CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Achieving State Space Reduction in Generated Ajax Web Application State Machine

by

Nadeem Fakhar

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Computing
Department of Computer Science

2023

Achieving State Space Reduction in Generated Ajax Web Application State Machine

By

Nadeem Fakhar

(PC113002)

DEPARTMENT OF COMPUTER SCIENCE

CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

ISLAMABAD

2023

To my parents and teachers.

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY
ISLAMABAD
Expressway, Kahuta Road, Zone-V, Islamabad
Phone:+92-51-111-555-666 Fax: +92-51-4486705
Email: info@cust.edu.pk Website: https://www.cust.edu.pk

CERTIFICATE OF APPROVAL

This is to certify that the research work presented in the thesis, entitled "**Achieving State Space Reduction in Generated Ajax Web Application State Machine**" was conducted under the supervision of **Dr. Aamer Nadeem**. No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the **Department of Computer Science, Capital University of Science and Technology** in partial fulfillment of the requirements for the degree of Doctor in Philosophy in the field of **Computer Science.** The open defence of the thesis was conducted on **March 09, 2023**.

Student Name : Nadeem Fakhar (PC113002) [illegible]

The Examination Committee unanimously agrees to award PhD degree in the mentioned field.

Examination Committee :

(a)	External Examiner 1:	Dr. Majid Iqbal Khan Professor COMSATS University, Islamabad	[illegible]
(b)	External Examiner 2:	Dr. Yaser Hafeez Professor UIIT, AAU, Rawalpindi	[illegible]
(c)	Internal Examiner :	Dr. Nayyer Masood Professor CUST, Islamabad	[illegible]

Supervisor Name :	Dr. Aamer Nadeem Professor CUST, Islamabad	[illegible]
Name of HoD :	Dr. Abdul Basit Siddiqui Associate Professor CUST, Islamabad	[illegible]
Name of Dean :	Dr. Muhammad Abdul Qadir Professor CUST, Islamabad	[illegible]

AUTHOR'S DECLARATION

I, **Nadeem Fakhar (Registration No. PC113002)**, hereby state that my PhD thesis titled, '**Achieving State Space Reduction in Generated Ajax Web Application State Machine'** is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/ world.

At any time, if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my PhD Degree.

[illegible]

(Nadeem Fakhar)

Dated: 09 March, 2023

Registration No: PC113002

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled "**Achieving State Space Reduction in Generated Ajax Web Application State Machine**" is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/ cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of PhD Degree, the University reserves the right to withdraw/ revoke my PhD degree and that HEC and the University have the right to publish my name on the HEC/ University Website on which names of students are placed who submitted plagiarized thesis.

[illegible]

(Nadeem Fakhar)

Dated: 09 March, 2023

Registration No: PC113002

List of Publications

It is certified that following publication has been made out of the research work that has been carried out for this thesis:-

Journal publications:

1. **N. F. Malik**, A. Nadeem, and M. A. Sindhu,“ Achieving State Space Reduction in Generated Ajax Web Application State Machine”, *Intelligent Automation and Soft Computing*, vol. 33(1), pp. 429–455 , 2022.

Nadeem Fakhar Malik
(PC113002)

Acknowledgement

All praise is due to Allah Almighty, who has all knowledge and has the power to grant from his knowledge and peace, mercy and blessing upon his last and final messenger and upon good-doers.
I would never have been able to finish my work without the support from my respected mother, my brother, sisters, and my friends.

One of the most notable of Allah's blessings upon me was in the form of my supervisor. I would like to thank my supervisors, Professor/Head of Center for Software Dependability (CSD) research group Professor Dr. Aamer Nadeem for his guidance and support. I am also grateful to Dr. Muddassar Azam Sindhu, Associate Professor QAU, whose reviews helped me a lot to improve my work. I feel very lucky to be part of CSD research group members whose discussion and constructive criticism maintained an environment that was conducive for research. Moreover, without the recreational activities of our CSD research group, I may have gone insane over the last few years. I also thank the faculty members of Capital University of Science and Technology who gave me the resources and healthy education environment.
Apart from the mentioned above, many people helped me reach this point. May Allah grant all of them peace and bless them with prosperity.

Abstract

Ajax (Asynchronous JavaScript and XML) constructs dynamic web applications by using Asynchronous communication and run time Document Object Model (DOM) manipulation. Ajax involves extreme dynamism, which induces novel kind of issues like state explosion, triggering state changes and unreachable states etc. Finite State Machines (FSM) provide an effective way to model the behaviour of software. However, the state model generated for an Ajax application can be enormous and may be hit by state explosion problem. Recent research has not addressed this issue comprehensively because existing techniques either apply partial reduction or compromise the effectiveness of model. This research uses soft computing based Fuzzy C Means (FCM) clustering algorithm to generate state machine model of an Ajax web application. The focus is on devising a framework to avoid the state explosion problem. The framework prioritizes the requirements and use cases based on requirements weightage, stakeholder weightage and user session based use case frequency. FCM uses this data to reduce the state space by identifying the most pivotal usage areas. The resultant DOM mutations for only these usage areas are considered to induce the finite state machine thus avoiding the state explosion. The framework considers, not only, usage patterns and use cases but also software requirements, which is a distinctive contribution that has not been explored in prior researches. By incorporating software requirements into the state space reduction process, the proposed framework enables more accurate and effective reduction, leading to better optimization and performance of the web applications. The experimental results demonstrate that the framework achieved a reduction in overhead of 50 and a reduction in efficiency of 0.342(34) and a reduction in overhead of 0.809(81) and a reduction in efficiency of 0.385(39) in different case studies. These findings suggest that the framework is effective in mitigating the state space explosion problem while maintaining reasonable efficiency.

Contents

List of Figures

List of Tables

Abbreviations

AHP	Anylitical Hierarchy Process
AJAX	Asynchronous JavaScript and XML
BDD	Binary Decision Diagrams
CBR	Case-Based Ranking
CV	Cumulative Voting
DOM	Document Object Model
ED	Euclidean Distance
FCM	Fuzzy C Means
FGKA	Fast Genetic K-means Approach
FSM	Finite State Machine
GA	Genetic Algorithm
GKA	Genetic K-means Algorithm
HAHP	Hierarchy AHP
HCV	Hierarchical Cumulative Voting
MD	Mahalanobis Distance
MST	Minimal Spanning Tree
POFOD	Probability of Failure on Demand
RV	Requirement Value
SRP	Software Requirements Prioritization
VBSE	Value based Software Engineerig
VIRP	Value-based Intelligent Requirement Prioritization
XML	Extensible Markup Language

Chapter 1

Introduction

Lately advanced web technologies under the umbrella of web 2.0 have appeared and this advancement has transformed web applications into rich single page applications from existing static multi page applications [1, 2]. Modern society heavily relies on interactive and smart web applications, which must be reliable, enhanceable, and secure. The increasing complexity of current web applications implies extensive questions to their reliability. The static analysis of web applications' code provides significant perception to their reliability, however highly dynamic character of current web applications has made dynamic analysis more crucial [3, 4].

Ajax based applications are growing day by day. Ajax uses asynchronous mechanisms to interact with users via responsive, graphic rich, and interactive web-browsers [5]. Ajax applications use DOM to manipulate information, and XML to accomplish interoperability [6]. It presents information using HTML and CSS and achieves data access from server by XMLHttpRequest object. It executes JavaScript code upon callback activation [3]. Ajax works on distributed application framework principles.

Ajax technology induces better user interaction [7], but not without a cost. The asynchronous, event driven, stateful nature, use of loosely typed scripting language, client-side extensive working, and exchange of page portions instead of full-page exchange makes Ajax more error prone [8–10]. It is server-agnostic client-side approach and can work with various scripting languages which makes it fit for autonomous and heterogeneous environments [9, 11]. These technology blends require more effort to verify and maintain Ajax applications [9].

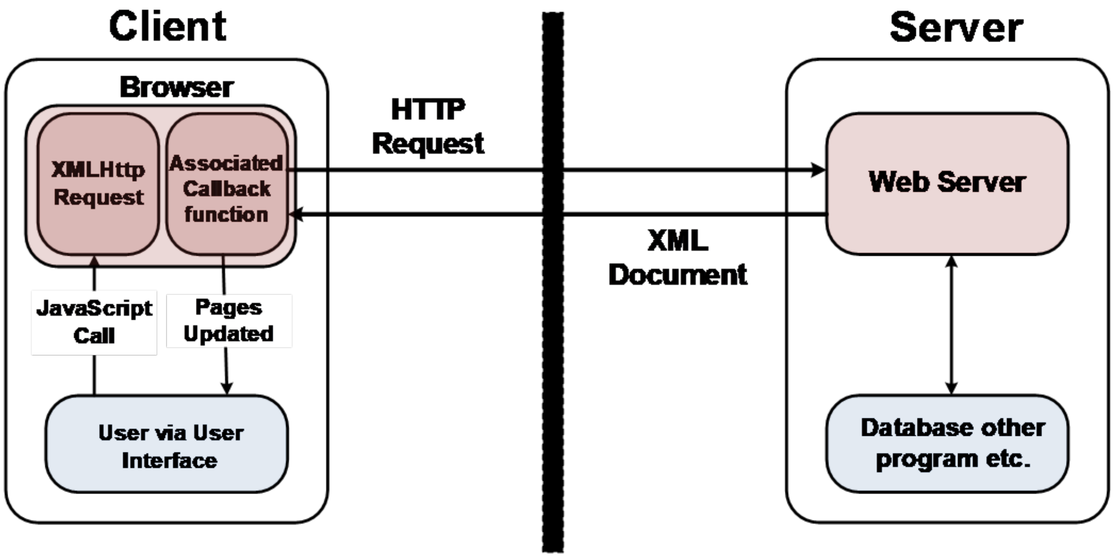


FIGURE 1.1: Ajax Web Application Working

Finite state machines (FSM) provide an effective way to model the behavior of software without going into its implementation details. Numerous earlier works have proposed methods to test applications using FSMs [8, 12, 13]. Just like all other desktop and web applications, Ajax applications can also be modeled by FSMs. Ajax web applications are single-page applications and theoretically, FSMs can model them completely but practically issues like state explosion, triggering state changes and unreachable states etc. are there to handle. Ajax applications process diverse user inputs as well as frequent client-server interactions resulting in abundant content change on the page. This causes number of DOM mutations leading to a large number of concrete states thus resulting in state explosion problem [3, 12]. Therefore, FSM-based method to model the system is only feasible if the FSM has limited states. Several state space reduction techniques have been proposed to avoid state explosion problem in different applications [14–16] but issues like partial reduction, processing overhead and effectiveness are still there to be addressed.

Verification and testing of web application proves to be a difficult task and the advent of Ajax applications has increased the complexity even further. Conventional techniques [17–21] lack in verification and testing of Ajax application features like asynchronous communication, client-server parallelism, and dynamic page segment updates. These features have added the issues like forward-back page navigation, enormous state changing elements, state explosion, and unreachable states.
To represent the behavior of a system without going into the implementation intricacies, one beneficial technique is to make use of finite state machines, as was just mentioned. By constructing the state machine, this research proposes a mechanism for simulating Ajax applications. In order to solve the critical issue of state space explosion, the methodology employs the utilization of a framework. Instead of modelling the complete system, the framework focuses on the most frequently used application components, which decreases the amount of state space that needs to be modeled. The method selects the most significant use cases by applying a fuzzy computing technique known as Fuzzy C Means (FCM) clustering, and thereafter creates a finite state machine for only those use cases. The approximation and learning nature of FCM makes it a good candidate to handle dynamic nature of Ajax application verification. These generated state machines can later be combined to construct an aggregated state machine of the given Ajax application.

1.1 Research Aims and Objectives

Like many software domains, web applications are becoming more complex. This complexity arises due to several factors, such as a larger number of hyperlinks, more complex interaction, and the increased use of distributed servers. The advent of Ajax has further added to this complexity due to its asyncronous nature and extreme dynamism. Modeling can help to understand these complex systems as it provides convenient and understandable graphical description of systems without going into the implementation details. System modeling helps in verification and validation at all stages of development. System modeling of Ajax applications face a significant issue of state explosion due to extreme dynamism of Ajax applications. The objectives in this research are to:

1. Identify and evaluate the limitations and inadequacies of the current techniques used for state space reduction in Ajax web applications, as reported in the existing literature.
2. Develop and propose an approach for constructing a reduced state machine that can effectively manage the state explosion problem in Ajax web applications.
3. Assess the effectiveness of the proposed approach in reducing the state machine size in Ajax applications and to determine the extent of its contribution towards achieving this goal.

1.2 Research Questions

RQ-1 What are the limitations or inadequacies, in the current techniques, employed for state space reduction in Ajax applications?
Objective: Our objective is to identify the weak areas in Ajax state space reduction techniques. These areas would then be addressed in our solution and an approach would be devised by filling the gaps in these areas.
Methodology: This research question will be answered by going through the current literature for its analysis. This findings of this analysis would then be evaluated to identify the gaps. These gaps would then be addressed in our solution.
This requirement is fully achieved in section 3.3 of chapter 3 of this thesis.
RQ-2 How to design an approach to build a reduced state machine?
Objective: State machines of applications notably Ajax applications commonly experience state explosion problem. Our objective is to design an approach that would help in construction of reduced state machine and would avoid state explosion.
Methodology: This question will be answered by looking through the literature

of past and current works and identifying the primary factors contributing to the state explosion problem in modeling Ajax applications. Existing solutions of this problem would also be looked upon and lastly we will focus on the most pivotal factors to be managed in order to find an efficient solution.
This requirement is fully achieved in chapter 4 of this thesis.

RQ-3 To what extent, does the proposed approach contribute to the reduction of the state machine in Ajax applications?

Objective: Computer scientists and software engineers face a major challenge in reducing state space. When state space is reduced, the effectiveness of a solution is often affected. Our goal is to establish a mechanism that, either by merging existing solutions or by developing a new one, can help reduce the state space of Ajax applications while having no or minimal impact on the solution's effectiveness.

Methodology: The existing state space reduction strategies, as well as their strengths and shortcomings, will be identified and evaluated to answer this question. Then, either by combining existing strategies or by establishing a new one, a state space reduction solution would be devised. Finally, experiments will be conducted to demonstrate that the space reduction approach is effective without compromising the solution's effectiveness.
This requirement is fully achieved in section 5.1 of chapter 5 of this thesis.

1.3 Existing Solutions

Existing solutions in the area of Ajax based state machine reduction consists of [6, 22]. Both these approaches are working to solve the state explosion problem in Ajax based applications. In [6] the authors have discussed the use of Binary Decision Diagrams (BDD) to avoid state explosion problem. In their approach the state machine is generated for the whole application and then is reduced. The mechanism starts by recording user sessions in xml log files and then by reading those files to generate the state machine for the Ajax application. The authors have claimed that the state machine is generated and reduced at the same time.

This mechanism imposes a constant overhead on the system by comparing and reducing the states all the time as the algorithm runs. Further the authors have not discussed the effects of this reduction on application testing, i.e., whether the reduced state machine has covered all the areas of the application under test. In [22] the authors have claimed that state explosion problem is handled as every session has got its own state machine. However this mechanism cannot guarantee in absolute about the handling of state explosion problem due to following reasons. Firstly the session of a large application can have large interacting events and corresponding changing elements resulting in state explosion. Secondly the framework constructs state machine for every session which is an overhead on the application.

1.4 Problem Statement

One of the key technologies facilitating today's dynamic web applications is Ajax. A significant challenge in exploring the state or search space of Ajax applications is state explosion problem.

Current solutions for state explosion problem majorly follow the approach of first exploring the whole state space and then reduce it. Some solutions construct state models by dividing the state space into smaller subspaces. Moreover, they do not consider software requirements during the reduction process. This omission can lead to incomplete or inaccurate models, of the web applications, which can affect the effectiveness of subsequent analysis and testing.

State space reduction also pose challenges like losing information or change in behavior of the system. The complete state space exploration of of a large application is not feasable and the solutions constructing state machine for every session have a big overhead associated. How much should we reduce the state space?

1.5 Proposed Solution

Our proposed solution consists of a framework whose objective is to achieve state space reduction in state machine of an Ajax web application. The framework uses soft computing based approach to achieve the obective.

Soft computing, as opposed to traditional computing, deals with approximate models and gives solutions to complex real-life problems. Unlike hard computing, soft computing is tolerant to imprecision, uncertainty, partial truth, and approximations [23]. In effect, the role model for soft computing is the human mind. One of the most important techniques of soft computing is fuzzy logic based clustering. Clustering or cluster analysis is a form of exploratory data analysis in which data is separated into groups or subsets such that the objects in each group share some similarity. Clustering has been used as a preprocessing step to separate data into manageable parts. Fuzzy C Means (FCM) [24] is a widely used clustering algorithm that works best in overlapping data domains. In FCM, every point has a degree of belonging to clusters thus the points on the edge of a cluster may be in the cluster to a lesser degree than points in the center of the cluster.

The framework uses FCM to achieve state space reduction by passing through a multistage progression to incrementally process the given information and generate results that help in the state machine construction.

1.6 Research Contribution

1. We analysed the current literature on Ajax based web application modeling to identify the challenges. There are many factors that contribute to the complexity of state machine construction of Ajax applications. Foremost of them is state explosion problem. We proposed a framework to handle state explosion problem in Ajax applications.

2. We designed and implemented the framework that generates reduced state machine for Ajax applications. The framework uses soft computing and generates the reduced state machine based on the requirements, use cases and usage patterns.

3. We evaluated the proposed framework to check the effectiveness of the approach. In this evaluation the effects of reduction in states on the coverage were analysed.

1.7 Thesis Outline

The thesis outline is as follows:

Chapter 2 gives the background of Ajax Web Application modeling and the challenges that are faced while creating the model of these dynamic applications.

Chapter 3 comprises a Literature Review in the area of Ajax applications, model based presentation of Ajax applications, the issues, the challenges, the gaps and the solutions that differnet researches propose.

Chapter 4 describes our proposed approach along with its details.

Chapter 5 is dedicated to the experimentaion and results. It discusses evaluation of our solution and its comparison with existing techniques.

Chapter 6 concludes the thesis and provides possible future directions for research.

Chapter 2

Background

Previously, users had to send requests to web servers via hyperlinks to update the page content of typical web applications. The web servers then do business logic executions for incoming requests and generate related web pages for client-side browsers to display updated content to users. When a client loads an Ajax-based web page, the browser on the client side creates Document Object Model (DOM) objects by parsing the HTML documents of the received web pages. This creates operation interfaces that programmes can use to change page content. The DOM is built in the form of a tree. Each HTML element has a DOM node corresponding to it, and each attribute in an HTML element has an attribute in DOM node corresponding to it. In addition to HTML elements, there are DOM nodes that represent browser windows and the HTML page itself, which programmes can use to obtain attribute values and capture events. In addition, the majority of well-known web browsers come equipped with integrated interpreters that allow documents written in HTML to run scripts such as JavaScript. Web applications can alter DOM nodes and attributes by asynchronously executing scripts, allowing partial content of a web page to be dynamically modified without requesting the entire page from servers. As a result, the amount of data sent between clients and servers is cut down.

The basic operation of an Ajax-based online application is for browsers to download the apps from a server, and then users can change the content of web pages by triggering events such as clicking buttons or interacting with browser screens.

Each triggered event has a callback handler that sends asynchronous requests to a server using the browser's XMLHttpRequest component, which processes the requests and returns results to clients. The replies are serialized in Extensible Markup Language (XML) format in most cases. When a client receives a response, it passes the content of the response to a callback function, which can update the partial content of the current web page based on the response and the script specified in the function. Client-side browsers and web servers can interchange data in the background in this asynchronous way, and a chunk of business logic executions can be migrated from servers to clients, reducing server load. Web applications that use the Ajax approach, allow users to interact with them in the same way that they would with desktop applications.

AJAX is intriguing because it opens up a plethora of new dynamic client-side possibilities in a domain where dynamism was previously restricted to server-side implementation of web applications. It proposes the concept of Web-2.0 [25], which is defined as interactive web applications that include user-generated content. Web User Interface (UI) widgets were also introduced via AJAX.This is similar to the widgets used in GUI application development. A table component, for example, can be referenced and altered without requiring the entire application to be updated or reloaded. The browser also ensures that the entire web application remains responsive until the table in the DOM is changed and the result is presented. AJAX opens up new possibilities for user interface design and enhances the overall user experience, but it also comes with a set of new technical problems. Using the browser's back button to return to the previous state, for example, will take you to a different state than you expected. In addition, the developer must handle DOM manipulation, which includes a DOM clean-up once the visibility is disabled. The DOM grows unexpectedly as a result of missing DOM clean-ups, which is a common error.

2.1 Challenges in Constructing FSM for AJAX Based Web Applications

AJAX presents whole new difficulties in the modeling and verification of web applications, particularly through the Reach, Trigger, and Propagate features, which make matters even more complicated [12, 13]. Additional research into this field carried out by A. Mesbah and A. van Deursen with the assistance of S. Lenselink and D. Roest uncovered a variety of difficulties [26–29]. This section presents the recognized obstacles, issues, and problems, and gives an overview of them along with an explanation for each challenge individually.

2.1.1 Reach Difficulty

Traditional web apps have different states that may be accessed using a specific URL that may also contain HTTP GET parameters. A unique URL is insufficient for navigating to AJAX states as the states are concealed by DOM-integrated JavaScript events. This indicates that in order to access AJAX states, such events must be found throughout the crawling process. In order to navigate and check the states, the crawler requires, in addition to recognizing such states, a way to execute the AJAX events that can lead to another state. In addition to the reach difficulty, there's also the bookmarking and sharing issue as web application only has a small number of unique URLs, which makes it difficult to create and share bookmarks.

2.1.2 Event Triggering

To determine whether an event resulted in a new state for a web application, the defined events must be triggered or executed within a verification environment. It's crucial in this period to not just start those events. AJAX states must take user input into account in addition to being connected to an event. Even though the same event is triggered, user input can alter the state. Identifying the data input points and inserting data prior to triggering an event that has already been detected are both prerequisite steps.

2.1.3 Result Dissemination

In order to make use of all available data inputs and events, the final output must be disseminated to other systems which causes a client-side DOM alteration.

In order to identify whether or not a new state has been reached, the altered DOM needs to be inspected and compared with known states. In addition to identifying the new state, it is necessary to identify the JavaScript defects in order to determine whether or not the new state is solely the result of a JavaScript event that was either not implemented correctly or was not implemented at all. The propagated result serves as the foundation for invariant or condition-based validation.

2.1.4 Asynchronous Behavior

When the browser needs to inquire about updated information from the server, it makes use of an AJAX engine. This engine runs an asynchronous XmlHTTPRequest. As a result, the user can continue using the browser and clicking on other items without having to wait for the AJAX answer to come back before continuing on.When modelling these asynchronous requests, the modelling approach must wait for the server's response and the DOM alteration caused by AJAX before comparing the propagated state to the previous state for state identification. Aside from waiting for the answer, the wait time is crucial for figuring out whether another candidate element event can be started without waiting for additional asynchronous AJAX responses.This complicates the process of DOM event detection and outcome comparison.

2.1.5 Statefulness

The increasing statefulness of the client-side makes it more difficult to derive AJAX states. A transition to examine and store the state-full behavior derived from the DOM states is also made when functionality is moved from the server to the client.This is particularly intriguing for drawing test cases from the web application that was crawled. To repeat the same state results during test case execution, the stateful behaviour must be duplicated within a stored test case. The newly crawled states must be based on the same user behavior, including the input values and navigation path.

2.1.6 Backtracking

A noted issue in modern AJAX web applications is that as the client-side experiences more stateful behavior, continuous backtracking create problems [29]. The gathered states must be accessible in both directions, saved, and compared in order to test backtracking. This indicates that the source state must have a backward edge from the target state. AJAX online applications have a well-known drawback known as the back-tracking issue. If the test framework supports reliable back-tracking for the crawling web application, it must support comparing the states.

2.1.7 DOM Tree Management

The proper handling of DOM trees is another issue that arises when AJAX is used in web applications. In other words, AJAX can be used to make particular DOM elements visible, or to add and delete elements from the page. AJAX-added DOM elements must be correctly removed in order to prevent the DOM tree from expanding and causing a slow browsing experience.

2.1.8 The Oracle Problem

According to the Oracle problem, fault positives might arise during state comparison and the creation that follows. This implies that states listed as two distinct states are same. The date component of a web page is a good illustration of this concept because it displays both the date and the current time in the form of a moving clock. If you compare the states without removing the information contained in the text, you would come up with new states. In order to prevent the production of new states that are not necessary, it has been recommended that oracle comparators remove all state information that is not essential prior to doing actual state comparisons [27].

2.1.9 DOM Validation

It is difficult to assess the veracity of the dynamic states. This refers to determining whether the state's Document Object Model is authentic. The states that have been found and gathered need to be checked to make sure they are correct.The Document Object Model is validated against the specification of the related HTML standard. Validation of the Document Object Model is performed by comparing its contents to the requirements outlined in the associated HTML standard.

2.1.10 Invariants

In order to ensure that the crawling phase is proceeding correctly, it is necessary to use application-specific invariants as an extra mechanism for validating the DOM [29]. These invariants may be used to determine if a state's DOM is also valid according to its application-specific DOM requirements.

2.1.11 State Explosion

A single web page can contain a wide variety of text field inputs, a significant number of actions on both the client and server sides, and a variety of dynamic content alterations. This leads to enormous and limitless tangible state possibilities. Any click has the potential to initiate a new state. Even a small web application can have an infinite number of states. In addition, the content could be different for each visitor or dependent on the time of day.

2.2 System Stakeholders

Before a system can be constructed, it is necessary for a project team to collect the requirements of the system from the many stakeholders. There are many ways to define the term "stakeholders." According to Freeman, the fundamental idea of stakeholders is as follows:
“A stakeholder is any group or individual who can affect or is affected by the achievement of the organization's objectives” [30].
The term, on the other hand, can also refer to “all those who have a stake in the change being considered, those who stand to gain from it and those who stand to

lose" [31].

In order to identify and select the key stakeholders of the software, several researchers have provided various definitions of stakeholders. The definitions are as follows:

According to Tom Gilb a stakeholder is "any person or organizational group with an interest in, or ability to affect, the system or its environment" [32].

"We define stakeholders as these participants together with any other individuals, groups or organizations whose actions can influence or be influenced by the development and use of the system whether directly or indirectly" [33].

"The people and organizations affected by the application" [34].

"System stakeholders are people or organizations who will be affected by the system and who have a direct or indirect influence on the system requirements" [35].

"Stakeholders are people who have a stake or interest in the project" [36].

"Anyone whose jobs will be altered, who supplies or gains information from it, or whose power or influence within the organization will increase or decrease" [37].

Primary, secondary, external, and extended stakeholders are the four broad categories into which stakeholders can generally be separated. [38].Because they will directly be affected by the project's outcomes and because they have a strong investment in the proposed system, the project's primary stakeholders are crucial to its success. The lack of these stakeholders may have a detrimental effect on the overall progression of the project as well as its ability to achieve its goals. Primary stakeholders are people who have a lot of power, authority, and responsibility over resources like money. Individuals that are influenced indirectly by the project's outcomes are referred to as secondary stakeholders. They could be people who have bought something or used a service. Despite not being involved in the construction of the initiative, they keep track of how their interests are being met. Even though they are not directly participating in the project, stakeholders from the outside (external) nonetheless contribute valuable insight. Finally, stakeholder who is frequently useful in assisting the aforementioned stakeholders in attaining their aims could be termed an extended.

When work begins on a project, there is usually a large number of people who are eager to take part in it. Because of the constraints imposed by the project, unfortunately, only few can be part of the roject. It's also true that some people are obligated but unwilling to engage in the process. To invite the proper individuals to join and contribute, it is necessary to identify, rank, and choose the stakeholders. This is extremely important given that improper engagement will lead to the collection of requirements that are both insufficient and incorrect, so putting the quality of the software in peril [39–41].

Stakeholders can be found by using the definitions provided above so that we know who might be impacted by or have influence over the project [39, 42]. In addition to system types, goals and methods for attaining those goals, as well as the systems' own domain, help in stakeholder identification [43]. Examining their contributions to the project is another way to find stakeholders. Examining how they interact, for instance, can help identify possible stakeholders [44]. Every stakeholder has a particular role they play which affects other roles. People can connect with one another in a variety of ways, such as through verbal and nonverbal contact, the exchange of information, and the pursuit of knowledge. This can be seen in the onion model [41]. Contexts are represented as onion rings, and each has certain responsibilities. Because each ring is connected to its neighbors, there are some jobs that are interconnected. The more actively a stakeholder participates in a discussion, the more substantial their participation is perceived to be. The conventional model of authority, legitimacy, and urgency can also be used to identify stakeholders [42]. It goes without saying that a stakeholder's involvement in a project is necessitated by their level of influence.

It's important to remember that stakeholders are real people with real expectations. Their backgrounds are diverse, and each has a unique set of skills. Additionally, they have a diversity of interests. The knowledge and interests of stakeholders must therefore be taken into consideration when selecting stakeholders [45]. Regarding knowledge, there are two categories of stakeholders: inner and outer [46]. Members of the inner group include developers and other producers who use technical expertise in their work. The producers are in need of the business expertise that is held by the stakeholders that make up the outer group. These stakeholders include consumers, advisors, and sponsors. In essence, RE requires that these two groups share what they know with each other. The terms education and professional experience can also be used to describe someone's knowledge. Numerous studies have found that one's experience and education level directly affect how individuals connect [47, 48]. Concepts such as personality analysis and group dynamics might be utilized while selecting stakeholders [39]. According to the findings of a number of studies, one way to establish the appropriateness of potential stakeholders is to first determine the level of interest they have in the project [45].

Knowledge is not adequate on its own. The ability to communicate and work together with other stakeholders is necessary. The process of developing software is frequently viewed as one in which numerous stakeholders must engage and communicate extensively. One of the most significant problems that exists in RE, according to a study, is that stakeholders do not possess the necessary skills to elicit requirements [41]. Stakeholders frequently have varying concerns, priorities, and responsibilities. Requirements frequently conflict during conversations with several stakeholders [49]. Negotiation and cooperation skills are needed to settle the conflicts and acquire better requirements [50, 51]. In fact, most businesses employ negotiation to help build shared understanding and set system boundaries and priorities [52]. Conversely, successful interaction and idea expression between stakeholders are made possible through communication skills [53]. Communication skills, both spoken and written, are also crucial. The usage of jargon by many stakeholders is one of the challenges that can occur during RE. To communicate effectively, domain specialists favour using commercial jargon, whereas developers are more comfortable using technical language. This can make stakeholders hide some requirements, which could lead to misconceptions [50]. Stakeholders must have appropriate oral and written skills to prevent poor communication.

2.3 Requirements Prioritization Techniques

The most important stakeholder requirements are determined by requirements engineers using the software requirements prioritization (SRP) approach before a software solution is developed [47]. Customers must be satisfied, and this can only be done if the specified quality, time, resources, and costs are adhered to strictly [54]. The SRP technique makes it easier to find conflicts between different functional requirements, figure out how to solve them, and figure out where to go next. The most difficult procedure, however, is choosing the right criteria in order to meet all crucial stakeholder needs and increase the product's market value [55]. The improper collection of software requirements raises the cost of system modification. The inappropriate criteria have a detrimental effect on the quality of the software.

Innovation in the development of distinctive and value-added software is a significant source of complexity. Complexity results from unclear user needs and goals. The only approach to reduce complexity is to choose the right requirements based on their level of importance or value. The requirements value is calculated using an appropriate requirements prioritization method. Putting the requirements under consideration in order of importance is also seen to be crucial for decision-making [56]. The process of prioritizing requirements is a time-consuming one [54, 57–59]. Therefore, in order to implement methodologies for requirements prioritization, professionals need to have not just a professional skill set but also a comprehensive understanding of the domain [60].

When a high-quality software product fulfils all of the fundamental requirements of its stakeholders or users, the contentment of those stakeholders or users is a factor that is taken into account [61, 62]. Because of constraints such as the available budget and amount of time for marketing, etc., it is not possible to take into account all of the requirements when developing the software. Only the most critical needs are taken into account in a single release [63]. Different characteristics of software requirements include risk [56, 59, 64, 65], importance [66], volatility [59], timing [65], and reliance on other requirements [67, 68].

On the grounds of the aforementioned characteristics, the insignificant requirements do not receive a great deal of attention. Requirements for high-quality software are based on only those features that bring value. The software's worth can be measured in terms of profit, efficiency, high performance, accurate data, and meeting the needs of the right users. To prioritise the most significant requirements, several SRP methodologies are used, and the requirements prioritization process still required a lot to be done [56].

Software engineers employ a variety of techniques based on factors such as cost, time, and relevance, however this approach has resulted in some disagreements. These conflicts arise as a result of the impact of one aspect on the other. Cost, for example, has a considerable impact on the priority of requirements. If the cost of a critical requirement is higher, there is a chance that the stakeholder will reconsider the requirement. As a result of this shift in the customer's mindset, the priority of that particular requirement is adjusted. [69]. When it comes to the SRP process, customers or other stakeholders who take part in the development of software have influence on the characteristics or capabilities that are required of the system. According to Donald Firesmith, many stakeholders have varying perceptions of the word ”prioritizing requirements”[70].

The selection of user requirements is done using a variety of SRP approaches in order to develop high-quality software. In a study, different SRP strategies were used to ”prioritize 13 well-defined quality requirements on a small telephone system,” with the statistics showing that AHP is more reliable than other techniques [71]. However, it comes accross scalability issues and thus is suitable for smaller sized projects having lesser number of requirements [58, 72].

2.3.1 Software Requirements Prioritization Techniques

A variety of software requirement prioritization approaches are suggested in order to rank them. Depending on the measurement scales, the requirements are ranked differently using various methodologies. These methods employ a variety of scales, including nominal, ordinal, interval, and ratio scales. The data are divided into many groups using these four main measurement scales. This section provides a condensed explanation of a few of the more significant requirements prioritizing strategies currently available.

