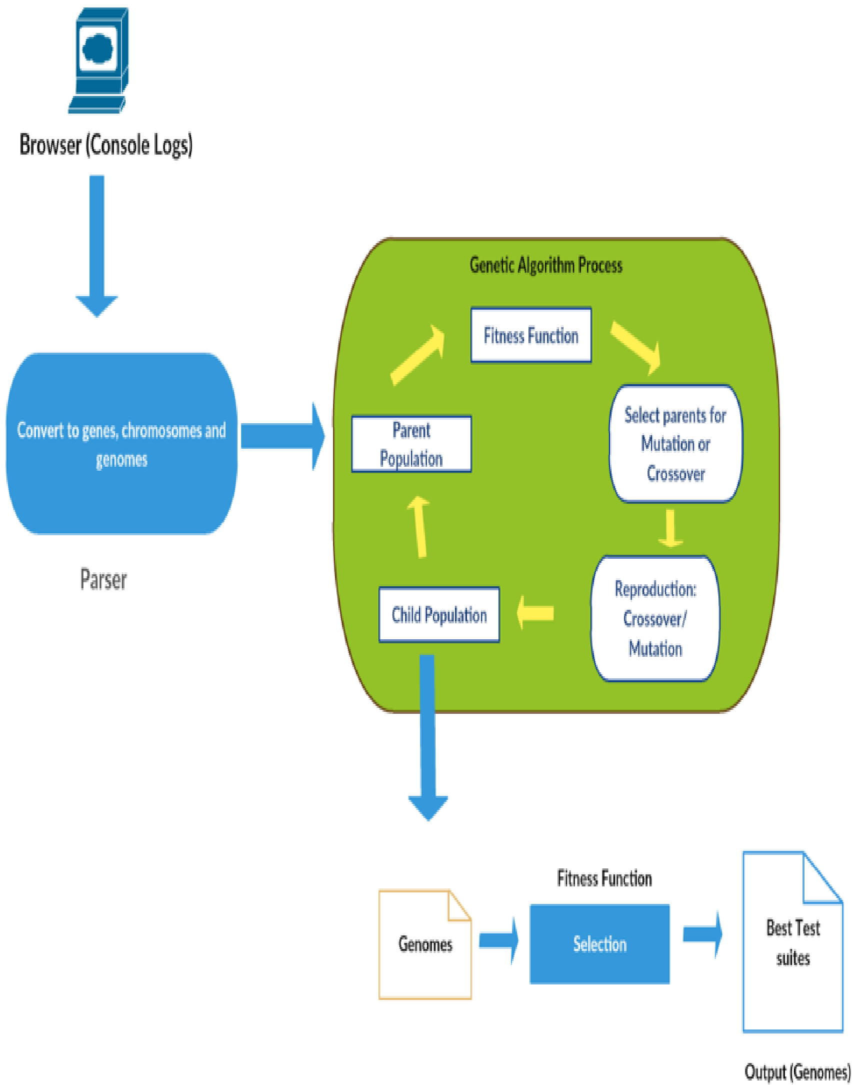


Figure 4.2: Overview of our approach.

4.3.2 Converting Browser logs into genomes

By giving a set of chromosomes, our parser processes and converts them into genome which is a sequence of chromosomes. The length of the genome is tunable, and a tester can change according to his preference. The length can be chosen based on the total number of chromosomes and how good each chromosome is based on the number of requests. Selecting a longer length may have increased coverage but may contain redundant requests. The initial population becomes the first parent population for the type of the genetic algorithm used.

4.3.3 Genetic Algorithm Process

Generating the initial population

The initial population of chromosomes is randomly selected from a set of chromosomes from the previous step. With the generated genome, the chromosomes are in their original and collected order. We used an initial fitness threshold to iter out the weakest chromosomes. More information on the fitness function is provided in the selection section. The threshold should be around the mean fitness of the genome pool. The initial population quota may not be fulfilled if we chose a high threshold. We may also miss out the better genomes if it too low.

Selection

Selection is a process where we select an individual from a genome pool based on their fitness. Tournament selection is the standard algorithm for selection in which we randomly select a fixed number of genomes, evaluate their fitness and chose the genomes with the best fitness which then becomes a parent. For second parent we repeat the process and apply crossover.

During Tournament selection, the random selection of a fixed number of genomes ensures that we do not select a few genomes as parents every time. The random selection distributes the probability of choosing a genome over whole genome pool. We would choose to pick a few genomes if we only select the best overall genomes. We may also lose out on the unique resources of the other genomes.

After the genetic algorithm process, we choose the best genome as our test suite by selecting the genome with the best fitness and by evaluating the last generation. A fitness function evaluates the fitness of a genome. As we have multiple objectives to optimize, our fitness function comprises of two metrics: length of the genome, Fault revelation. In general, we want to maximize the fault revelation while minimizing the length of the genome. The fitness function can be altered according to the needs of the tester. The overall result of the genetic algorithm depends on the fitness function since it favours genomes with higher fitness and the way we defined the fitness also dictates the output.

Reproduction

Genetic algorithm has two methods of reproduction: crossover and mutation. Genomes go through either crossover or mutation in each cycle. These reproduction algorithms can be designed at the chromosome or genome level. For this thesis, we focused on designing crossover and mutation for genomes because web applications have persistent state and therefore realistic usage and fault revelation is highly dependent on the sequence of requests.

Depending on the mutation threshold, the algorithm chooses whether to reproduce using mutation or crossover. We mutate the genome if the randomly generated number is higher than the mutation threshold. We will mutate 50 per cent of the time if the threshold is set to 0.5.

When the desired number of child population is generated, we stop the reproduction as the generated child population then becomes the parent population for the next cycle of the genetic algorithm.

Crossover

The crossover function takes two genomes as inputs and produces two child genomes. Each child genome has traits from both the parents. Figure 4.3 illustrates the crossover for a genome. In each genome, a split point is randomly chosen. The part in the genome A which is before the split point is joined with the part in the genome B which is after the split point and vice versa. The chromosomes from both the parents result in two child genomes.

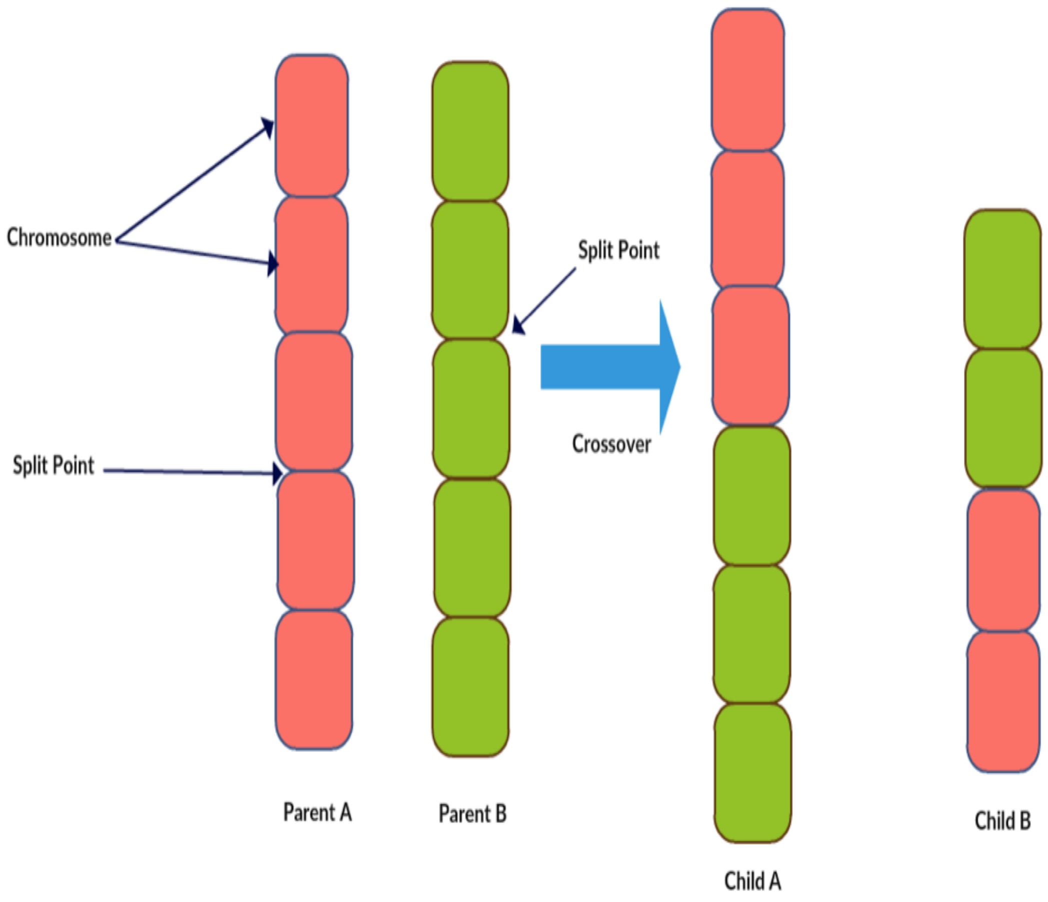


Figure 4.3: Crossover for a genome.

The resource coverage of the test suite is increased by Crossover and increases the effectiveness of finding fault within a web application. Tournament selection selects the best genomes i.e. parent genomes in terms of fitness from a randomly picked number of genomes. The resulting child genomes have higher fitness as we always choose genomes with higher fitness.

Mutation

Mutation operates on one genome where a random chromosome picked is swapped with another chromosome from a chromosome pool. Figure 4.4 illustrates mutation for a genome.

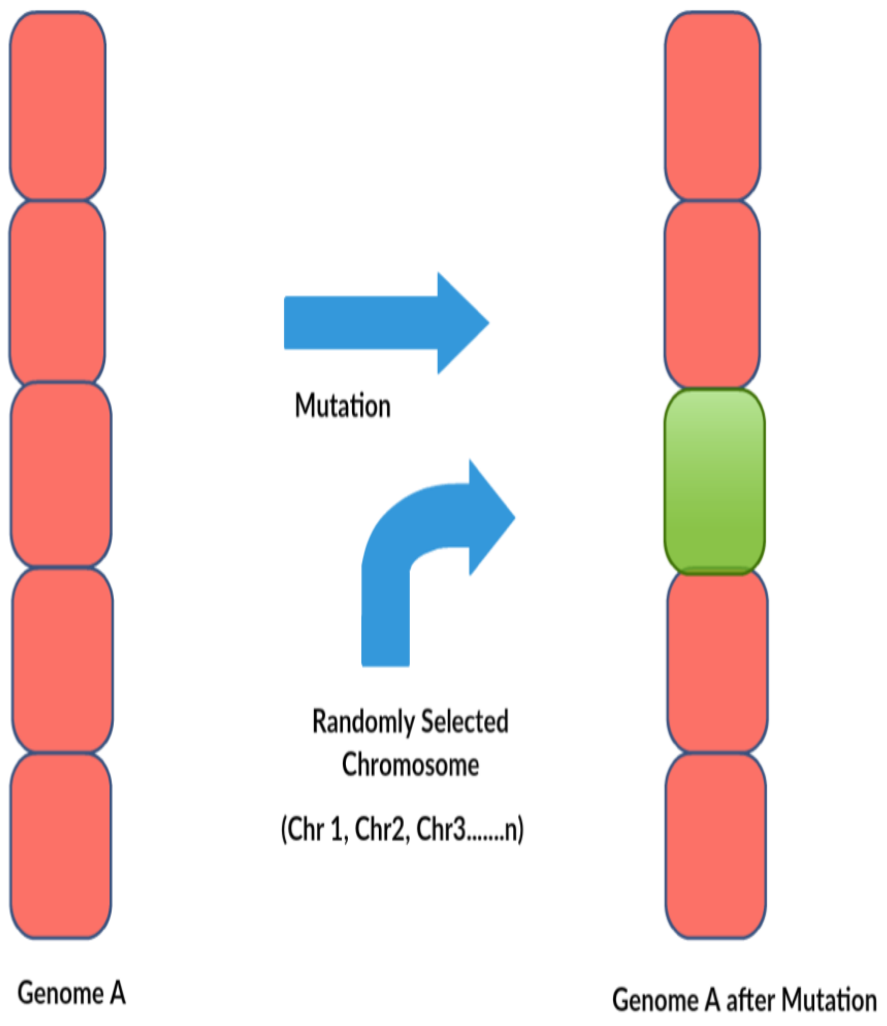


Figure 4.4: Mutation for a genome.

The mutation increases the diversity in our population. Since chromosome from a genome pool is replaced with another chromosome, new chromosomes are added into the existing pool. More diversity means we have a wider variety of genomes with a different set of chromosomes in the genome pool.

4.3.4 Implementation

We implemented our approach by using gremlins for test cases, selenium web-driver to automate browser and NSGA-2 algorithm for multi-objective search, based on the DEAP Python framework [15].

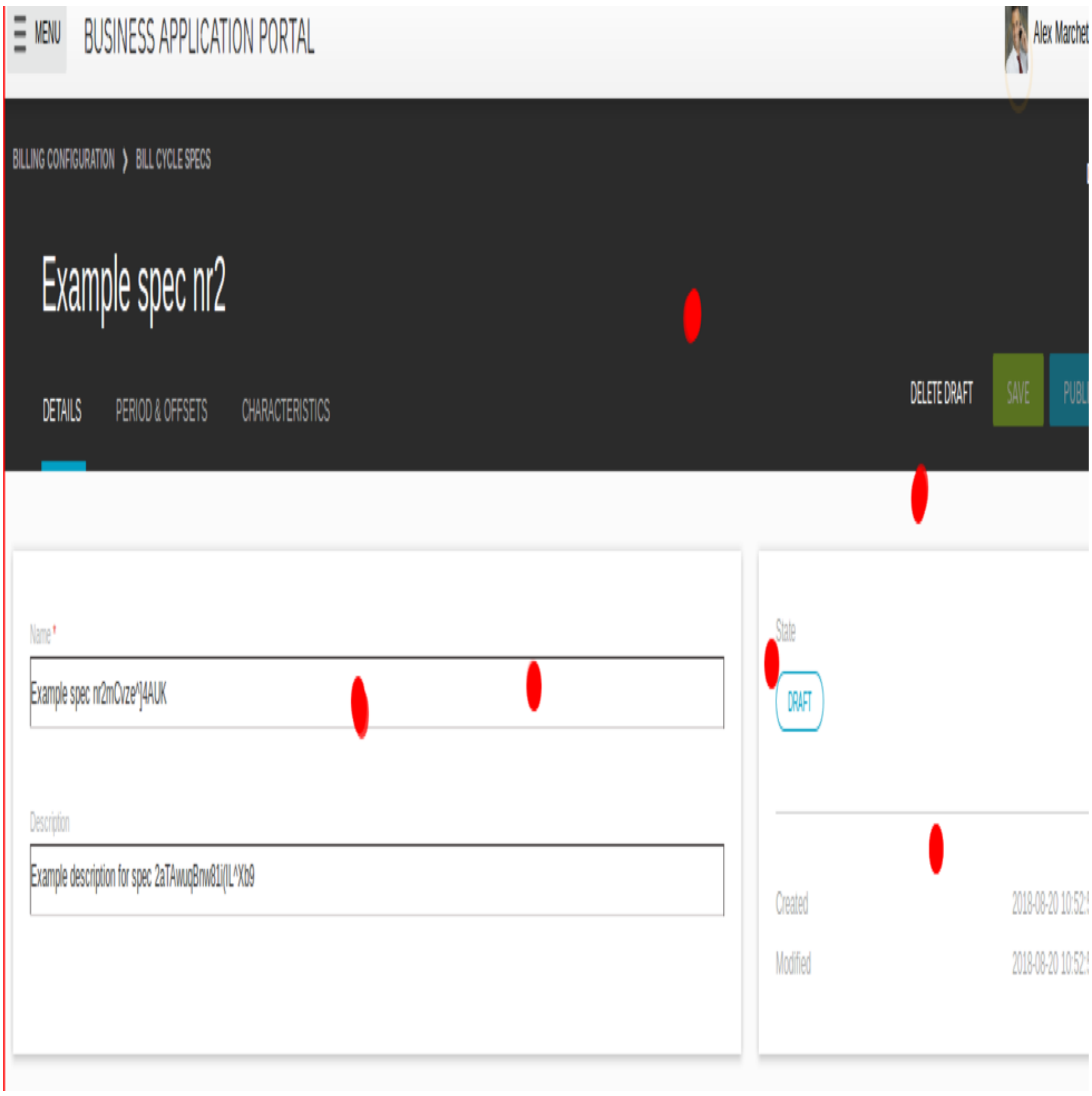
Gremlins.js Implementation

In our approach, we used gremlins to check the robustness of web applications by unleashing a horde of undisciplined gremlins. A gremlins horde is an army of specialized gremlins which is ready to mess up the web application [34]. Gremlins run with as little or as much control as we need. It has a low time cost for initial setup.

The default gremlins horde includes all five available types of randomly generated user interactions, otherwise known as "species of gremlins", that include:

- **formFillerGremlin** fills in inputs with data, interacts with other standard form elements and clicks buttons/checkboxes.
- **clickerGremlin** clicks anywhere on the visible application.
- **toucherGremlin** touches anywhere on the visible application.
- **scrollerGremlin** scrolls the viewport on the visible application.
- **typerGremlin** triggers random typing on a text field.

Gremlins provide several gremlin species: some click everywhere on the page, others scroll the window in every possible direction, others enter data in form inputs, etc. When triggered, gremlins will leave a visual indication on the screen for the action that was performed as shown in Figure 4.5 with red dots. By default, gremlins will be randomly triggered in 10-millisecond intervals for a total of 1000 times.



Example spec nr2

Name *

Example spec nr2mCvze^]4AUK

Description

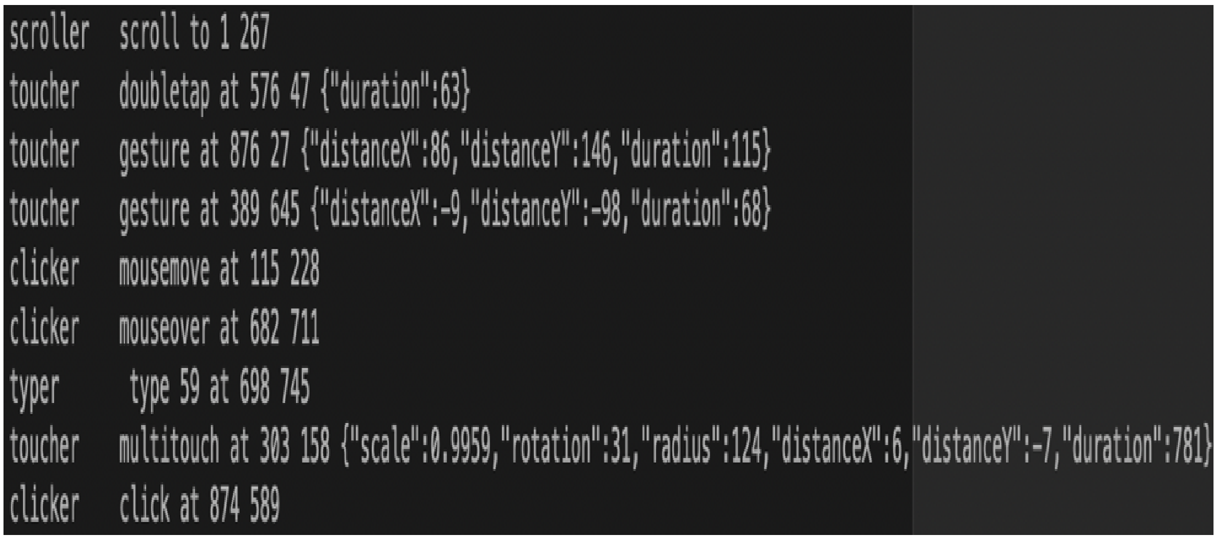
Example description for spec 2aTAwuqBnw81i(IL^Xb9

Created 2018-08-20 10:52:

Modified 2018-08-20 10:52:

Figure 4.5: Gremlins testing of web applications.

In Figure 4.5, we are testing the Ericsson web application to store the log files of the actions they took, which can be found in the developer console, along with any additional data associated with that species. These log files are test cases which were exported and saved in a CSV format file (.csv). They'll look something like the example as shown in Figure 4.6.



```
scroller   scroll to 1 267
toucher    doubletap at 576 47 {"duration":63}
toucher    gesture at 876 27 {"distanceX":86,"distanceY":146,"duration":115}
toucher    gesture at 389 645 {"distanceX":-9,"distanceY":-98,"duration":68}
clicker    mousemove at 115 228
clicker    mouseover at 682 711
typer       type 59 at 698 745
toucher    multitouch at 303 158 {"scale":0.9959,"rotation":31,"radius":124,"distanceX":6,"distanceY":-7,"duration":781}
clicker    click at 874 589
```

Figure 4.6: Gremlins test cases in .CSV file

Selenium WebDriver Implementation

In our approach, we used a Selenium WebDriver for automation testing framework to test the web applications. WebDriver is an interface that doesn't need any special server to execute tests. Instead, it just opens a browser instance [35]. WebDriver can be implemented on different browsers, but in our approach, we choose Chrome Driver for testing on chrome web browsers.

While running the test in a chrome web browser, we came across such situations where the tests failed displaying an error, i.e., WebElement could not locate the elements for different reasons. This happened because of the slow network where it takes more time to load the application; the second reason was that the web server could not serve the web page quickly due to resources constraints. It was noticed that during the test, WebElement was not loaded on the web page by the time the test script tried to identify it. To overcome such problem, the average wait time was calculated and configured so that WebDriver wait for the WebElements to load on the page before it displayed NoSuchElementException error. Implicit wait time and explicit wait time can be implemented by which it forces to make WebDriver wait for the WebElements.

- **Implicit Wait:**
  Implicitly wait is associated with a global timeout period and is common to all the WebElements. This is used to configure the wait time of WebDriver for the AUT. Depending on the network speed test execution time can be increased or decreased. If an application is hosted on a local machine, it always takes less time comparing the same application hosted on a remote server which may take more time to load a web page. In addition to that,

wait time would be configured considering the cases accordingly so that tests do not spend more time to wait for the page to be loaded or spend less time and timeout. Selenium WebDriver provides a method called manage() to handle such issues to set the implicitly wait time. Using implicitly wait time, the maximum wait time can be set for all WebElements on the web page.

Furthermore, Before WebDriver reaches the maximum time out; it polls the DOM for the WebElements for a certain period on the page. However, the default timeout is set at 0. If a negative timeout is set, the page load would be indefinite. Once set, it is set for the life of its object instance. In case of searching for multiple elements, WebDriver polls the DOM until it finds at least one element, or the timeout will expire. If XPath which is a slower strategy uses the locator, it is better to increase implicitly wait for timeout judiciously due to its adverse effect.

```
//wait for 5 secs
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
//change the theme
driver.findElement(By.xpath("//div[@*='MOa']/div[2]/div[@*='Gpa']
contains(text(),'Change theme')]")).click();
```

In the above code, we can notice that maximum timeout is set to 5 seconds in the web browser to wait for the WebElements since it is not present immediately in the page. If it cannot load the elements within the given period it throws a NoSuchElementException error; otherwise, it proceeds further with the rest of the test. We evaluate the effect of tuning the following parameters on our resulting test suite:

- **Explicitly wait:**
  Explicitly wait time configured to look for only a particular WebElement in the DOM before the test proceeds further. On the other hand, Thread.sleep() sets the condition to wait for the specified period which makes the test extremely slower. The main goal of using explicitly wait is to execute the automation testing faster. The following code was taken from our test:

  ```
  //wait for 2 secs
  Thread.sleep(2000);
  ```
  One of the ways, it can be accomplished with Expected Conditions. Below it is shown taken from the real test:

```
//wait for the element 35 secs max
WebDriverWait wait = newWebDriverWait(driver, 35);
WebElement elementForLogOut = wait.until (ExpectedConditions.elementToBe
Clickable(By.xpath("//*[@id='gb']/div[1]/div[1]/div[2]/div[5]/div[1]/a/span")));
elementForLogOut.click();
```

Here we can notice that in the above code snippet, that driver waits a maximum 35 seconds before it throws a TimeoutException error; else it continues with the next step. In every 500 milliseconds, WebDriver calls by default the ExpectedConditions until it returns the WebElement successfully. However, a successful return is always Boolean true or not null values for all ExpectedConditions types. ExpectedCondition is an interface used to implement the conditional wait for a particular WebElement. Thus, implicitly wait is overridden exclusively for the elements.