2.3.1.1 Analytical Hierarchy Process

Pairwise comparisons are the foundation of the statistical method known as the Analytical Hierarchy Process (AHP). Saaty proposed AHP to prioritize requirements [73]. The requirements are evaluated in pairs, and then, based on the results of those comparisons, the priorities are established. For a limited requirements set, this approach performs well and accurately. The pairwise comparisons are computed using the following formula: $n \times (n-1)/2$.

2.3.1.2 Minimal Spanning Tree

Repetitive comparisons are reduced using the Minimal Spanning Tree (MST), reducing the total comparisons to $n-1$. According to MST, the fewer comparisons are adequate to determine the relative importance of the requirements [71].

2.3.1.3 Cost-Value Approach

The relationship between the importance and implementation costs of requirements is investigted by cost-value approach [63]. The term cost relates to the time and money required to meet the requirement, and benefits means the advantage associated with the requirement. AHP is then used to determine the above mentioned relationship.

2.3.1.4 Hierarchy AHP

The Hierarchy AHP (HAHP) was designed to address the AHP's [71] scalability problem. When there are more requirements in an AHP model, there will also be more comparisons, which will cause the overall efficiency to decrease. AHP is also only suitable for projects with a limited requirements; less suitable for projects with a high or medium number of requirements. As a result, the new HAHP approach is introduced, however, in comparison to AHP, HAHP is diffult to implement and less reliable [71] .

2.3.1.5 Numerical Assignment

The stakeholders classify the requirements into separate groups using the numerical assignment technique [74]. Determining the precise group that meets a particular condition is one of the technique's major limitations. All requirements are placed in the critical group using this strategy. As a result, all of the requirements must be met, making the strategy less effective [75]. The three main groups critical, standard, and optional are introduced. As a result, these three classes will cover all of the requirements, and there may be more. However, the requirements in each category have the same importance, and there is competition between them [76].

2.3.1.6 Theory W

The Win-Win Model, often known as Theory W, is focused on initial planning, risk assessment, and management. In the first step, the stakeholders rank all of the needs using the value based software engineerig (VBSE) practises. As a result, all required value negotiations take place in the initial step. Value and success ctiteria are the two basic concepts of this philosophy. [77].

2.3.1.7 Top-Ten Requirements

This method is built on the notion of users or stakeholders selecting the top 10 most important requirements. However, this method does not account for each requirement's specific priority. The technique can be used to pick a critical set of requirements from a small dataset of requirements. The technique's granularity isn't particularly excellent [74]. Stakeholders are treated equally, and all entities are given equal weight. Consequently, the process lacks credibility as a prioritization technique.

2.3.1.8 Planning Game

The planning game is a method for prioritizing tasks based on the needs of the client. Short sentences make up the stories that serve as representations of the criteria. Experts determine which stories to include in the initial release, and they also estimate the time commitment. Customers choose the stories for the next software releases based on the time or effort estimates provided by the experts. Because of its adaptability, the technique is well suited to creative developmental models such as extreme programming. Release and iteration planning are the two essential components of the planning procedure in this technique. In first type of planning, it is established which requirements to be handled in the initial and subsequent software releases. Subsequent to completion of the requirements, iteration planning is used to plan the various development actions.

2.3.1.9 Cumulative Voting or The Hundred Dollar Test

The hundred dollar test, also known as cumulative voting (CV) [78], is based on conversations. During the conversations, the needs are examined holistically. Depending on the significance of the requirements, stakeholders can allocate their $100 allocation to different requirements. A stakeholder can assign any amount of money to any demand. The dollars assigned to a certain requirement by various stakeholders are tallied, and the final priority of the requirement is determined by calculating the sum.

2.3.1.10 B-Tree Prioritize

The B-Tree prioritisation method [79] can deal with the problem of changing requirements and how they change over time. Using the priority values derived from the prioritized list of requirements, the function f is utilized in the B-Tree to illustrate the requirements.

2.3.1.11 Ranking

Because of the application of the ordinal scale, numerical assignment and ranking have a lot in common with each other [76]. The requirements are ordered starting with the highest priority requirement, which is given a value of 1, moving on to the next greatest priority requirement, which is given 2 , and so on. The value n, which represents total requirements number, is assigned to the final requirement.

Different algorithmic methods for prioritization criteria like bubble sort or binary search tree etc. can be used in ranking [80].

2.3.1.12 Bubble Sort

The method of ranking is utilized in bubble sort in order to prioritize the requirements [71]. The importance of the requirement determines the priority. However, how important is it is not taken into consideration. The steps involved are as follows:

1. Requirements are organized in column format.
2. Priority based swapping is done starting from top two requirements. The highest-priority requirement is prioritized first.
3. To determine the priority of each requirement, compare all of them and switch them around.
4. The top two requirements are compared again, and the process is repeated from the beginning.

2.3.1.13 Hierarchical Cumulative Voting

The scalability problem is addressed by hierarchical cumulative voting (HCV) [81]. HCV has the same notion as CV and requirements are scored like in CV with only differenc being the selection criterion of the requirement. HCV organizes the needs into a number of hierarchies, and then assigns priority to each of those according on the concept of CV.

2.3.1.14 Priority Groups

The technique is introduced by Karlsson et al. and the following steps are conducted in this technique [71].

1. Make a list of all the requirements.
2. Define the various requirements' priority levels, including high, medium, and low.
3. The number of requirements can serve as a guide for the formation of other subgroups, and the requirements can be adjusted within these subgroups.
4. Up until a one-on-one relationship is established, repeat step 3.
5. From left to right, read the requirements.

2.3.1.15 Value-Based Intelligent Requirement Prioritization (VIRP)

VIRP technique is comprised of following three steps.

1. The process of eliciting requirements and prioritizing at stakeholder level.
2. Prioritization at expert level.
3. Prioritization of Requirements Based on Fuzzy Logic.

The stakeholders elicit and prioritize the requirements in step one. The RV function, which is based on the prioritization of requirements at an expert level, is used in the second phase to evaluate the requirement value. There are two sorts of required factors in the RV function. rRCFs stand for requirement-specific requirement classification factors, while pRCFs stand for project-specific requirement classification factors. In the third step of the VIRP, Fuzzy C-Means is used to group the requirements into several clusters.

2.3.1.16 Interactive Genetic Algorithm-Based Prioritization

A posteriori analysis is performed using an interactive GA and pairwise comparison as the two primary methods [82, 83]. This technique's major goal is to gather pertinent information from the user. Extensive pairwise comparisons are impractical given current approaches. As a result, the GA is used to limit the number of comparisons. With this method, the fitness function receives incremental input that is only partially understood when it is first applied to the data.

2.3.1.17 Case-Based Ranking (CBR)

An interactive preference elicitation system using the CBR technique has been suggested [84]. Boolean values are used in the pairwise comparison for the purpose of eliciting preference. The CBR method incorporates an acquisition policy, which makes recommendations for the pairs of instances that should be investigated first. To forecast or approximate the preference values of unknown couples, a machine learning approach is used. To get the precise approximated ranks of the requirements, the error and effort minimization approach is used. Another approach Case-based reasoning is also discussed alongside case based ranking but two are different approaches.

Case-based reasoning is a problem-solving methodology that involves finding a solution to a novel problem, by identifying similarities with previously solved problems. The process involves storing a library of past cases and retrieving a similar case to the current problem, then adapting the solution to fit the new situation. In other words, Case-based reasoning uses past experiences to inform the solution to a new problem [85–87].

On the other hand, ranking is a technique for assessing and prioritizing a set of options or alternatives. Ranking involves assigning a numerical value or score to each option, based on some criteria or attributes, and then ordering the options according to their scores. Ranking is often used to make decisions when there are multiple options to choose from and limited resources to allocate [88, 89].

In summary, Case-based reasoning concentrates on solving new problems based on similarities with past experiences, while ranking is a technique for evaluating and prioritizing options based on predefined criteria or attributes. While both approaches can be useful in decision-making, they are applied in different contexts and have different goals.

2.4 User Sessions

A session is a collection of user interactions with the web application that occur over a set period of time. Multiple webpage views, events, social interactions, and ecommerce transactions can all be part of a single session. A session can be thought of as a wrapper for the actions that a user takes on a website.

S. Elbaum et.al [90, 91] explore a user session-based approach to testing a Web application. They also introduce three approaches for creating test cases: US-1, US-2, and US-3. Individual user sessions are successively replayed in US-1. US-2 entails re-enacting a combination of interactions from many users. US-3 entails combining routine user requests with those that are likely to cause problems (e.g., navigating backward and forward while submitting a form). This study also demonstrated that the efficiency of user session strategies improves as the amount of gathered user sessions grows in quantity.

Jessica Sant et.al [92] proposed a method for constructing a web application model using user session logs and storing it on the web server. The captured user session log data reveals an application's dynamic actions, which can be useful in resolving web application verification issues. Their technologies are capable of generating usage patterns with great coverage and accurate mapping of user behavior.

Several researches [93–96] have discussed the utilization of user sessions for software and GUI verification

T.Deenadayalan et.al [97] investigates the clusters strategy based on user sessions. This approach chooses a collection of demonstrative user sessions from each cluster based on the service profile and personalizes them by augmentation with additional requests to cover the relationships of web page dependencies. Furthermore, it allows the coverage of the implementation structure by attaching requests with dependency relationships to requests contained in user sessions generated during user events.

2.5 Clustering

The assigning of a set of observations into subsets (clusters) is known as clustering [98, 99]. Clustering is the partitioning of data into groups of items that are similar. Each group (cluster) is made up of things that are comparable to one another but not to objects from other groups. Clustering is an unsupervised learning approach and a common statistical data analysis methodology used in a variety of domains such as machine learning, data mining, pattern recognition, image analysis, and bioinformatics. The fundamental premise of software engineering is that error-prone software modules will have comparable software measures and, as a result, will most likely be located in the same cluster with other modules that share same characteristics. Similarly, the modules that are not prone to failure will most likely be clustered together in the same cluster (s) [100]. The distance criterion is used to determine similarity: two or more objects belong to the same cluster if they are "close" in terms of distance. Distance-based clustering is the term for this method. Another type of clustering is conceptual clustering, which involves two or more objects belonging to the same cluster if one defines a concept that is shared by all of them. To put it another way, objects are grouped based on how well they fit descriptive concepts rather than simple similarity measures. [101].

For developing a software quality estimation system, many clustering algorithms have been presented thus far. .

Serban et al. [102] identified crosscutting issues using clustering algorithms. The comparison was primarily performed from the perspective of aspect mining, employing a set of quality measures (SSE, PAM, ACC).

Berkhin [103] concentrated on clustering algorithms in the context of data mining. Data mining adds to the complexity of clustering very large datasets with many different sorts of attributes. This necessitates specific computational needs for clustering techniques. A number of algorithms that match these conditions have recently appeared and have been effectively applied to real-world data mining situations.

Xu et al. [104] concentrated on clustering algorithms, going over a wide range of methodologies that have been proposed in the literature. These algorithms come from many research communities, trying to handle diverse issues, and each has its own set of advantages and disadvantages.

Kumar et al. [105] offered a comparison of six widely used software quality prediction modelling techniques: Classification and Regression Trees, Multiple Linear Regression, Artificial Neural Networks, Case-Based Reasoning, Rule-Based Systems, and Fuzzy Systems. Fuzzy systems were found to produce better results than previous strategies.

Gupta et al. [106] created a software quality estimation method using clustering algorithms. To eliminate the need for an expert, software metrics thresholds are applied. Hundreds of software modules were clustered into a small number of coherent groups using the K-means and Fuzzy C-means clustering methods. If at least one metric of the mean vector is higher than the threshold value of that metric, a cluster is predicted to be fault-prone after this phase.

Fast Genetic K-means Approach was proposed by Lu et al. [107] as a new clustering algorithm (FGKA). The Genetic K-means Algorithm (GKA) proposed by Krishna and Murty in 1999 was the inspiration for FGKA, but it has numerous enhancements over GKA. Experiments show that, while the K-means method may converge to a local optimum, both FGKA and GKA eventually converge to the global optimum, with FGKA being substantially faster than GKA.

2.5.1 Basic Clustering Techniques

Clustering techniques are divided into two main categories: hierarchical and partitional. [98, 100, 103, 108].

2.5.1.1 Hierarchical Clustering

The use of hierarchical clustering results in the formation of a hierarchy of clusters, which can be depicted as a tree-like structure known as a dendrogram.The tree's root is made up of a single cluster that contains all of the data, while the leaves correspond to individual observations. They are either agglomerative (from the bottom up) or divisive (from the top down) (top-down) [108, 109].

2.5.1.2 Agglomerative Algorithms

Begin at the leaves, with each object forming its own cluster, then gradually combine groupings based on a distance metric [108]. The clustering may come to an end when all of the items are in a single group, or at any other time the user desires.

2.5.1.3 Divisive Algorithms

They begin at the root with a single group that contains all of the objects and then proceed to divide that group into smaller and smaller subgroups until each object is placed in a single cluster, or as desired. At each phase, dividing the data items into distinct groups and following the same pattern until all objects fall into a different cluster is a dividing strategy [110]. This is comparable to the divide-and-conquer strategy used by divide-and-conquer algorithms. The basic method of hierarchical clustering begins with a set of N items to be clustered and a $N \times N$ distance (or similarity) matrix.

1. Begin by allocating each item to a cluster; if you have N items, you will now have N clusters, each holding only one item. Allow the clusters' distances (similarities) to be equal to the distances (similarities) between the items they include.
2. Find the most similar (closest) pair of clusters and merge them into a single cluster, resulting in one less cluster.
3. Calculate the distances (similarities) between each of the previous clusters and the new cluster.
4. Steps 2 and 3 should be repeated until all items are grouped into a single N. (*) cluster.

2.5.1.4 Partitional Clustering

A partitional clustering algorithm creates k partitions of a database of n objects, with each cluster optimising a clustering criterion inside each cluster, such as the minimization of the sum of squared distance from the mean.Partitional clustering comes in a variety of forms:

K-Means Clustering

The K-means clustering method is one of the most basic clustering methods. The k-means method allocates each point to the cluster with the nearest centre (also known as centroid). The centre is the arithmetic mean of all the points in the cluster — that is, its coordinates are the arithmetic mean of each dimension across all the points in the cluster [111, 112].
The steps of the algorithm are as follows: [113, 114]

1. Determine the number of clusters, k.
2. Create k clusters at random and determine the cluster centers, or create k random points as cluster centers.
3. Assign each point to the cluster center that is closest to it, where "closest" is defined in terms of one of the distance measurements given above.
4. Calculate the new cluster centres again.
5. Repeat the previous two steps till you reach a convergence threshold.

This algorithm seeks to reduce an objective function, in this case a squared error function, to the smallest possible value. The objective function is shown in the equation 2.1.

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^{(j)} - c_j||^2 \tag{2.1}$$

where $||x_i^{(j)} - c_j||^2$ is the distance between a data point $x_i^{(j)}$ and the cluster centre

c_j, and n denotes the distance between the n data points and their individual cluster centres.

Fuzzy C-Means Clustering

Bezdek created a method called the fuzzy c-means clustering approach [115, 116]. For this technique, each instance can belong to any cluster with a distinct membership grade between 0 and 1 [117]. A dissimilarity function is minimized, as in equation 2.2, and centroids that minimise this function are found.

$$J(U, V|Z) = \sum_{i=1}^{c}\sum_{j=1}^{N}(\mu_{ij})^m||x_j - c_i||^2 \tag{2.2}$$

In this equation, V is a vector of cluster prototypes (centers) and m is a constant. Also, in equation 2.3 we have

$$D_{ijA}^2 = ||x_j - c_i||^2{}_A = (x_j - c_i)^T A(x_j - c_i) \tag{2.3}$$

If A = I, we're dealing with a Euclidian norm and the clusters will be round in shape. If we use the Mahalanobis norm, A is defined, as shown in the equation 2.4 and the clusters will be ellipsoidal in form [118].

$$A = \left(\sum_{j=1}^{N}(x_j - \overline{x})(x_j - \overline{x})^T\right) \tag{2.4}$$

The algorithm steps are

1. Randomly initialize the membership function, as in 2.5.
2. Determine the centroids, as in 2.6
3. Calculate the degree of dissimilarity, as in 2.2. Stop if the improvement over the previous iteration is less than a certain threshold.
4. Make a new u calculation, as in 2.7. Go to step 2.

$$\sum_{i=1}^{c} u_{ij} = 1, \forall j = 1, ..., n \tag{2.5}$$

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^m x_j}{\sum_{j=1}^{n} u_{ij}^m} \tag{2.6}$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{c}\left(\frac{D_{ij}}{D_{kj}}\right)^{2/(m-1)}} \tag{2.7}$$

c_i is i^{th} cluster's centroid, u is between 0 and 1, D_{ij} is the Euclidean distance between centroid and the data point, m is a weighting exponent which is between 1 and ∞. The most significant benefit of Fuzzy C Means is that it always converges.

2.6 FSM Based Web Application Modeling

Finite state machine is a very effective method to model a web application. Few researches have used this technique to model Ajax applications.

2.6.1 Finite State-Machine (FSM) Based Verification

This method infers a finite state machine from the web application's traces [9]. When necessary, the produced FSM can undergo additional redefinition so that the verification model can be fine-tuned with the assistance of interacting events' sequences. Manual re-definition of the FSM is a requirement of this technique.

2.6.2 Invariant Based FSM

This method infers an FSM using invariants [119]. The FSM contains several states, each of which is represented by a node, and events are what cause transitions between states. The application's FSM serves as the foundation for creating more complex models.

2.6.3 Crawl Based AJAX States

It is a tough piece of work to crawl the web application by utilizing Crawljax and derive a finite state machine [5]. The WebDriver application programming interface (API), which is utilised by Crawljax, makes it possible to use and control the WebDriver browser interface. Crawljax is able to crawl the application thanks to the WebDriver browser interface, which provides full browser support. Crawljax is able to carry out JavaScript-related actions thanks to the support provided by browsers. Crawler employs a variety of strategies to infer application states, but they are not without chellanges, the most prominent of which being state explosion [3].

Ajax is a technique for building dynamic web applications that allows for asynchronous communication and the manipulation of the Document Object Model

(DOM) at runtime. However, the highly dynamic nature of Ajax can lead to issues such as state explosion, state changes being triggered, and unreachable states. Finite State Machines (FSMs) are an effective way to model the behavior of software, but when used to model an Ajax application, the resulting state model can become extremely large and may suffer from the state explosion problem due to a large number of user-triggered state changes and the high level of dynamism in the application.

To build a system, the project team must gather the requirements for the system from various stakeholders. If stakeholders are not involved in the process, it can hinder the progress of the project and prevent it from achieving its objectives. Requirements engineers use a software requirements prioritization approach to identify the most critical stakeholder requirements before creating a software solution.

The way a user interacts with a system can provide valuable information on which areas should be tested more extensively. A session refers to a series of interactions between a user and a web application over a specified time period. It can be conceptualized as a grouping of a user's actions on a website.

Soft computing-based learning algorithms can assist in identifying which areas of an application are more important and which are less important. Clustering involves dividing data into groups of similar items. Each group, or cluster, consists of items that are similar to each other but not to those in other groups.

Finite state machines can be a useful approach to modeling a web application. Few researches have applied this technique to model Ajax applications.

Chapter 3

Literature Review

3.1 State Model of AJAX

Web applications are represented using a variety of notations and diagrammatic methods, but finite state machines are the most convenient way to show their dynamic behavior [17]. A state machine is a convenient technique to simulate software behavior without having to deal with implementation difficulties [17]. A finite state machine model can be very helpful in illustrating this dynamic behaviour since object states in Ajax web applications alter in response to user or server-driven events at run time [9, 120].

3.1.1 Web Application Vs Traditional Application

The complexity of web-based applications has substantially expanded over time, according to Donley and Offutt [121]. Even software experts have trouble distinguishing between web-based and traditional applications. They stressed the point that in order to test web-based applications, it is very vital to first obtain a deeper understanding of the applications themselves. This is a very critical step in the testing process. They talked about the key distinctions between traditional and web-based applications.

According to this study, online software does not fall into any of the classic categories of shrink wrapped, bundled, contractual, or embedded software. Web developers upload the web application that they have created onto a web server, which is a computer that is linked to the World Wide Web and is capable of accepting requests via the HyperText Transfer Protocol (HTTP) . End users then utilize browsers on their client computers to access the software. This is a significant deviation from past software deployment methods: All users share a single copy of the application.

3.1.2 Execution Trace Data based State Machine Construction

the application. Arora and Sinha [122] examine two popular testing techniques:

invariant-based and state-based testing. Despite the fact that these strategies are still effective in a variety of applications, numerous concerns and problems remain, such as scalability issues. The issues of ”gathering session data”, ”limiting state space”, and ”advancing in FSM retrieval to automatically determine user session-based test scenarios” are still open. The authors agreed that a more meaningful method for DOM to FSM advancement is required. Experiments using this technique were able to generate test cases for semantically interacting sequences, with data indicating that longer sequences yield more test cases with higher fault exposing potential. Finally, the authors emphasized that testing is heavily dependent on the technology through which it is carried out, and that future testing methodologies must adapt to the diverse and dynamic nature of web-based applications. Marchetto et al. acquired the web application’s execution trace data and used it to create a finite state machine. [120]. This method’s foundation is an Ajax application’s dynamically extracted state machine; however, the method is only partially dynamic and manual validation is also utilized for model extraction. According to the findings of this study, finite state machine retrieval is primarily an untapped area that requires further development. [120]. In Ajax testing, dynamic state identification is a challenging process that demands constant attention.

As a result, a dynamic analysis technique was needed to build the application's state-based model. The authors primarily concentrated on identifying groups of event sequences that interact semantically and are used to generate test cases. [123].Their understanding was that the length of these sequences has an effect on their fault exposing capability, i.e., more length equals more faults, and the results of the studies that were carried out verified this understanding. [22, 123–125].The method generates a very high number of test cases, which leads to state explosion, and also involves events that have no connection to one another. Marchetto only contributed to the reduction of the number of test cases that used asynchronous communication. This method only addresses a few aspects of asynchronous communication; other testing issues, such as how to get runtime DOM changes and transition between several DOM states, still require investigation. Furthermore, dynamic analysis-based finite state machine retrieval is also necessary.
Arora and Sinha [6] concentrated on testing the issues with asynchronous web applications' runtime behavior. They came up with a number of experiments for it. They provided a method that creates a state machine to find all dynamically generated states, their associated events, and DOM changing elements. To achieve scalability in state machine generation, they used a technique which is based on Model Checking and that technique lessened the state space paths to evade the problem of state explosion. However the approach contains processing overheads. Moreover the effects of state reduction on testing effectiveness are not discussed and remain unclear.They employed a method that is based on model checking to accomplish scalability in the production of state machines, and that method reduced the state space paths to avoid the issue of state explosion. Processing overheads are present in the method, though. Furthermore, the consequences of state reduction on test efficacy are not covered and are still unknown. the application.

3.1.3 Crawler based State Machine Construction

Key elements of AJAX-based Web 2.0 applications include client-side runtime manipulation of the Domain Object Model (DOM) tree and stateful asynchronous client/server communication. They are not only fundamentally different from conventional Web apps as a result, but they are also more prone to errors and more difficult to test.

VeriWeb is a tool that was developed by Benedikt et al. [126] that automatically explores the pathways of multi-page websites using a crawler and a detector for anomalies such as navigation and page problems (which are configurable through plugins). In contrast to more traditional tools of this type (such as spiders), which frequently only look at static links, VeriWeb can automatically investigate the dynamic contents of a website, including form submission and client-side script execution[126]. For form-based pages, VeriWeb extracts potential input values using SmartProfiles. User-specified SmartProfiles are essentially collections of pairs of attributes and their values. Then, forms are automatically filled out with these attribute-value pairs. The configuration of the SmartProfile does not depend in any way on the structure of the website that is being evaluated. It is unclear if VeriWeb's crawling technique may be utilized for testing in AJAX apps, despite the fact that it has some support for client-side scripting execution.the application.

Mesbah et al. [13, 26] suggested that Ajax user interfaces be tested automatically using invariants. The primary objective of this work was to crawl Ajax applications using the CRAWLJAX tool, which simulates actual user activities on various clickable application interface elements and derives the model from a state flow graph. Additionally, he advocated the usage of CRAWLJAX-based automatic invariant detection. Crawljax uses the Levenshtein method [127] to find out what's different about two DOM instances by figuring out how far apart they are in terms of edits. The authors stated in their study that the optimal path seeding strategy for automated testing of web applications is capture and replay, which did not apply in his work. According to Mesbah's perspective in his work [13], invariant-based testing is a poor substitute for an oracle. The authors also noted that employing capture and reply techniques is the best way to manage dynamic state extraction because Ajax dynamism makes extensive testing extremely difficult.

Crawljax has access to client-side code and can recognize clicked components that cause a state change in the built-in DOM of the browser. The user interface's states and any potential (event-based) transitions between them are represented in a state-flow graph once the state changes have been identified. In an AJAX application, a UI state change is defined as a modification to the DOM tree structure brought on by either server-side or client-side state changes. These DOM modifications' paths are also noted.

The user interface is compared to various constraints after the various dynamic states have been identified. These constraints are stated as DOM tree invariants, allowing any state to be checked.

In accordance with a fault model, Mesbah and van Deursen [13, 26] divide these invariants into: DOM-tree invariants, DOM-state invariants, and application-specific invariants categories. Below is a description of the common DOM-tree invariants [128].

3.1.3.1 Validated DOM

This invariant ensures a correct DOM structure on all execution paths. Following the completion of each state transition, the DOM tree that was generated is then converted into an HTML instance. To make sure there are no mistakes or warnings, a W3C validator serves as an oracle. Many browsers do not display errors due to minor HTML code mistakes, but all HTML validators expect the structure and content of the source code to exist. Consequently, this is significant. On the other hand, modifications to the user interface of a single-page app built using AJAX are brought about by partially updating the Document Object Model (DOM) with JavaScript. This is a concern since client-side JavaScript cannot be validated by HTML validators. the application.

3.1.3.2 DOM Error Messages

the application. This invariant guarantees that a string pattern representing an error message will never be present in the states. Error messages should be automatically recognized, whether they are client-side errors like "404 Bad Request" and "400 Not Found" or server-side errors like "500 Internal Server Error" and "MySQL Error."

3.1.3.3 Other Invariants

These contain invariants for additional purposes like finding linkages, adding more security restrictions, and invariants that may lead to enhanced accessibility at any time during the crawling process, among other things.

3.1.3.4 No Dead Clickables

This invariant's primary responsibility is to ensure that an AJAX application does not have any physical links that are "dead" or "broken." This is important because any link in an AJAX application that can be clicked has the potential to really change the state of the programme via background data retrieval form server utilizing the susceptible to errors JavaScript. The AJAX engine will typically conceal these kinds of error signals, and it will ensure that the user interface is not made aware of any broken links. You can find any broken links or clickables on the website by monitoring the client-server request-and-response flow subsequent to each event.

3.1.3.5 Consistent Back Button

The malfunctioning back button in the browser is one of the most frequent problems that occur with AJAX-based web applications. The browser entirely leaves the application's page when the back button is clicked. Crawling enables a comparison between the expected state of a graph and its actual state after the back button has been executed. This comparison enables the automatic detection of any inconsistencies or errors that may have occurred.
The authors of [129] and [130] figure out hash values based on how the state is set up (structure) and what it contains. However, despite the fact that these techniques can eliminate superfluous copies of the same state, they are insufficient for

the task of identifying pages that are almost identical to one another.
[27] improves the algorithm's state equivalence mechanism by first performing Oracle Comparator Pipelining (OCP) before computing hash values. Each comparator is designed to remove an irrelevant substring from the DOM string, which may result in meaningless variations between two states, such as time stamps or marketing banners.
In jAk, the term "Jaccard similarity" is used [131]. In this study, the authors define two pages as comparable if their Jaccard indices are higher than a specific threshold and they have the same normalized URL. After removing query values and sorting query parameters lexicographically, a normalized URL is returned. The Jaccard similarity is calculated using the sets of JavaScript events, HTML forms, and links that are present on the web pages.
In [132], two methods for enhancing any state equivalence mechanism based on the DOM are provided. The first seeks to identify unneeded dynamic content by repeatedly loading and refreshing a page. The second recognizes session parameters and does not take into account requests and answers that change because of them.

The DOM uniqueness tool that is described in [133] finds pages that have a similar DOM structure by locating repeated patterns and reducing them to a canonical DOM representation. This representation is then hashed into a single integer value. When configuring the algorithm, the user can choose which HTML tags should be included in the canonical representation as well as whether or not to include their textual content. This method keeps track of structural changes, like adding or taking away rows from a table. Since it includes sorting the items in each DOM subtree, it is likewise unaffected by elements shuffles. Nevertheless, a misleading distinction between two nearly identical states might arise as a result of adjustments that are not identified as being components of a structural pattern.

This process is further extended by the method found in [134], which divides a DOM tree into several subtrees, each of which corresponds to an individual application component, such as widgets. Because the DOM uniqueness technique is applied to each component on its own, this prevents an explosion in the number of alternative states that could occur when the same data is displayed using a variety of different configurations.

A web application's state machine model is constructed by grouping related pages [135] according to a page's structure. A prefix tree is used by the authors to model a page utilizing its links (anchors and forms). The Abstract Page Tree (APT), a different prefix tree that is used to hold these trees, is vectorized. Analyzing APT subtrees enables the discovery of related pages.

A distinct way of clustering application states into multiple equivalence clusters emerges in software tools that describe and test Rich Internet Applications (RIAs) using execution traces, such as RE-RIA [136], CrawlRIA [137], and CreRIA [138]. The process of clustering is accomplished by analysing a number of equivalence criteria, all of which are dependant on the DOM set of elements, event listeners, and event handlers. If one set includes the other as a subset, then two DOMs are regarded as being comparable. Both memory use and computation time are high for this strategy.

In-depth discussions of the problems and difficulties encountered when crawling modern asynchronous apps were conducted by Deursen et al [3]. Modern web applications have made tremendous progress toward the single-page approach, in which the JavaScript engine maintains the DOM-based user interface and interaction. This results in a great deal of analysis and comprehension problems, most of which cannot be solved by the static analysis techniques available in the contemporary day. They include state explosion, navigation through state changes and the inability to access states that can't be reached. The authors of this work have discovered, through the use of automated crawling, a method by which these problems can be overcome. For future study, they mentioned dynamic web application analysis, such as benchmarking, guided crawling and example-based crawls; model-based web application analysis; and cyber security as attractive areas for future research.

3.1.4 State Machine Construction from Requirements and Design Documents

Sabharwal et al [139] came up with a methodology to model the navigation mechanism of online applications. This mechanism is based on user requirements and

design that too at lower level. Their algorithm collects the information from the requirements and the low level design to construct a navigation graph of the page and this graph is further used for the synthesis of test sequences. This method has the advantage of using workflows that can be positive or negative. The tester can thus use the path navigation graph for any of the above mentioned test sequences. The problems of page and link explosion are also focused in this research.
Ajax Web applications are DOM based applications which are functioned by user event connected message handlers or by server messages. It is evident from above mentioned studies that Ajax is vulnerable because of features like stateful client, asynchronous communication, delta updates, un-typed JavaScript, client-side DOM manipulation, event handling, timing, back/ forward button and browser dependence. As Ajax application work as a single page applications, they have to handle large traffic on one page. Inputs to text fields, client side event handling server side activities, and other dynamic changes on single page might result in unbounded concrete states, i.e., state explosion. As discussed earlier in this section, several state space reduction techniques have been proposed but not without issues. Issues like processing overheads, effects of state reduction on testing effectiveness remain unaddressed. Moreover, techniques adopt exhaustive methods of state machine creation which results in scalability issues. This research proposes a framework to address these issues by limiting the state space by identifying the most frequently used areas of the application under test. The framework addresses the issues of state explosion without compromising the effectiveness of testing.

3.2 State Explosion

When it comes to building and validating software systems, one of the most well-known methodologies is called state-based modelling [140]. It is a method of brute force verification that can automatically and methodically examine the state space (SS) and specification of a given system to show whether or not its attributes are fully satisfied. Clarke Clarke et al. [140], Queille, and Sifakis [141] have all separately advocated this strategy. The level of trust in the system is greatly increased by the brute-force check of SS in state-based modelling.

However, state-based modelling has some restrictions because of the expanding state space (SSE). SSE happens when a system's state space grows at a rate that is proportional to the number of its components and quickly fills up the computer's memory. As a result, only a limited number of SS will be able to be examined. But the promising benefits of model-checking have still pushed researchers to try to solve the SSE problem, which has led to the main direction of state-based modelling research. [142, 143].
There are four primary categories of state space explosion prevention methods. [144]:

1. Cut the number of states to explore.
2. Reduce the amount of memory required to store investigated states.
3. Use a distributed environment or parallel processing.
4. Give up the completeness constraint and only examine a portion of the state space.

The description of these four different techniques is presented below.

3.2.1 State Space Reductions

When examining the state spaces of some simple models, it is immediately apparent that they include significant redundancy. The obvious solution is to try to take advantage of this redundancy and cut down on the states that need to be searched in. It is important to specify which states are excluded from the search in order to put this theory into practice. Additionally, it is necessary to show that the method is correct—specifically, in terms of state space coverage as per some criterion, typically bi-simulation or stutter equivalence.

3.2.1.1 State based Reductions

State-based reductions take advantage of the fact that if two states are bisimilar, it is enough to look at the states that come after (successors) only one of them.

Either dynamically, while the exploration is taking place, or statically, before the exploration begins, by a modification of the model, the reduction can be carried out. Examples of these reductions include symmetry reduction [145–150], live variable reduction [151, 152], cone of influence reductions, and slicing [153, 154].