Prototype Implementation:

We implemented our approach primarily in a python programming language. First, we created test cases by parsing Browser Logs by using gremlins. Then we convert the test case into inputs for the genetic algorithm namely: genes, chromosomes and genomes. For a certain number of generations, genomes are reproduced and chosen after their fitness evaluation in each turn and selecting the ones with the highest fitness to reproduce using tournament selection. After the reproduction cycle, our output is the genome with the highest fitness value.

In the remainder of this section, we describe the implementation in detail.

We parsed the browser logs of the web application to create test cases based on the user interaction with the web application and for this, we used a Selenium WebDriver for automation testing framework to test the web applications. Each test case was saved in a CSV format file (.csv). The test cases were converted into sub-components of genetic algorithm.

We model a chromosome as an event on the web application. For example, click, scroll, touch, etc. and a sequence of chromosomes make up a genome. Our initial genome pool has a length of 100 genomes that are randomly made by randomly choosing a number. These can also be tuned as required. During this process, if the initial fitness value of each genome is above the initial fitness threshold, they are put in our initial population pool. This process is required to iter out the genomes with weak fitness. Our initial fitness threshold was 0.1 since that was the median fitness of the genome pool. We should have at least 100 genomes with fitness which are equal to or above the fitness threshold and iter out the weakest genomes. Since our test cases are from the actual interactions from the web application, some of them may only contain few events. Therefore, we iter out the test cases with weakest fitness value.

The initial population is our first parent population. If the randomly generated number between 0 and 1 exceeds the mutation threshold, the algorithm performs a crossover. Otherwise, the algorithm performs a mutation. We mutate 10 per cent of the time as our base mutation threshold is set to 0.1. We perform crossover more often than the mutation since we want to increase fault revelation while minimizing the sequence length. We keep doing crossover and mutation until we have the specified number of child population as they become the parent population for the next cycle of genetic algorithm.

The number of generations we chose is tunable and we repeat the process for a fixed number of generations. The fitness stays the same or increases with every iteration. We evaluate the fitness of each genome in our latest child generation when we reach our generation limit. The genome which has the highest fitness becomes our test suite.

To measure how many events a genome covers, we fixed the number of unique events in the genome and divide it with the total number of unique events interacted with the web application. A unique event has an event type and associated parameters, such as its location in the browser window. We get the total number of unique events in the web application by parsing all the test cases and making a list of unique events. The higher the fault revelation, the higher the fitness and the longer the sequence length, the lower the fitness. We have two weights: w1 and w2. These weights determine our prioritization of fault revelation and sequence length. If we want to prioritize fault revelation more, then w1 will be bigger than w2 and vice versa. Since the primary goal of this thesis is to maximize fault revelation, our w1 is higher than w2. But if w1 is too low, then the output genome will be too long and won't be cost-effective.

```
**Input**: Population $P$, crossover probability $p$, mutation
        probability $q$
**Output**: Offspring $Q$
$Q \leftarrow \emptyset$;
for $i$ in $range(0, |P|)$ do
    generate $r \sim U(0,1)$;
    if $r < p$ then                                    ▷ apply crossover
        randomly select parent individuals $x_1$, $x_2$;
        $x'_1, x'_2 \leftarrow uniformCrossover(x_1, x_2)$;
        $Q \leftarrow Q \cup x'_1$
    else if $r < p + q$ then                           ▷ apply mutation
        randomly select individual $x_1$;
        ▷ vary test cases within the test suite $x_1$
        $x \leftarrow shuffleIndexes(x_1)$;
        for $i$ in $range(1, |x|, step\ 2)$ do
            generate $r \sim U(0,1)$;
            if $r < q$ then
                $x[i-1], x[i] \leftarrow onePointCrossover(x[i-1], x[i])$;
        ▷ vary test events within the test case $x[i]$
        for $i$ in $range(0, |x|)$ do
            generate $r \sim U(0,1)$ ;
            if $r < q$ then
                $x[i] \leftarrow shuffleIndexes(x[i])$;
        $Q \leftarrow Q \cup x$
    else $Q \leftarrow Q \cup$ (randomly selected $x_1$); ▷ apply reproduction
return $Q$;
```

Figure 4.7: Pseudo code of NSGA-2 algorithm

From Algorithm Figure 4.7, We define a whole test suite variation operator to manipulate individuals. It applies one of the finer-grained crossover, mutation and reproduction operators on each individual (at test suite level). The inter-individual variation is achieved by using a uniform set element crossover among individuals (test suites). A more complex mutation operator manipulates the inter-individual variation. Since each individual is a test suite containing several test cases, the operator first randomly shuffles test case orders and then performs a single-point crossover on two neighbouring test cases with probability q, where the prior shuffle operation aims to improve crossover diversity. Subsequently, the more fine-grained test case mutation operator shuffles the test events within each test case with probability q, by randomly swapping event positions. Although atomic events include (mutable) parameters, we choose instead to mutate the execution order of the events, thereby reducing the complexity of the variation operator. Mutants are possible to operate on new GUI widgets not exercised by an initial test case because the timing of the operations is mutated. The reproduction operator leaves a randomly chosen individual unchanged.

We conducted all the experiments on a machine described in the test machine section 4.2.1. We assigned 10 minutes for our tool for each subject. We used the browser-reported console error as the automated test oracle. For Gremlins, we used its default uniform distribution when generating different types of events. Our tool, we set a crossover and mutation rate to 0.4 and 0.3 respectively and limited the maximum number of generations to 100, with a population size of 10, and with each individual containing 3 test sequences. These parameters remain the same through all experiments and are not tuned specifically for any certain subjects.

The scripts used to run the experiments can be referred from the appendix section.

Main.py: The main script to start the automation framework

This script starts the automation framework. This script starts python Flask server, starts gremlins on the specified URL. Gremlins perform suitable actions on the web application using selenium web driver. It clicks anywhere on the screen, scrolls, touches and tries to fill forms where text fields are available. The events such as click, touch, scroll etc are read by another script file readgremlinslogs.py from the browser console logs. Now using the Flask all the events are saved into a CSV file as test cases. Each CSV file is a test suite and each test suite contain multiple test cases (test event sequences). After the test cases have been saved to CSV file, the automation framework starts the NSGA-2 algorithm utils.py automatically which can be referred below. The commands below describe the automation framework steps:

- **To start logs flask server:**

  pip install -r requirements.txt export FLASK APP=app.py nohup flask run > noflask.out

- **To run gremlins:**
  run cmd = nohup npm run start –url http:localhost

Utils.py: Python based NSGA-2 Algorithm.

Test event sequences are fed to the algorithm to perform the necessary actions on the web application such as click, touch, scroll etc. The arguments in the script ngen, npop and nchromo represent the number of generations, number of populations and number of chromosomes respectively. Apart from these, there is one more argument to run the NSGA-2 algorithm in headless mode. The command to run the script is specified below:
python utils.py –url http://localhost –ngen 100 –npop 10 –nchromo 3 –headless ".format(args.url, args.ngen, args.npop, args.nchromo, args.headless)

4.4 Data Evaluation

Since our test data generation techniques are language independent, requiring just the application but not the source code for testing, our techniques can be easily extended to other web technologies as well.

For this thesis, we evaluate our approach on one of the Ericsson web applications which are built upon Angular JS. The application consists of a backend, a client browser and a web server. The content on the web application is due to the request and response from the web server. Browser logs were previously collected during the initial run. We converted the logs as inputs for the algorithm.

We run our prototype implementation for the evaluation of our hypothesis choosing and tuning the parameters as required. The initial fitness threshold during the initial population generation is fixed for all the runs of the test case generation algorithm. We find the average fitness for all our inputs and set the threshold around that so that we at least have 100 genomes for our initial population but at the same time filtering our weakest genomes. Moreover, we gave more priority to increasing the fault revelation than the length by using weights in our fitness function. For example (w=2, w2 = 1).

For every generation, We measure fault revelation capacity and a number of requests (length) in each genome as well as the time taken for each run of our algorithm. These metrics are also the components of the fitness: The higher the fault revelation, the higher the fitness and the longer the sequence length, the lower the fitness.

We evaluate the effect of tuning the following parameters on our resulting test suite:

- An initial number of chromosomes per genome: We performed different experiments with the following values for the initial number of chromosomes per genome: 25, 50, 100. The number of generations was fixed at 30 and the mutation threshold was fixed at 0.1.
- Mutation threshold: We performed three experiments with mutation threshold of 0.1, 0.25 and 0.5 respectively. In these experiments, the number of generations was fixed at 30 and the initial number of chromosomes per genome was fixed at 100.
- Number of generations: We performed one experiment with 100 generations to see how the number of generations affect fitness of the genomes. During this experiment, the mutation threshold was fixed at 0.2 and the initial number of chromosomes per genome was fixed at 100.

Each experiment was repeated 30 times taking the mean value of length for every generation and counting the faults in the web application. We also calculated the standard deviation for the length of the genome in every generation.

To evaluate the performance of the algorithm, CPU usage is also considered as a parameter. CPU performance is considered and compared against number of generations, number of populations and number of chromosomes as described in Figure 5.7.

Chapter 5

Results and Analysis

5.1 Results

This chapter of the document deals with the results obtained from the experimental procedure described in the previous section 4.3.

5.1.1 Comparing the size of the initial population

The size of the initial population is obtained by considering a sample of different chromosomes in the algorithm which are tunable. Chromosomes are generated by the automated framework and can be found from the file genes.csv as described in the previous section 4.5.3. The script utils.py is made to run continuously for the specified number generations. Each genome or individual or test case is a sequence of events. 5 runs of the script means 5 generations for the algorithm to populate each time.

Figure 5.1 provides an overview of the effect of increasing the initial number of chromosomes per genome. The y-axis represents the length in terms of a number of events of our test suite from the last generation. The x-axis represents the initial number of chromosomes per genome. The initial population of the algorithm is represented by the generation starting with 0.

As the initial number of chromosomes per genome increases, the length of the test suite and the probability of identifying more faults increases. As we increase the initial number of chromosomes per genome length increases steadily. Since our fitness is composed of fault revelation and length, we decreased the weight of length and increased the other as we want to identify defects in the lowest test sequence length as possible. The dependency between chromosomes and test suite length is that each test suite (.csv) contains multiple test cases (test event sequences) as described in Figure 4.1. The number of chromosomes per genome or individual means each genome again contains multiple test cases. So the test cases increases and also the test suite length and vice versa.