3.2.1.2 Path based Reductions

Path-based reductions take advantage of the fact that occasionally only one of two action sequences needs to be investigated because they are simply two alternative linearizations of "independent" actions and hence have the same outcome. These reductions aim to cut down on the amount of similar interleavings. Examples of these reductions include [155, 156], partial order reduction [157–161], confluence [162], and simultaneous reachability analysis [163].

3.2.1.3 Compositional Methods

Systems are frequently described as being composed of several parts. There are two techniques to take advantage of this structure: compositional state space generation [164] and the assume-guarantee approach [165–167].

3.2.1.4 Storage Size Reductions

Memory constraints are typically the key barrier in model checking. As a result, we can use some sort of time-memory trade-off to conserve some memory at the expense of requiring more time. The memory requirements of the method are mostly driven by the structure that is examined, which maintains a history of previously traversed states. As a result, approaches that aim to reduce the amount of memory required do so by focusing on this structure in particular.

3.2.1.5 State Compression

Each state is represented as a byte vector during the search procedure, which might have a very high size (e.g., 100 bytes). This vector can either be compressed

[168–174] to take up less space, or its components that are shared can be used interchangeably [175]. We can represent the entire implicitly visited structure as a minimum deterministic automaton rather than compacting individual states[176]. This is an alternative to compressing individual states. [176].

3.2.1.6 Caching and Selective Storing

We can save only a subset of the visited structure's states as opposed to storing all of them. This strategy may result in revisiting some states, which may in turn cause an increase in runtime; but, it does not waste memory. These kinds of methods include, for instance:

- caching [177–179] is used to purge some previously stored data, as soon as system is out of memory,
- only selected states are saved through selective storing [172, 180], in accordance with predetermined heuristics,
- sweep line [181–183] utilizes the progress function, which ensures that some states will not be used in the future, allowing them to be removed from memory.

3.2.1.7 Randomized Techniques and Heuristics

We can utilise randomised techniques and heuristics if the memory needs of the search remain too high even after applying the aforementioned techniques. These methods only look at a small part of the space of possible states. As a result, they can only assist in the identification of errors; we cannot rely on them to help us establish that something is accurate.

3.2.1.8 Heuristic Search

(This is also called a ”directed” or ”guided” search.) The order in which states are visited is set by some heuristics [184–186]. As an alternative heuristic technique, a genetic algorithm can be used in which the objective state is ”evolved” over time [187].

3.2.1.9 Random Walk and Partial Search

The random walk method retains no data and always visits one successor of the present state. [188, 189]. A number of modifications can be made to this basic strategy, such as visiting some of the successors as opposed to just one, storing certain states in the visited structure, or combining random walk and local breadth-first search. [189–193].

3.2.1.10 Bitstate Hashing

The technique does not record entire states but instead simply saves one bit of information for each state in a large hash table [194]. When there is a collision, the search skips over some states. This method can also be performed using Bloom filters, which is a more complex approach [195, 196].

Table 3.1 shows the summary of included studies while table 3.2 shows the summary of studies handling state space explosion along with their advantages and limitations.

TABLE 3.1: Summary of Included Studies

Sr. No.	Study	Approach
State Model Construction		
1	[6, 22, 120, 122–125]	Execution trace data based construction
2	[3, 13, 26, 27, 126–138]	Crawler based construction
3	[139]	Requirements based construction
State Space Explosion		
1	[145–154]	State based reductions
2	[155–163]	Path based reductions
3	[164–167]	Compositional methods
4	[168–176]	State compression
5	[177–183]	Caching and selective storing
6	[184–187]	Heuristic search
7	[188–193]	Random walk and Partial search
8	[194–196]	Bistate hashing

Table 3.2: Summary of Studies Handling State Space Explosion

Sr. No.	Study	Problem solved	Advantages	Limitations
1	[145–154]	State based reductions	Works well for system having bisimilar states	Poor performance for systems with distinct states
2	[155–163]	Path based reductions	Works well if alternate paths of same actions are present	Weak in absence of alternative linearizations
3	[164–167]	Compositional reductions	Strong for discretely structured systems	Weak for system components with vague boundaries
4	[168–176]	State compression	Works well for systems having state byte vectors with interchangeable components	Weak for systems with large state byte vectors
5	[177–183]	Caching and selective storing	Save only a subset of the visited structure's states	Issue of state revisiting causing increased runtime
6	[184–187]	Heuristically guided search	Works well when guided by an accurate heuristic function	Terminates late if the heuristic function is not admissible
7	[188–193]	Random state selection	Simple to implement	Computationally expensive
8	[194–196]	Hashing	Good hash function can reduce the number of hash collisions	Weak for large or dynamic data here

3.3 ReactJS - A Modern Web Technology

ReactJS is a JavaScript library that was produced by Facebook for building user interfaces. It has become one of the most popular front-end frameworks for web development, known for its simplicity, reusability, and scalability. ReactJS allows developers to create components that can be reused across multiple projects and scales easily from small to large applications. In this section, we will discuss the advantages and disadvantages of ReactJS, how we can model it using finite state machines, and how this can help with dependability and testing of software. [197].

3.3.0.1 Advantages of ReactJS

1. Reusability: One of the leading advantages of ReactJS is the ability to create reusable components. This can be a major time-saver in development, as it allows developers to create components once and use them across multiple projects. Additionally, it can lead to more consistent and maintainable code.

2. Scalability: React JS is designed to scale easily from small to large applications. Its component-based architecture permits for easy management of complex user interfaces and data flow. This can make it a good choice for projects that may need to grow and evolve over time.

3. Performance: ReactJS is known for its high performance. It uses a virtual DOM (Document Object Model) that updates only the essential changes, reducing the number of manipulations to the actual DOM and improving the application's speed. This can lead to faster rendering times and a better user experience.

4. Community: ReactJS has a large and active community of developers, which means there are many resources available online for learning and troubleshoot-

ing. Moreover, there are many open-source libraries and tools, that can be used, with React to extend its functionality.

3.3.0.2 Disadvantages of ReactJS

1. Learning Curve: While ReactJS is relatively easy to learn for experienced developers, it can be challenging for beginners to handle its concepts. It requires knowledge of JavaScript, HTML, and CSS, as well as additional concepts like JSX (JavaScript XML) and the virtual DOM.
2. Complexity: ReactJS can be complex to manage in larger applications. There are many dynamic parts and dependencies, which can make debugging and testing more difficult. Additionally, the component-based architecture can lead to code duplication and other issues if not managed carefully.
3. Limited Scope: ReactJS is a front-end library and does not include features such as routing or HTTP requests, which require additional libraries. This can make it less viable for certain types of projects or applications.

3.3.0.3 Modeling ReactJS using Finite State Machines

Finite State Machines (FSM) are mathematical models that depict the behavior of a system with a finite number of states and transitions between those states. Using FSMs to model ReactJS can help with dependability and testing of software. By defining the states and transitions of React components, it is possible to identify edge cases and ensure that the application behaves as expected. The Xstate library is a common tool for modeling ReactJS using Finite State Machines. It allows developers to define the state and transitions of a component using a declarative syntax. Xstate also provides a visual representation of the state machine, making it easier to reason about the behavior of the component [198, 199].
Benefits of using Finite State Machines for modeling ReactJS:

1. Improved Dependability: By modeling React components using Finite State Machines, developers can detect edge cases and ensure that the application

behaves as expected. This can lead to more dependable software that is less likely to fail or exhibit unexpected behavior.

2. Easier Testing: Finite State Machines provide a clear and obvious representation of the state and transitions of a React component, making it easier to write tests for the application. This can lead to more reliable and robust testing.

3. Simplified Debugging: By breaking down the behavior of a React component into finite states and transitions, it becomes easier to mark and debug errors. This can save time and effort in the development process.

3.3.0.4 Issues with Modeling ReactJS using Finite State Machines

1. State Explosion: The more states and transitions a React component has, the more complex and challenging it can be to model using Finite State Machines. This can result in a state explosion, where there are too many states and transitions to manage effectively. This can lead to code that is difficult to understand and maintain.

2. Tooling: While there are many tools and libraries available for modeling React JS using Finite State Machines, there is deficiency of standardization in the field. This can make it challenging to choose the right tools and ensure compatibility with other parts of the development stack. Additionally, the tools themselves can be complex and difficult to learn, requiring a significant investment in time and resources.

3. Limited Application: While modeling React JS using Finite State Machines can be helpful for specific types of applications, it may not be suitable for all projects. Some applications may have complex or dynamic behavior that is difficult to model using Finite State Machines.

4. Maintenance: As with any software development tool, modeling React JS using Finite State Machines demands ongoing maintenance and updates. This can be a significant investment of time and resources, especially for large or complex applications.

Overall, modeling React JS using Finite State Machines can be a powerful tool for enhancing the dependability and testing of software. However, it is important to be aware of the potential issues and limitations of this approach, and to carefully evaluate whether it is the right choice for a particular project [200, 201].

3.4 Conclusion and Research Gaps

This chapter depicts following conclusions.

1. Due to the fact that online software only consists of a single copy of the application that is shared by all users, it does not fall under any of the traditional classifications for software.
2. Better user engagement is induced by current Ajax-based web applications, but at a price. Ajax is more error prone due to its asynchronous, event-driven, stateful, usage of a loosely typed scripting language, client-side substantial functioning, and exchange of page fragments rather than the entire page. [RQ1].
3. The use of finite state machines is a convenient technique to demonstrate the dynamic behavior of web applications without dealing with challenges of actual implementation. [RQ2].
4. Modeling Ajax applications comes with a whole new set of problems, such as state navigation, state change triggers, states that can't be reached, and, most importantly, state explosion. [RQ1].
5. Recent studies focused on modelling Ajax applications have only partially addressed the problem of state explosion by applying partial reduction or by reducing the model's effectivness. [RQ1].
6. The majority of research on preventing state explosions only looks at user session data. Other crucial application elements, such as the significance of the requirements or the stakeholders, are not taken into account. [RQ1].

7. According to some studies, the state machine can be generated and reduced simultaneously. As the algorithm runs, this mechanism constantly compares and reduces the states, adding overhead to the system. Also not included are the impacts of this reduction on application verification. [RQ1,RQ3].

Chapter 4

StateReduceAjax

This chapter proposes a framework *StateReduceAjax* to achieve state space reduction in state machine of an Ajax web application. In the context of modeling web applications, the state explosion problem refers to the difficulty of dealing with a large number of possible states that a web application can be in, and the resulting combinatorial explosion of possible paths that a user can take through the application. This can make it challenging to model and test the application effectively, as it becomes impractical to test all possible paths, and combinations, of user interactions.

In particular, Ajax web applications, which use asynchronous requests to update parts of a page without requiring a full page reload, can exacerbate the state explosion problem. This is because Ajax interactions can introduce additional states, and interactions, that must be considered when modeling the application.

To address the state explosion problem in modeling Ajax web applications, techniques such as model checking, symbolic execution, and state space reduction can be used to explore the state space of the application more efficiently and effectively. These approaches aim to identify and prioritize the most important paths, and interactions, to test while avoiding redundant or irrelevant paths that would not significantly contribute to the overall test coverage. Larger the state space, more this issue becomes unmanageable. The main objective behind the construction of *StateReduceAjax* is to reduce the state space.

Prior to delving into the specifics of the proposed framework, it is important to

examine the connection between an approach and a framework. An approach refers to a general method or strategy that is used to solve a problem or achieve a goal. It is a broad set of principles and guidelines that guide decision-making and action in a particular field or discipline. For example, Agile is an approach to software development that emphasizes iterative and incremental development, collaboration, and customer feedback. On the other hand, a framework is a structured and organized set of guidelines, concepts, and tools that can be used to apply an approach or solve a particular problem. A framework provides a specific structure or template for the implementation of an approach. For example, Scrum is a framework that is based on the Agile approach and provides a specific set of roles, events, and artifacts to guide the development process. In summary, an approach provides a high-level strategy or viewpoint, while a framework provides a specific set of guidelines and tools to implement that approach.
It is possible for a framework to integrate different approaches or methodologies. A framework is typically designed to provide a structure and guidance for a specific domain or problem space, and it may draw on different approaches or techniques to achieve its goals. For example, the Six Sigma framework for quality management integrates elements of statistical process control, design of experiments, and other approaches to improve processes and reduce defects. Similarly, the Lean Startup framework for entrepreneurship combines lean manufacturing principles with agile software development methodologies to help startups build and iterate on products more efficiently. In these cases, the framework serves as a placeholder for multiple approaches, providing a unified structure and language for practitioners to work within.

The proposed framework *StateReduceAjax* passes through a multistage progression to incrementally process the given information and generate results that help in the state machine construction. Initial process starts by gathering and storing the stakeholder information along with the use cases and system requirements.
Requirements pass through prioritization process and the result is stored back in the form of prioritized value for each requirement. This prioritized requirement value is used in use case requirement mapping to calculate Use Case Requirement Weightage (UiW) for each use case. The use case based session recording calcu-

lates how frequently a use case is executed.
The framework uses Fuzzy C Means (FCM) [24] to achieve state space reduction by segregating system use cases as high priority and low priority. The functionality of only these high priority use cases are considered for state machine construction. This reduced use case set helps to control state explosion in state machine of an Ajax application.
Fuzzy C Means (FCM) takes set of UiW values along with the use case frequency to identify the most pivotal use cases. These pivotal use cases comprise of the most frequently performed user actions. It is practically impossible to record all possibilities and then to extract and test these event and element mappings. One way to limit the possibilities is considering only the most frequently used events and then mapping them to the corresponding elements. *StateReduceAjax* runs only these pivotal use cases and maps user actions to Document Object Model (DOM) mutations using DOM tools [202, 203]. These DOM mutations are the DOM states and *StateReduceAjax* finally connects these DOM states to construct the state machine. Figure 4.1 shows the complete working of the proposed solution. In this research the proposed framework accomplishes its objectives by using Algorithm I shown in Table 4.1.

This algorithm initiates by taking requirement set as input and then sends it to requirement prioritization module. Requirement prioritization module applies requirement prioritization and returns prioritized requirement set. The algorithm passes this data to the use case prioritization module. This module takes use case set data as input and performs user session recordings, use case requirement mapping, and use case frequency calculation to generate use case frequency data and use case requirement weightage. This data is then sent to FCM that generates the prioritized use case set which is returned to the main algorithm. The algorithm then iterates through the prioritized use case set and invokes *generateFSM* module for each use case. The *generateFSM* module takes Source DOM of the Ajax application and use case set as input and implements event element mapping to generate state Log file. It then uses this log file to append the states and transition information to FSM data file for each use case. Finally it sends FSM data file to drawGraph function to generate the state machine for the use case.

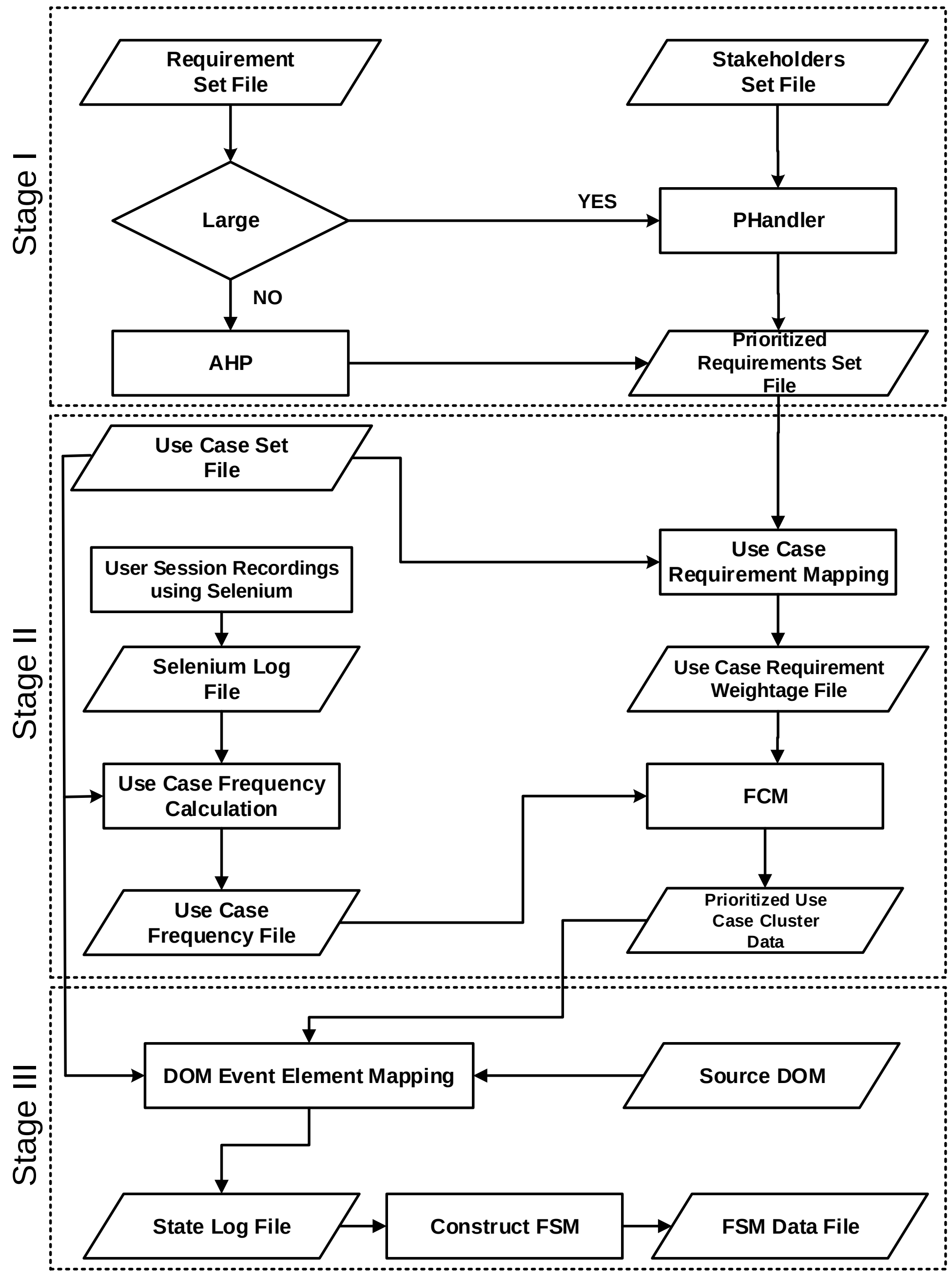

Stage I
Requirement Set File
Stakeholders Set File
Large
YES
NO
PHandler
AHP
Prioritized Requirements Set File
Stage II
Use Case Set File
User Session Recordings using Selenium
Use Case Requirement Mapping
Selenium Log File
Use Case Requirement Weightage File
Use Case Frequency Calculation
FCM
Use Case Frequency File
Prioritized Use Case Cluster Data
Stage III
DOM Event Element Mapping
Source DOM
State Log File
Construct FSM
FSM Data File

FIGURE 4.1: *StateReduceAjax* Framework

4.1 Stage I

Requirements of a software may vary from smaller to larger in number and there are many techniques to prioritize requirements. Requirements could be simple or complex having muliple features to be considered for prioritization. Because

TABLE 4.1: Algorithm I

procedure stateReduceAjax ( )		
Declare	:	Prioritized_Requirements_Set, Prioritized_Usecase_Set, FSM_Data
Input	:	Requirements_Set, Stakeholder_Set, UseCase_Set
Output	:	FSM_Data_File
1		Prioritized_Requirements_Set = PrioritizeRequirements(Requirements_Set, Stakeholder_Set)
2		Prioritized_Usecase_Cluster_Data = PrioritizeUsecases (Prioritized_Requirements_Set)
3		for each Prioritized_Usecase_id in Prioritized_Usecase_Cluster_Data.Usecase_id
4		gererateFSM()
end_procedure		

of its ease of use, variety, and high level of accuracy, the Analytical Hierarchy Process, often known as the AHP, is one of the multi-criteria decision making (MCDM) techniques that is utilized most frequently by researchers from all over the world [204]. AHP employs pair-wise comparisons as opposed to traditional approaches, which improves the accuracy of the findings and permits verbal assessments. The comparisons between pairs are used to find the right ratios and scales for the priorities. AHP offers a tried-and-true method for handling complex decision-making and can help with defining and weighing criteria, assessing the information gathered, and speeding up the decision-making process. [205]. AHP technique, however, is only appropriate for a few requirements [58] and this research uses AHP to prioritize smaller set of requirements.
There are many requirement prioritization techniques for larger requirement sets. This research uses PHander which is considered as a recent and robust requirement prioritization technique [206–208]. It is an expert system that employs value-based intelligent prioritization (VIRP) method and neural networks to prioritize a vast collection of requirements [209, 210]. Phandler initially uses a machine learning technique to group related requirements, then an artificial neural network (ANN) is used for additional prioritization, and finally AHP is used for concluding comparisons [211].

The first stage of the solution reads system requirements and stakeholder information from the file for further processing. If the requirement set is smaller than the threshold value than AHP is applied on it. If the requirement set is larger than the threshold value than it is input to PHandler alonwith the stakeholder information. Many researches have taken threshold value from experts and this research also uses the same so the requirement set having less than equal to expert given threshold value is considered small otherwise large [207, 212, 213]. Figure 4.2 shows the block diagram of Stage I.

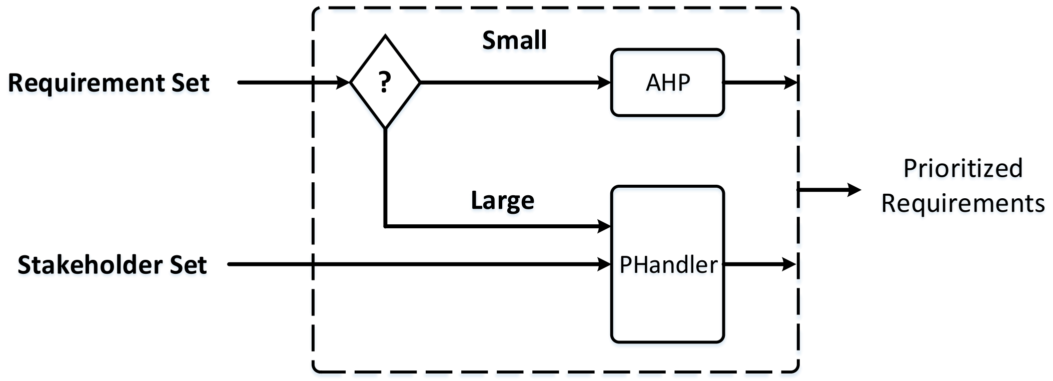


FIGURE 4.2: Stages I Block Diagram

4.1.1 Requirements Prioritization

The first step of Stage I prioritizes the requirements. The requirement set having small set of requirements fall in small project group and AHP [214] methodology is utilised in this study to rank it. A potential candidate for PHandler is the requirement set that has a significant number of requirements [215].

4.1.1.1 AHP

AHP is a requirement prioritization technique to rank the contesting requirements. Prioritization process uses pair wise comparisons of the requirements to identify which requirement has higher precedence and to what degree on a gauge of 1 to 9. Requirements with equal importance have a pair wise comparison value 1 while 9 depicts substantial difference in importance. The steps of AHP are shown in table 4.2 while results are shown in tables 4.3, 4.4, and 4.5.

TABLE 4.2: AHP

procedure AHP ( )		
Declare	**:**	Prioritized_Requirements_Set
Input	**:**	Requirements_Set
Output	**:**	Prioritized_Requirements_Set
1		Matrix construction of order n (n is the number of requirements).
2		Comparisons of the requirements using the pairwise method
3		Normalizing the matrix and figuring out the eigenvalues
4		Valuing each requirement
5		Prioritized_Requirements_Set ← Result accuracy estimation
end_procedure		

In the first step, AHP reads the requirements file and formulates an n x n matrix where n is the number of candidate requirements e.g. R1 to R6 in case of six requirments.

It then performs pair wise comparisons on these requirements and assigns ranking values ranging between 1 and 9. The value of '1' shows both requirements are equally important and '9' shows a significant difference in importance. For two requirements R1 and R2, if R1 has value '5' then it is strongly more important than R2 and AHP places this value at the intersection of row R1 and column R2. Row R2 and column R1 gets the reciprocal value, i.e., 1/5. A comparison of any requirement with itself gives '1' showing equality in importance thus the diagonal of matrix remains '1'.

TABLE 4.3: Pairwise Requirement Values

	R1	**R2**	**R3**	**R4**	**R5**	**R6**
R1	1.0	2.0	1.0	2.0	5.0	1.0
R2	0.5	1.0	0.5	0.2	1.0	1.0
R3	1.0	2.0	1.0	3.0	2.0	1.0
R4	0.5	5.0	0.3	1.0	1.0	0.5
R5	0.2	1.0	0.5	1.0	1.0	0.5
R6	1.0	1.0	1.0	2.0	2.0	1.0

Next step takes the sum of every column of the matrix and then normalizes the matrix by dividing each element in every column by the sum of that column. AHP then takes the average of each row in the normalized matrix. This process adds value of each element of a row and then divides this sum by total element count of that row.

TABLE 4.4: Normalized Requirement Values

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**SUM**	**PM**
R1	0.222	0.166	0.230	0.210	0.416	0.200	1.446	0.241
R2	0.111	0.083	0.115	0.052	0.083	0.200	0.645	0.107
R3	0.222	0.166	0.230	0.315	0.166	0.200	1.302	0.217
R4	0.111	0.416	0.076	0.105	0.083	0.100	0.893	0.148
R5	0.111	0.083	0.115	0.105	0.083	0.100	0.598	0.099
R6	0.222	0.083	0.230	0.210	0.166	0.200	1.113	0.185

The succeeding step checks the consistency by multiplying the average row value with each row element of the original matrix, resulting in a consistency matrix.

TABLE 4.5: Prioritized Requirement Values

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**SUM**	**Priority V**
R1	0.241	0.482	0.241	0.482	1.205	0.241	2.893	12.000
R2	0.053	0.107	0.053	0.053	0.107632	0.107	0.484	4.500
R3	0.217	0.434	0.217	0.651	0.434	0.217	2.170	10.000
R4	0.074	0.744	0.049	0.148	0.148	0.074	1.240	8.333
R5	0.049	0.099	0.049	0.099	0.099	0.049	0.448	4.500
R6	0.185	0.185	0.185	0.371	0.371	0.185	1.484	8.000

The last step takes the dot product sum of each row of the consistency matrix by 1/W resulting in weight assignments to each requirement.

4.1.1.2 PHandler

This research applies the PHandler to prioritize large requirement set. The PHandler uses three stages to prioritize the requirements. In the first task of stage I, experts evaluate just the requirements without any other details. In the next task, the experts assess the stakeholder data, which comprise of their brief profiles, their expectations about the system and system functionality of their choice. These profiles form the basis to quantify and identify the weightage of the stakeholders in reference to the system. Experts perform the quantification of the stakeholders using STAR triangle ranking method [216]. This method assigns a 1-10 range value to each stakeholder based on some key attributes which include the amount of significance, domain expertise, involvement, dependency, control, and decision-making authority. PHandler uses the quantified stakeholder profiles in later stages to resolve the conflicts among contending requirements. The third task takes requirement classification factors (RCFs) and then uses these factors to calculate requirement value (RV) for each requirement. These RCFs are project related (projRCFs) or requirement related (reqRCFs). While the reqRCFs are completeness, consistency, understandability, within the scope, and non-redundancy, the projRCFs are feasibility, modifiability, urgency, traceability, and testability. Equation 4.1 uses these RCFs to determine value of each requirement (RV) in the range of 0 to 5.

$$RV = 0.35 + 0.42 \sum_{i=1}^{5} pRCFi + \sum_{i=1}^{5} rRCFi \tag{4.1}$$

RCFi in equation 4.1 indicates the specific classification factor whose existence or nonexistence affects the RV of a requirement [214].

The second stage of the PHandler applies exceptions on the requirements with similar RV. These exceptions, along with RV, take profile values of the stakeholders as input and assign the requirement/s to different priority clusters. The stakeholder profile value depends on the influence, role, interest, and urgency of the stakeholder in reference to the project.The PHandler comes across five different cases to resolve competing requirements

Case 1: Requirements with Same RV and Different projRCF Value

For two requirements R1 and R2 having same RV, the PHandler compares the projRCF value and places the requirement with greater projRCF value in priorityCluster1. The example data in Table 4.6 shows the assignment of R2 to priorityCluster1.

TABLE 4.6: Requirements with Same RV and Different projRCF value

Req_ID	RV	Feasibility	Modifiability	Urgency	Traceability	Testability	projRCF
R1	0.71	2	2	2	1	2	9
R2	0.71	2	2	1	3	2	10

Case 2: Requirements with Same RV, projRCF and Different valST Value

For two requirements R1 and R2 having same RV and projRCF, the PHandler compares the valST value and places the requirement with greater valST value in priorityCluster2.The example data in Table 4.7 shows that R1 will be placed in priorityCluster2.

TABLE 4.7: Requirements with Same RV, projRCF and Different valST value

Req_ID	RV	Feasibility	Modifiability	Urgency	Traceability	Testability	projRCF	Stakeholder: valST
R1	0.71	3	2	1	2	2	10	S1:15
R2	0.71	2	2	1	3	2	10	S2:14

Requirements can have same requirement value (RV), project requirement classification factors (projRCF) value, and stakeholder profile value (valST). For all such cases, the PHandler calculates whether a stakeholder falls in high, medium or low value groups, using following Equation 4.2.

$$Average_{(N)} = \frac{\sum_{i=1}^{n} valST}{N} \tag{4.2}$$

PHandler takes the average of profile values of all the stakeholders who are part

of the elicitation process and then places the stakeholders, having profile values greater than or equal to the calculated average, in High Stakeholder Value group. The stakeholders having profile values less than the average are remaining stakeholders (remST). The PHandler repeats the above process for the remaining stakeholders to establish Medium and Low Value stakeholder groups. The process first calculates the remaining number of stakeholders using following Equation 4.3.

$$Average_{(remST)} = \frac{\sum_{i=1}^{remN} valST}{remN} \tag{4.3}$$

remN are the number of stakeholders we get by subtracting Number of stakeholders in high stakeholder group from the total number of stakeholders. The PHandler places the stakeholders with profile values higher than the average in Medium Stakeholder Value group and the remaining in Low Stakeholder Value Group.

Case 3: Requirements with Same RV, projRCF, valST and High Stakeholder Value

For two requirements R1 and R2 having same RV, projRCF, and valST, PHandler calculates the stakeholder value and places the requirements with high Stakeholder value in priorityCluster3. The example data in Table 4.8 shows that R1 and R2 are assigned to priorityCluster3.

TABLE 4.8: Requirements with Same RV, projRCF, valST and High Stakeholder Value

Req_ID	RV	Feasibility	Modifiability	Urgency	Traceability	Testability	projRCF	StakeholdervalST
R1	0.71	3	2	1	2	2	10	S3:13
R2	0.71	2	2	1	3	2	10	S4:13

Case 4: Requirements with Same RV, projRCF, valST and Medium Stakeholder Value

For two requirements R1 and R2 having same RV, projRCF, and valST, PHandler calculates the stakeholder value and places the requirements with high Stakeholder value in priorityCluster4. The example data in Table 4.9 shows that R1 and R2 will be placed in priorityCluster4.

TABLE 4.9: Requirements with Same RV, projRCF, valST and Medium Stakeholder Value

Req_ID	RV	Feasibility	Modifiability	Urgency	Traceability	Testability	projRCF	StakeholdervalST
R1	0.71	3	2	1	2	2	10	S5:8
R2	0.71	2	2	1	3	2	10	S6:8

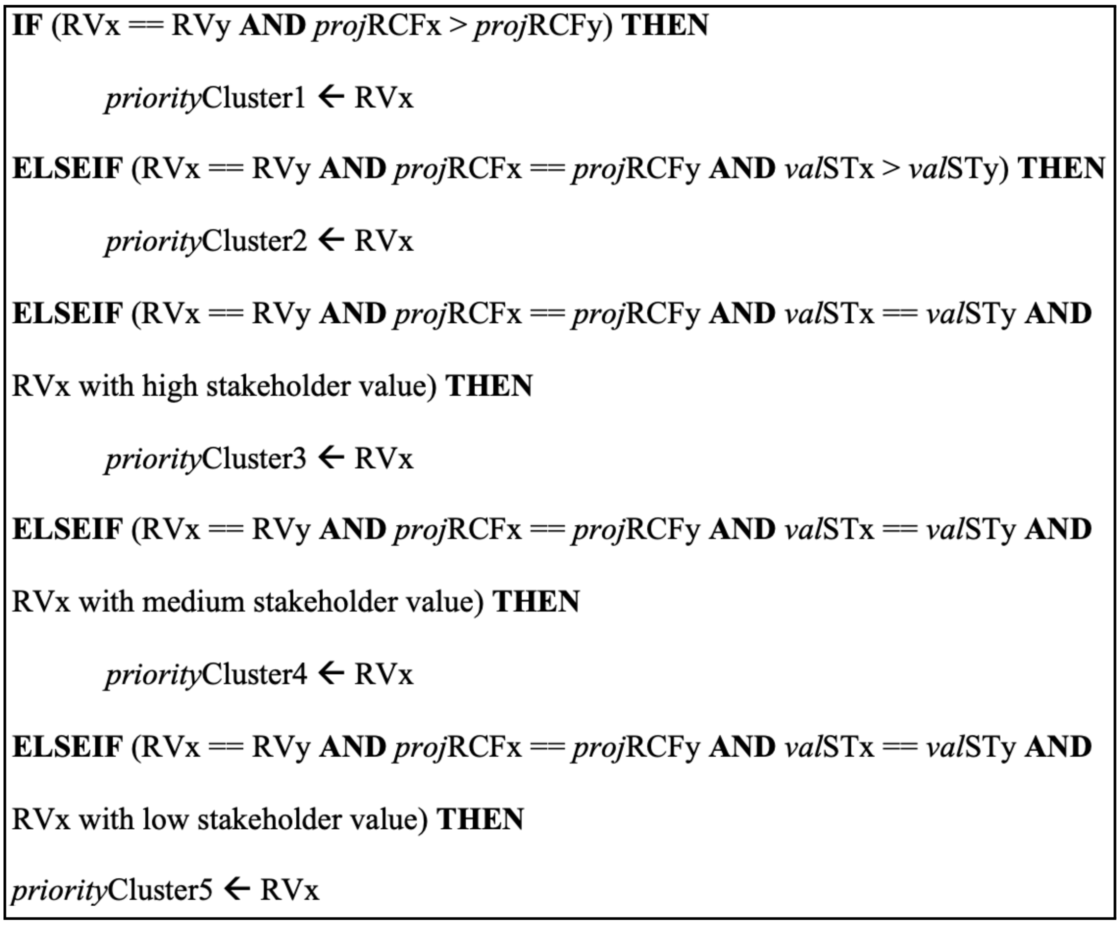
Case 5: Requirements with Same RV, projRCF, valST and Low Stakeholder Value

For two requirements R1 and R2 having same RV, projRCF, and valST, PHandler calculates the stakeholder value and places the requirements with high Stakeholder value in priorityCluster5. The example data in Table 4.10 shows that R1 and R2 would be placed in priorityCluster5.