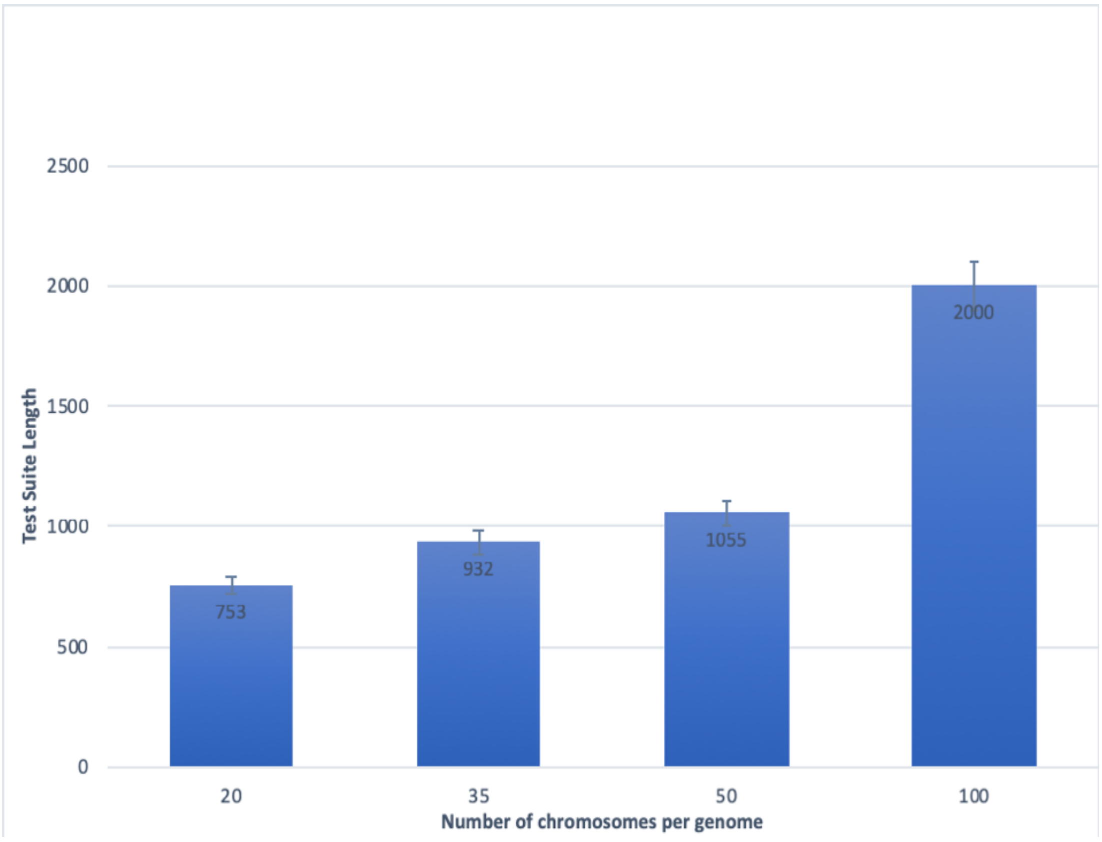


Figure 5.1: Effect of the initial number of chromosomes per genome on the length

Figure 5.2 gives an overview of effect of increasing the initial number of chromosomes per genome vs. the average time taken to run the framework for 5 generations. The y-axis represents the average time taken by the framework and the x-axis represents the initial number of chromosomes per genome. The average time taken to run the framework increases as we increase the initial number of chromosomes.

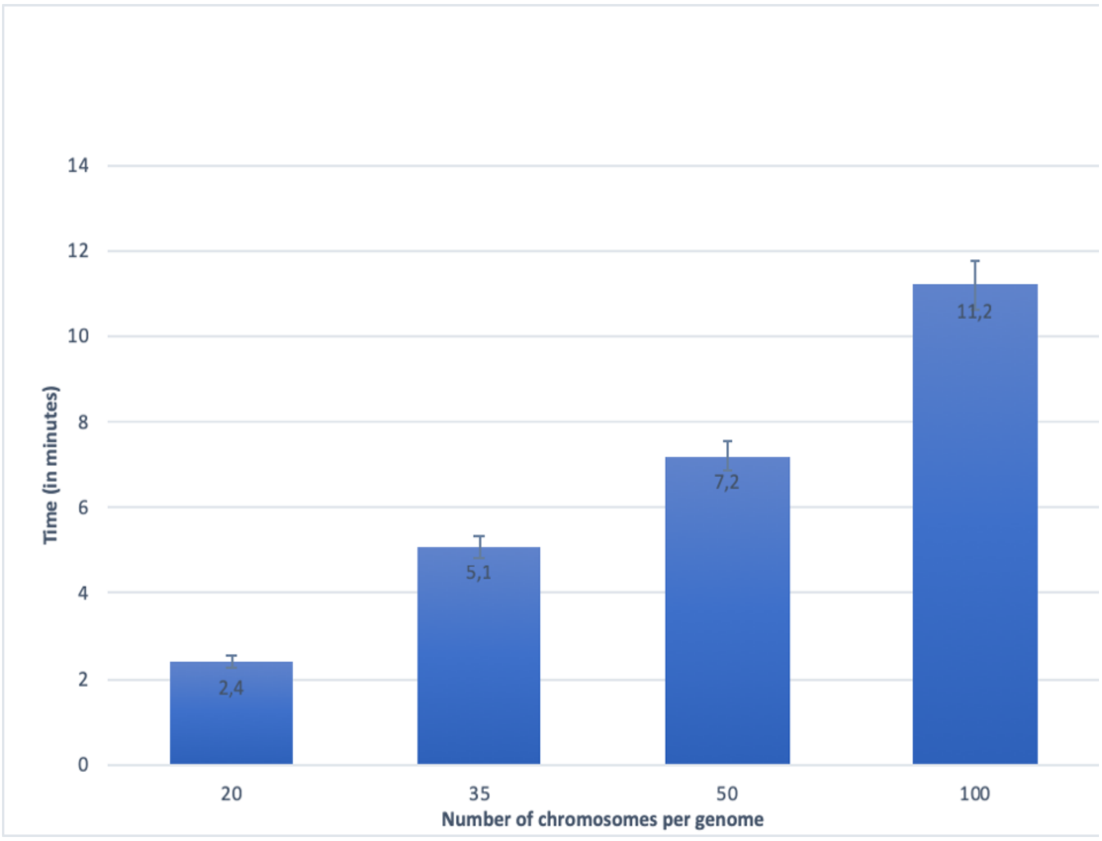


Figure 5.2: Effect of changing an initial number of chromosomes per genome on the average time taken to run GA for five generations.

5.1.2 Mutation Threshold

Mutation threshold is also another factor which is responsible for the overall increase in the test suite length. As described in section 4.3.3 mutation threshold is a value set in the algorithm to perform mutation. I.e inserting new genes into the chromosome pool. For this experiment, we chose 3 different mutation threshold values such as 0.1, 0.25 and 0.5 respectively. We run the experiments continuously for 10 generations with variable mutation threshold values and check to see if it affects the test suite length. The algorithm picks the test cases, performs mutation and generates new test cases for every run. In the later stage, we also calculated the time taken by the framework to complete the process for 10 generations and used utils.py script to run this experiment.

Figure 5.3 gives an overview of the effect of changing mutation threshold on our overall test suite. The y-axis represents the length in terms of events of our test suite from our last generation and x-axis represents the mutation threshold. Increase in the mutation threshold increases the length of the test suite.

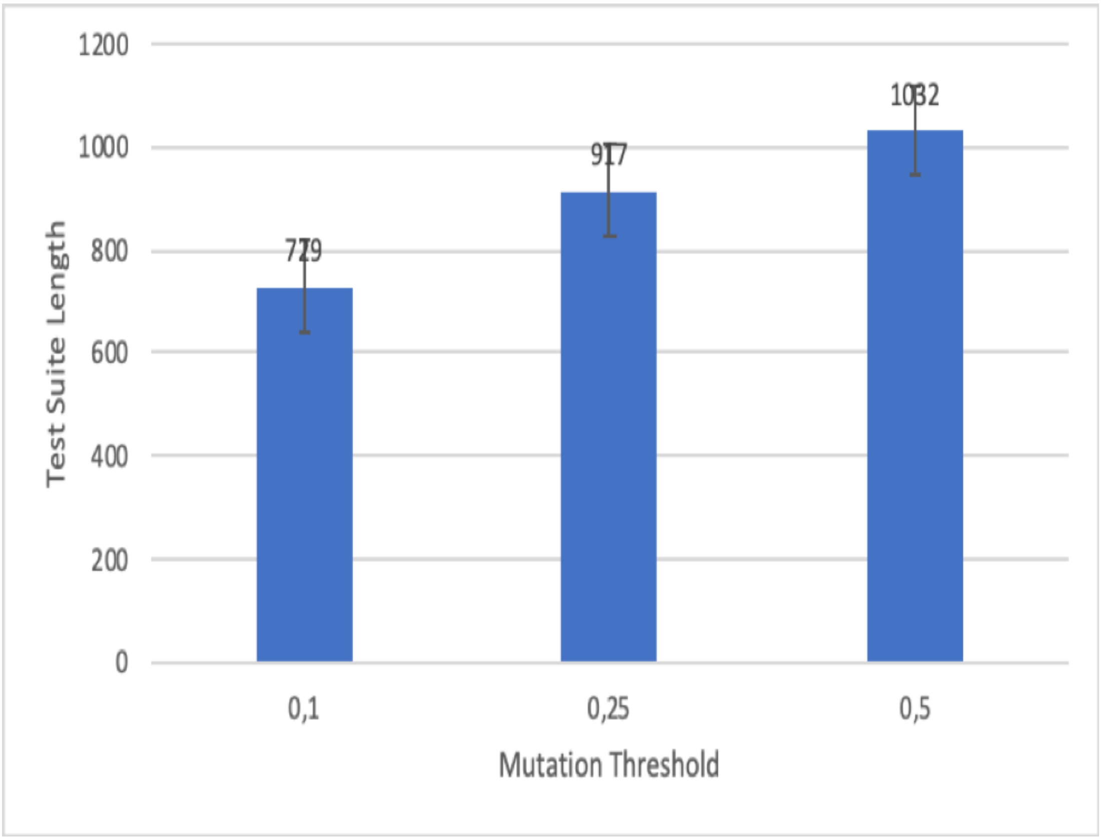


Figure 5.3: Effect of changing mutation threshold on length of the output test suite.

Figure 5.4 gives an overview of the effect of changing mutation threshold on the average time taken to run the framework for 10 generations. The y-axis represents the average time taken and the x-axis represents the mutation threshold. The average time is taken to run the framework increases as we increase the mutation threshold.

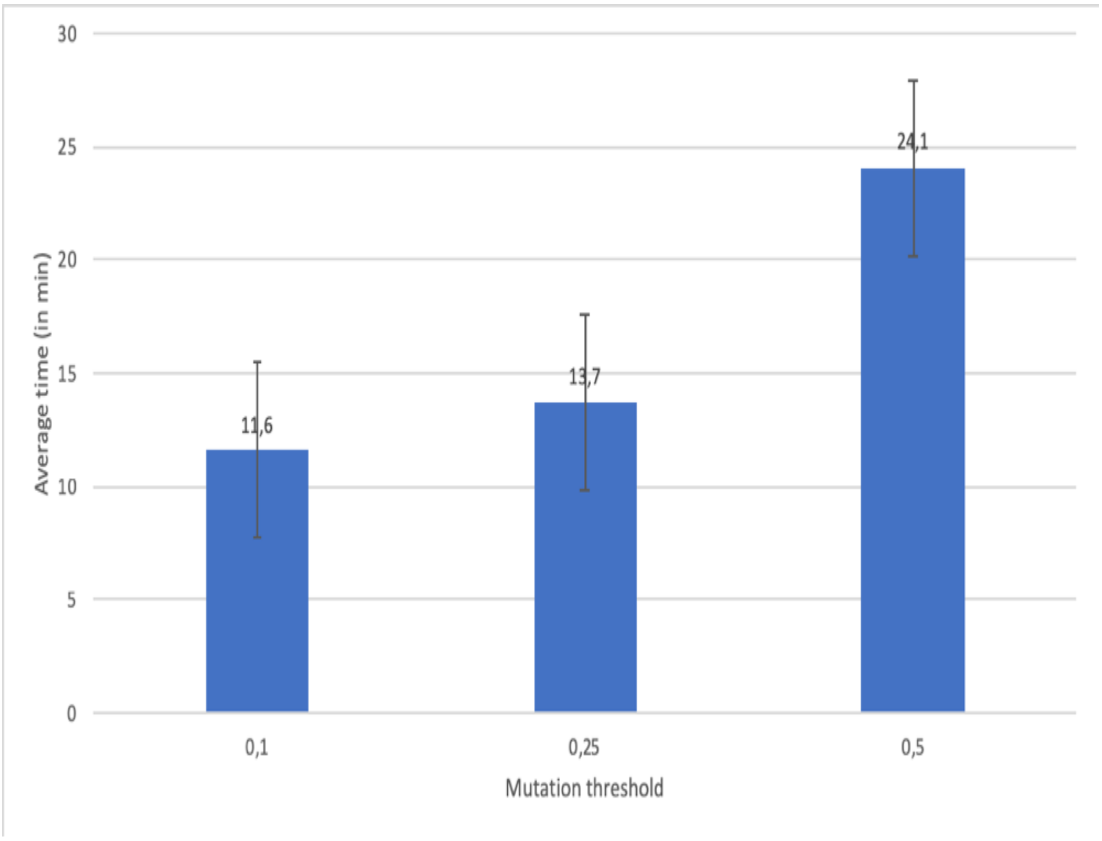


Figure 5.4: Effect of changing mutation threshold on the average time taken to run GA for 10 generations.

5.1.3 Number of generations

We ran the experiments for 100 generations to see the effect of a number of generations on the result test suite. Figure 5.5 demonstrates the result of the experiments. The y-axis represents the length, in terms of a number of events of the best genome in each generation and x-axis represents the number of generations.

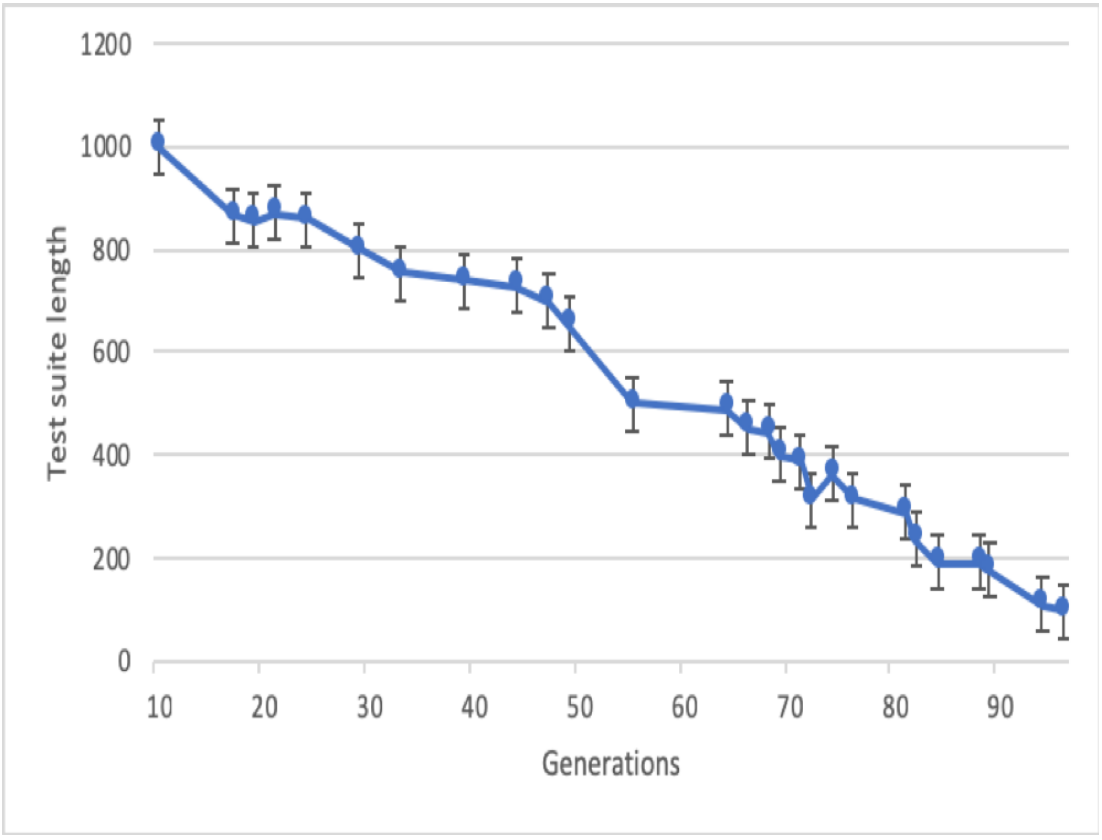


Figure 5.5: Effect of number of generations on length of the output test suite

Figure 5.6 shows the effect of increasing the number of generations on the average time taken by the framework to run. The y-axis represents the average time taken in minutes and the x-axis represents the number of generations. The average time to run the framework increases as the number of generations increases and vice-versa.

The results of this experiment support our hypothesis that increasing the number of generations will yield better results.

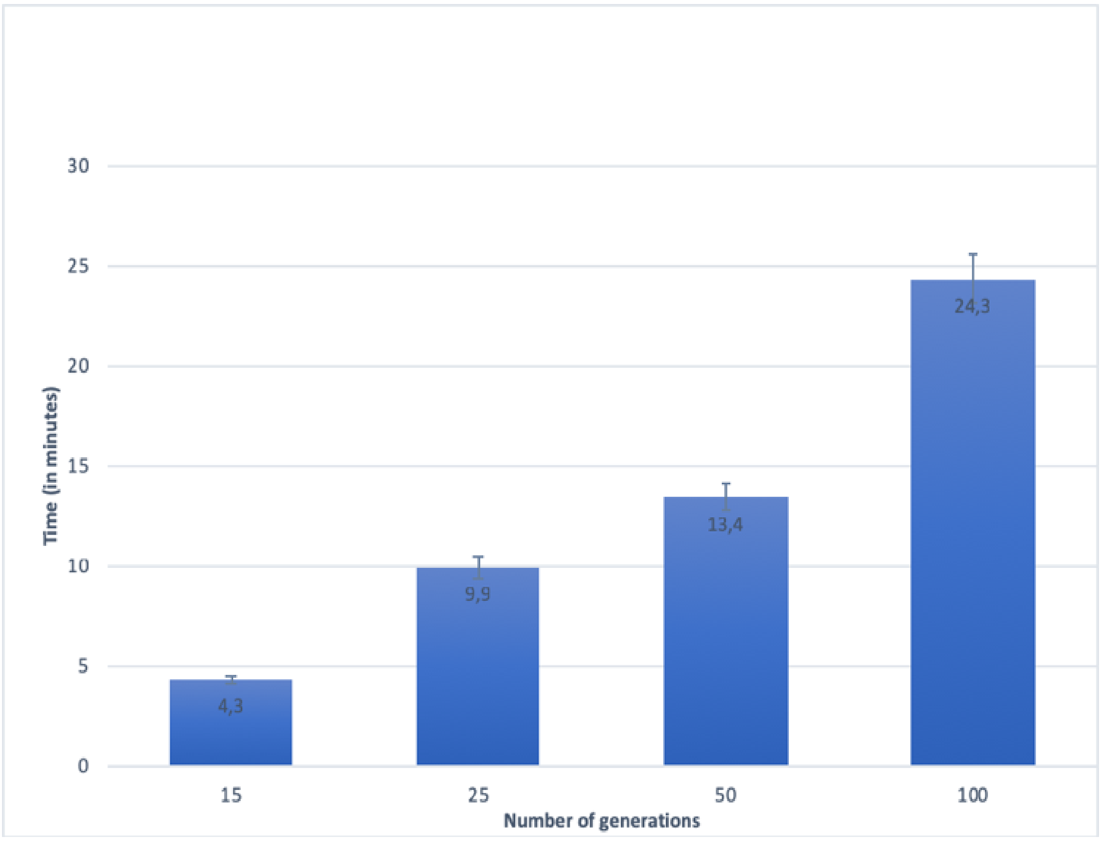


Figure 5.6: Effect of an increasing number of generations on the average time taken for the framework.

5.1.4 CPU Performance

Figure 5.7 compares the average CPU load performance with the ngen, npop, nchromo. Here, ngen, npop and nchromo is a number of generations (11, 15, 20, 32, 45, 65), number of population/evaluations (15, 20, 13, 19, 33, 41) and number of chromosomes (10, 15, 25, 27, 45, 74). From figure 4.5, we can conclude that the CPU load percentage value will increase and may vary when ngen, npop, nchromo values are increased.

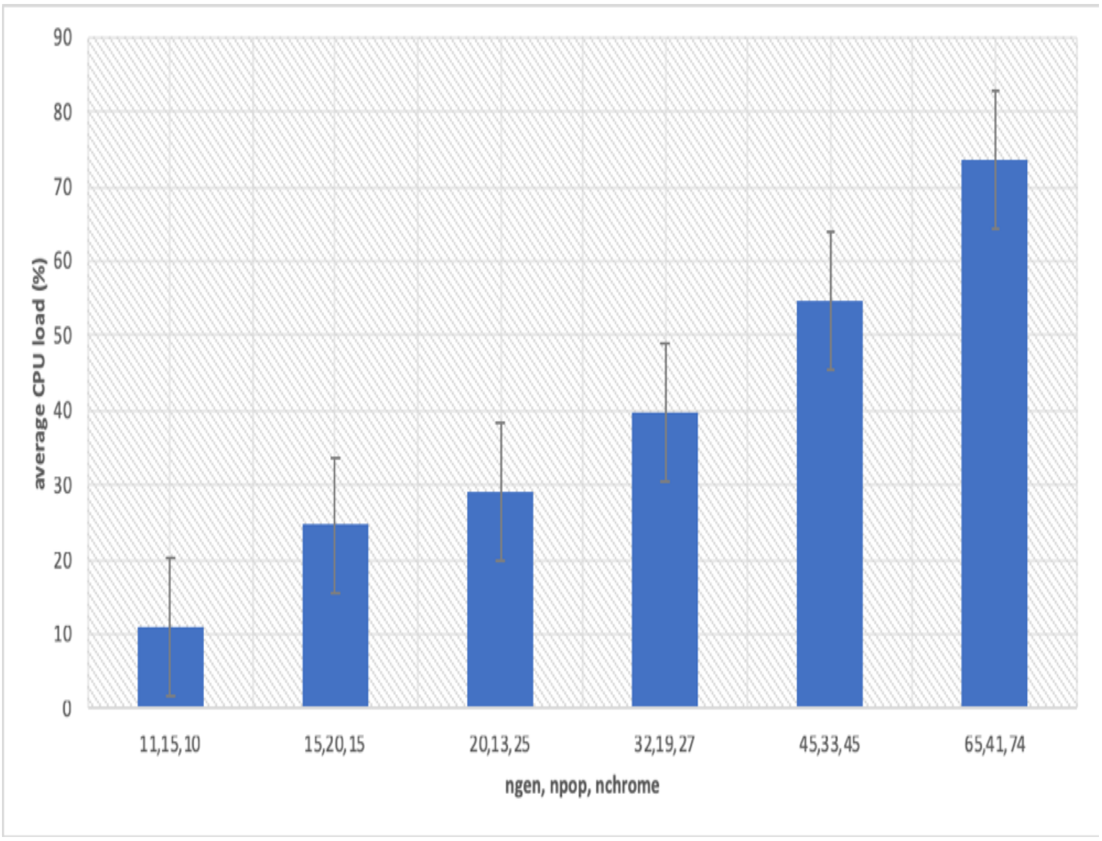


Figure 5.7: Comparison of CPU performance with ngen, npop, nchromo.

5.2 Analysis

In this section, we will analyze the results from the previous section.

From Figure 5.1, we can see that increasing the initial number of chromosomes per genome increases the length of the test suite. Increasing the initial number of chromosomes also increases the total number of chromosomes in the genome pool. There are more choices of chromosomes for the algorithm, so therefore it can be iter out the weakest ones with the better ones.