Table 4.10: Requirements with Same RV, projRCF, valST and Low Stakeholder Value

Req_ID	RV	Feasibility	Modifiability	Urgency	Traceability	Testability	projRCF	StakeholdervalST
R1	0.71	3	2	1	2	2	10	S7:6
R2	0.71	2	2	1	3	2	10	S8:6

Figure 4.3 shows the exceptions for a given requirements x in comparison to a requirement y



```
IF (RVx == RVy AND projRCFx > projRCFy) THEN
        priorityCluster1 ← RVx
ELSEIF (RVx == RVy AND projRCFx == projRCFy AND valSTx > valSTy) THEN
        priorityCluster2 ← RVx
ELSEIF (RVx == RVy AND projRCFx == projRCFy AND valSTx == valSTy AND
RVx with high stakeholder value) THEN
        priorityCluster3 ← RVx
ELSEIF (RVx == RVy AND projRCFx == projRCFy AND valSTx == valSTy AND
RVx with medium stakeholder value) THEN
        priorityCluster4 ← RVx
ELSEIF (RVx == RVy AND projRCFx == projRCFy AND valSTx == valSTy AND
RVx with low stakeholder value) THEN
priorityCluster5 ← RVx
```

Figure 4.3: Exceptions between Same Priority Requirements

The third stage of the PHandler applies AHP to remove the contest among the competing requirements. AHP produces prioritized lists for the given clusters and finally, the PHandler combines all the priority lists to generate final priority list. Figure 4.4 shows the woking stages of PHandler.

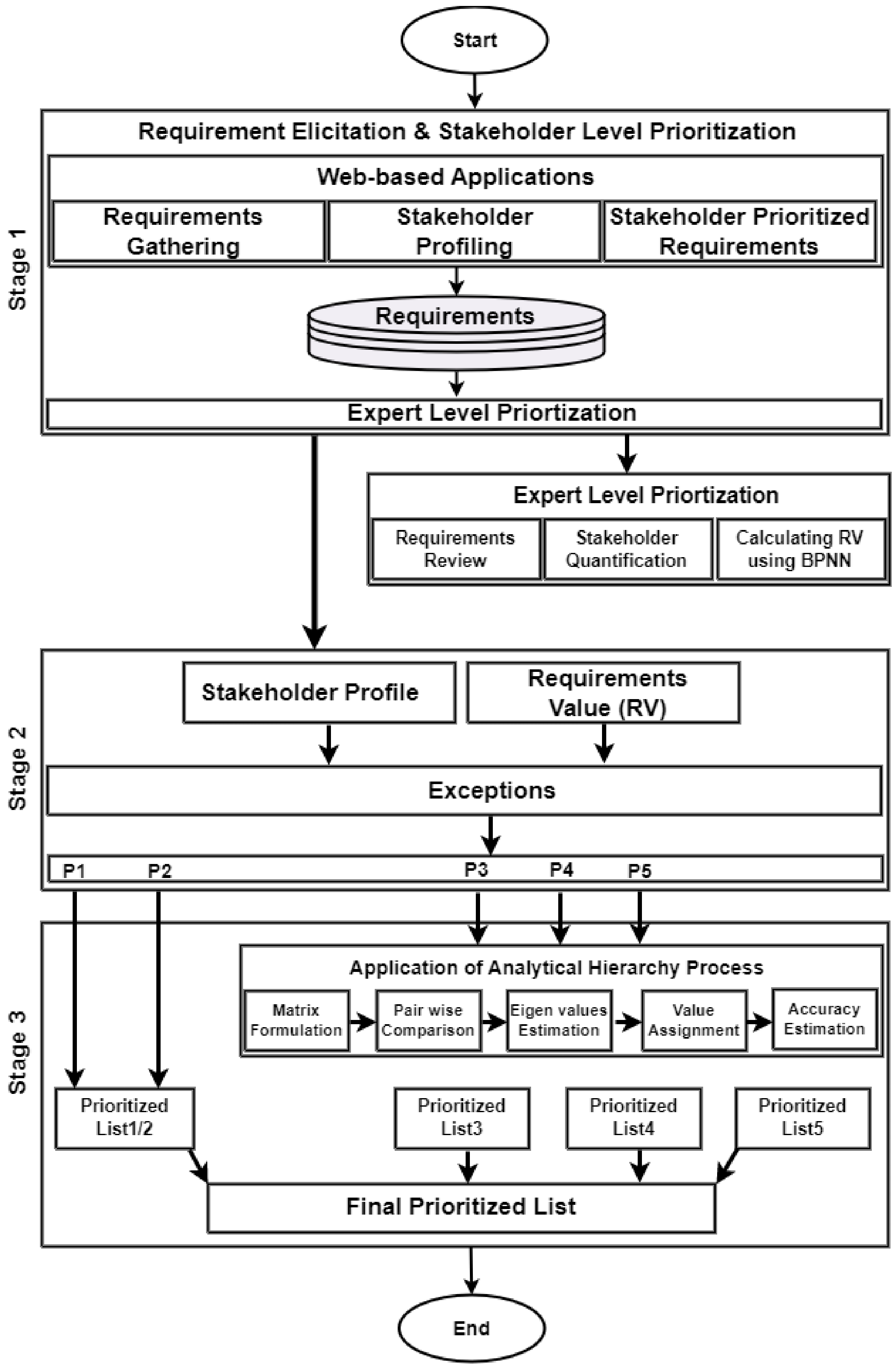


FIGURE 4.4: Stages of PHandler

Table 4.11 shows the algorithm for stage I. Algorithm II takes requirement set as input and the output of this module is Prioritized Requirement Set.

TABLE 4.11: Algorithm II

```
   function prioritizeRequirements();
1  Declare  : THRESHOLD ← n
2  Input    : RequirementsSet, StakeholderSet, n
3  Output   : PrioritizedRequirementsSet
4  if (RequirementsSet.count()≥THRESHOLD)
5      PrioritizedRequirementsSet ←
       PHandler(RequirementsSet,StakeholderSet)
6  else
7      PrioritizedRequirementsSet ← AHP(RequirementsSet)
   endfunction
```

4.1.1.3 Validation Mechanisms

To ensure objectivity and validity in the proposed framework's use of expert input, for determining requirement importance, we can implement validation mechanisms. One possible approach would be to have multiple experts provide their opinions independently and then compare and reconcile their ratings to establish a consensus. Alternatively, we can use statistical methods to, analyze the ratings and, identify any outliers or inconsistencies. These validation mechanisms would help to lessen the potential biases and errors that could arise from subjective and expert opinion setups.
Following are few examples of statistical methods that could be used to analyze ratings from experts and identify, outliers or, inconsistencies:

1. Inter-rater reliability: This measures the degree of agreement or consistency among multiple raters. It can be calculated using statistical measures such as Cohen's kappa or Fleiss' kappa, which compare the ratings of different raters to see how closely they align.

2. Mean, median, and mode: These are basic descriptive statistics that can help to recognize outliers. For example, if the mean rating is significantly

higher or lower than the median or mode, it could suggest that one or more ratings are outliers.

3. Box plots: Box plots are a visual tool for displaying the distribution of ratings. They display the median, interquartile range, and outliers of the data set. Outliers can be identified as any points that fall outside of the whiskers of the box plot.

4. Principal component analysis (PCA): PCA is a statistical technique that can be used to mark patterns and relationships in complex data sets. It can help to identify outliers by detecting any data points that are far from the others in the data set.
5. Z-scores: A z-score measures the number of standard deviations a data point is from the mean. If a rating has a z-score that is considerably higher or lower than the others, it could be an outlier.

These are just a few examples of statistical methods that could be used to validate expert input and identify, outliers or, inconsistencies. The choice of method will depend on the specific context and the nature of the data being analyzed.

4.2 Stage II

Stage II initiates by reading use cases from Use Case Set for later use in different levels of this stage. It then performs user session recording, using Selenium [217], to gather the user interaction information with reference to the system functionality. Figure 4.5 shows the block diagram of Stage II.

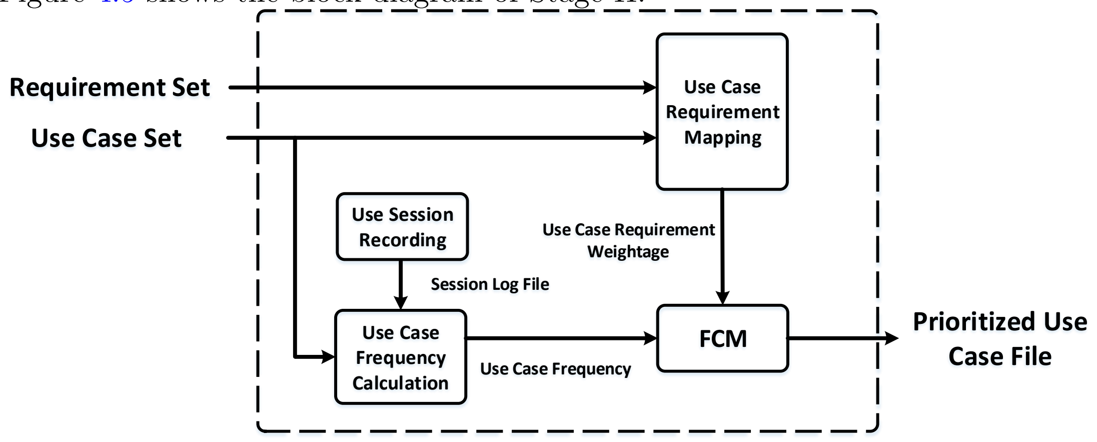
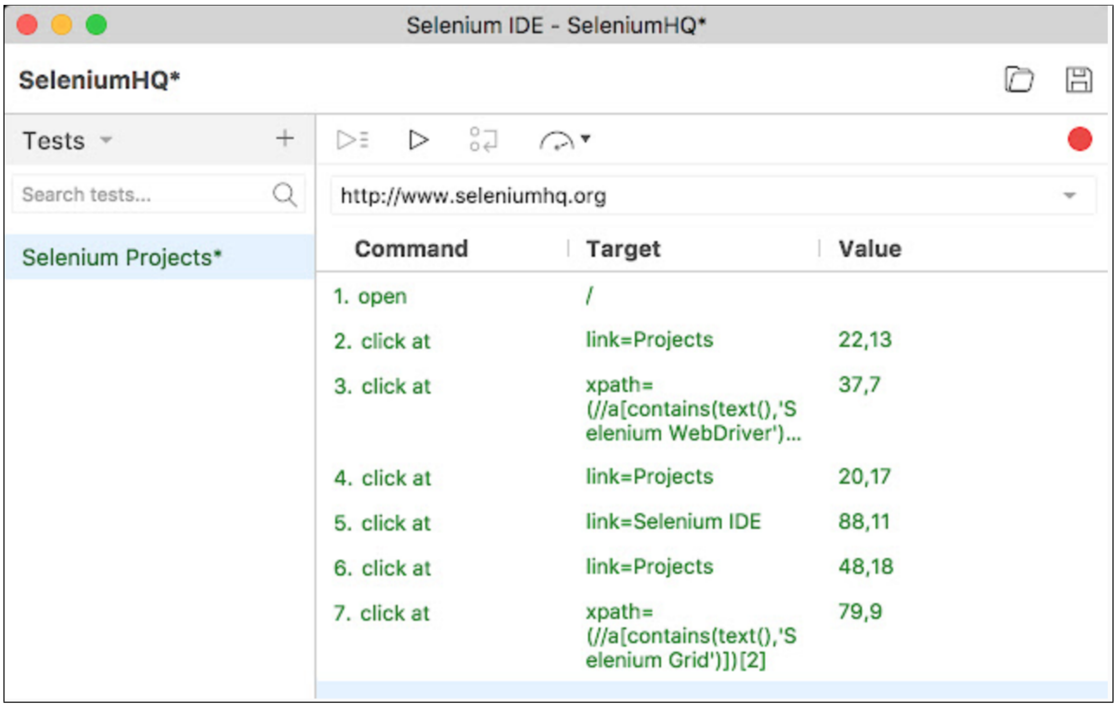

Requirement Set
Use Case Set
Use Case Requirement Mapping
Use Session Recording
Use Case Requirement Weightage
Session Log File
Use Case Frequency Calculation
Use Case Frequency
FCM
Prioritized Use Case File

FIGURE 4.5: Stages II Block Diagram

Selenium is a tool that records the user events and subsequent behavior as the user interacts with system. Selenium can then re-run these recorded steps as and when required. It uses JavaScript, iframes, and test automation engine to run, record, and rerun the user actions within the browser [218, 219]. Selenium provides basic as well as advanced commands to the users including, opening URIs, clicking elements, typing into fields, and verification commands. User can stipulate anticipated value or behavior using verification commands. Selenium also provides the option to insert programming logic while evaluating the software. Two other tools available with Selenium are the IDE and Remote Control. Selenium IDE is web browser plugin and woks on record/replay base mechanism, while Remote Control server enables the users to write their scripts in programming language of their choice. Figure 4.6 shows the Selenium IDE interface.



http://www.seleniumhq.org

Command	Target	Value
1. open	/	
2. click at	link=Projects	22,13
3. click at	xpath=(//a[contains(text(),'Selenium WebDriver')...	37,7
4. click at	link=Projects	20,17
5. click at	link=Selenium IDE	88,11
6. click at	link=Projects	48,18
7. click at	xpath=(//a[contains(text(),'Selenium Grid')])[2]	79,9

FIGURE 4.6: Selenium IDE

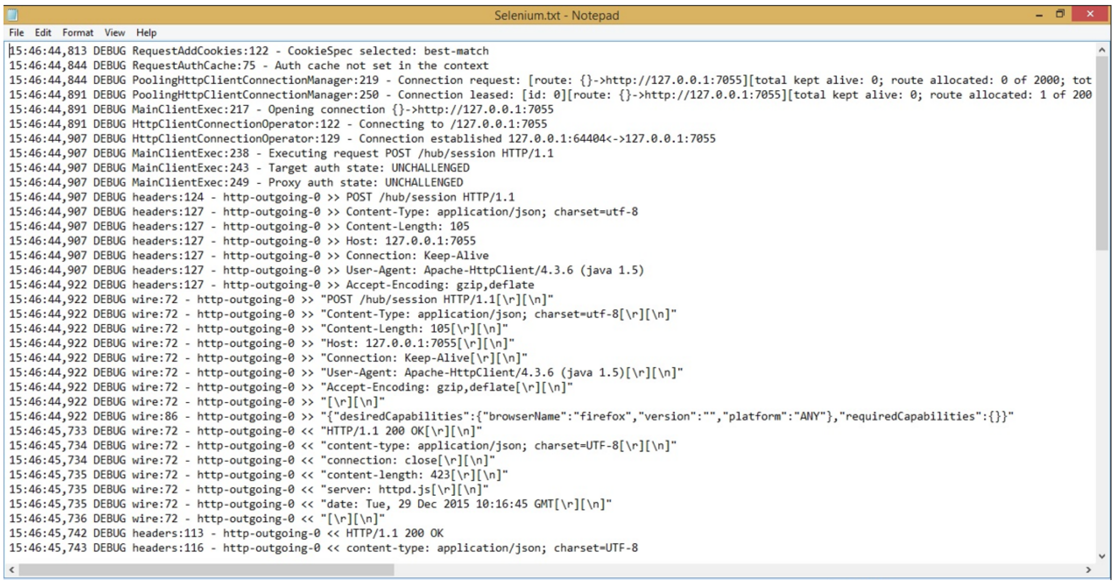
StateReduceAjax uses these recordings to calculate the use case frequency which along with Use Case Requirement Weightage UiW is input to Fuzzy C Means (FCM) classification algorithm [24]. (FCM) then generates the most pivotal use cases which are later used for DOM event element mapping in the next stage for State Machine generation. UiW is calculated by mapping each use case with its relevant prioritized requirements to generate Use Case Requirements Weightage Set. The output of this stage as mentioned before is prioritized Use Case Set. Table 4.12 depicts the algorithm for stage II.

TABLE 4.12: Algorithm III

```
     function PrioritizeUsecases ();
1    Declare  : Usecase Requirement Weight ← 0,
                Usecase Count ← 0,
                Usecase Frequency ← 0,
                Intermediate Usecase Frequency ← 0,
                Intermediate Usecase Count ← 0,
                Fuzzy Data File, Number of Clusters ← 2,
                Usecase Requirement Weightage Set,
                Selenium Log File,
                Usecase Frequency Data ← 20
2    Input    : Prioritized Requirements Set
                Usecase Set
                Ajax URL
                Session Id
3    Output   : Prioritized Usecase Set
4    foreach Usecase in Usecase Set
5        foreach Step in Usecase
6            foreach Prioritized Requirement in Prioritized
7                if ( Map(Prioritized Requirement, Step) == True)
8                    Usecase Requirement Weight +=
                     Prioritized Requirements Set.Prioritized Requirement.Weight
9            Usecase Requirement Weightage Set.append
             ([Usecase.Usecase id,Usecase Requirement Weight])
10           Usecase Requirement Weight ← 0
11           Selenium Log File ← Selenium.Execute(Session Id, Ajax URL)
12           while !EOF (Selenium Log File)
13               foreach Session id in Selenium Log File.Session id
14                   foreach Usecase id in Usecase Set.Usecase id
15                       Intermediate Usecase Count ←
                         Calculate Usecase Count
                         (Session id,Usecase id,Usecase Set, Selenium Log File)
16                       Intermediate Usecase Frequency ←
                         Usecase Count / Selenium Log File.SessionTime
17                       Intermediate Usecase Frequency Data.append
                         (Usecase id,Intermediate Usecase Frequency)
18           foreach Usecase id in Usecase Set
19               while !EOF() Intermediate Usecase Frequency Data
20                   if (Usecase id == Intermediate Usecase Frequency Data.Usecase id)
21                       Usecase Frequency ← Usecase Frequency +
                         Intermediate Usecase Frequency Data.
                         Intermediate Usecase Frequency
22                       Usecase Frequency Data.appned(Usecase id, Usecase Frequency)
23                       Usecase Frequency ← 0
24           foreach Usecase in Usecase Frequency Data
25               Fuzzy Data File.append() ← [Usecase Frequency Data.Usecase.
                 Usecase id,Usecase Frequency Data.Usecase Frequency,
                 Usecase Requirement Weightage Set.Usecase Requirement Weight]
26               Prioritized Usecase Set ← FCM.Execute
                 (Fuzzy Data File, Number of Clusters)
     endfunction
```

4.2.1 User Session Recording and Log File Generation

In this level the framework records the user sessions to identify the usage patterns of the application. Selenium records the sessions as user performs different actions on the system [219]. The user interaction with the system triggers different events and Selenium records the user action data, resulting events, and the corresponding application traces in a log file. Figure 4.7 shows the sample selenium log file.



```
15:46:44,813 DEBUG RequestAddCookies:122 - CookieSpec selected: best-match
15:46:44,844 DEBUG RequestAuthCache:75 - Auth cache not set in the context
15:46:44,844 DEBUG PoolingHttpClientConnectionManager:219 - Connection request: [route: {}->http://127.0.0.1:7055][total kept alive: 0; route allocated: 0 of 2000; tot
15:46:44,891 DEBUG PoolingHttpClientConnectionManager:250 - Connection leased: [id: 0][route: {}->http://127.0.0.1:7055][total kept alive: 0; route allocated: 1 of 200
15:46:44,891 DEBUG MainClientExec:217 - Opening connection {}->http://127.0.0.1:7055
15:46:44,891 DEBUG HttpClientConnectionOperator:122 - Connecting to /127.0.0.1:7055
15:46:44,907 DEBUG HttpClientConnectionOperator:129 - Connection established 127.0.0.1:64404<->127.0.0.1:7055
15:46:44,907 DEBUG MainClientExec:238 - Executing request POST /hub/session HTTP/1.1
15:46:44,907 DEBUG MainClientExec:243 - Target auth state: UNCHALLENGED
15:46:44,907 DEBUG MainClientExec:249 - Proxy auth state: UNCHALLENGED
15:46:44,907 DEBUG headers:124 - http-outgoing-0 >> POST /hub/session HTTP/1.1
15:46:44,907 DEBUG headers:127 - http-outgoing-0 >> Content-Type: application/json; charset=utf-8
15:46:44,907 DEBUG headers:127 - http-outgoing-0 >> Content-Length: 105
15:46:44,907 DEBUG headers:127 - http-outgoing-0 >> Host: 127.0.0.1:7055
15:46:44,907 DEBUG headers:127 - http-outgoing-0 >> Connection: Keep-Alive
15:46:44,907 DEBUG headers:127 - http-outgoing-0 >> User-Agent: Apache-HttpClient/4.3.6 (java 1.5)
15:46:44,922 DEBUG headers:127 - http-outgoing-0 >> Accept-Encoding: gzip,deflate
15:46:44,922 DEBUG wire:72 - http-outgoing-0 >> "POST /hub/session HTTP/1.1[\r][\n]"
15:46:44,922 DEBUG wire:72 - http-outgoing-0 >> "Content-Type: application/json; charset=utf-8[\r][\n]"
15:46:44,922 DEBUG wire:72 - http-outgoing-0 >> "Content-Length: 105[\r][\n]"
15:46:44,922 DEBUG wire:72 - http-outgoing-0 >> "Host: 127.0.0.1:7055[\r][\n]"
15:46:44,922 DEBUG wire:72 - http-outgoing-0 >> "Connection: Keep-Alive[\r][\n]"
15:46:44,922 DEBUG wire:72 - http-outgoing-0 >> "User-Agent: Apache-HttpClient/4.3.6 (java 1.5)[\r][\n]"
15:46:44,922 DEBUG wire:72 - http-outgoing-0 >> "Accept-Encoding: gzip,deflate[\r][\n]"
15:46:44,922 DEBUG wire:72 - http-outgoing-0 >> "[\r][\n]"
15:46:44,922 DEBUG wire:86 - http-outgoing-0 >> "{"desiredCapabilities":{"browserName":"firefox","version":"","platform":"ANY"},"requiredCapabilities":{}}"
15:46:45,733 DEBUG wire:72 - http-outgoing-0 << "HTTP/1.1 200 OK[\r][\n]"
15:46:45,734 DEBUG wire:72 - http-outgoing-0 << "content-type: application/json; charset=UTF-8[\r][\n]"
15:46:45,734 DEBUG wire:72 - http-outgoing-0 << "connection: close[\r][\n]"
15:46:45,735 DEBUG wire:72 - http-outgoing-0 << "content-length: 423[\r][\n]"
15:46:45,735 DEBUG wire:72 - http-outgoing-0 << "server: httpd.js[\r][\n]"
15:46:45,735 DEBUG wire:72 - http-outgoing-0 << "date: Tue, 29 Dec 2015 10:16:45 GMT[\r][\n]"
15:46:45,736 DEBUG wire:72 - http-outgoing-0 << "[\r][\n]"
15:46:45,742 DEBUG headers:113 - http-outgoing-0 << HTTP/1.1 200 OK
15:46:45,743 DEBUG headers:116 - http-outgoing-0 << content-type: application/json; charset=UTF-8
```

Figure 4.7: Selenium Log File

In order to record multiple sessions, multiple users use the application. Figure 4.8 shows the Selenium sessions.

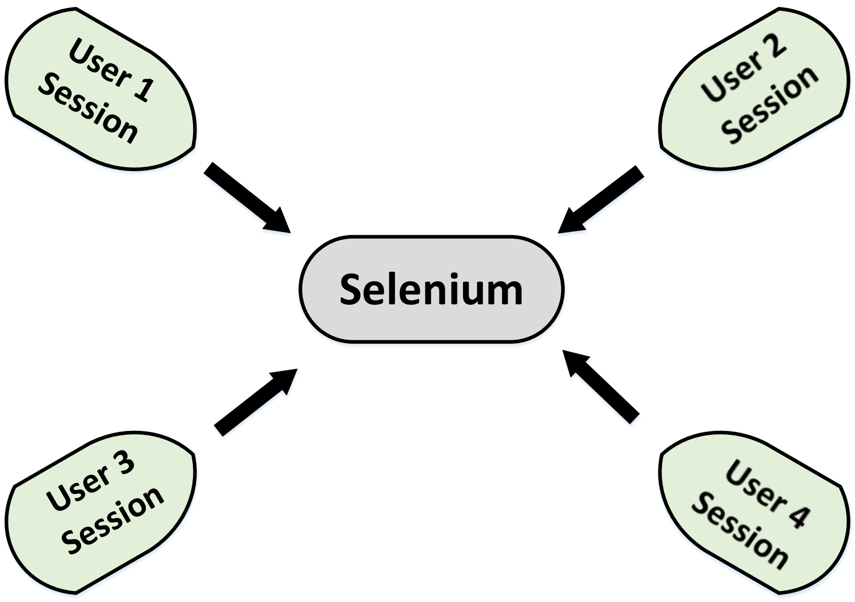


Figure 4.8: Selenium Sessions

Selenium records the usage data from multiple sessions in log files, which later merge into a central log file referred as Selenium Log File in this research. Line 8 in the Algorithm III in Table 4.12 shows the working of user session recording functionality.

4.2.2 Calculation of Use Case Frequency

The next level of Stage II calculates the use case frequency of the system which depicts the system usage in a quantifiable manner [220]. It models the way users operate the system, the actions they perform, the corresponding function calls, and the parameter value distributions. This identifies the most frequently used functions thus helping in focusing on most pivotal system areas. In this research, the identification of the most used functions becomes the base to discover and work on only those DOM changes, which emerge because of this usage pattern. This helps in identifying those DOM mutations that result from the most frequently used operations of the system.

This level starts by taking Selenium Log File and Use case set. Use case set is used to identify and map steps read from Selenium Log File with corresponding use cases. Initially the use case count is calculated for each user session from Selenium Log File and then is divided by the total user session time to calculate the use case frequency for that session. The final use case frequency is calculated by adding frequencies from all user sessions for each use case. Line 9 to 20 of the Algorithm III in Table 4.12 shows the working of use case frequency calculation. The output of use case frequency calculation is the use case frequency data. Figure 4.9 shows the Selenium sessions for Use Case Frequency Calculation.

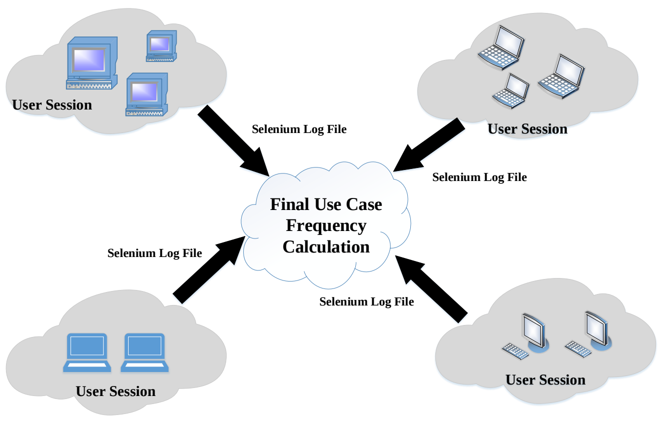


FIGURE 4.9: Use Case Frequency Calculation

4.2.3 Use Case Requirement Mapping

The next level of this stage calculates the use case requirement mapping. The requirement weightage already calculated is used in this stage to calculate the overall weightage of the use case. Every use case comprises of multiple requirements and here the solution identifies the requirements related to each use case. Figure 4.13 and 4.14 shows the requirement weightage and the requirement use case mapping.

TABLE 4.13: Requirement Weightage

ID	Weightage
R1	R1 - Weightage
R2	R2 - Weightage
R3	R3 - Weightage
R4	R4 - Weightage
.	. - Weightage
.	. - Weightage
.	. - Weightage
Rn	Rn - Weightage

TABLE 4.14: Requirement Use Case Mapping

ID	Requirment Desc.	Use Case 1	Use Case 2	Use Case 3	.	.	Use Case n
R1	R1-Description	I	.	I	.	.	I
R2	R2-Description		I		I	I	.
R3	R3-Description	I	I			I	.
R4	R4-Description		I		I		I
.	.-Description		.	.	.	.	.
.	.-Description		.	.	.	.	.
.	.-Description		.	.	.	.	.
Rn	Rn-Description				I		I

In 4.14 the 'I' at the Requirement Use Case intersection shows the use case in which the current requirement is included in. It then sums up the prioritized weights of

the requirements related to the use case and calculates the overall weight of the use case. This use case weight alongside the use case frequency is another important factor to consider for use case prioritization. Figure 4.15 shows the weightage of each use case alongwith its frequency.

TABLE 4.15: Use Case Weightage and Frequency

Use Case ID	**Use Case Weightage**	**Use Case Frequency**
Use Case 1	Sum of Weightages of all the requirements in Use case 1	Use Case 1 Frequency
Use Case 2	Sum of Weightages of all the requirements in Use case 2	Use Case 2 Frequency
Use Case 3	Sum of Weightages of all the requirements in Use case 3	Use Case 3 Frequency
.	Sum of Weightages of all the requirements in Use case .	Use Case . Frequency
.	Sum of Weightages of all the requirements in Use case .	Use Case . Frequency
.	Sum of Weightages of all the requirements in Use case .	Use Case . Frequency
Use Case n	Sum of Weightages of all the requirements in Use case n	Use Case n Frequency

Use case frequency addresses a use case in terms of its usage while the weightage of a use case addresses it in terms of the requirements it is fulfilling and their importance to the system. The more the weight of a use case, the more important it is to the system. Both these factors are then input to FCM for overall priority calculation of each use case. Line 1 to 7 of the Algorithm III in Table 4.12 shows the working of use case requirement mapping. The output of this step is Use Case Requirement Weightage Set.

4.2.4 Application of Fuzzy C Mean Clustering

The last level of Stage II implements Fuzzy C Means (FCM) classification algorithm [24] that generates the most pivotal use cases. Fuzzy is a dominant unsupervised clustering technique for data analysis and model construction. FCM is a soft clustering algorithm which processes different data elements, gives them membership labels, and refers them to one or more clusters. In comparison to other clustering algorithms, FCM demonstrates better efficiency, reliability, and robustness in most situations or applications [221–225]. Table 4.16 shows the comparison of clustersing techniques.

Table 4.16: Comparison of Clustering Techniques

Algorithm	**Input Parameters**	**Cluster Structure**	**Computational Complexity**
Hirarcical	Number of Clusters, Branching Factor, Diameter Threshold	Spherical	O(N2) (time), O(N2) (space)
K Means	Number of Clusters	Spherical	O(NKd) (time), O(N+K) (space)
Fuzzy C Mens	Number of Clusters	Spherical, Ellipsoidal	O(NCT) Near O(N)

Previously studies have used FCM and its variations in the area of software requirements prioritization [214] and pattern recognition [226]. FCM identifies key features of dataset objects and based on this information group them into clusters. It computes the correct location of an object in the dataset and assigns it to its designated cluster in a set of multiple clusters. Fuzzification parameter m determines the degree of fuzzification, in the range [1-n], of an object into a cluster. FCM starts by identifying the central points or centroids, and these centroids are referred as the mean of each cluster. The algorithm then creates the distance matrix using the Euclidean distance formula given in Equation 4.4.

$$d(x,y) = \sqrt{\sum_{i=1}^{d} |x_i - y_i|^2} \tag{4.4}$$

d is the Euclidean distance between two objects x, and y and xi and yi are the attributes of the objects. FCM assigns a membership level to every object in each cluster. It iteratively updates the centroid and membership levels and then readjusts the centroid location in each cluster of dataset. FCM uses an objective function to adjust the centroid position. This objective function takes the membership level of every object in the cluster and then calculates the distance of the object from centroid. Equation 4.5 is used in FCM to compute membership level during iterative optimization process of the FCM.

$$U_{ij} = \frac{1}{\sum_{k=1}^{c} [\frac{||x_i - c_j||}{||x_i - c_k||}]^{\frac{2}{m-1}}} \tag{4.5}$$

Equation 4.6 denotes the objective function used in the FCM.

$$J = \sum_{i=1}^{N} \cdot \sum_{j=1}^{N} U_{ij}^{m} ||x_j - v_i||^2 \tag{4.6}$$

In Equation 4.5 and 4.6 Uij is the membership of xj in ith cluster, vi is the center of the ith cluster, the bars $||\ldots||$ represent the norm metric and m constant is associated with the degree of fuzzification. FCM assigns higher values of membership to the data values closer to the centroid and minimizes the cost function while it assigns lower membership values to the data objects far away from the centroid.