As we increase the initial number of chromosomes per genome, the average time taken to run the framework for 5 generations increases and our input size will be bigger since we have a higher number of chromosomes in our genome pool. The algorithm needs to evaluate fitness and do crossover to rejoin genomes and Computationally the system will have to do more processing due to this. This also increases the run time.

The slight increase in the length of the test suite as seen in Figure 5.1 can be explained by the high number of chromosomes per genome in each generation. Increasing the mutation threshold increases the length of the test suite and also reduces fitness. Mutation does not filter out the chromosomes using the fitness function which is done randomly. Increasing the mutation threshold fewer number of weaker genomes are filtered out as we do only a few crossovers. Therefore, the length reduction is lower. We also get a higher standard deviation as random chromosomes of different length are swapped by the algorithm using mutation.

The average time taken to run the framework decreases as we increase the mutation threshold for 10 generations which can be seen in Figure 5.4. Fewer crossovers are done using higher mutation threshold. Crossover requires rejoining genomes while mutation is just a swap. Hence mutations are faster than a crossover. Moreover, fewer crossovers decrease the possibility of having genomes with large length.

From Figure 5.5, we can see that the increase in the number of generations reduces the length of the output test suite as it increases the fitness of the output test suite. We do a number of crossovers that maximizes the fitness since we run the experiments for more generations. This is done by tournament selection. Among the randomly chosen genomes, parents chosen by tournament selection for crossover have the highest fitness. Therefore, the child generation has higher mean fitness than the parent generation during a cycle of the algorithm. The mean fitness for the child generation increase as they become the parent generation for the next cycle. This result as we run the algorithm for more generations.

Crossover increases fitness by reducing the length. The decrease in the length continues until the 85th generation and then eventually decreases after 90th generation. The algorithm has maximized the reduction and any further reduction might lead to a reduction in fault revelation as we do crossovers and mutations.

From Figure 5.6, we can see that increasing the number of generations also increases the average time taken to run by the framework. We will have genomes of longer length as we run the framework for more number of generations. The also increases the computation by the system. This results in an increase in the average time taken.

Chapter 6

Conclusion and Future work

The attempt of testing of web applications, was partly succeeded. This kind of robustness testing in our approach was strongly depended on the efficiency of our algorithm. We have also demonstrated how the generated test cases can be used for detecting faults in web application. From this approach, we can conclude that we have found as many faults as we can by reducing the test sequence length. In our approach, by using these two objectives, we can reduce the human effort, cost of testing and time-consuming. We also believe that our approach is a practical and useful testing tool since it was able to find 2 unique crashes by minimizing test sequence length to 8 in a single test of the web application. Furthermore, this framework can provide a basis for studying fault localization techniques and test suite minimization for JavaScript web applications.

6.1 Answers to Research Questions

RQ1: How can robustness testing of the web application will be automated and optimized to maximize defect identification in a BSS system?

Ans: The opinion of the user is considered as the key factor in the qualitative approach. But it is essential that the web applications need to be tested for robustness during the run time. This ensures that the web application is robust enough to be released to real users. The system can be reconfigured based on its robustness. In the quantitative approach, the Experimental generated approach is considered as a suitable approach, as described in section 4.1. If n random actions can make an application fail, it's better to acknowledge it during testing, rather than letting users discover it. For this purpose, the gremlins monkey testing library is considered as a suitable technique to automatically test the web application robustly. To optimize and maximize defect identification in the subject application a search-based meta-heuristic algorithm is considered.

RQ1: How effective is the proposed approach?

Ans: The efficiency and effectiveness of our proposed approach can be measured by the cost-effective test suites. To measure the effectiveness, it is also necessary to detect the parameters that affect the results. The parameters contributing to degrading the performance of the framework and the algorithm are identified based on each test case length and the time taken by the framework on every run. The effect of increasing the number of chromosomes on the result test suite is clearly shown and analyzed in section 5. The effect of mutation threshold can also be considered as another parameter as it increases the total time to run the framework. Increasing the number of generations will yield more cost-effective test suite. Increasing the initial number of chromosomes per genome will increase the resulting test suite's ability to find maximum faults. Since our automated test case generation technique is language independent; our techniques can be easily extended to other web-based technologies.

6.2 Future work

Future work in evaluating the algorithm and improving the model include:

- Coverage metric: In this thesis, we are considering two objectives fault revelation and length of the test sequence. To find more faults coverage can be considered as extra objective. Future work should be done with optimizing coverage for web applications.

- Pluggable fitness function: There might be various sections of the web application that are used frequently and also critically. By creating a new fitness function, we can calculate the fitness based on the most frequently used parts of the web application the test suite covers. This fitness function will increase the coverage and can be used as a high priority test suite that covers the most commonly used parts of the web application.

- Taking persistent state into account during crossover and mutation. Many faults occur due to corrupted data or trying to access data that does not exist. Future work should be done by taking the persistent states of the web app into consideration while making test cases.

- Implementing Mutation Analysis: Mutation analysis [43] inserts seeded faults into a software system, in the hope that test cases that reveal them may also reveal real faults. Mutation analysis can be used for run-time test optimization, in a way that evaluates the effectiveness of the test suites and eliminates those weak ones through potential multiple rounds of evolution.

- Evaluation: Do experiments on different applications. Evaluate test suites based on their ability to reveal faults.

References

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," Tech. Rep. 2, 2002. [Online]. Available: https://www.iitk.ac.in/kangal/Deb_NSGA-II.pdf

[2] J. Oh, S. Lee, and S. Lee, "DIGITAL FORENSIC RESEARCH CONFERENCE Advanced Evidence Collection and Analysis of Web Browser Activity." [Online]. Available: https://www.dfrws.org/sites/default/files/session-files/paper-advanced_evidence_collection_and_analysis_of_web_browser_activity.pdf

[3] Jim Conallen, "Modeling Web Application Architectures with UML," Tech. Rep. [Online]. Available: https://pdfs.semanticscholar.org/d68b/40db21c4d7db05b9dfb67e06e54f1d76d0f5.pdf

[4] M. R. Prasad, R. L. Naik, and V. Bapuji, "Cloud Computing : Research Issues and Implications," *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, vol. 2, no. 2, 1 2013. [Online]. Available: http://www.iaesjournal.com/online/index.php/IJ-CLOSER/article/view/1963

[5] "03.120.10 - Quality management and quality assurance." [Online]. Available: https://www.iso.org/ics/03.120.10/x/

[6] G. J. Myers, T. Badgett, and T. M. Thomas with Corey Sandler, "The Art of Software Testing, Second Edition," Tech. Rep., 2004. [Online]. Available: www.Wiley.com.

[7] G. M. D. Gandhi and A. S. Pillai, "Challenges in GUI Test Automation," *International Journal of Computer Theory and Engineering*, vol. 6, no. 2, pp. 192–195, 2 2014.

[8] A. Shahrokni and R. Feldt, "A Systematic Review of Software Robustness," Tech. Rep. [Online]. Available: http://www.robertfeldt.net/publications/shahrokni_2013_sysrev_robustness.pdf

[9] K. Mao, "Multi-objective Search-based Mobile Testing," Tech. Rep., 2017. [Online]. Available: http://discovery.ucl.ac.uk/1553273/2/ke_phd_thesis_final.pdf

[10] J. Dahmann, J. A. Lane, G. Rebovich, and R. Lowry, "Systems of systems test and evaluation challenges," in *2010 5th International Conference on System of Systems Engineering.* IEEE, 6 2010, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/5543979/

[11] M. Fewster and D. Graham, *Software test automation : effective use of test execution tools.* Addison-Wesley, 1999. [Online]. Available: https://www.academia.edu/17763647/Software_Test_Automation_Effective_Use_of_Test_Execution_Tools._By_Mark_Fewster_and_Dorothy_Graham._Published_by_Addison_Wesley_Harlow_Essex_U.K._1999._ISBN_0_201_33140_3_574_pages._Price_U.K._26.95_Soft_Cover

[12] B. P. Kar, "TEST SUIT REDUCTION BY FINDING COST OPTIMAL REPRESENTATIVE SET." [Online]. Available: https://www.academia.edu/7907093/TEST_SUIT_REDUCTION_BY_FINDING_COST_OPTIMAL_REPRESENTATIVE_SET

[13] M. Douglas Jacyntho, D. Schwabe, and G. Rossi, "A Software Architecture for Structuring Complex Web Applications," Tech. Rep. [Online]. Available: http://www-di.inf.puc-rio.br/schwabe/papers/OOHDMJava2%20Report.pdf

[14] C. Hutchison, M. Zizyte, P. E. Lanigan, D. Guttendorf, M. Wagner, C. Le Goues, P. Koopman, and C. L. Goues, "Robustness Testing of Autonomy Software," vol. 10, 2018. [Online]. Available: https://doi.org/10.1145/3183519.3183534

[15] F.-A. Fortin, U. Marc-André Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy François-Michel De Rainville," Tech. Rep., 2012. [Online]. Available: http://deap.gel.ulaval.ca,

[16] "Web-based application development." [Online]. Available: https://futureforward.nl/en/blog/web-based-application-development/

[17] "Web Application Architecture | Existek Blog." [Online]. Available: https://existek.com/blog/web-application-architecture/

[18] "Web application architecture: Components, models and types." [Online]. Available: https://www.scnsoft.com/blog/web-application-architecture

[19] "What is HTTP Request, Request Line, Request Header &amp; Request Body?" [Online]. Available: https://www.toolsqa.com/client-server/http-request/

[20] "What are web applications and dynamic web pages." [Online]. Available: https://helpx.adobe.com/dreamweaver/using/web-applications.html

[21] S. Al-Zain, D. Eleyan, and Y. Hassouneh, "Comparing GUI Automation Testing Tools for Dynamic Web Applications," Tech. Rep., 2013. [Online]. Available: www.ajouronline.com

[22] "User session management concepts." [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSPREK_9.0.0/com.ibm.isam.doc/wrp_config/concept/con_usr_sess_mgmt_conc.html

[23] ACM Sigsoft. and Institute of Electrical and Electronics Engineers., *ESEM 2009 : 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM : October 15-15, 2009, Lake Buena Vista, Florida USA.* IEEE, 2009. [Online]. Available: https://www.researchgate.net/publication/221494931_How_do_testers_do_it_An_exploratory_study_on_manual_testing_practices

[24] P. Mahajan, H. Shedge, and U. Patkar, "Automation Testing In Software Organization," *International Journal of Computer Applications Technology and Research*, vol. 5, no. 4, pp. 198–201, 4 2016. [Online]. Available: http://ijcat.com/archives/volume5/issue4/ijcatr05041004.pdf

[25] E. Enoiu, D. Sundmark, A. Causevic, and P. Pettersson, "A Comparative Study of Manual and Automated Testing for Industrial Control Software," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST).* IEEE, 3 2017, pp. 412–417. [Online]. Available: http://ieeexplore.ieee.org/document/7927994/

[26] D. Graham and M. Fewster, "Praise for Experiences of Test Automation," Tech. Rep. [Online]. Available: http://ptgmedia.pearsoncmg.com/images/9780321754066/samplepages/0321754069.pdf

[27] J.-C. Fernandez, L. Mounier, and C. Pachon, "A Model-Based Approach for Robustness Testing," 2005, pp. 333–348. [Online]. Available: http://link.springer.com/10.1007/11430230_23

[28] W. Johansson, M. Svensson, U. E. Larson, M. Almgren, and V. Gulisano, "T-Fuzz: Model-Based Fuzzing for Robustness Testing of Telecommunication Protocols," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation.* IEEE, 3 2014, pp. 323–332. [Online]. Available: http://ieeexplore.ieee.org/document/6823894/

[29] P. Koopman, K. Devale, and J. Devale, "Interface Robustness Testing: Experience and Lessons Learned from the Ballista Project," in *Dependability Benchmarking for Computer Systems.* Hoboken, NJ, USA: John Wiley & Sons, Inc., pp. 201–226. [Online]. Available: http://doi.wiley.com/10.1002/9780470370506.ch11

[30] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, “Evaluating Fuzz Testing,” p. 16, 2018. [Online]. Available: https://doi.org/10.1145/3243734.3243804

[31] P. Tsankov, M. T. Dashti, and D. Basin, “SECFUZZ: Fuzz-testing Security Protocols,” Tech. Rep. [Online]. Available: http://www.infsec.ethz.ch/research/software/.

[32] P. Mcminn, “Search-Based Software Testing: Past, Present and Future,” Tech. Rep. [Online]. Available: https://mcminn.io/publications/c18.pdf

[33] R. Roshan and C. M. Sharma, “Review of Search based Techniques in Software Testing Rabins Porwal ITS Ghaziabad UP,INDIA,” Tech. Rep. 6, 2012. [Online]. Available: https://pdfs.semanticscholar.org/95d0/365c987e5ccd3a23536c0828ebf35d9c58f3.pdf

[34] “An Intro to Monkey Testing with Gremlins.js | CSS-Tricks.” [Online]. Available: https://css-tricks.com/intro-monkey-testing-gremlins-js/

[35] S. Gojare, R. Joshi, and D. Gaigaware, “Analysis and Design of Selenium WebDriver Automation Testing Framework,” *Procedia Computer Science*, vol. 50, pp. 341–346, 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050915005396

[36] J. McCall, “Genetic algorithms for modelling and optimisation,” *Journal of Computational and Applied Mathematics*, vol. 184, no. 1, pp. 205–222, 12 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377042705000774

[37] K. Mao, M. Harman, and Y. Jia, “Sapienz: Multi-objective Automated Testing for Android Applications,” Tech. Rep. [Online]. Available: http://github.com/Rhapsod/sapienz

[38] M. Harman, Y. Jia, and Y. Zhang, “Achievements, open problems and challenges for search based software testing,” Tech. Rep. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/mharman/icst15.pdf

[39] C. Cadar and K. Sen, “Symbolic Execution for Software Testing: Three Decades Later,” Tech. Rep. [Online]. Available: https://people.eecs.berkeley.edu/~ksen/papers/cacm13.pdf

[40] S. Artzi, J. Dolby, S. Holm Jensen, A. Møller, and F. Tip, *A Framework for Automated Testing of JavaScript Web Applications*, 2011. [Online]. Available: www.dhtmlgoodies.com

[41] S. Mirshokraie, A. Mesbah, and K. Pattabiraman, “JSEFT: Automated JavaScript Unit Test Generation,” Tech. Rep. [Online]. Available: http://salt.ece.ubc.ca/publications/docs/icst15.pdf

[42] H. H. Elkatawneh, “Comparing Qualitative and Quantitative Approaches,” *SSRN Electronic Journal*, 2016. [Online]. Available: http://www.ssrn.com/abstract=2742779

[43] K. Nishiura, Y. Maezawa, H. Washizaki, and S. Honiden, “Mutation Analysis for JavaScript Web Application Testing,” Tech. Rep. [Online]. Available: http://www.honiden.nii.ac.jp/

Appendix A
Source code for our approach

A.1 Python based REST API to log the browser interactions

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from urllib.parse import unquote
app = Flask(__name__)
CORS(app)
@app.route("/")
def log():
    with open("genes.csv", "a") as f:
        s = unquote(request.args.get("event"))
        if s.find('fps') != -1:
            return jsonify({})
        f.write(unquote(request.args.get("event")))
        f.write("\n")
    f.close()
    return jsonify({})
```

A.2 NSGA-2 algorithm for our approach

```
import random
from deap import base
from deap import creator
from deap import tools
import array
import random
import json
import numpy as np
from math import sqrt
from deap import algorithms
from deap import base
from deap import benchmarks
from deap.benchmarks.tools import diversity, convergence
from deap.benchmarks.tools import hypervolume
from deap import creator
from deap import tools
from random import shuffle
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from  selenium.webdriver.common.action_chains import ActionChains
import argparse
from read_gremlins_logs import parse_line, Ind
from datetime import datetime
from selenium.webdriver.common.touch_actions import TouchActions
from time import sleep, time
from peewee import *
import os
from selenium.webdriver.chrome.options import Options
db = SqliteDatabase('errors.db')

class ErrorModel(Model):
    timestamp = IntegerField()
    sequence_length = IntegerField()
    number_errors = IntegerField()
    sequence = TextField()
    errors = TextField()
    class Meta:
        database = db # This model uses the "people.db" database.
db.create_tables([ErrorModel])
parser = argparse.ArgumentParser()
parser.add_argument(
    "--url",
    help="echo the url you use here")
parser.add_argument(
    "--ngen",
    default=10,
    help="echo the number of generations you use here")
parser.add_argument(
    "--npop",
    default=2,
    help="echo the number of populations you use here")
parser.add_argument(
    "--nchromo",
    default=2,
    help="echo the number of chromosomes you use here")
parser.add_argument(
    "--headless",
    default=1,
    help="headless evaluation; 0 for no, 1 for yes")
args = parser.parse_args()
num_chromosomes = int(args.nchromo)
creator.create("FitnessMulti", base.Fitness, weights=(-1.0, 1.0))
creator.create("TestCase", list, fitness=creator.FitnessMulti,
testcase=None, best=None)
to_cur = 0

def read_large_file(file_object):
    global to_cur
    """
    Uses a generator to read a large file lazily
    """
    while True:
        skip = 0
        while skip <= to_cur:
            data = file_object.readline()
            skip += 1
        to_cur = skip
        if not data:
            break
        yield data

def initTestCase(size):
    ret = []
    path = "genes.csv"
    try:
        with open(path) as file_handler:
            i = 0
            for line in read_large_file(file_handler):
                # process line
                ret.append(line)
                i += 1
                if i >= size:
                    break
    except (IOError, OSError):
        print("Error opening / processing file")
    return ret

toolbox = base.Toolbox()
toolbox.register("testcase", initTestCase,
creator.TestCase, size=num_chromosomes)
```

```
def uniform(low, up, size=None):
    try:
        return [random.uniform(a, b) for a, b in zip(low, up)]
    except TypeError:
        return [random.uniform(a, b) for a, b in zip([low] * size,
        [up] * size)]

toolbox.register("attr_generator", initTestCase, num_chromosomes)
toolbox.register("individual", tools.initIterate,
creator.TestCase, toolbox.attr_generator)
toolbox.register("population", tools.initRepeat, list,
toolbox.individual)

def evaluate(individual, report=False, driver=None):
    errors = []
    # close_after_eval = False
    if not driver:
        # close_after_eval = True
        chrome_options = Options()
        if bool(int(args.headless)):
            chrome_options.add_argument("--headless")
        driver = webdriver.Chrome(executable_path="./chromedriver",
        chrome_options=chrome_options)
    driver.get(args.url)
    """coverageJSON = driver.execute_script("return
    jscoverage_serializeCoverageToJSON();")
    file = open("jscoverage.json","w")
    file.write(coverageJSON)
    file.close()"""
    dict_dims = driver.get_window_size()
    max_width = dict_dims['width']
    max_height = dict_dims['height']
    for tc in individual:
        if type(tc) is str:
            o = parse_line(tc)
        else:
            o = tc
        if o is None:
            continue
        o.normalize_dimensions(max_width, max_height)
        if o.event == "type" or o.event == "input":
            elems = driver.find_elements_by_tag_name(o.dom)
            for elem in elems:
                try:
                    if elem and elem.is_displayed():
                        elem.send_keys(o.input)
                        elem.send_keys(Keys.RETURN)
                except:
                    actions = ActionChains(driver)
                    actions.send_keys(Keys.RETURN)
                    actions.perform()
                    # sleep(25)
        elif o.event == "scroll":
            actions = TouchActions(driver)
            actions.scroll(o.locx, o.locy)
            try:
                actions.perform()
            except:
                pass
        elif (o.event == "tap" or o.event == "doubletap"
        or o.event == "multitouch"):
            actions = TouchActions(driver)
            actions.tap_and_hold(o.locx, o.locy)
            try:
                actions.perform()
            except:
                pass
        else:
            actions = ActionChains(driver)
            actions.move_by_offset(o.locx, o.locy)
            actions.click()
            try:
                actions.perform()
            except:
                pass
    for entry in driver.get_log('browser'):
        errors.append(entry)
    # if close_after_eval:
    #     driver.close()
    obj1 = len(individual)
    obj2 = len(errors)
    if report is not False:
        report.write("Number of Errors: {} \n".format(len(errors)))
        report.write("Test Cases Length: {} \n"
        .format(len(individual)))
        report.write("Errors Found:\n")
        report.write(str(errors))
        report.write("\n")
        report.write("Test Cases:\n")
        report.write(str(list(individual)))
        row = ErrorModel(timestamp=int(time()),
        sequence_length=len(individual), number_errors=len(errors),
                     sequence=str(list(individual)),
                     errors=str(errors))
        row.save()
    return obj1, obj2

def mate(a, b):
    shuffle(a)
    new_genes = random.choice([8, 16, 32, 64, 128])
    for i in range(new_genes):
        a.insert(np.random.randint(0, len(a)), synthesize())
    choice = np.random.randint(0, 2)
    if choice == 0:
        ret = a.extend(b)
    elif choice == 1:
        ret = b.extend(a)
    return ret

def synthesize():
    if np.random.randint(0, 1e3) % 2 == 0:
        tc = Ind(_e="click")
    elif np.random.randint(0, 1e3) % 3 == 0:
        tc = Ind(_e="scroll")
    else:
        tc = Ind(_e="type")
    tc.fuzzy()
    return tc

MUTPB = 0.3

def mutate(individual, indpb):
    # shuffle seq
    individual, = tools.mutShuffleIndexes(individual, indpb)
    # crossover inside the suite
    for i in range(1, len(list(individual)), 2):
        if random.random() < MUTPB:
            if len(list(individual)) <= 2:
                continue  # sys.exit(1)
            if len(list(individual)) <= 2:
                continue  # sys.exit(1)
            individual[i - 1],
            individual[i] = tools.cxBlend(individual[i - 1],
            individual[i], 0.7)
    # shuffle events
    for i in range(len(list(individual))):
        if random.random() < MUTPB:
            if len(list(individual)) <= 2:
                continue  # sys.exit(1)
            list(individual)[i], = tools.mutShuffleIndexes
            (list(individual)[i], indpb)
    return individual