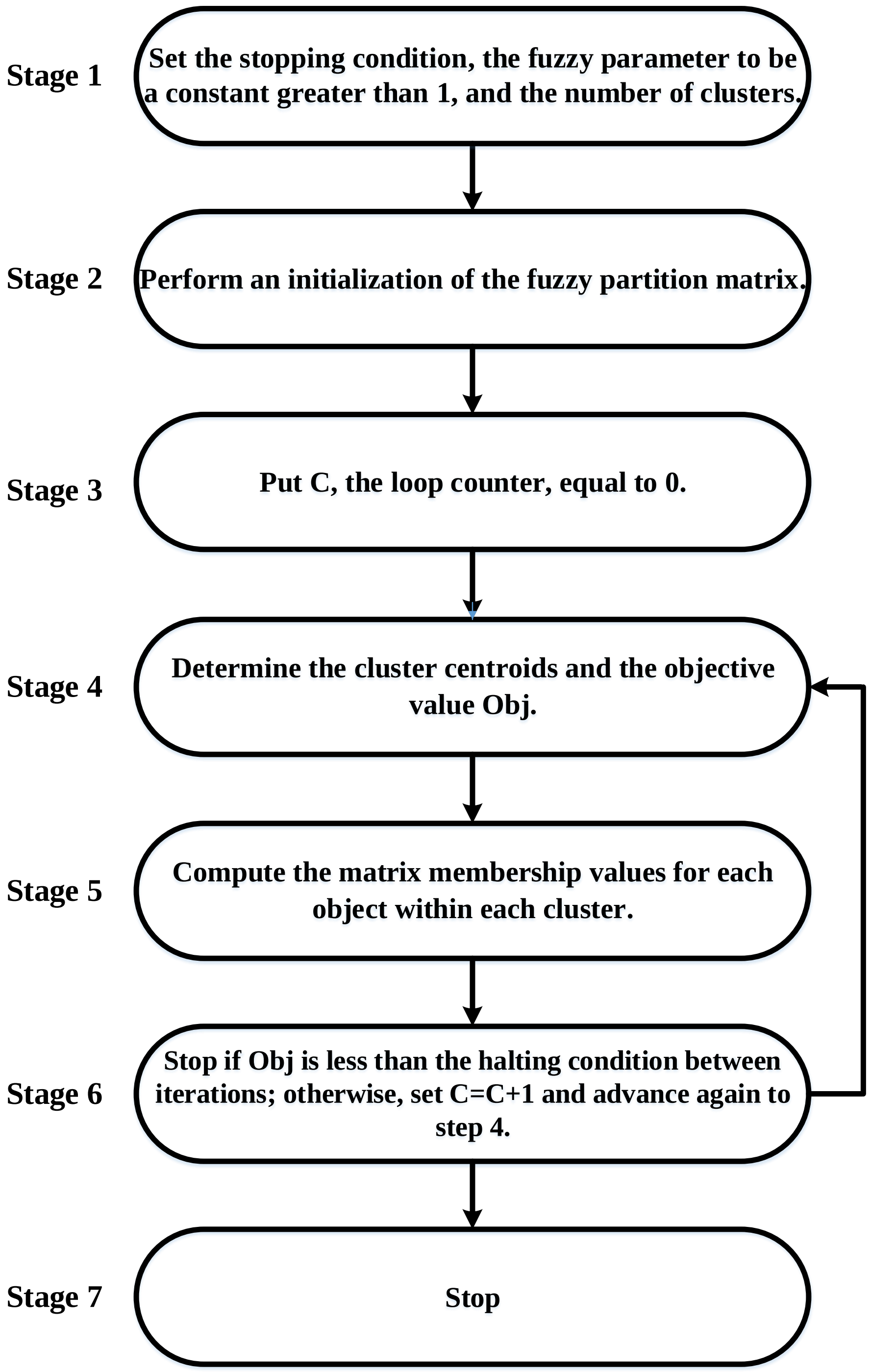

Stage 1
Set the stopping condition, the fuzzy parameter to be a constant greater than 1, and the number of clusters.
Stage 2
Perform an initialization of the fuzzy partition matrix.
Stage 3
Put C, the loop counter, equal to 0.
Stage 4
Determine the cluster centroids and the objective value Obj.
Stage 5
Compute the matrix membership values for each object within each cluster.
Stage 6
Stop if Obj is less than the halting condition between iterations; otherwise, set C=C+1 and advance again to step 4.
Stage 7
Stop

FIGURE 4.10: Fuzzy C Means Clustering Algorithm Working

The probability of association of a given data object with a cluster is shown with the membership function. This probability depicts the distance of an object from its cluster centroid. Figure 4.10 shows the working of Fuzzy C Means Clustering Algorithm.

The above mentioned situation is candidate for the use of soft computing, i.e., a learning algorithm to generate trusted results as traditional methods might not be able to handle such scenarios. We cannot anticipate the usage pattern of any user beforehand and every new user might change the patterns of the input data. We need a robust solution here, which changes itself with the change of input data and that is why FCM is used here to generate the results. Figure 4.11 shows the general FCM diagram.

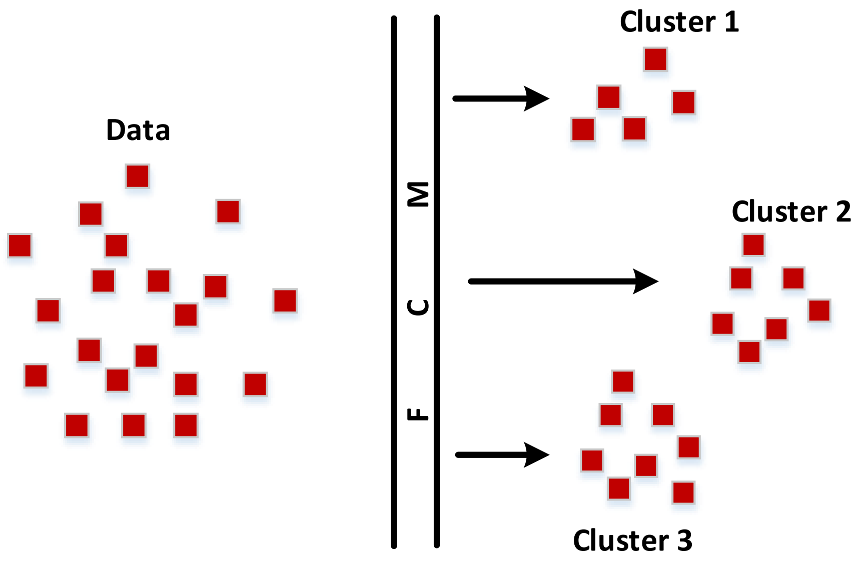


Figure 4.11: FCM Clustering

FCM in this research places data into two clusters labeled as pivotal and non-pivotal. Pivotal cluster contains the pivotal use cases which in this research are input to next stage for the generation of state machine. Line 21 to 23 of the Algorithm III in Table 4.12 shows the functionality of FCM. The output of FCM is prioritized use case set cluster.

4.3 Stage III

Stage III of the solution takes the pivotal use case list generated from the previous stage as input along with the source DOM and Use Case Set. In this state the use cases listed in the FCM generated pivotal list are rerun using Selenium to identify the pivotal user actions, resulting triggered events and the corresponding changing elements of DOM. Figure 4.12 shows the block diagram of Stage III.

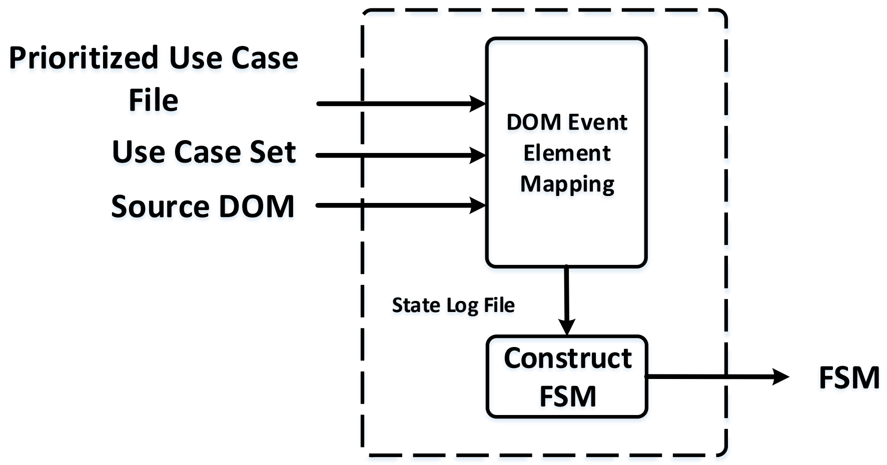
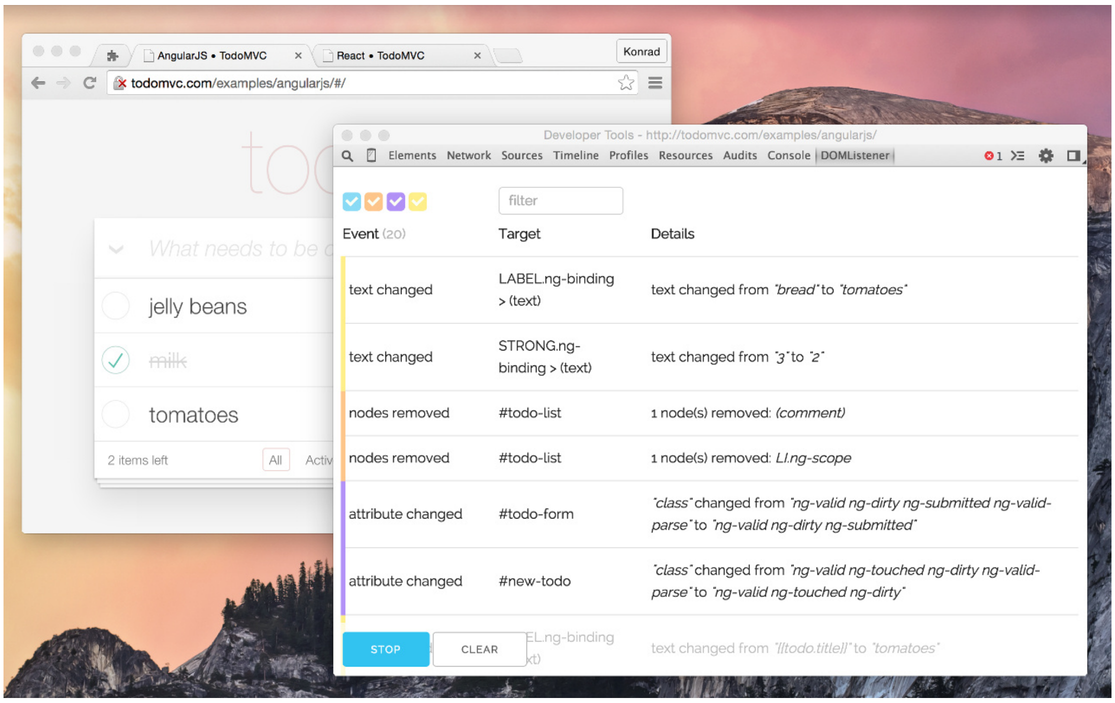


FIGURE 4.12: Stages III Block Diagram

Here it is important to consider that the actions performed by user on the Ajax web application only identify the front-end changes and the Selenium records the same. It implies that the action resulting DOM changes do not link automatically with the browser history. This means that the orthodox forward back mechanisms on the browser do not work in the Ajax application, as the DOM changes occur on the same page. In order to link the functional changes recorded by user actions with the DOM tree changes, generated as a result by those functional changes, the solution requires recording DOM tree mutations as well. For this purpose, the solution collects all application execution traces by executing pivotal use cases and subsequently matches the user actions to DOM events by employing DOM Listener [202] and HTML DOM Navigation [203] tools.

On a webpage, when there is a trigger such as a user interaction or browser manipulation it triggers of client-side JavaScript event. These events effect the DOM by executing a JavaScript function. Events include anything from a click to mouse hover over an element to a webpage loading or refreshing. An event handler, also known as an event listener, is a section of programming code—typically a user-defined JavaScript function—that gets executed whenever one of the available events is triggered.

DOM Listener provides interface for observing DOM changes i.e. node removal and addition, attribute and text modifications, resulting from executions of JavaScript handler functions in response to the fired events or user interaction. This tool is a browser extension or add-on, which runs from within the browser to help observe the DOM changes. During this process, the user can apply sorting on the observed DOM changes according to the path of elements or nature of DOM changes, and view desired DOM changes. Figure 4.13 shows the DOM Listener interface.


AngularJS • TodoMVC
React • TodoMVC
Konrad
todomvc.com/examples/angularjs/#/
Developer Tools - http://todomvc.com/examples/angularjs/
Elements Network Sources Timeline Profiles Resources Audits Console DOMListener
filter
What needs to be
jelly beans
~~milk~~
tomatoes
2 items left
All
Activ
STOP
CLEAR

Event (20)	Target	Details
text changed	LABEL.ng-binding > (text)	text changed from *"bread"* to *"tomatoes"*
text changed	STRONG.ng-binding > (text)	text changed from *"3"* to *"2"*
nodes removed	#todo-list	1 node(s) removed: *(comment)*
nodes removed	#todo-list	1 node(s) removed: *LI.ng-scope*
attribute changed	#todo-form	*"class"* changed from *"ng-valid ng-dirty ng-submitted ng-valid-parse"* to *"ng-valid ng-dirty ng-submitted"*
attribute changed	#new-todo	*"class"* changed from *"ng-valid ng-touched ng-dirty ng-valid-parse"* to *"ng-valid ng-touched ng-dirty"*
	EL.ng-binding xt)	text changed from *"{{todo.title}}"* to *"tomatoes"*

FIGURE 4.13: DOM Listener Interface

Every DOM has elements associated with it and programming tools access these elements through their respective DOM nodes. These elements have properties like position, appearance, content etc. A front-end change can alter more than one

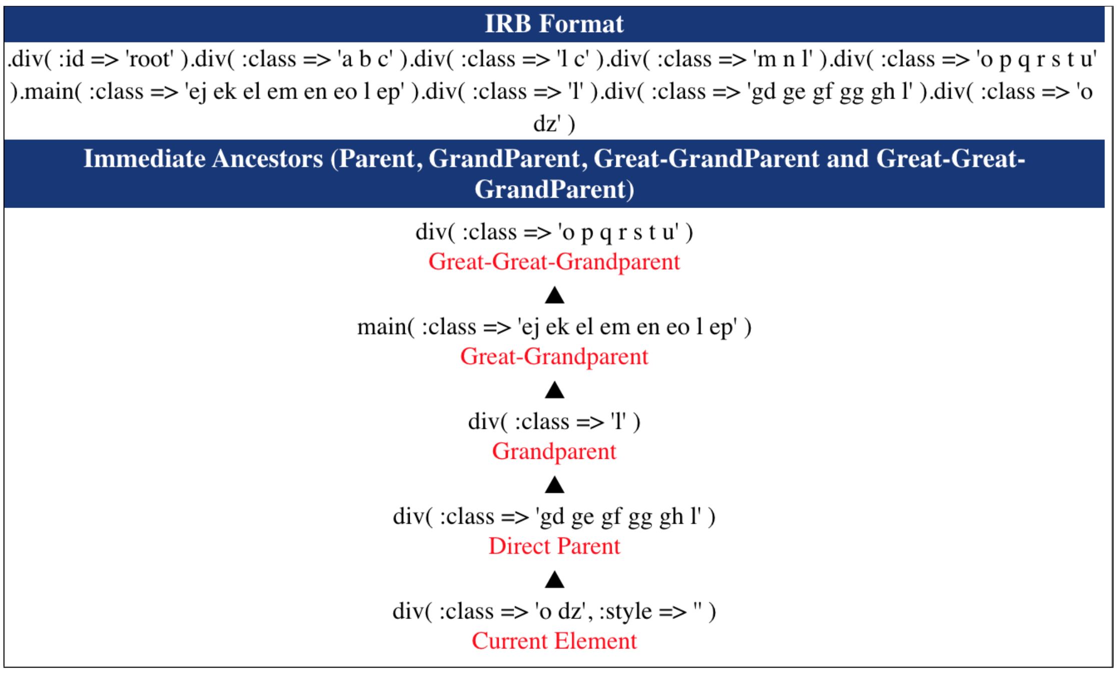
DOM nodes or elements. Therefore, there should exist a mechanism to traverse through these interrelated DOM nodes or set of elements. HTML DOM Naviga-

tion Tool provides a way of traversing DOM and helps in identifying the relative position of DOM elements or nodes. It works as a browser extension or add-on and provides a mechanism to identify DOM path corresponding to the front-end elements. User can switch between multiple views to extract useful information from the DOM paths.

Figure 4.14 shows the working of HTML DOM Navigation Tool.

The solution then maps the extracted events to corresponding DOM elements. It finally uses this event-element data to construct the state machine where the event denotes the transitions between states while the element list denotes the states itself. The output of this stage is the sate machine.

Algorithm for stage III can be seen here in Table 4.17.



IRB Format

.div(:id => 'root').div(:class => 'a b c').div(:class => 'l c').div(:class => 'm n l').div(:class => 'o p q r s t u').main(:class => 'ej ek el em en eo l ep').div(:class => 'l').div(:class => 'gd ge gf gg gh l').div(:class => 'o dz')

Immediate Ancestors (Parent, GrandParent, Great-GrandParent and Great-Great-GrandParent)

div(:class => 'o p q r s t u')
Great-Great-Grandparent
▲
main(:class => 'ej ek el em en eo l ep')
Great-Grandparent
▲
div(:class => 'l')
Grandparent
▲
div(:class => 'gd ge gf gg gh l')
Direct Parent
▲
div(:class => 'o dz', :style => '')
Current Element

FIGURE 4.14: HTML DOM Navigation Tool

Table 4.17: Algorithm IV

```
     function prioritizeRequirements();
1    Declare  : Current DOM,
                New DOM,
                Element List,
                State List,
                transition,
                targetState,
                startState,
                state Log File
2    Input    : Prioritized Usecase id, SourceDOM, Usecase Set
3    Output   : FSM Data File
4    Current DOM ← SourceDOM
5    while !EOF(Usecase Set)
6      if (Usecase Set.Usecase id == Prioritized Usecase id)
7          foreach Step in Usecase
8              Event ← Selenium.execute(Step)
9              newDOM ← browser.fetchDom(Ajax URL)
10             if (isDifferent(currDOM, newDOM))
11                 Element List ← ChangedElements(NewDOM)
12                 Curr DOM ← New DOM
13                 State List ← [Event , Element List]
14                 State Log File.append(State List)
15                 State List ← 0
16                 Element List ← 0
17   if (State Log File not empty)
18       StartState = SourceDOM
19       while (! EOF (State Log File)
20           transition ← State Log File.Event
21           targetState ← State Log File.Element List
22           FSM Data File.append
              (startState, transition, targetState) StartState ← targetState
23     drawGraph(FSM Data File)
24     State Log File ← NULL
     endfunction
```

4.3.1 DOM Event Element Mapping

State changes in an Ajax application are the changes in its DOM tree. The user action based changes recorded in Selenium are only functional in nature and a DOM change identification requires a link from the user action to the changed DOM element. In order to get these DOM changes, the solution first identifies the link between selenium-recorded actions with the triggered events and then maps those events to the modified DOM elements.

The aggregate number of DOM elements and probable concrete DOM states are commonly huge and exponential. However, this research only takes a prioritized subset of total use cases resulting in considering only the prioritized user actions, triggered events and corresponding changing elements. This results in processing only pivotal and discarding less pivotal triggered events and linked elements thus avoiding state explosion problem [227–230].

The process starts by rerunning the FCM identified prioritized use cases in browser alongside DOM listener and HTML navigation tools. A Use case comprises of user actions that trigger Ajax events [120] which are detected using DOM Listener tool. The xpath of the corresponding changing elements are identified by HTML Navigation Tool, which works along with the browser as its extension and performs internal DOM processing to identify and log dynamically changing elements as a result of trigged events.

In oredr to understand this step of solution we can take the example of Ajax powered Coffee Maker which is one of our case studies. Ajax powered Coffee Maker is an Ajax application to allow the office workers to order coffee online. There is a main coffee maker (Coffee Maker 1) and a supporting coffee maker (Coffee Maker 2), both Coffee makers can brew one cup at a time. Once a user places the order and no coffee is brewing, the main coffee maker handles the order. If an order is brewing in Coffee Maker 1 then Coffee Maker 2 will handle the additional order. If both coffee makers are busy then the user gets a wait alert. User can order coffee by entering the name (optional), selecting the beverage type and coffee cup size. Default beverage type is Mocha and default cup size is small. Suppose the user orders coffee by entering the name and default beverage and cup size. The use case is order coffee by entering name and default settings. User action is to submit the order, and parameters are user name, coffee size, and coffee type. Ajax event is OnClick () and the corresponding java script function is OrderCoffee (). Elements changed are

//INPUT[@id = 'name'],

//DIV[@id = 'controls1']/FORM[1]/P[2]/INPUT[1],

//DIV[@id = 'controls1']/FORM[1]/P[3]/INPUT[1],

//DIV[@id = 'controls1']/FORM[1]/P[4]/INPUT[1], and

//DIV[@id = 'controls1']/FORM[1]/P[4].

Figure 4.15 shows the complete working of Event Element mapping.

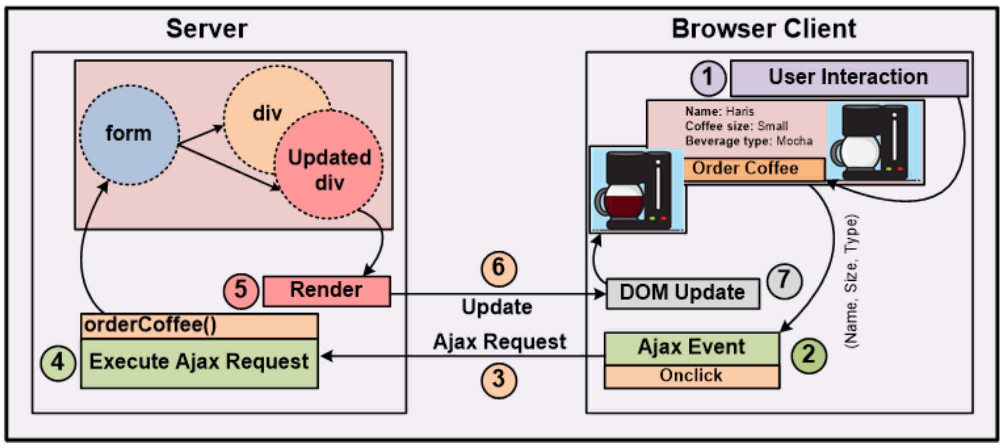


FIGURE 4.15: Event Element Mapping

Line 1 to 13 of Algorithm IV in Table 4.17 shows the working of event element mappings functionality that outputs State Log File. This file is then used in subsequent phases to construct the Sate Machine.

4.3.2 State Machine Construction

In the final phase of stage III state machine is constructed from the State Log File generated in previous phase. In this method of state machine construction, every use case has its own state machine. Table 4.18 shows the sample State Log File chunk.

TABLE 4.18: State Log File

Sr. No.	Source DOM	Destination DOM
1	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Idle) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Idle) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)
2	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Idle) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status" value(Brewing<UserID,Size,BeverageType>) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)
3	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Brewing<UserID,Size,BeverageType>) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Idle) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)
4	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Brewing<UserID,Size,BeverageType>) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status" value(Brewing<UserID,Size,BeverageType>) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status" value(Brewing<UserID,Size,BeverageType>)
5	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Brewing<UserID,Size,BeverageType>) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Brewing<UserID,Size,BeverageType>)	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Idle) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status" value(Brewing<UserID,Size,BeverageType>)
6	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Brewing<UserID,Size,BeverageType>) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Brewing<UserID,Size,BeverageType>)	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status" value(Brewing<UserID,Size,BeverageType>) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)
7	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Idle) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Brewing<UserID,Size,BeverageType>)	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status" value(Brewing<UserID,Size,BeverageType>) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status" value(Brewing<UserID,Size,BeverageType>)
8	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Idle) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Brewing<UserID,Size,BeverageType>)	html/body/div"wrapper"/div"coffeemaker1"/div"coffeemaker1-status"value(Idle) html/body/div"wrapper"/div"coffeemaker2"/div"coffeemaker2-status"value(Idle)

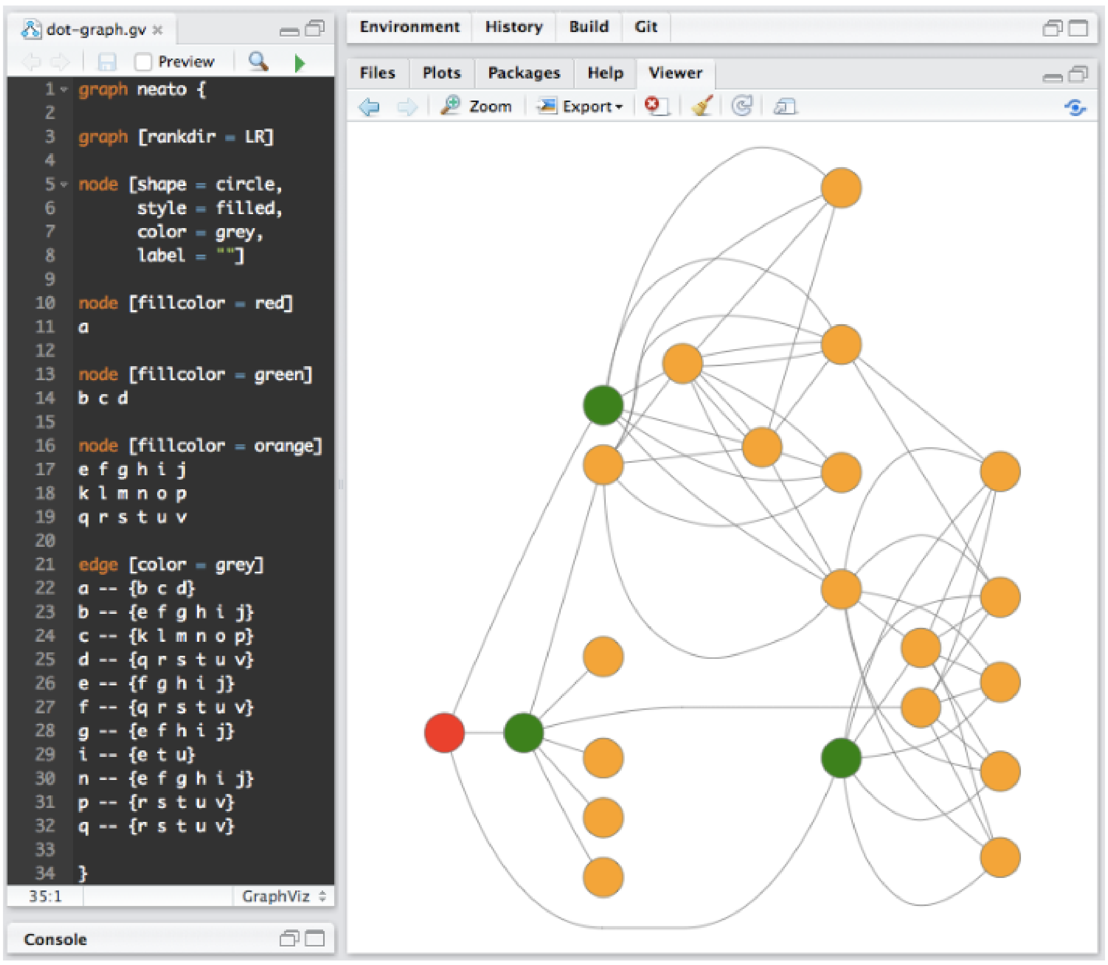
A use case depicts a partial behavior of the system and a state machine generated from the use case portrays the same.

There can be many use cases of an Ajax application, thus many state machines. However, in this research we have used FCM that filters out the most pivotal use cases of the application resulting in a prioritized use case set. The aggregate of these prioritized use cases depicts the aggregated system behavior and the same is true for state machines, their aggregate will show the overall prioritized behavior of the system. However we have left the aggregation of these state machines for the future work and this research is limited to separate state machine per use case.
A major challenge in modeling Ajax application is the number of DOM states, which can be unlimited thus making it difficult to construct a state machine. Our solution handles this state explosion problem by considering only the prioritized use case set. This prioritized set possesses a lot lesser number of user actions as compared to overall action possibilities. This limited user action set triggers a lot lesser number of events as compared to all the event possibilities resulting in only the most essential element changes. In this way, the most crucial DOM changes become the basis of state machine construction.
Line 14 to 22 in Table 4.17 shows the working of state machine construction which outputs FCM Data File.

This file is ultimately submitted to the Draw Graph function, which will draw the state machine. The Draw Graph method generates the graph by utilizing Graphviz. The DOT language is the name of the graph description language that is included in Graphviz. Additionally, Graphviz includes a variety of tools that are able to handle the DOT language. DOT is very modifiable, and it gives you control over a wide variety of layout characteristics, like the colors of lines, the shapes of arrows and nodes, and many others.

The description of the DOT graph can be sent to Graphviz in one of three different ways: as a string, as a link to a Graphviz file (file extension .gv), or as a text connection. All of these methods are supported.
First, a directive must specify whether or not an undirected or directed graph is desired for use in the Graphviz graph specification. From a semantic perspective, this shows whether or not there is a natural direction connecting one of the edge's nodes to the other. A graph can optionally also be characterised as strict. This

prohibits the development of many edges. It is possible for an undirected graph to have a maximum of one edge connecting the same two nodes, but not more. The edge with the previously defined one will be identified by subsequent edge statements using the same two nodes, and any attributes specified in the edge statement will be applied. Figure 4.16 shows the sample Graphviz interfce.



```
graph neato {

graph [rankdir = LR]

node [shape = circle,
      style = filled,
      color = grey,
      label = ""]

node [fillcolor = red]
a

node [fillcolor = green]
b c d

node [fillcolor = orange]
e f g h i j
k l m n o p
q r s t u v

edge [color = grey]
a -- {b c d}
b -- {e f g h i j}
c -- {k l m n o p}
d -- {q r s t u v}
e -- {f g h i j}
f -- {q r s t u v}
g -- {e f h i j}
i -- {e t u}
n -- {e f g h i j}
p -- {r s t u v}
q -- {r s t u v}

}
```

FIGURE 4.16: Graphviz Interface

Table 4.19 shows the use case steps along with DOM states.

TABLE 4.19: Use Case Steps with DOM States

S. No.	**Use Case Steps**	**DOM state XPath**
1	Use Case Step 1	//*[**@XPath of Use Case Step 1**]
2	Use Case Step 2	//*[**@XPath of Use Case Step 2**]
3	Use Case Step 3	//*[**@XPath of Use Case Step 3**]
4	Use Case Step 4	//*[**@XPath of Use Case Step 4**]
5	Use Case Step 5	//*[**@XPath of Use Case Step 5**]

The steps of the sample use case are input to Graphviz for state machine construction. Figure 4.17 shows the state machine for proposed used case.

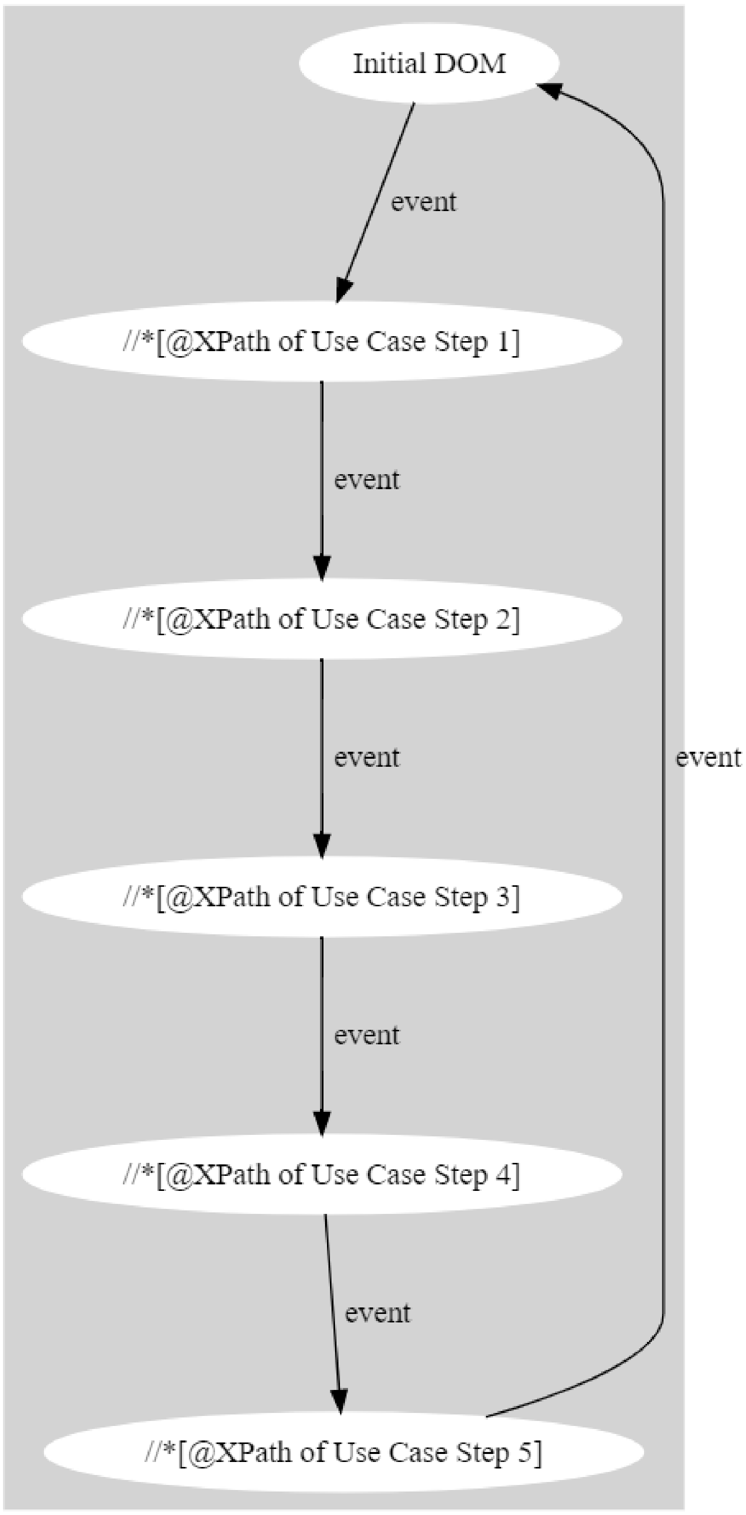

Initial DOM
event
//*[@XPath of Use Case Step 1]
event
//*[@XPath of Use Case Step 2]
event
event
//*[@XPath of Use Case Step 3]
event
//*[@XPath of Use Case Step 4]
event
//*[@XPath of Use Case Step 5]

FIGURE 4.17: Proposed Use Case

This chapter proposes a framework *StateReduceAjax* to reduce the state space of the state machine of an Ajax web application. As previously mentioned, one major challenge in model-based testing of Ajax applications is the issue of state explosion, where the larger the state space becomes, the more difficult it is to manage. The main goal of this framework is to reduce the state space.

StateReduceAjax involves a multi-stage process that progressively processes input information and generates results to aid in the construction of the state machine. The initial step involves collecting and storing stakeholder information, use cases, and system requirements.

The requirements undergo a prioritization process, and the resulting prioritized values for each requirement are stored. These prioritized requirement values are used in the use case requirement mapping to calculate the Use Case Requirement Weightage for each use case. The use case-based session recording determines how often a use case is executed.

The framework employs FCM to reduce the state space by dividing system use cases into high-priority and low-priority groups. Only the functionality of the high-priority use cases is taken into account when constructing the state machine. This reduced set of use cases helps to prevent state explosion in the state machine of an Ajax application.

FCM uses a set of use case weightage values and frequency data to identify the most critical use cases, which consist of the most commonly performed user actions. It is impractical to record all possibilities and then test the event and element mappings. To limit the number of possibilities, the framework focuses on the most frequently used events and maps them to the corresponding elements. It then runs only the pivotal use cases and uses DOM tools to map user actions to DOM mutations. The framework uses these DOM mutations, which are DOM states, to construct the state machine.

The proposed framework in this research aims to reduce the state space of generated AJAX web applications, and it comprises three stages i.e. requirement prioritization, pivotal use case identification, and state machine construction. The noteworthy contribution of this research lies in its unique approach of utilizing requirements alongside use cases and usage patterns to reduce the state space of AJAX applications. This approach has not been used before in any other research.

The proposed framework also uses a learning algorithm, Fuzzy C Means (FCM), in its second stage to identify the most pivotal use cases. With more data, the FCM algorithm ensures that the framework evolves and performs better over time. This feature is essential in today's ever-changing technological world.
The comprehensive method of the proposed framework ensures that it captures the necessary information to generate a state machine that accurately reflects the application's behavior. The use of FCM allows for more efficient and accurate identification of the most pivotal use cases, making the state machine construction more efficient. In summary, the proposed framework offers a unique approach to reduce the state space of AJAX web applications, utilizing requirements alongside use cases and usage patterns. The use of FCM as a learning algorithm ensures that the framework can evolve to perform better over time. The framework's comprehensive approach ensures that the generated state machine accurately reflects the application's behavior, making it a valuable contribution to the field of web application development.
Next chapter discusses the experimental results and analysis on those results.

Chapter 5

Experimentation and Results

This section presents the experiments conducted to prove the hypothesis stated earlier. The experiments implement *StateReduceAjax* discussed in the previous section. *StateReduceAjax* uses Ajax Powered Coffee Maker [231] and Ajax based ToDo list [232] as its case studies. These case studies are selected as they are used in many researches [11, 22, 27, 120, 233–236].

The first case study taken in this regard is a small Ajax based application called Ajax powered Coffee Maker [231].

The process initiates by reading the requirements from Requirement Set. This is a small system having 20 requirements thus a candidate for the AHP to prioritize the requirements. AHP initiates by constructing an ‘n x n’ matrix where ‘n’ denotes the total number of requirements. AHP then carries out all the steps as mentioned in section 4.1.1 to generate Weighted Requirements Matrix. Table 5.1, 5.2 and 5.3 show the Weighted Requirements Matrix for Ajax powered coffee maker.

The output of AHP is the Prioritized Requirement Set for Ajax powered Coffee Maker.

StateReduceAjax then reads the use cases from the Use Case Set. Table 5.4 shows the use cases of Ajax powered Coffee Maker.

Table 5.1: Ajax Powered Coffee Maker Requirements Weightage

R/R	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**	**R12**	**R13**	**R14**	**R15**	**R16**	**R17**	**R18**	**R19**	**R20**
R1	1.0	2.0	2.0	3.0	2.0	1.0	3.0	2.0	2.0	3.0	2.0	2.0	2.0	1.0	2.0	1.0	2.0	1.0	2.0	2.0
R2	0.5	1.0	2.0	2.0	1.0	2.0	2.0	0.5	1.0	2.0	1.0	1.0	1.0	2.0	1.0	0.5	1.0	0.5	1.0	2.0
R3	0.5	0.5	1.0	3.0	2.0	2.0	3.0	2.0	2.0	3.0	2.0	2.0	2.0	1.0	2.0	1.0	2.0	2.0	2.0	1.0
R4	0.3	0.5	0.3	1.0	0.5	3.0	1.0	3.0	0.5	1.0	0.5	0.5	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3
R5	0.5	1.0	0.5	2.0	1.0	0.5	2.0	0.5	1.0	0.5	2.0	2.0	1.0	0.5	1.0	2.0	1.0	0.5	1.0	0.5
R6	1.0	0.5	0.5	0.3	2.0	1.0	0.3	1.0	2.0	3.0	2.0	0.5	2.0	1.0	2.0	1.0	0.5	1.0	2.0	1.0
R7	0.3	0.5	0.3	1.0	0.5	3.0	1.0	0.3	0.5	1.0	0.5	2.0	0.5	3.0	0.5	0.3	0.5	0.3	0.5	0.3
R8	0.5	2.0	0.5	0.3	2.0	1.0	3.0	1.0	2.0	3.0	2.0	0.5	2.0	1.0	2.0	1.0	0.5	1.0	2.0	1.0
R9	0.5	1.0	0.5	2.0	1.0	0.5	2.0	0.5	1.0	2.0	1.0	1.0	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5
R10	0.3	0.5	0.3	1.0	2.0	0.3	1.0	0.3	0.5	1.0	0.5	2.0	0.5	0.3	0.5	3.0	0.5	0.3	0.5	3.0
R11	0.5	1.0	0.5	2.0	0.5	0.5	2.0	0.5	1.0	2.0	1.0	1.0	1.0	2.0	1.0	0.5	1.0	2.0	1.0	0.5
R12	0.5	1.0	0.5	2.0	0.5	2.0	0.5	2.0	1.0	0.5	1.0	1.0	1.0	2.0	1.0	2.0	1.0	0.5	1.0	2.0
R13	0.5	1.0	0.5	2.0	1.0	0.5	2.0	0.5	1.0	2.0	1.0	1.0	1.0	2.0	1.0	0.5	1.0	2.0	1.0	0.5
R14	1.0	0.5	1.0	3.0	2.0	1.0	0.3	1.0	2.0	3.0	0.5	0.5	0.5	1.0	2.0	1.0	0.5	1.0	2.0	1.0
R15	0.5	1.0	0.5	2.0	1.0	0.5	2.0	0.5	1.0	2.0	1.0	1.0	1.0	0.5	1.0	0.5	1.0	2.0	1.0	0.5
R16	1.0	2.0	1.0	3.0	0.5	1.0	3.0	1.0	2.0	0.3	2.0	0.5	2.0	1.0	2.0	1.0	2.0	1.0	0.5	1.0
R17	0.5	1.0	0.5	2.0	1.0	2.0	2.0	2.0	1.0	2.0	1.0	1.0	1.0	2.0	1.0	0.5	1.0	2.0	1.0	0.5
R18	1.0	2.0	0.5	3.0	2.0	1.0	3.0	1.0	2.0	3.0	0.5	2.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0
R19	0.5	1.0	0.5	2.0	1.0	0.5	2.0	0.5	1.0	2.0	1.0	1.0	1.0	0.5	1.0	2.0	1.0	2.0	1.0	2.0
R20	0.5	0.5	1.0	3.0	2.0	1.0	3.0	1.0	2.0	0.3	2.0	0.5	2.0	1.0	2.0	1.0	2.0	1.0	0.5	1.0

Table 5.2: Averaging over Normalized Column

R/R	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**	**R12**	**R13**	**R14**	**R15**	**R16**	**R17**	**R18**	**R19**	**R20**	**SUM**	**PM**
R1	0.0833	0.0976	0.1379	0.0756	0.0784	0.0411	0.0786	0.0945	0.0755	0.0818	0.0816	0.0870	0.0851	0.0423	0.0800	0.0484	0.0976	0.0455	0.0909	0.0923	1.5949	0.0797
R2	0.0417	0.0488	0.1379	0.0504	0.0392	0.0822	0.0524	0.0236	0.0377	0.0545	0.0408	0.0435	0.0426	0.0845	0.0400	0.0242	0.0488	0.0227	0.0455	0.0923	1.0533	0.0527
R3	0.0417	0.0244	0.0690	0.0756	0.0784	0.0822	0.0786	0.0945	0.0755	0.0818	0.0816	0.0870	0.0851	0.0423	0.0800	0.0484	0.0976	0.0909	0.0909	0.0462	1.4515	0.0726
R4	0.0278	0.0244	0.0230	0.0252	0.0196	0.1233	0.0262	0.1417	0.0189	0.0273	0.0204	0.0217	0.0213	0.0141	0.0200	0.0161	0.0244	0.0152	0.0227	0.0154	0.6486	0.0324
R5	0.0278	0.0244	0.0230	0.0252	0.0196	0.1233	0.0262	0.1417	0.0189	0.0273	0.0204	0.0217	0.0213	0.0141	0.0200	0.0161	0.0244	0.0152	0.0227	0.0154	0.6486	0.0324
R6	0.0417	0.0488	0.0345	0.0504	0.0392	0.0205	0.0524	0.0236	0.0377	0.0136	0.0816	0.0870	0.0426	0.0211	0.0400	0.0968	0.0488	0.0227	0.0455	0.0231	0.8716	0.0436
R7	0.0833	0.0244	0.0345	0.0084	0.0784	0.0411	0.0087	0.0472	0.0755	0.0818	0.0816	0.0217	0.0851	0.0423	0.0800	0.0484	0.0244	0.0455	0.0909	0.0462	1.0494	0.0525
R8	0.0278	0.0244	0.0230	0.0252	0.0196	0.1233	0.0262	0.0157	0.0189	0.0273	0.0204	0.0870	0.0213	0.1268	0.0200	0.0161	0.0244	0.0152	0.0227	0.0154	0.7005	0.0350
R9	0.0417	0.0976	0.0345	0.0084	0.0784	0.0411	0.0786	0.0472	0.0755	0.0818	0.0816	0.0217	0.0851	0.0423	0.0800	0.0484	0.0244	0.0455	0.0909	0.0462	1.1508	0.0575
R10	0.0417	0.0488	0.0345	0.0504	0.0392	0.0205	0.0524	0.0236	0.0377	0.0545	0.0408	0.0435	0.0426	0.0211	0.0400	0.0242	0.0488	0.0227	0.0455	0.0231	0.7556	0.0378
R11	0.0278	0.0244	0.0230	0.0252	0.0784	0.0137	0.0262	0.0157	0.0189	0.0273	0.0204	0.0870	0.0213	0.0141	0.0200	0.1452	0.0244	0.0152	0.0227	0.1385	0.7892	0.0395
R12	0.0417	0.0488	0.0345	0.0504	0.0196	0.0205	0.0524	0.0236	0.0377	0.0545	0.0408	0.0435	0.0426	0.0845	0.0400	0.0242	0.0488	0.0909	0.0455	0.0231	0.8676	0.0434
R13	0.0417	0.0488	0.0345	0.0504	0.0196	0.0822	0.0131	0.0945	0.0377	0.0136	0.0408	0.0435	0.0426	0.0845	0.0400	0.0968	0.0488	0.0227	0.0455	0.0923	0.9935	0.0497
R14	0.0417	0.0488	0.0345	0.0504	0.0392	0.0205	0.0524	0.0236	0.0377	0.0545	0.0408	0.0435	0.0426	0.0845	0.0400	0.0242	0.0488	0.0909	0.0455	0.0231	0.8872	0.0444
R15	0.0833	0.0244	0.0690	0.0756	0.0784	0.0411	0.0087	0.0472	0.0755	0.0818	0.0204	0.0217	0.0213	0.0423	0.0800	0.0484	0.0244	0.0455	0.0909	0.0462	1.0261	0.0513
R16	0.0417	0.0488	0.0345	0.0504	0.0392	0.0205	0.0524	0.0236	0.0377	0.0545	0.0408	0.0435	0.0426	0.0211	0.0400	0.0242	0.0488	0.0909	0.0455	0.0231	0.8238	0.0412
R17	0.0833	0.0976	0.0690	0.0756	0.0196	0.0411	0.0786	0.0472	0.0755	0.0091	0.0816	0.0217	0.0851	0.0423	0.0800	0.0484	0.0976	0.0455	0.0227	0.0462	1.1676	0.0584
R18	0.0417	0.0488	0.0345	0.0504	0.0392	0.0822	0.0524	0.0945	0.0377	0.0545	0.0408	0.0435	0.0426	0.0845	0.0400	0.0242	0.0488	0.0909	0.0455	0.0231	1.0197	0.0510
R19	0.0833	0.0976	0.0345	0.0756	0.0784	0.0411	0.0786	0.0472	0.0755	0.0818	0.0204	0.0870	0.0213	0.0423	0.0200	0.0484	0.0244	0.0455	0.0227	0.0462	1.0717	0.0536
R20	0.0417	0.0488	0.0345	0.0504	0.0392	0.0205	0.0524	0.0236	0.0377	0.0545	0.0408	0.0435	0.0426	0.0211	0.0400	0.0968	0.0488	0.0909	0.0455	0.0923	0.9656	0.0483

Table 5.3: Weighted Requirements Matrix

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**SUM**	**Priority**
1	**0.08**	0.11	0.15	0.1	0.06	0.04	0.16	0.07	0.12	0.11	0.08	0.09	0.1	0.04	0.1	0.04	0.12	0.05	0.11	0.1	1.82	22.78
2	**0.04**	0.05	0.15	0.06	0.03	0.09	0.1	0.02	0.06	0.08	0.04	0.04	0.05	0.09	0.05	0.02	0.06	0.03	0.05	0.1	1.2	22.88
3	**0.04**	0.03	0.07	0.1	0.06	0.09	0.16	0.07	0.12	0.11	0.08	0.09	0.1	0.04	0.1	0.04	0.12	0.1	0.11	0.05	1.67	23.03
4	0.03	0.03	0.02	0.03	0.02	0.13	0.05	0.11	0.03	0.04	0.02	0.02	0.02	0.01	0.03	0.01	0.03	0.02	0.03	0.02	0.69	21.28
5	0.04	0.05	0.04	0.06	0.03	0.02	0.1	0.02	0.06	0.02	0.08	0.09	0.05	0.02	0.05	0.08	0.06	0.03	0.05	0.02	0.98	30.21
6	0.08	0.03	0.04	0.01	0.06	0.04	0.02	0.04	0.12	0.11	0.08	0.02	0.1	0.04	0.1	0.04	0.03	0.05	0.11	0.05	1.17	26.76
7	0.03	0.03	0.02	0.03	0.02	0.13	0.05	0.01	0.03	0.04	0.02	0.09	0.02	0.13	0.03	0.01	0.03	0.02	0.03	0.02	0.78	14.87
8	0.04	0.11	0.04	0.01	0.06	0.04	0.16	0.04	0.12	0.11	0.08	0.02	0.1	0.04	0.1	0.04	0.03	0.05	0.11	0.05	1.35	38.41
9	0.04	0.05	0.04	0.06	0.03	0.02	0.1	0.02	0.06	0.08	0.04	0.04	0.05	0.02	0.05	0.02	0.06	0.03	0.05	0.02	0.89	15.5
10	0.03	0.03	0.02	0.03	0.06	0.01	0.05	0.01	0.03	0.04	0.02	0.09	0.02	0.01	0.03	0.12	0.03	0.02	0.03	0.14	0.83	22.04
11	0.04	0.05	0.04	0.06	0.02	0.02	0.1	0.02	0.06	0.08	0.04	0.04	0.05	0.09	0.05	0.02	0.06	0.1	0.05	0.02	1.02	25.81
12	0.04	0.05	0.04	0.06	0.02	0.09	0.03	0.07	0.06	0.02	0.04	0.04	0.05	0.09	0.05	0.08	0.06	0.03	0.05	0.1	1.06	24.41
13	0.04	0.05	0.04	0.06	0.03	0.02	0.1	0.02	0.06	0.08	0.04	0.04	0.05	0.09	0.05	0.02	0.06	0.1	0.05	0.02	1.03	20.83
14	0.08	0.03	0.07	0.1	0.06	0.04	0.02	0.04	0.12	0.11	0.02	0.02	0.02	0.04	0.1	0.04	0.03	0.05	0.11	0.05	1.16	26.05
15	0.04	0.05	0.04	0.06	0.03	0.02	0.1	0.02	0.06	0.08	0.04	0.04	0.05	0.02	0.05	0.02	0.06	0.1	0.05	0.02	0.97	18.87
16	0.08	0.11	0.07	0.1	0.02	0.04	0.16	0.04	0.12	0.01	0.08	0.02	0.1	0.04	0.1	0.04	0.12	0.05	0.03	0.05	1.37	33.16
17	0.04	0.05	0.04	0.06	0.03	0.09	0.1	0.07	0.06	0.08	0.04	0.04	0.05	0.09	0.05	0.02	0.06	0.1	0.05	0.02	1.15	19.74
18	0.08	0.11	0.04	0.1	0.06	0.04	0.16	0.04	0.12	0.11	0.02	0.09	0.02	0.04	0.03	0.04	0.03	0.05	0.03	0.05	1.25	24.43
19	0.04	0.05	0.04	0.06	0.03	0.02	0.1	0.02	0.06	0.08	0.04	0.04	0.05	0.02	0.05	0.08	0.06	0.1	0.05	0.1	1.1	20.57
20	0.04	0.03	0.07	0.1	0.06	0.04	0.16	0.04	0.12	0.01	0.08	0.02	0.1	0.04	0.1	0.04	0.12	0.05	0.03	0.05	1.3	26.83

Table 5.4: Ajax Powered Coffee Maker Use Cases

Sr. No.	**Use Case**
UC_1	Place Order with Default Settings
UC_2	Place Order by Entering Name and Default Settings
UC_3	Place Order bySelecting Size and Default Settings
UC_4	Place Order by Selecting Beverage Type and Default Settings
UC_5	Place Order by Entering Name, Size and Beverage Type
UC_6	Acknowledge Wait Request

It uses Prioritized Requirement Set and Use Case Set to apply use case-requirements mapping. Table 5.5 shows the requirement weightage.

Table 5.5: Ajax Powered Coffee Maker Requirement Weightage

Requirement ID	**Requirement Weightage**
1	22.77872007
2	22.87712412
3	23.02900707
4	21.27860694
5	30.20582427
6	26.76265
7	14.86613933
8	38.4093347
9	15.49634389
10	22.04218892
11	25.80990133
12	24.40635319
13	20.82877419
14	26.045751
15	18.87053865
16	33.15820938
17	19.74255956
18	24.43362155
19	20.57234037
20	26.83404868

Table 5.6 shows the requirement use case mapping.

Table 5.6: Ajax Powered Coffee Maker Requirement Use Case Mapping

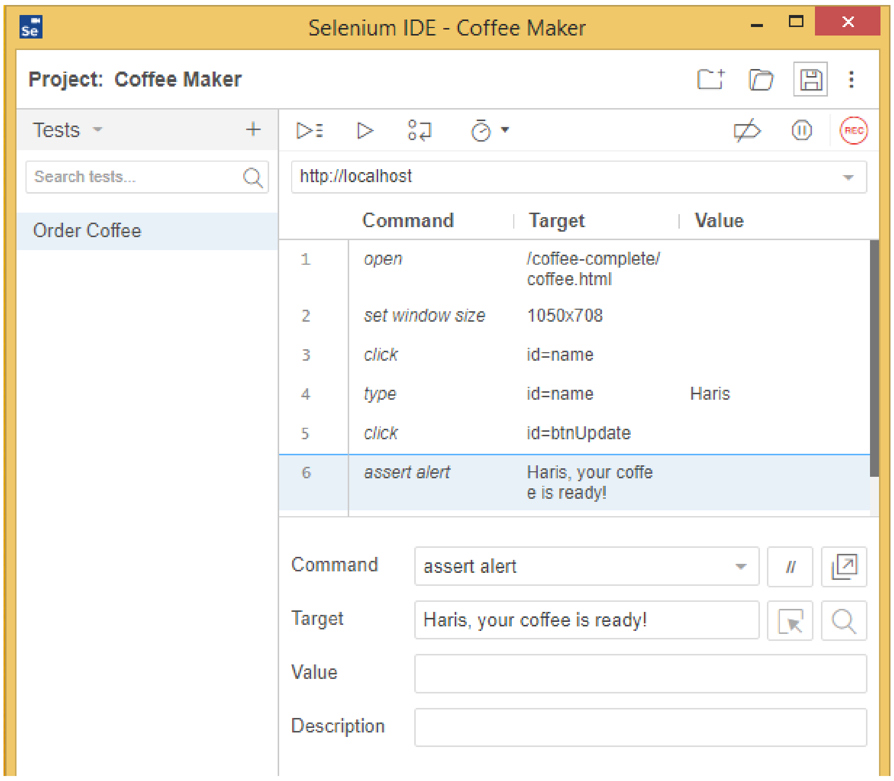
Use Case	**Mapped Requirements**
UC1	9,10,11,12,13,14
UC2	2,3,9,10,11,12,13,14
UC3	4,5,9,10,11,12,13,14
UC4	6,7,9,10,11,12,13,14
UC5	9,10,11,12,13,14
UC6	15,16,17,18,19

Table 5.7 shows the use case weightage.

TABLE 5.7: Ajax Powered Coffee Maker Use Case Weightage

Use Case	Weightage
UC1	134.6293125
UC2	180.5354437
UC3	186.1137437
UC4	176.2581018
UC5	134.6293125
UC6	116.7772695

The output of this function is the Use case Requirement Weightage Data having Use cases along with their weights. The next stage of StateReduceAjax uses Selenium and record user sessions to identify usage patterns. Selenium archives the user action statistics in a log file. This leads StateReduceAjax to calculate use case frequency of the system. This identifies the most frequently used functions thus helping in focusing on most pivotal system areas for testing. Figure 5.1 shows the Selenium IDE session for Coffer Maker.



	Command	Target	Value
1	open	/coffee-complete/coffee.html	
2	set window size	1050x708	
3	click	id=name	
4	type	id=name	Haris
5	click	id=btnUpdate	
6	assert alert	Haris, your coffee is ready!	

Command: assert alert
Target: Haris, your coffee is ready!
Value:
Description:

FIGURE 5.1: Selenium IDE for Coffee Maker

In this research, the identification of the most used functions becomes the base to discover and work on only those DOM changes, which emerge because of this usage pattern. The output of this step is the Use Case Frequency Data. The last level of Stage II implements Fuzzy C mean classification algorithm (FCM) [24]. FCM takes features of the objects and classifies them into clusters. It iteratively calculates the appropriate position the objects, identify their suitable cluster among multiple clusters and place them in it. Table 5.8 shows the Ajax Powered Coffee Maker Fuzzy Input Set taken from Use Case Requirement Weightage Data and Use Case Frequency Data respectively.

TABLE 5.8: Ajax Powered Coffee Maker Fuzzy Input

Use Case	Weightage	Frequency
UC1	**134.6293125**	**7**
UC2	**180.5354437**	**23**
UC3	**186.1137437**	**15**
UC4	**176.2581018**	**14**
UC5	**134.6293125**	**8**
UC6	**116.7772695**	**12**

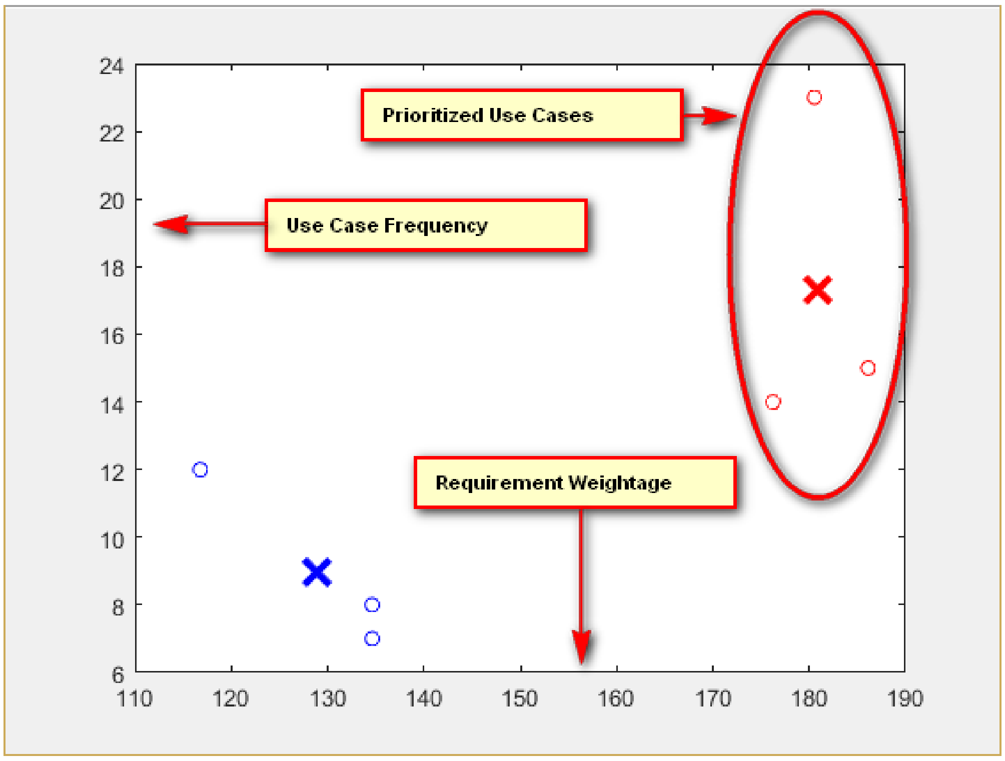
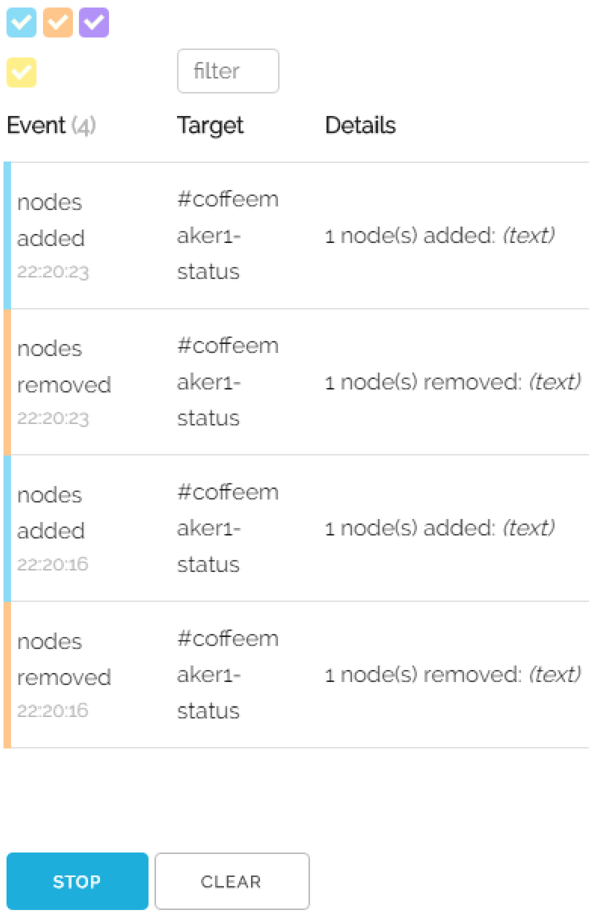


FIGURE 5.2: FCM Results for Ajax Powered Coffee Maker

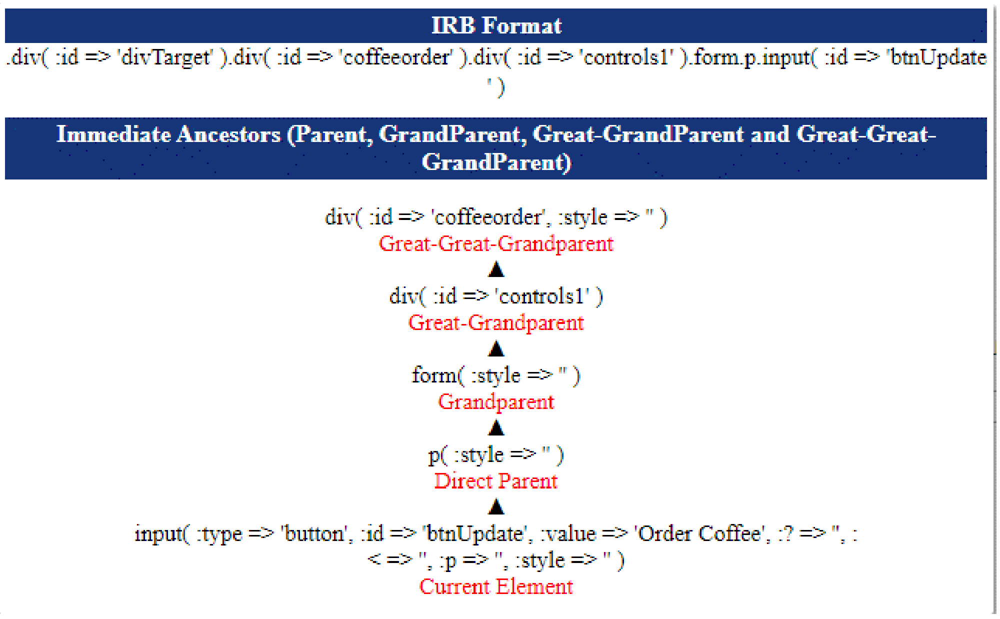
The output of FCM is the Prioritized Use Case Set having two clusters, i.e., pivotal and non-pivotal. Pivotal cluster represents the prioritized use cases. These use cases give the most essential user actions and the solution identifies event element mapping only for these user actions. The event element mapping of this reduced action set gives limited events, thus reduced DOM mutation set, and state machine. However, as the identified action set comprise the most vital user actions so the state machine models the most essential system behavior. Figure 5.2 shows the Fuzzy C Mean results of the Coffer Maker.
DOM event element relationship links a triggered event, because of user action, with resulting elements that change their behavior. Here the model will create a list of elements which alter behavior in reference to associated event. Here the solution uses DOM Listener [202] and HTML DOM Navigation [203] tools to rerun application execution traces by executing pivotal use cases and subsequently matches the user actions to DOM events. Figure 5.3 shows the DOM Listener for Coffee Maker.



Event (4)	Target	Details
nodes added 22:20:23	#coffeemaker1-status	1 node(s) added: *(text)*
nodes removed 22:20:23	#coffeemaker1-status	1 node(s) removed: *(text)*
nodes added 22:20:16	#coffeemaker1-status	1 node(s) added: *(text)*
nodes removed 22:20:16	#coffeemaker1-status	1 node(s) removed: *(text)*

FIGURE 5.3: DOM Listener Interface for Coffee Maker

Figure 5.4 shows the HTML DOM Navigation for Coffee Maker.



IRB Format

.div(:id => 'divTarget').div(:id => 'coffeeorder').div(:id => 'controls1').form.p.input(:id => 'btnUpdate ')

Immediate Ancestors (Parent, GrandParent, Great-GrandParent and Great-Great-GrandParent)

div(:id => 'coffeeorder', :style => '')
Great-Great-Grandparent
▲
div(:id => 'controls1')
Great-Grandparent
▲
form(:style => '')
Grandparent
▲
p(:style => '')
Direct Parent
▲
input(:type => 'button', :id => 'btnUpdate', :value => 'Order Coffee', :? => '', :< => '', :p => '', :style => '')
Current Element

Figure 5.4: Coffee Maker HTML DOM Navigation Interface

Table 5.9 shows the "Place Order by Entering Name and Default Settings" use case steps along with DOM states.

Table 5.9: Place Order by Entering Name and Default Settings

S. No.	Use Case Steps	DOM state XPath
1	Enter Name	**//div[@id='controls1']/form/p/input**
		//div[@id='controls1']/form/p/input value = "Name"
2	Order Coffee	**//div[@id="controls1"]/form/p[4]/input**

Figure 5.5 shows the state machine for use case.

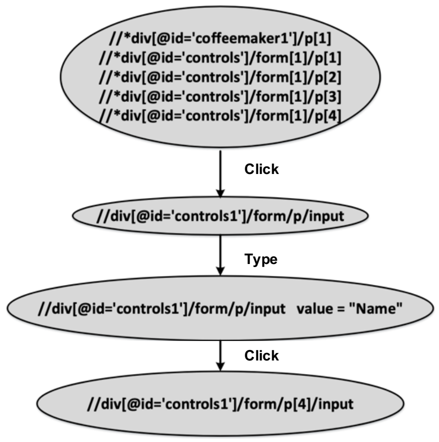

//*div[@id='coffeemaker1']/p[1]
//*div[@id='controls']/form[1]/p[1]
//*div[@id='controls']/form[1]/p[2]
//*div[@id='controls']/form[1]/p[3]
//*div[@id='controls']/form[1]/p[4]
Click
//div[@id='controls1']/form/p/input
Type
//div[@id='controls1']/form/p/input value = "Name"
Click
//div[@id='controls1']/form/p[4]/input

Figure 5.5: State Machine of "Place Order by Entering Name and Default Settings" Use Case

Table 5.10 shows the "Place Order by Selecting Size and Default Settings" use case steps along with DOM states.