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.5)
toolbox.register("select", tools.selNSGA2)

def main(seed=None):
    random.seed(seed)
    NGEN = int(args.ngen)
    MU = int(args.npop)
    CXPB = 0.9
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean, axis=0)
    # stats.register("std", np.std, axis=0)
    stats.register("min", np.min, axis=0)
    stats.register("max", np.max, axis=0)
    logbook = tools.Logbook()
    logbook.header = "gen", "evals", "min", "avg", "max"
    pop = toolbox.population(n=MU)
    # Evaluate the individuals with an invalid fitness
    invalid_ind = [ind for ind in pop if not ind.fitness.valid]
    fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit
    '''This is just to assign the crowding
    distance to the individuals'''
    # no actual selection is done
    pop = toolbox.select(pop, len(pop))
    record = stats.compile(pop)
    logbook.record(gen=0, evals=len(invalid_ind), **record)
    print(logbook.stream)
    # Begin the generational process
    for gen in range(1, NGEN):
        # Vary the population
        offspring = tools.selTournamentDCD(pop, len(pop))
        offspring = [toolbox.clone(ind) for ind in offspring]
        for ind1, ind2 in zip(offspring[::2], offspring[1::2]):
            if random.random() <= CXPB:
                toolbox.mate(ind1, ind2)
            toolbox.mutate(ind1)
            toolbox.mutate(ind2)
            del ind1.fitness.values, ind2.fitness.values
        # Evaluate the individuals with an invalid fitness
        invalid_ind =
        [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit
        # Select the next generation population
        pop = toolbox.select(pop + offspring, MU)
        record = stats.compile(pop)
        logbook.record(gen=gen, evals=len(invalid_ind), **record)
        print(logbook.stream)
    print("Final population hypervolume is %f" % hypervolume
    (pop, [11.0, 11.0]))
    return pop, logbook
if __name__ == "__main__":
    pop, stats = main()
    chrome_options = Options()
    if bool(int(args.headless)):
        chrome_options.add_argument("--headless")
    driver = webdriver.Chrome(executable_path="./chromedriver",
    chrome_options=chrome_options)
    driver.get(args.url)
    with open("report_{:%B_%d_%Y}.txt".format(datetime.now()),
    "w") as f:
        for ind in pop:
            f.write(str(ind))
            f.write("\n")
            evaluate(ind, f, driver)
    driver.close()
    #os.rename("genes.csv", "genes-{}.csv".format(datetime.now()))
    import matplotlib.pyplot as plt
    gen, avg, min_, max_ = stats.select("gen", "avg", "min", "max")
    np_min = np.array(min_)
    np_avg = np.array(avg)
    np_max = np.array(max_)
    fig, axes = plt.subplots(2, 3)
    # first objective
    axes[0, 0].plot(gen, np_avg[:, 0], label="average length")
    axes[0, 0].set_title("Average Test Case Length")
    axes[0, 0].set_xlabel("Generation")
    axes[0, 0].set_ylabel("Objective #1: Test Case Length")
    axes[0, 1].plot(gen, np_min[:, 0], label="minimum length")
    axes[0, 1].set_title("Minimum Test Case Length")
    axes[0, 1].set_xlabel("Generation")
    axes[0, 1].set_ylabel("Objective #1: Test Case Length")
    axes[0, 2].plot(gen, np_max[:, 0], label="maximum length")
    axes[0, 2].set_title("Maximum Test Case Length")
    axes[0, 2].set_xlabel("Generation")
    axes[0, 2].set_ylabel("Objective #1: Test Case Length")
    # second objective
    axes[1, 0].plot(gen, np_avg[:, 1], label="average errors")
    axes[1, 0].set_title("Average Error Revelation")
    axes[1, 0].set_xlabel("Generation")
    axes[1, 0].set_ylabel("Objective #2: Number of Errors")
    axes[1, 1].plot(gen, np_min[:, 1], label="minimum errors")
    axes[1, 1].set_title("Minimum Error Revelation")
    axes[1, 1].set_xlabel("Generation")
    axes[1, 1].set_ylabel("Objective #2: Number of Errors")
    axes[1, 2].plot(gen, np_max[:, 1], label="maximum errors")
    axes[1, 2].set_title("Maximum Error Revelation")
    axes[1, 2].set_xlabel("Generation")
    axes[1, 2].set_ylabel("Objective #2: Number of Errors")
    plt.tight_layout()
    plt.show()
```

A.3 To extract browser logs

```
import os
import json
import numpy as np
from chomsky import chomsky

class Ind:
    event = None
    input = None
    locx = None
    locy = None
    duration = None
    distancex = None
    distancey = None
    scale = None
    rotation = None
    radius = None
    dom = None
    def __init__(self, _e, _i=None, _lx=None, _ly=None, _du=None,
    _dx=None, _dy=None,
                 _scale=None, _rotation=None, _radius=None,
                 _dom=None):
        self.event = _e
        self.input = _i
        self.locx = _lx
        self.locy = _ly
        self.duration = _du
        self.distancex = _dx
        self.distancey = _dy
        self.scale = _scale
        self.rotation = _rotation
        self.radius = _radius
        self.dom = _dom

    def __str__(self):
        return "{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}".format(
            self.event, self.input, self.locx, self.locy,
            self.duration, self.distancex, self.distancey,
            self.scale, self.rotation, self.radius,
            self.dom
        )

    def fuzzy_location(self):
        if self.locx is not None and self.locx < 500:
            self.locx += np.random.randint(0, 100)
        elif self.locx is not None:
            self.locx -= np.random.randint(0, 100)
        else:
            self.locx = np.random.randint(0, 500)
        if self.locy is not None and self.locy < 500:
            self.locy += np.random.randint(0, 100)
        elif self.locy is not None:
            self.locy -= np.random.randint(0, 100)
        else:
            self.locy = np.random.randint(0, 500)
    def fuzzy_input(self):
        self.input = chomsky(np.random.randint(3, 20))
        self.dom = "input"

    def fuzzy(self):
        if self.event == "type" or self.event == "input":
            self.fuzzy_input()
        else:
            self.fuzzy_location()
    def normalize_dimensions(self, max_width, max_height):
        if self.locx is not None and int(self.locx) >= max_width:
            self.locx = max_width - 50
        if self.locy is not None and int(self.locy) >= max_height:
            self.locy = max_height - 50

def parse_line(l):
    parts = l.split()
    e = parts[1]
    o = None
    if e == "type":
        o = Ind(_e=e, _i=chomsky(np.random.randint(3, 5)),
        _dom='input')
    elif e == "click" or e == "scroll" \
            or e == "mousedown" or e == "mouseout" \
            or e == "mouseover" or e == "dblclick" \
            or e == "mousemove" or e == "mouseup":
        o = Ind(_e=e, _lx=parts[3], _ly=parts[4])
    elif e == "gesture":
        params = json.loads(parts[5])
        o = Ind(_e=e, _lx=parts[3], _ly=parts[4],
                _dx=params['distanceX'],
                _dy=params['distanceY'],
                _du=params['duration'])
    elif e == "tap" or e == "doubletap":
        params = json.loads(parts[5])

        o = Ind(_e=e, _lx=parts[3], _ly=parts[4],
                _du=params['duration'])
    elif e == "multitouch":
        params = json.loads(parts[5])

        o = Ind(_e=e, _lx=parts[3], _ly=parts[4],
                _dx=params['distanceX'],
                _dy=params['distanceY'],
                _du=params['duration'],
                _scale=params['scale'],
                _rotation=params['rotation'],
                _radius=params['radius']
                )
    elif e == "input":
        o = Ind(_e=e, _i=chomsky(np.random.randint(3, 7)),
        _dom='input')
    return o
```

```
def read_log(path):
    if not os.path.exists(path):
        return None

    events = set()
    atomic_sequences = []

    with open(path, "r") as f:
        lines = f.readlines()
        for l in lines:
            if l.find("VM26 gremlinsClient.js:") == -1:
                continue
            parts = l.split()
            if parts[2] == "gremlin":
                # print(parts[5:])
                events.add(parts[4])
                e = parts[4]
                o = None
                if e == "type":
                    o = Ind(_e=e, _i=parts[5], _lx=parts[7],
                    _ly=parts[8])
                elif e == "click" or e == "scroll"\
                        or e == "mousedown" or e == "mouseout"\
                        or e == "mouseover" or e == "dblclick"\
                        or e == "mousemove" or e == "mouseup":
                    o = Ind(_e=e, _lx=parts[6], _ly=parts[7])
                elif e == "gesture":
                    o = Ind(_e=e, _lx=parts[6], _ly=parts[7],
                    _dx=int(parts[9].replace(",", "")),
                            _dy=int(parts[11].replace(",", "")),
                            _du=int(parts[13].replace("}", "")))
                elif e == "tap" or e == "doubletap":
                    o = Ind(_e=e, _lx=parts[6], _ly=parts[7],
                    _du=int(parts[9].replace("}", "")))
            elif e == "multitouch":
                o = Ind(_e=e, _lx=parts[6], _ly=parts[7],
                _dx=int(parts[15].replace(",", "")),
                        _dy=int(parts[17].replace(",", "")),
                        _scale=float(parts[9].replace(",","")),
                        _rotation=int(parts[11].replace(",","")),
                        _radius=int(parts[13].replace(",",""))
                        )
                elif e == "input":
                    o = Ind(_e=e, _i=parts[5], _dom=''.join(str(e)
                    for e in parts[7:]))

                if o is not None:
                    print(o)
                    atomic_sequences.append(o)
```

A.4 Main file to start the framework

```
import os
import shutil
import subprocess
import argparse
from read_gremlins_logs import parse_line
from multiprocessing import Pool
from time import sleep
parser = argparse.ArgumentParser()
parser.add_argument(
    "--url",
    help="echo the url you use here")
parser.add_argument(
    "--ngen",
    default=10,
    help="echo the number of generations you use here")
parser.add_argument(
    "--npop",
    default=2,
    help="echo the number of population you use here")
parser.add_argument(
    "--nchromo",
    default=2,
    help="echo the number of chromosomes you use here")
parser.add_argument(
    "--headless",
    default=1,
    help="headless evaluation; 0 for no, 1 for yes")
args = parser.parse_args()

def create_if_not_exists(dir):
    if os.path.exists(dir):
        return
    os.mkdir(dir)

def open_file_replace(file_path, old, new):
    with open(file_path, "r") as f:
        content = f.read()
        print(content)
    f.close()
    new_content = content.replace(old, new)
    with open(file_path, "w") as f:
        f.write(new_content)
        print(new_content)
    f.close()

def init_gremlins(blackbox_url, instances, headless,
devtools, timeout):
    # clone repo
    repo_url = "https://github.com/karthik3583/gremlinsmonkey"
    gremlins_dir = "gremlinsmonkey"
    print("RUN: {}".format("Locating Gremlins Puppet Repository"))
    if not os.path.exists(gremlins_dir):
        clone_cmd = "git clone {}".format(repo_url)
        print("RUN: {}".format(clone_cmd))
        os.system(clone_cmd)
    else:
        print("Gremlins Puppet Repository Found")
    # start logs flask server
    flask_cmd = "pip install -r requirements.txt &&
    export FLASK_APP=app.py && nohup flask run > noflask.out &"
    print("RUN: {}".format(flask_cmd))
    os.system(flask_cmd)
    # npm install
    chdir_cmd = "cd {} && git pull origin master"
    .format(gremlins_dir)
    install_cmd = "{} && npm install &&
    npm audit fix".format(chdir_cmd)
    print("RUN: {}".format(install_cmd))
    os.system(install_cmd)
    # edit configs
    default_configs_path = "{}/config.default"
    .format(gremlins_dir)
    custom_configs_path = "{}/config".format(gremlins_dir)
    shutil.rmtree(custom_configs_path, ignore_errors=True)
    shutil.copytree(default_configs_path, custom_configs_path)
    # page config file
    page_config_path = "{}/config/page.js".format(gremlins_dir)
    open_file_replace(page_config_path,
    "http://localhost:8888/home", blackbox_url)
    # gremlins config file
    page_config_path = "{}/config/gremlins.js"
    .format(gremlins_dir)
    open_file_replace(page_config_path, "instances: 100000,",
    "instances:{},".format(instances))
    open_file_replace(page_config_path, "timeout: 100000,",
    "timeout:{},".format(timeout))
    # browser config file
    if headless:
        page_config_path = "{}/config/browser.js"
        .format(gremlins_dir)
        open_file_replace(page_config_path, "headless: false,",
        "headless: true,")
    if devtools:
        page_config_path = "{}/config/browser.js"
        .format(gremlins_dir)
        open_file_replace(page_config_path, "devtools: false,",
        "devtools: true,")
    # run gremlins
    run_cmd = "{} && nohup npm run start --url {}
    &".format(chdir_cmd, blackbox_url)
    print("RUN: {}".format(run_cmd))
    p = subprocess.call(run_cmd, shell=True)
    # run evolutionary
    run_cmd = "python utils.py --url {} --ngen {} --npop {}
    --nchromo{}--headless {}".format(args.url, args.ngen,
    args.npop, args.nchromo, args.headless)
    print("RUN: {}".format(run_cmd))
    sleep(5*60)
    os.system(run_cmd)

def main():
    init_gremlins(args.url,
                  instances=5,
                  headless=False,
                  devtools=False,
                  timeout=60*60*1)
if __name__ == "__main__":
    main()
```