Table 5.10: Place Order by Selecting Size and Default Settings

S. No.	**Use Case Steps**	**DOM state XPath**
1	Select Size	**//div[@id="controls1"]/form/p[2]/input[2]**
2	Order Coffee	**//div[@id="controls1"]/form/p[4]/input**

Figure 5.6 shows the state machine for use case.

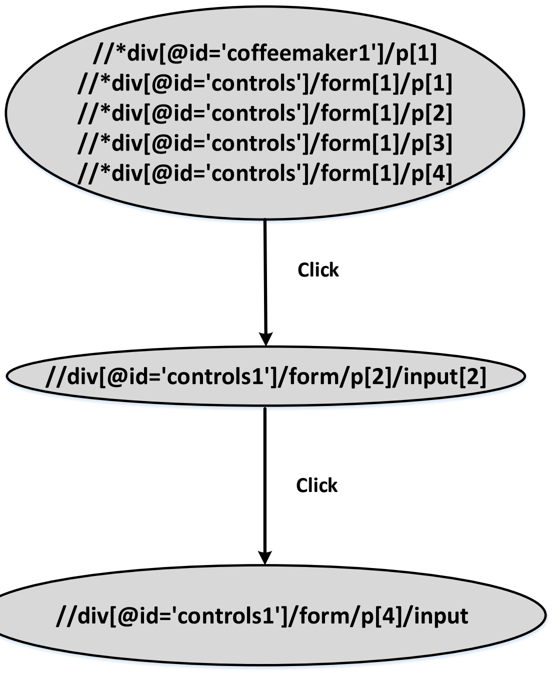

//*div[@id='coffeemaker1']/p[1]
//*div[@id='controls']/form[1]/p[1]
//*div[@id='controls']/form[1]/p[2]
//*div[@id='controls']/form[1]/p[3]
//*div[@id='controls']/form[1]/p[4]
Click
//div[@id='controls1']/form/p[2]/input[2]
Click
//div[@id='controls1']/form/p[4]/input

Figure 5.6: State Machine of "Place Order by Selecting Size and Default Settings" Use Case

Table 5.11 shows the "Place Order by Selecting Beverage Type and Default Settings" use case steps along with DOM states.

Table 5.11: Place Order by Selecting Beverage Type and Default Settings

S. No.	**Use Case Steps**	**DOM state XPath**
1	Select Beverage Type	**//div[@id="controls1"]/form/p[3]/input[2]**
2	Order Coffee	**//div[@id="controls1"]/form/p[4]/input**

Figure 5.7 shows the state machine for use case.

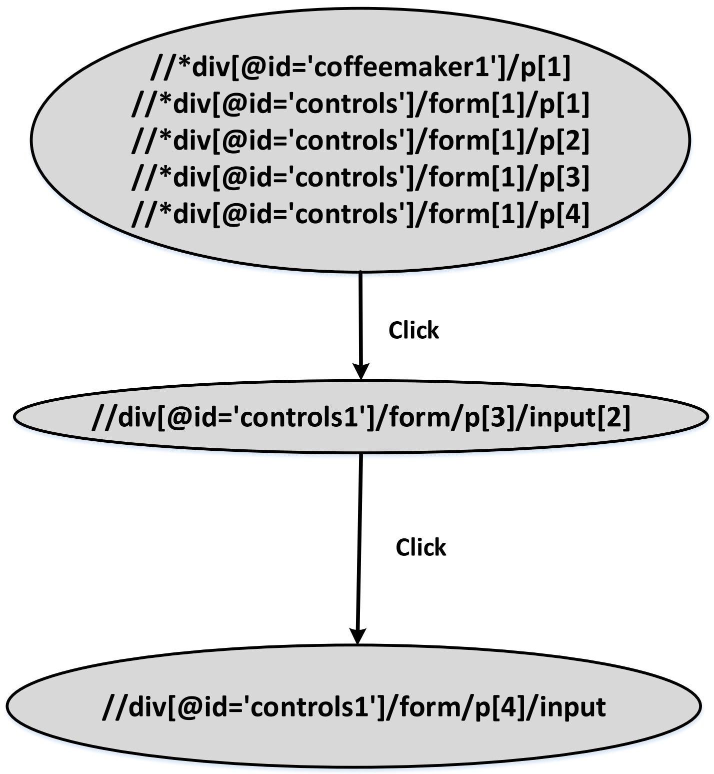

//*div[@id='coffeemaker1']/p[1]
//*div[@id='controls']/form[1]/p[1]
//*div[@id='controls']/form[1]/p[2]
//*div[@id='controls']/form[1]/p[3]
//*div[@id='controls']/form[1]/p[4]
Click
//div[@id='controls1']/form/p[3]/input[2]
Click
//div[@id='controls1']/form/p[4]/input

FIGURE 5.7: State Machine of "Place Order by Selecting BeverageType and Default Settings" Use Case

Table 5.12 shows the "Place Order by Entering Name, Size and Beverage Type" use case steps along with DOM states.

TABLE 5.12: Place Order by Entering Name, Size and Beverage Type

S. No.	Use Case Steps	DOM state XPath
1	Enter Name	//div[@id='controls1']/form/p/input
		//div[@id='controls1']/form/p/input value = "Name"
2	Select Size	//div[@id="controls1"]/form/p[2]/input[2]
3	Select Beverage Type	//div[@id="controls1"]/form/p[3]/input[2]
4	Order Coffee	//div[@id="controls1"]/form/p[4]/input

Figure 5.8 shows the state machine for use case.

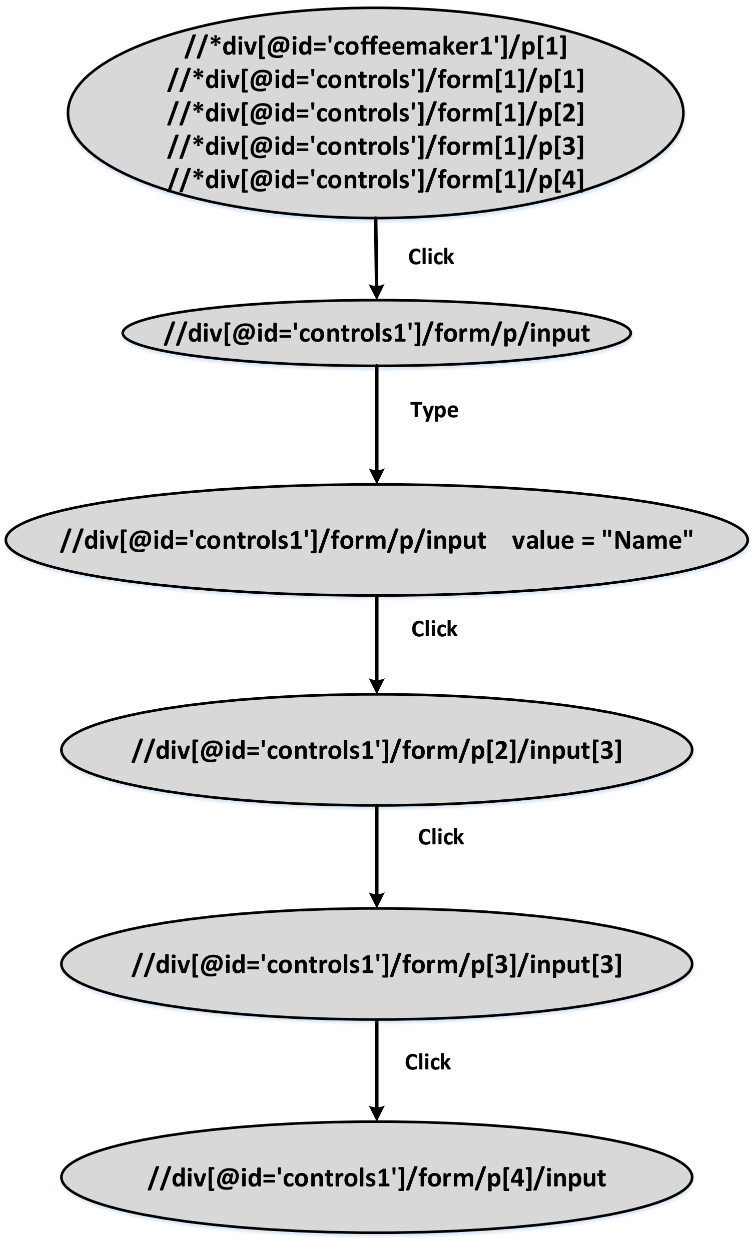

//*div[@id='coffeemaker1']/p[1]
//*div[@id='controls']/form[1]/p[1]
//*div[@id='controls']/form[1]/p[2]
//*div[@id='controls']/form[1]/p[3]
//*div[@id='controls']/form[1]/p[4]
Click
//div[@id='controls1']/form/p/input
Type
//div[@id='controls1']/form/p/input value = "Name"
Click
//div[@id='controls1']/form/p[2]/input[3]
Click
//div[@id='controls1']/form/p[3]/input[3]
Click
//div[@id='controls1']/form/p[4]/input

FIGURE 5.8: Place Order by Entering Name, Size and Beverage Type

Second case study is an Ajax based ToDo list [232]. The ToDo List helps the users to manage daily task list in Ajax style. It includes the features like multiple lists, task notes, tags, due dates, task priority, and task sorting searching. The system falls in large requirement set so PHandler is used to prioritize the requirements. PHandler takes stakeholder and requirements data to carry out the prioritization process. Requirement classification factors projRCFs and reqRCFs are used to calculate the value of requirements RV. Table 5.13 shows the RV values of some ToDo list requirements. The output of PHandler is the Prioritized Requirement

Table 5.13: RV Value of ToDo List Requirements

Feasibility	Modifiability	Urgency	Traceability	Testability	Completeness	Consistency	Understandability	Within Scope	Non Redundant	Total
5	4	5	5	5	5	5	5	5	5	1.33
2	1	1	3	2	1	2	2	1	1	0.67
2	1	2	3	2	1	2	2	1	1	0.69
3	2	2	1	2	2	2	2	2	1	0.73
4	3	4	3	3	3	4	3	3	3	1.01
2	1	2	1	2	1	2	2	1	1	0.65
2	1	1	1	2	1	2	2	1	1	0.63

Set. The solution then reads ToDo list use cases from Use Case Set and uses both these sets to apply use case-requirements mapping. Table 5.14 shows a chunk of ToDo list use cases.

Table 5.14: ToDo List Use Case Chunk

Sr. No.	**Use Case**
UC10	Filter task/s by tag/s
UC11	Delete a task
UC12	Mark a task as complete
UC13	Select option to view incomplete tasks
UC14	Rename a list
UC15	Delete a list
UC16	Clear completed tasks
UC17	Export a list
UC18	Publish a list
UC19	Move a task from one list to another list
UC20	Arrange the lists by hand

The output of this function is the Use Case Requirement Weightage Data. StateReduceAjax records user sessions to identify usage patterns in Selenium Log File. StateReduceAjax then calculates the use case frequency of the system and logs into Use Case Frequency Data. The next stage of StateReduceAjax implements

(FCM). Table 5.15 shows the ToDo List Fuzzy input Set taken from Use Case Requirement Weightage Data and Use Case Frequency Data respectively.

TABLE 5.15: ToDo List FCM Input Data

Use Case	Weightage	Frequency	Use Case	Weightage	Frequency	Use Case	Weightage	Frequency	Use Case	Weightage	Frequency
UC1	3.4	7	UC12	4.2	103	UC23	1.1	61	UC34	4.3	3
UC2	3	141	UC13	5.1	44	UC24	1	11	UC35	2.8	1
UC3	6.8	63	UC14	1.1	5	UC25	1	31	UC36	2.2	1
UC4	3.5	33	UC15	1.3	3	UC26	1	26	UC37	1.6	1
UC5	4.2	8	UC16	1.1	13	UC27	0.9	15	UC38	3.1	2
UC6	3.4	13	UC17	2	21	UC28	1	71	UC39	2.2	2
UC7	2.6	19	UC18	1	2	UC29	1	57	UC40	4.7	7
UC8	1.2	127	UC19	1.2	17	UC30	0.9	16	UC41	1.5	1
UC9	1.2	35	UC20	1	9	UC31	1	6	UC42	2.1	17
UC10	2	17	UC21	1	30	UC32	1.7	2			
UC11	1	7	UC22	1	54	UC33	2.5	0			

The output of FCM is the prioritized use case set having two clusters, i.e., pivotal and non-pivotal. Figure 5.9 shows the Fuzzy C Mean results of the ToDo list.

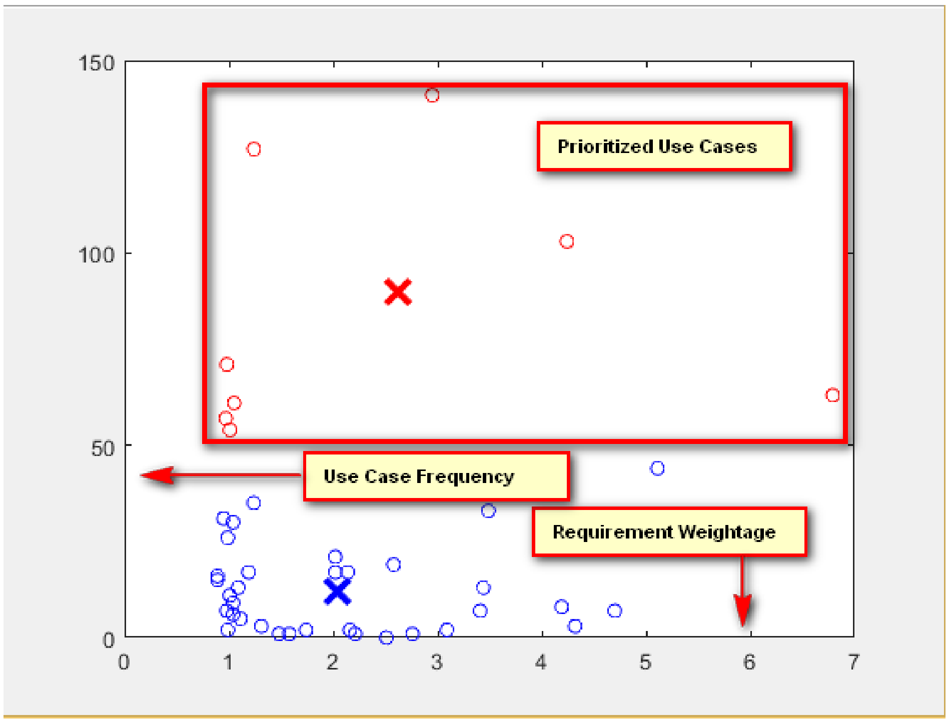


FIGURE 5.9: FCM Results of ToDo List

These FCM identified use cases give the most frequently used user actions and the solution identifies event element mapping only for these user actions. The event element mapping of this reduced action set gives limited events, thus reduced DOM mutation set, and state machine. However, as the identified action set comprise the most vital user actions so the state machine models the most frequent system behavior.

StateReduceAjax uses DOM Listener and HTML DOM Navigation tool to identify the links among front end user actions to the corresponding triggered events and further to the changed elements. These changing elements represent the DOM mutations which in fact are the Ajax application state changes.

The output of DOM event element mapping is the state log file having event element mapping records for each use case. This state log file for each use case is input to Construct FSM function.

Table 5.16 shows the "Add Task with Add Button to an Already Created List" use case steps along with DOM states.

TABLE 5.16: Add Task with Add Button to an Already Created List

S. No.	**Use Case Steps**	**DOM state XPath**
1	Select List	**//li[@id="list_ name"]/a/span**
2	Enter Task Name	**//form[@id="newtask_form"]/input** **//form[@id="newtask_form"]/input value "task_name"**
3	Click Add Button	**//form[@id="newtask_form"]/div**

Figure 5.10 shows the state machine for proposed used case.

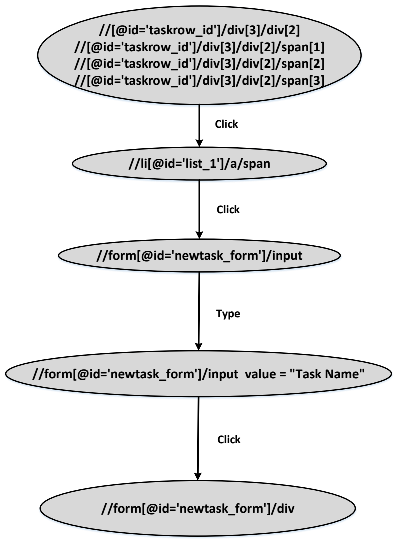

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//li[@id='list_1']/a/span
Click
//form[@id='newtask_form']/input
Type
//form[@id='newtask_form']/input value = "Task Name"
Click
//form[@id='newtask_form']/div

FIGURE 5.10: Add Task with Add Button

TABLE 5.17: Add Task with Advance Button to an Already Created List

S. No.	Use Case Steps	DOM state XPath
1	Select List	**//li[@id='list_ name']/a/span**
2	Enter Task Name	**//form[@id='newtask_form']/input**
		//form[@id='newtask_form']/input
		value 'task_name'
3	Click Advanced Button	**//div[2]/div[2]/div/div/a/span**
4	Select Priority	**//form['taskedit_form']/div/select**
		//form['taskedit_form']/div/select
		value "priority_value"
5	Select Due Date	**//form[@id='taskedit_form']/div[2]/img**
		//div[@id='ui-datepicker-div']
		/table/tbody/tr[2]/td[4]/a
6	Enter Note	**//form[@id'taskedit_form']//div[5]**
		/textarea
		//form[@id'taskedit_form']//div[5]
		/textarea value 'note_text'
7	Enter Tag	**//form[@id='taskedit_form']/div[6]**
		/table/tbody/tr/td/input
		//form[@id='taskedit_form']/div[6]
		/table/tbody/tr/td/input value "tag_name"
8	Click Save Button	**//form[@id='taskedit_form']/div[8]/input**

Figure 5.11 shows the state machine for proposed used case.

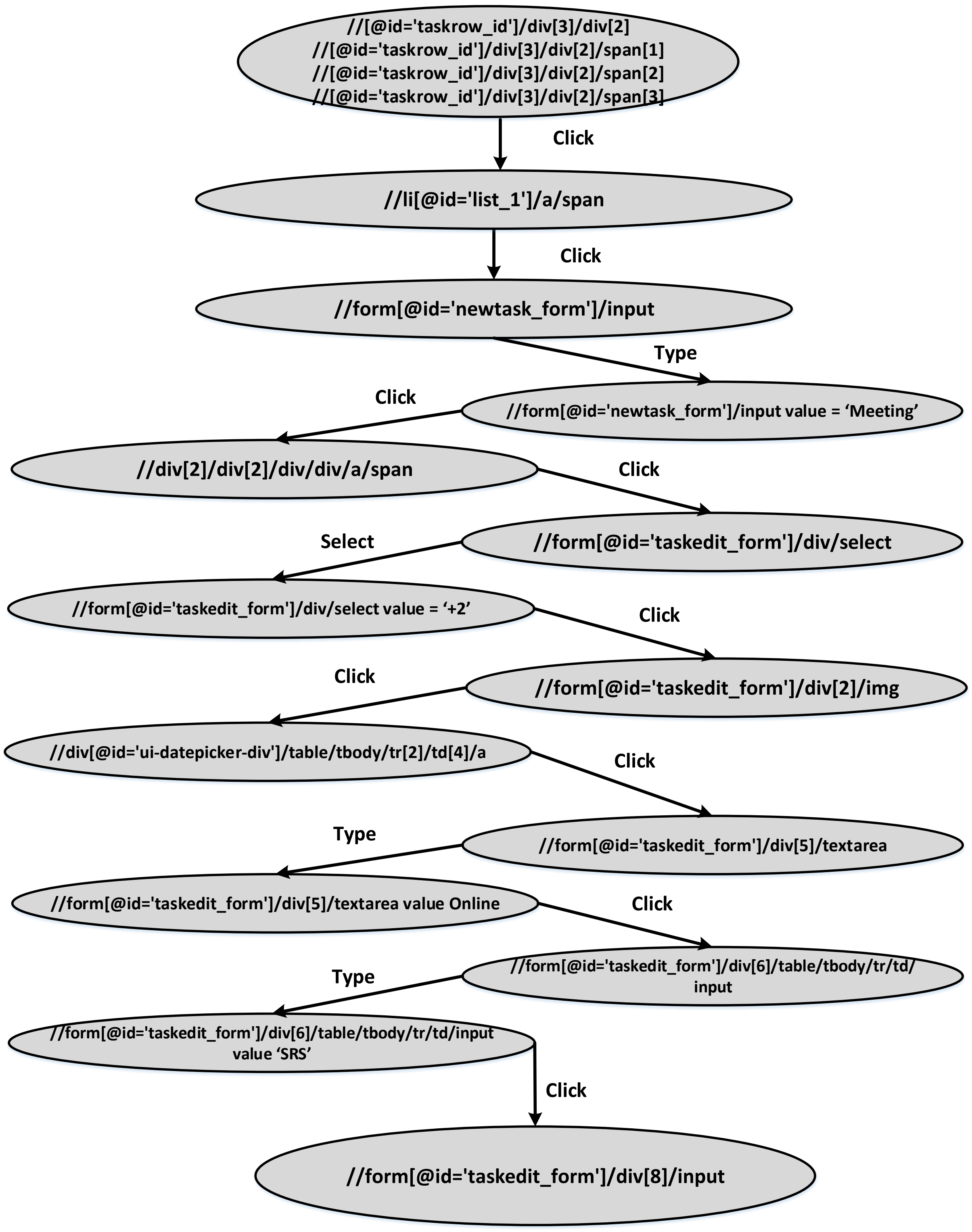

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//li[@id='list_1']/a/span
Click
//form[@id='newtask_form']/input
Type
//form[@id='newtask_form']/input value = 'Meeting'
Click
//div[2]/div[2]/div/div/a/span
Click
//form[@id='taskedit_form']/div/select
Select
//form[@id='taskedit_form']/div/select value = '+2'
Click
//form[@id='taskedit_form']/div[2]/img
Click
//div[@id='ui-datepicker-div']/table/tbody/tr[2]/td[4]/a
Click
//form[@id='taskedit_form']/div[5]/textarea
Type
//form[@id='taskedit_form']/div[5]/textarea value Online
Click
//form[@id='taskedit_form']/div[6]/table/tbody/tr/td/input
Type
//form[@id='taskedit_form']/div[6]/table/tbody/tr/td/input value 'SRS'
Click
//form[@id='taskedit_form']/div[8]/input

FIGURE 5.11: Add Task with Advance Button

Table 5.18 shows the "Search Task by Label" use case steps along with DOM states.

TABLE 5.18: Search Task by Label

S. No.	Use Case Steps	DOM state XPath
1	Select List	//li[@id='list_1']/a/span
2	Enter Task Name in Search Bar	//div[@id='toolbar']/div/div[2]/div/input
		//div[@id='toolbar']/div/div[2]/div/input
		value = 'task_name'
3	Click Search Buton	//div[@id='toolbar']/div/div[2]/div/div

Figure 5.12 shows the state machine for proposed used case.

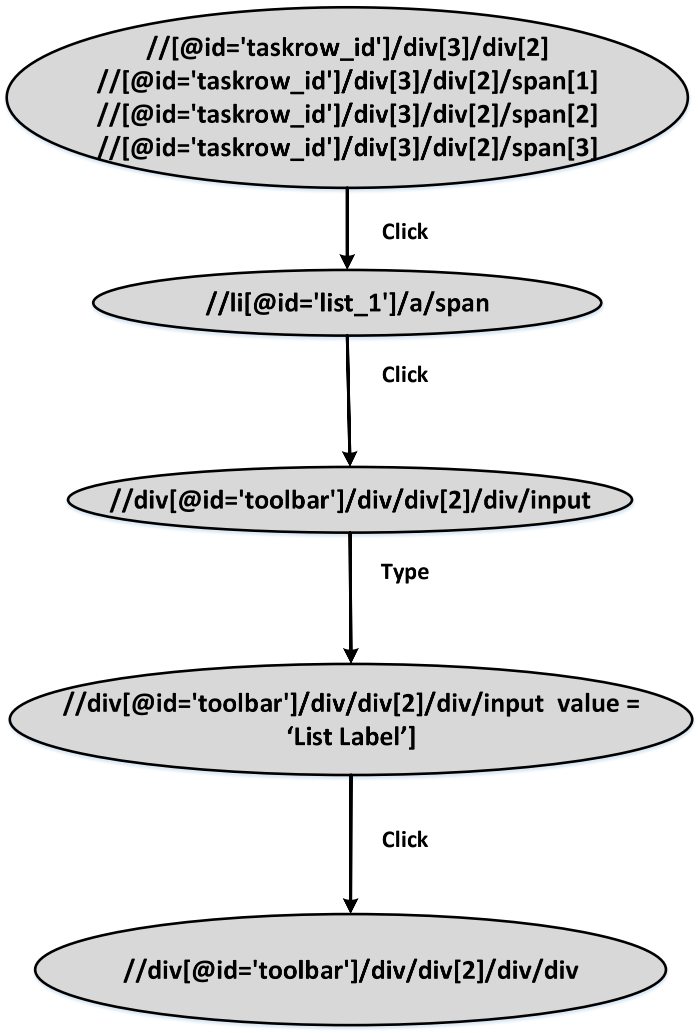

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//li[@id='list_1']/a/span
Click
//div[@id='toolbar']/div/div[2]/div/input
Type
//div[@id='toolbar']/div/div[2]/div/input value = 'List Label']
Click
//div[@id='toolbar']/div/div[2]/div/div

FIGURE 5.12: Search Task by Label

Table 5.19 shows the "Mark Task As Complete" use case steps along with DOM states.

TABLE 5.19: Mark Task as Complete

S. No.	Use Case Steps	DOM state XPath
1	Select List	**//li[@id='list_1']/a/span**
2	Check the Task to be Marked as complete	**//a[contains(@href, '#list/id')] //li[@id='taskrow_num']/div/div /label/input**

Figure 5.13 shows the state machine for proposed used case.

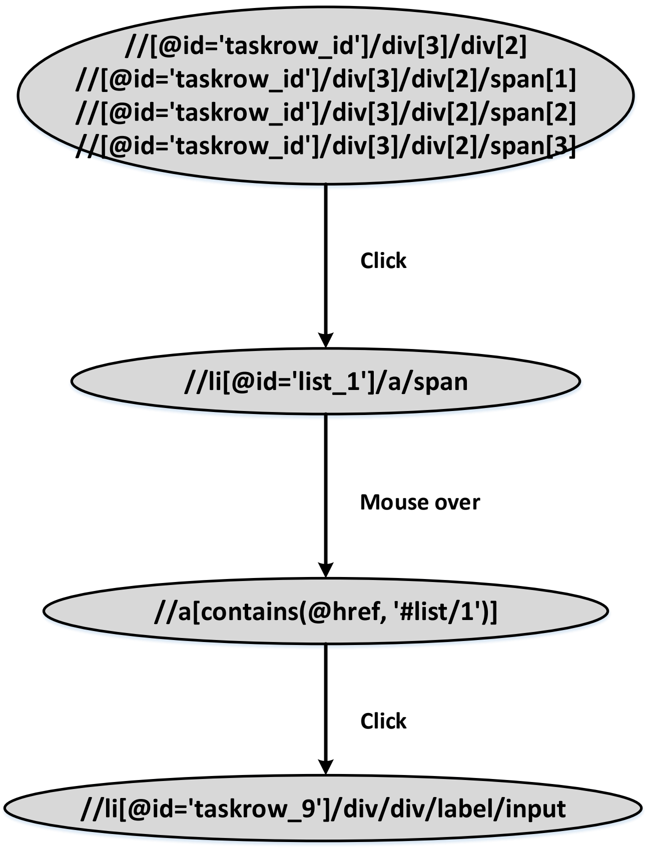

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//li[@id='list_1']/a/span
Mouse over
//a[contains(@href, '#list/1')]
Click
//li[@id='taskrow_9']/div/div/label/input

FIGURE 5.13: Mark Task as Complete

Table 5.20 shows the "Sort List by Priority" use case steps along with DOM states.

TABLE 5.20: Sort List by Priority

S. No.	Use Case Steps	DOM state XPath
1	Select List	**//li[@id='list_1']/a/span**
2	Select List Action Button	**//li[@id='list_id']/a/div**
3	Select 'Sort List by prioity option'	**//div[@id='listmenucontainer'] /ul/li[12]**

Figure 5.14 shows the state machine for proposed used case.

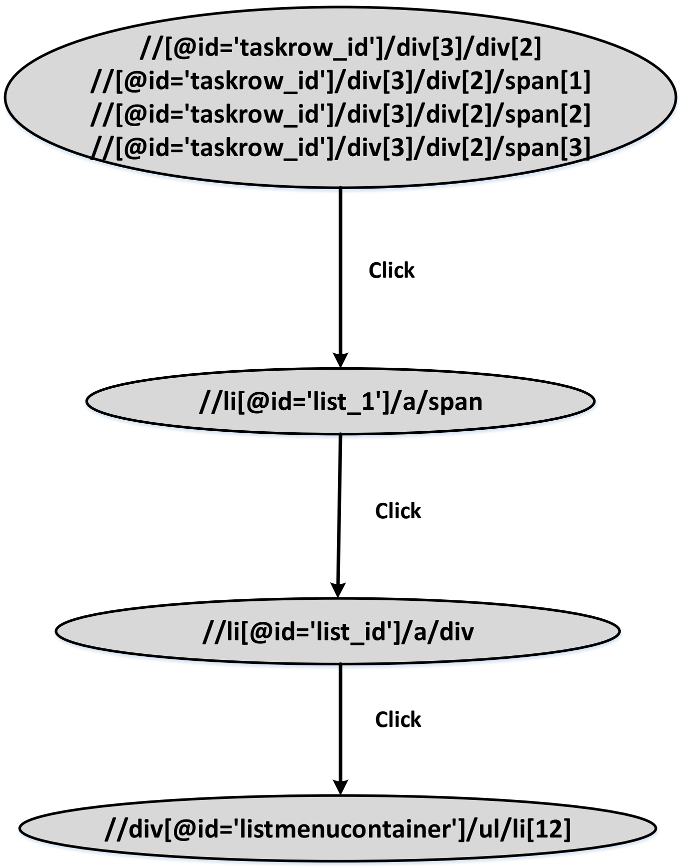

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//li[@id='list_1']/a/span
Click
//li[@id='list_id']/a/div
Click
//div[@id='listmenucontainer']/ul/li[12]

FIGURE 5.14: Sort List by Priority

Table 5.21 shows the "Sort List By Due Date" use case steps along with DOM states.

TABLE 5.21: Sort List by Due Date

S. No.	Use Case Steps	DOM state XPath
1	Select List	**//li[@id='list_1']/a/span**
2	Select List Action Button	**//li[@id='list_id']/a/div**
3	Select 'Sort List by Due Date'	**//div[@id='listmenucontainer'] /ul/li[13]**

Figure 5.15 shows the state machine for proposed used case.

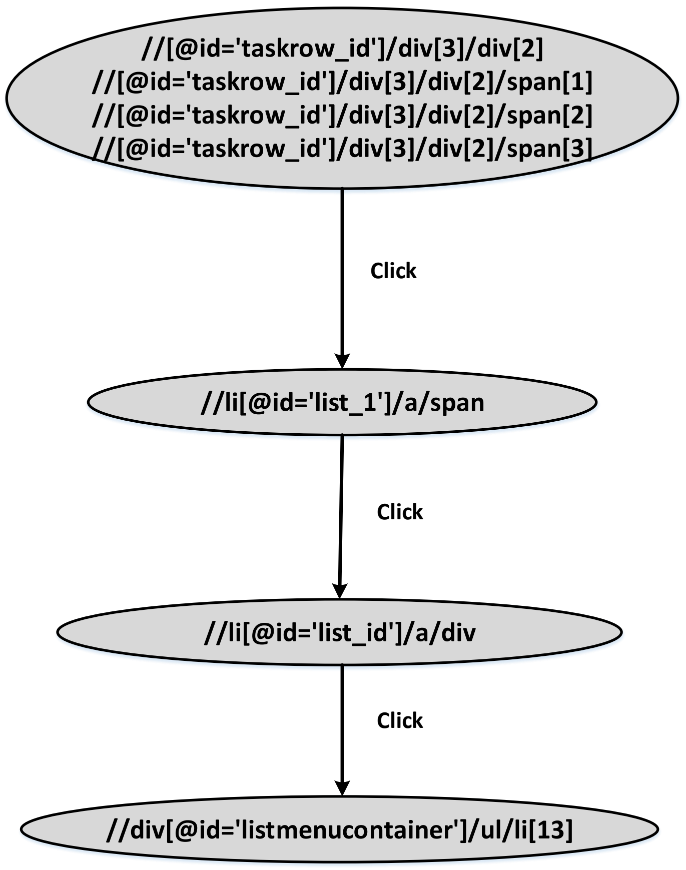

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//li[@id='list_1']/a/span
Click
//li[@id='list_id']/a/div
Click
//div[@id='listmenucontainer']/ul/li[13]

FIGURE 5.15: Sort List by Due Date

Table 5.22 shows the "Sort All Tasks By Priority" use case steps along with DOM states.

TABLE 5.22: Sort All Tasks by Priority

S. No.	Use Case Steps	DOM state XPath
1	Select Task or List Selection option	**//div[@id='tabs_buttons'] /div/div/span**
2	Select All Tasks option	**//li[@id='slmenu_list:-1']/a**
3	Select 'Select All Tasks Action Button'	**//div[@id='list_all']/a/div**
4	Select 'Select Sort by Priority option'	**//li[@id='sortByPrio']/div**

Figure 5.16 shows the state machine for proposed used case.

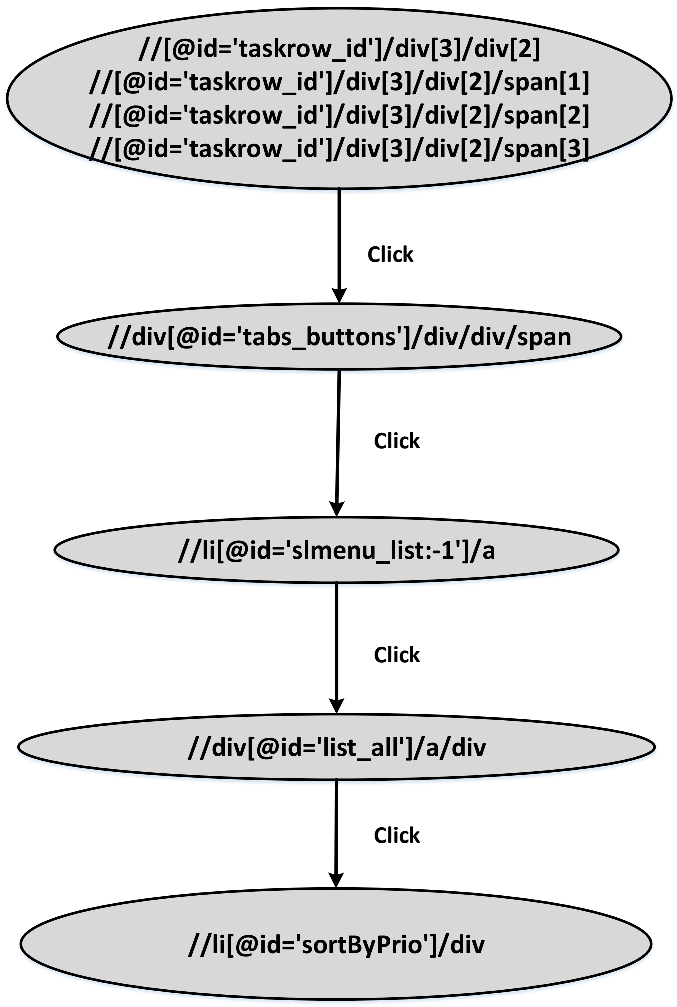

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//div[@id='tabs_buttons']/div/div/span
Click
//li[@id='slmenu_list:-1']/a
Click
//div[@id='list_all']/a/div
Click
//li[@id='sortByPrio']/div

FIGURE 5.16: Sort All Tasks by Priority

Table 5.23 shows the "Sort All Tasks By Due Date" use case steps along with DOM states.

TABLE 5.23: Sort All Tasks by Due Date

S. No.	**Use Case Steps**	**DOM state XPath**
1	Select Task or List Selection option	**//div[@id='tabs_buttons'] /div/div/span**
2	Select All Tasks option	**//li[@id='slmenu_list:-1']/a**
3	Select 'Select All Tasks Action Button'	**//div[@id='list_all']/a/div**
4	Select 'Select Sort by Due Date option'	**//div[@id='listmenucontainer'] /ul/li[13]**

Figure 5.17 shows the state machine for proposed used case.

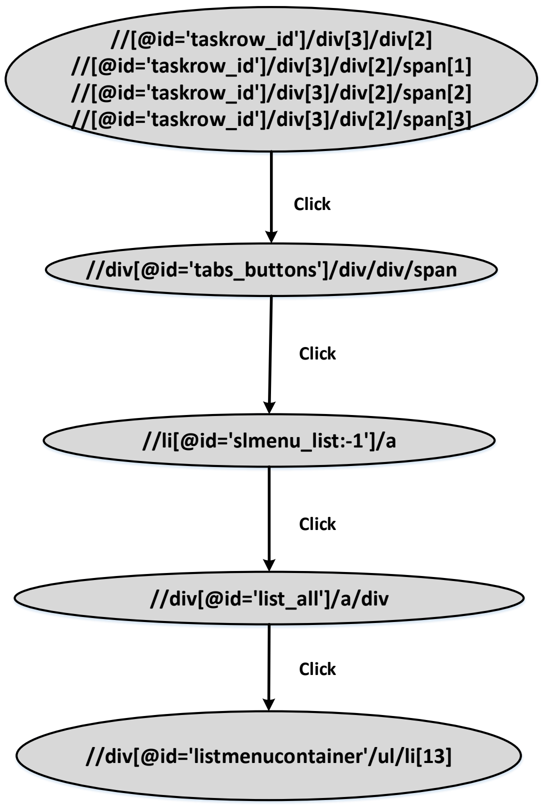

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//div[@id='tabs_buttons']/div/div/span
Click
//li[@id='slmenu_list:-1']/a
Click
//div[@id='list_all']/a/div
Click
//div[@id='listmenucontainer'/ul/li[13]

FIGURE 5.17: Sort All Tasks by Due Date

Table 5.24 shows the "Change Password Protection Settings" use case steps along with DOM states.

TABLE 5.24: Change Password Protection Settings

S. No.	Use Case Steps	DOM state XPath
1	Click Settings	**//div[@id='mtt_body']/div/div[2] /div[3]/span/a**
2	Enable Password Protection	**//form[@id='settings_form']/table /tbody/tr[3]/td/label/input**
3	Set Password	**//form[@id='settings_form']/table /tbody/tr[4]/td/input //form[@id='settings_form']/table /tbody/tr[4]/td/input value ="password"**
4	Submit Changes	**//form[@id='settings_form']/table /tbody/tr[15]/td/input**

Figure 5.18 shows the state machine for proposed used case.

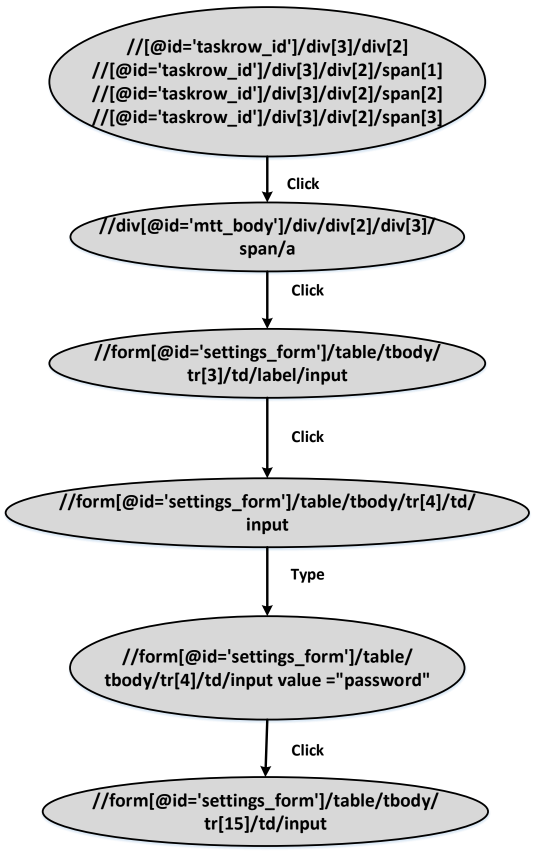

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//div[@id='mtt_body']/div/div[2]/div[3]/span/a
Click
//form[@id='settings_form']/table/tbody/tr[3]/td/label/input
Click
//form[@id='settings_form']/table/tbody/tr[4]/td/input
Type
//form[@id='settings_form']/table/tbody/tr[4]/td/input value ="password"
Click
//form[@id='settings_form']/table/tbody/tr[15]/td/input

FIGURE 5.18: Change Password Protection Settings

Table 5.25 shows the "Set Date Format" use case steps along with DOM states.

TABLE 5.25: Set Date Format

S. No.	Use Case Steps	DOM state XPath
1	Click Settings	//div[@id='mtt_body']/div/div[2] /div[3]/span/a
2	Select Date Format	//form[//form[@id='settings_form']/table /tbody/tr[10]/td/select //form[@id='settings_form']/table /tbody/tr[10]/td/select label = "selected format"
3	Select Short Date Format	//form[//form[@id='settings_form']/table /tbody/tr[11]/td/select //form[@id='settings_form']/table /tbody/tr[11]/td/select label = "selected format"
4	Select Short Date (current year) Format	//form[//form[@id='settings_form']/table /tbody/tr[12]/td/select //form[@id='settings_form']/table /tbody/tr[12]/td/select label = "selected format"
4	Submit Changes	//form[@id='settings_form']/table /tbody/tr[15]/td/input

Figure 5.19 shows the state machine for proposed used case.

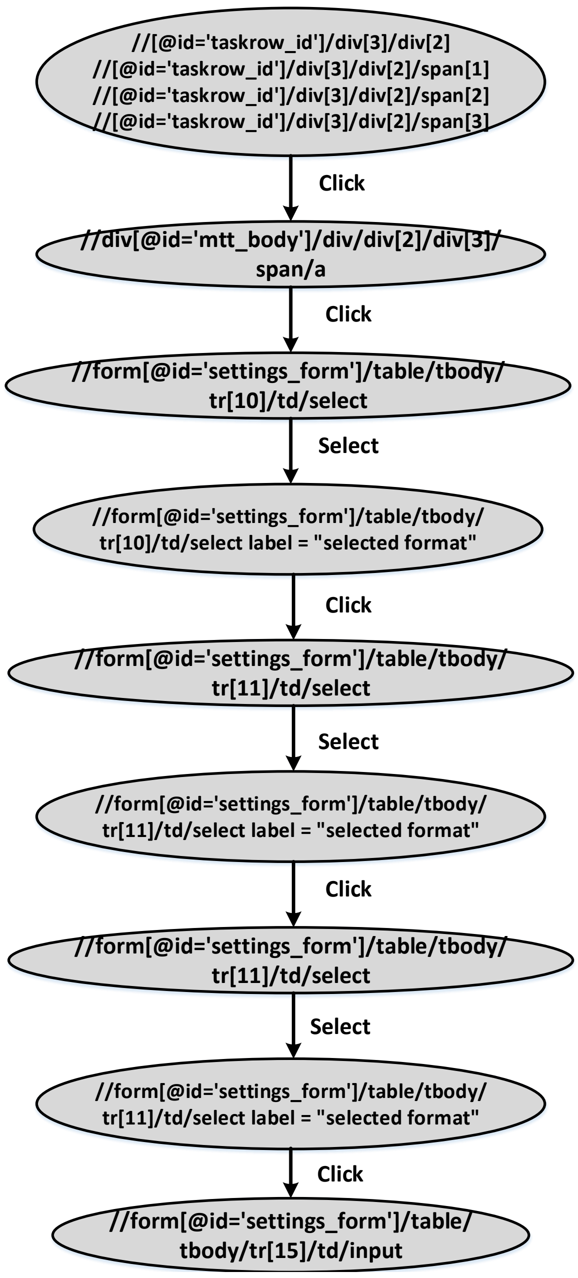

//[@id='taskrow_id']/div[3]/div[2]
//[@id='taskrow_id']/div[3]/div[2]/span[1]
//[@id='taskrow_id']/div[3]/div[2]/span[2]
//[@id='taskrow_id']/div[3]/div[2]/span[3]
Click
//div[@id='mtt_body']/div/div[2]/div[3]/span/a
Click
//form[@id='settings_form']/table/tbody/tr[10]/td/select
Select
//form[@id='settings_form']/table/tbody/tr[10]/td/select label = "selected format"
Click
//form[@id='settings_form']/table/tbody/tr[11]/td/select
Select
//form[@id='settings_form']/table/tbody/tr[11]/td/select label = "selected format"
Click
//form[@id='settings_form']/table/tbody/tr[11]/td/select
Select
//form[@id='settings_form']/table/tbody/tr[11]/td/select label = "selected format"
Click
//form[@id='settings_form']/table/tbody/tr[15]/td/input

FIGURE 5.19: Set Date Format

5.1 Evaluation of the Proposed Solution

In order to evaluate the proposed approach we need to use some metric. We have considered here, the impact of reduction of FSM on testing effectiveness. For this purpose, we calculated testing cost and testing effectiveness. Our experiments show that the reduction in cost is significant as compared to the loss of effectiveness. The parameters we used in order to gauge the effectiveness of testing, are faults detected and frequency of execution of different elements of the system under test.

In order to evaluate effectiveness of our approach, we used fault seeding. This technique is a combination of artificially induced faults and the measure whether testing is capable enough to uncover them. The comparison of detected and undetected seeded faults gives a confidence measure of testing [237]. One of the issues to cater in this approach is identification of potential fault seeding areas in the application. Randomly placed faults may prove to be an overhead as the seeded areas may be less or not executed. Two factors that are pivotal in identifying the fault seeding areas are usage patterns and the testers' perspective of the application usage. One perspective is from user and the other is from tester point of view and both could be quite different from one another. There should be a systematic way to incorporate both in fault seeding process.

Use cases are the main source to record user interaction with the application and in turn to identify usage patterns. These usage patterns help in identifying the application areas that are more used than the others. The faults in these more used areas have greater impact on the system behavior. This argument supports the idea that all the faults do not have same significance. The faults in areas encountered more by the users have different effect on reliability than the same type of faults in less encountered areas [238]. Almost all studies consider the faults alone as the measure of testing effectiveness without considering their frequency of execution [239–243]. This frequency of execution or the frequency profile provides the quantitative characterization of system usage [244]. Software testing based on this frequency of execution confirms that most frequently used operations are focused thus achieving maximum reliability in available testing time [244]. One important thing to consider here is the fault types to be seeded. This research uses classification of faults given in [245] and their distribution given in [246]. Grigorjev et al. [246] give a classification of faults and their distribution in the program, as shown in Table 5.26

Table 5.26: Fault Distribution

Class of Faults	**Percentage**
Logic Control Faults	32
Data Faults	24
Interface Faults	18
Computational Faults	13
Initialization Faults	13

There are a number of ways to seed faults in a system, i.e., seeding in random manner [247], in isolated manner [248], or by a human expert [249]. In this research, we have seeded one fault in every use case of our case studies from different classes of faults, and used fault severity definitions as given in [246]. Only those severity types are used that do not lead to system crashes, i.e., 3 to 5. Table 5.27 shows the severity levels as given by [246].

Table 5.27: Fault Severity Levels

Severity	**Description**
1	Catastrophic–Bug causes system crash
2	Major– Bug makes the product unusable
3	Moderate–Bug affecting product usage
4	Minor– Bug is not affecting product usage
5	Nuisance– Easily reparable faults

Several reliability metrics are available to measure the reliability of a system. One such metric is Probability of Failure on Demand (POFOD) [250, 251]. The argument to prove the effectiveness of proposed solution initiates by seeding a fault in each use case. The idea is that the seeded fault is encountered once the user executes the use case. Since all use cases do not have same execution frequency, the faults in use cases do not trigger the same number of times, which makes some faults more critical than others. In order to assess the reliability of the system, testing is performed using test cases. For simplicity, we assume that all use cases have equal number of test cases, i.e., one test case per use case. In order to calculate effectiveness of testing we first need to calculate effectiveness of a test case. Effectiveness of a test case is the ratio of its corresponding use case's frequency to the sum of frequencies of all use cases. Effectiveness of testing would then be calculated by taking the sum of Effectiveness of only the high category test cases. High category test cases are the test cases corresponding to the high category use cases identified by FCM. Cost of testing is the ratio of number of test cases executed to the total number of test cases. Executing all test cases incurs higher cost, our approach reduces the number of test cases by reducing the state machine, and thus testing cost is reduced. equation 5.1, 5.2 and 5.3 show the definitions of Effectiveness of a test case, Effectiveness of testing and Cost respectively.

$$Effectiveness_{tc} = \frac{Frequencyofcorrospondingusecase}{\sum_{i=1}^{n} Frequencyofuc_i} \tag{5.1}$$

$$Effectiveness_{High} = \sum_{i=1}^{h} Effectiveness_{tc_i} \tag{5.2}$$

$$Cost_{testing} = \frac{\sum_{j=1}^{h} tc_j}{\sum_{i=1}^{n} tc_i} \tag{5.3}$$

Here ‘n’ and ‘h’ show the total number of use cases and total number of high category use cases respectively. More test cases means more testing cost and vice versa. However the decrease in test cases may affect the effectiveness of testing. Here, we evaluate the impact of reduction in testing cost, on effectiveness of testing. As we mentioned earlier, we are writing one test case per use case, so reduced use case set would mean reduced test case set. As we are using POFOD, we argue that the use cases executed more often have the functionality that is requested frequently by the user and the faults in these use cases are encountered more by the user. The state machine generated by our solution comprise of the states corresponding to the most frequently executed use cases. The discarded group of use cases decreases the testing cost. We compare the relationship between this decreased cost and effectiveness of testing to prove the usefulness of our solution. Table 5.28 shows the frequency and effectiveness of a use case for an example scenario.

TABLE 5.28: Effectiveness of a Use Case

Use Case/Test Case	**Frequency**	**Effectiveness**
UC1/TC1	2	0.0625
UC2/TC2	3	0.09375
UC3/TC3	10	0.3125
UC4/TC4	5	0.15625
UC5/TC5	12	0.375

FCM results show that UC3 and UC5 fall in high category and their corresponding test cases would be considered for calculating the testing effectiveness. By using Eq. (8) the testing effectiveness of system is 68.7 and by using Eq. (9) the cost of testing is 40. This shows that the cost is reduced by 60 but the effectiveness is not reduced that much and is reduced only by 31.3.

We have used Coffee Maker and ToDo List case studies to further strengthen the argument regarding effectiveness of our solution. We have seeded a fault in each use case and these seeded faults ensure that whenever the user runs that use case, it triggers the seeded fault. Our solution implements the FCM algorithm to prioritize the use cases. This information depicts the most frequently used application areas and the faults in these areas tend to be encountered the most. The discarded use cases are not considered for testing which reduces the testing cost.

FCM categorizes use cases into two groups, i.e., high priority and low priority. FCM processes the given data and calculates a threshold value. The use cases falling above the threshold value belong to high priority group and rest belong to low priority group. Our solution discards the low priority use cases. equation 5.4 shows the relationship between high and low priority use cases.

$$UC_{All} = UC_{High} U UC_{Low} \tag{5.4}$$

The cost of testing is determined as the ratio of test cases executed to the total number of test cases. Thus cost of solution calculates the percentage of high category use cases in reference to all the use cases. equation 5.5 shows this relationship.

$$Cost(UC_{High}) = \frac{|UC_{High}|}{|UC_{All}|} \tag{5.5}$$

In our solution we claim that although we have reduced the testing cost but the effectiveness of testing is not compromised. This implies that effectiveness of testing is reduced less as compared to number of use cases reduced. Equation 5.6 shows the effectiveness calculation.

$$Effectiveness(UC_{testing}) = \frac{\sum_{j=1}^{h}(freq(uc_j))_{High}}{\sum_{i=1}^{n} freq(uc_i)_{All}} \tag{5.6}$$

Here n and h represent the total number and the number of high category use cases respectively. In order to find the reduction in overhead of use cases and in efficiency of testing, equation 5.7 and equation 5.8 are used respectively.

$$Reduction_{Cost}(UC_{All}) = 1 - Cost(UC_{High}) \tag{5.7}$$

And

$$Reduction_{Effectiveness}(UC_{All}) = 1 - Effectiveness(UC_{High}) \tag{5.8}$$

Table 5.29 shows the results of these calculations on the Coffee Maker and ToDo List case studies.

TABLE 5.29: Calculation on Case Studies

Case Study/ Calculations	**Coffee Maker**	**ToDo List**
No of Use Cases	6	42
Frequency of Use Cases	79	1102
High Category Use Cases	3	8
Frequency of High Category Use Cases	52	677
Overhead of High Category Use Cases	50%	19%
Efficiency of High Category Use Cases	66%	61%
Reduction in Cost	50%	81%
Reduction in Effectiveness	34%	39%

Coffee Maker case study has got a total of six use cases. Out of these six use cases three use cases (UC2, UC3, and UC4) fall in high category group and in turn are used by our solution to generate state machine. FCM has placed remaining 3 use cases in low category. Our solution has discarded these low category use cases. The percentage of these high category use cases is 0.5(50). The Coffee Maker case study use cases are executed 79 times by the users during frequency calculation. Each use case is seeded by one fault so the system comes across these faults 79 times during this use case set execution. Our solution considers only the use cases falling in high category group. The statistics show that these high category use cases are executed 52 times out of 79. This implies that although the use cases are reduced by 50 percent, the efficiency of these reduced use cases remains 0.658(66). Here the calculations show that the reduction in overhead is 50 while the reduction in efficiency is 0.342(34). These numbers support our claim that the reduction in use case number into half by our solution does not reduce the reliability into half. Todo List case study has got a total of 42 use cases which are categorized into two groups by FCM, i.e., high and low. In the ToDo List case study eight use cases (UC2, UC3, UC8, UC12, UC22, UC23, UC28 and UC29) fall in high category group and in turn are used by our solution to generate the state machine. The percentage of these high category use cases is 0.19(19). 34 use cases are discarded by our solution. The percentage of this low category use cases is 0.809(81). The ToDo List case study use cases are executed 1102 times by the users during frequency calculation. Each use case is seeded by one fault so the system comes across these faults 1102 times during this use case execution. Our solution considers only the use cases falling in high category group. The statistics show that these use cases are executed 677 times out 1102. This shows that although the use cases are reduced by 81 percent the efficiency of the use cases remain 0.614(61). Thus, the reduction in overhead is 0.809(81) while reduction in efficiency is 0.385(39). This number supports our claim that the reduction in use case number by 81 does not reduce the reliability by 81 percent.

5.2 Comparison with Existing Techniques

In this section we have performed comparative analysis of our technique with other techniques. We have discussed various approaches in related work section and here we have selected [6, 22] for comparison. The basis of this selection is that both these approaches are working to solve the state explosion problem in Ajax based applications.

In [6] the authors have discussed the use of Binary Decision Diagrams (BDD) to avoid state explosion problem. In their approach the state machine is generated for the whole application and then is reduced. The mechanism starts by recording user sessions in xml log files and then by reading those files to generate the state machine for the Ajax application. The authors have claimed that the state machine is generated and reduced at the same time. This mechanism imposes a constant overhead on the system by comparing and reducing the states all the time as the algorithm runs. Further the authors have not discussed the effects of this reduction on application testing, i.e., whether the reduced state machine has covered all the areas of the application under test.

In [22] the authors have claimed that state explosion problem is handled as every session has got its own state machine. However this mechanism cannot guarantee in absolute about the handling of state explosion problem due to following reasons. Firstly the session of a large application can have large interacting events and corresponding changing elements resulting in state explosion. Secondly the framework constructs state machine for every session which is an overhead on the application.

Our solution handles these issues in a more efficient way. Firstly the pivotal use cases are identified using FCM and state machine is generated only for those use cases. These limited use cases are the most frequently used actions of the user regarding the application thus depicting the most pivotal areas of the application.

Our framework records the triggered events, corresponding elements, and DOM changes only for these use cases thus avoiding state explosion. These use cases in fact cover all the important functions of the application thus effectiveness of the state machine regarding application coverage remain intact.

Table 5.30 shows the comparison amongst the approaches.

Table 5.30: Comparison of Approaches

Factors/ Approach	**Arora et. al**	**Arora et. al**	**StateReduceAjax Using FCM**
Scalability	Low (Difficult to implement as lot manual effort and human intervention required)	High (No additional functionality or extensive modification required)	High (No additional functionality or extensive modification required)
Cost	High (Machine is first built and then reduced)	High (Machine is built for every session)	Low (Machine is built only for most pivotal use cases)
Effectiveness	Low (Effects of reduction not addressed)	Low (Do not guarantee state explosion avoidance)	High (All the pivot areas are tested)
Adaptability	Low (Difficult to manage large size, complexity, and non-determinism of modern applications)	Low (Difficult to perform verification and accuracy checks on machine due to heterogeneity of new applications)	High (No additional functionality or extensive modification required as machine is built only for most pivotal use cases)
Maintainability	Low(problems with scalability, accuracy, and expressiveness as manual effort and human intervention required)	Low (Framework may become too large or complex to be easily modified, which can hinder the maintainability)	High (Smaller and more manageable state machines with reduced computational complexity, which are easier to understand, alter, and maintain)
Usability	Low (complexity and learning curve associated with using the BDD tool, the difficulty in understanding and inferring the resulting BDDs)	Medium (complex but delivers a view of the system that allows for more thorough analysis of the state space)	High (user-friendly because of its simplified nature, making it easier to learn and use for developers)

Quantitative comparison of the approaches was not viable due to the lack of specific quantitative metrics and data, such as the number of users, data points, and processing time. Following are some details.

1. Lack of relevant data: In some cases, the data required to make a quantitative comparison may be difficult or impossible to obtain. For example, if one of the approaches is relatively new and has not been widely used, it may be difficult to find sufficient data to make a meaningful comparisons.
2. Differences in context: The approaches being compared may be designed for different contexts, such as different types of applications or different end users. This can make it difficult to compare them in a meaningful way, as their performance may be influenced by factors that are unique to their respective contexts.
3. Non-linear relationships: Some aspects of software development, such as cost or effectiveness, may not have a linear relationship with the metrics being used for comparison. This can make it difficult to compare approaches based solely on quantitative metrics, as the results may not correctly reflect their actual performance.
4. Subjectivity of metrics: Even when metrics are well-defined, their interpretation can be subjective. For example, different people may have different opinions on what makes good maintainability or usability. This can lead to discrepancies in the results of a quantitative comparison.
5. Qualitative factors: Some aspects of software development, such as usability and adaptability, may be hard to measure quantitatively. In these cases, qualitative methods such as user surveys or expert evaluations may be more appropriate.

In conclusion, while quantitative comparisons can be useful in some cases, they may not always be feasible or appropriate. Instead, a qualitative comparison was conducted based on several criteria, including scalability, cost, effectiveness, adaptability, maintainability, and usability. The comparison has revealed that the first approach has low scalability, high cost, low effectiveness, low adaptability, low maintainability, and low usability. The second approach has high scalability, high cost, low effectiveness, low adaptability, low maintainability, and medium usability. The third approach has high scalability, low cost, high effectiveness, high adaptability, high maintainability, and high usability. Therefore, a qualitative analysis was performed to identify the strengths and weaknesses of each approach based on available information, expert opinions, and real-world examples. This qualitative comparison delivers valuable insights into the trade-offs involved in each approach and forms a basis for making informed recommendations.

Chapter 6

Conclusion and Future Work

Web applications of today are getting more difficult to use because there are now more hyperlinks, interactions are getting more complicated, and more distributed servers are being used. Ajax has made things even more complicated because it works in an asynchronous way and is very dynamic. Modeling, which gives convenient and intelligible graphical descriptions of systems without diving into the implementation details, can aid in the understanding of these complicated systems. At each level of development, system modeling facilitates verification and validation. Due to the extreme dynamism of Ajax applications, system modeling of Ajax apps is hampered by the issue of state explosion. The focus of this study was on addressing the state explosion issue in modelling Ajax applications. The research began by analysing the gaps in existing state space reduction strategies for Ajax applications, and then proceeded to propose a methodology for constructing a state machine with reduced states. This was accomplished by constructing a framework that generates a reduced state machine, which avoids the problem of state explosion. Finally, the solution's effectiveness was assessed.

6.1 Conclusion

The results of the implemented solution show that the proposed approach is feasible to generate a state machine of an Ajax application. The proposed solution offers a soft computing based process that detects all dynamically generated states and the relevant events and DOM changing elements. The state explosion problem addressed here is also handled comprehensively without compromising the dynamic behavior of these applications using FCM. The approach not only generates a reduced state machine but also does it without compromising the efficiency of testing. Experimental results prove that StateReduceAjax controls the size of generated state machine without compromising the quality of testing. In case of Coffee Maker, statistical results show that 50% of reduction in number of use cases reduces the efficiency by 34%. In the other case study of ToDo List the use cases are reduced by 81% while the efficiency is reduced merely by 39%.

6.2 Future Work

For each of the selected use cases in this study, a reduced finite state machine was created. FCM was employed in the use case selection process. In the future, we can aggregate all of these created state machines into a single FSM for the entire system.

Using the aggregated FSM, test cases for the system's most frequently used components can be generated. This process can apply standard coverage criteria to the FSMs and a technique for aggregating tests defined for single use case-based FSMs into tests for the aggregate FSM.

In the future, we plan to propose an algorithm for traversing the FSM to generate test sequences that satisfy a specific coverage criterion. We also aim to automatically generate input test data, which can be used to convert abstract test cases into concrete test cases. We can also address the "oracle" problem, which involves determining whether the results are correct.

Bibliography

[1] S. Raj, R. Krishna, and A. Nayak, "Distributed component-based crawler for ajax applications," in *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, 2018, pp. 1–6.

[2] S. Khalid, S. Khusro, and I. Ullah, "Crawling ajax-based web applications: Evolution and state-of-the-art," *Malaysian Journal of Computer Science*, vol. 31, pp. 35–47, 01 2018.

[3] A. van Deursen, A. Mesbah, and A. Nederlof, "Crawl-based analysis of web applications: Prospects and challenges," *Science of Computer Programming*, vol. 97, pp. 173–180, 2015, special Issue on New Ideas and Emerging Results in Understanding Software.

[4] M. V. Bharathi and S. Rodda, "Survey on testing technique for modern web application-rookies vantage point," *International Journal of Networking and Virtual Organisations*, vol. 21, no. 2, pp. 277–288, 2019.

[5] A. Mesbah, E. Bozdag, and A. van Deursen, "Crawling ajax by inferring user interface state changes," in *2008 Eighth International Conference on Web Engineering*, 2008, pp. 122–134.

[6] A. Arora and M. Sinha, "Avoiding state explosion problem of generated ajax web application state machine using bdd," in *2013 Sixth International Conference on Contemporary Computing (IC3)*, 2013, pp. 381–386.

[7] H. Z. Jahromi, D. T. Delaney, and A. Hines, "Beyond first impressions: Estimating quality of experience for interactive web applications," *IEEE Access*, vol. 8, pp. 47 741–47 755, 2020.

[8] A. Mesbah and A. van Deursen, "A component- and push-based architectural style for ajax applications," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2194–2209, 2008, best papers from the 2007 Australian Software Engineering Conference (ASWEC 2007), Melbourne, Australia, April 10-13, 2007.

[9] A. Marchetto, F. Ricca, and P. Tonella, "A case study-based comparison of web testing techniques applied to ajax web applications," *STTT*, vol. 10, pp. 477–492, 12 2008.

[10] S. Pradhan, M. Ray, and S. Patnaik, "Clustering of web application and testing of asynchronous communication," *International Journal of Ambient Computing and Intelligence*, vol. 10, pp. 33–59, 07 2019.

[11] A. Arora, "Test case generation using progressively refined genetic algorithm for ajax web application testing," *Recent Patents on Computer Science*, vol. 11, 10 2018.

[12] A. van Deursen and A. Mesbah, "Research issues in the automated testing of ajax applications," in *SOFSEM 2010: Theory and Practice of Computer Science*, J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, and B. Rumpe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 16–28.

[13] A. Mesbah and A. Deursen, "Invariant-based automatic testing of ajax user interfaces," 01 2009, pp. 210–220.

[14] E. M. Clarke and O. Grumberg, "Avoiding the state explosion problem in temporal logic model checking," in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 294–303. [Online]. Available: https://doi.org/10.1145/41840.41865

[15] S. Kimura and E. Clarke, "A parallel algorithm for constructing binary decision diagrams," in *Proceedings., 1990 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1990, pp. 220–223.

[16] K. Böhmer and S. Rinderle-Ma, "A systematic literature review on process model testing: Approaches, challenges, and research directions," 09 2015.

[17] A. Andrews, J. Offutt, and R. Alexander, "Testing web applications by modeling with fsms," *Software and System Modeling*, vol. 4, pp. 326–345, 07 2005.

[18] G. A. Di Lucca and A. R. Fasolino, "Testing web-based applications: The state of the art and future trends," *Information and Software Technology*, vol. 48, no. 12, pp. 1172–1186, 2006, quality Assurance and Testing of Web-Based Applications.

[19] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher II, "Leveraging user-session data to support web application testing," *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 187–202, 2005.

[20] F. Ricca and P. Tonella, "Analysis and testing of web applications," 06 2001, pp. 25– 34.

[21] S. Sampath and S. Sprenkle, "Chapter four - advances in web application testing, 2010–2014," ser. Advances in Computers, A. Memon, Ed. Elsevier, 2016, vol. 101, pp. 155–191.

[22] A. Arora and M. Sinha, "A sustainable approach to automate user session based state machine generation for ajax web applications," *Journal of Theoretical and Applied Information Technology*, vol. 54, pp. 401–419, 08 2013.

[23] D. Ibrahim, "An overview of soft computing," *Procedia Computer Science*, vol. 102, pp. 34–38, 2016, 12th International Conference on Application of Fuzzy Systems and Soft Computing, ICAFS 2016, 29-30 August 2016, Vienna, Austria.

[24] A. Ahmad and S. S. Khan, "Survey of state of the art mixed data clustering algorithms," *IEEE Access*, vol. 7, pp. 31 883–31 902, 2019.

[25] A. Mesbah and A. Deursen, "An architectural style for ajax," *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, 08 2006.

[26] A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling ajax-based web applications through dynamic analysis of user interface state changes," *ACM Trans. Web*, vol. 6, no. 1, Mar. 2012. [Online]. Available: https://doi.org/10.1145/2109205.2109208

[27] D. Roest, A. Mesbah, and A. van Deursen, "Regression testing ajax applications: Coping with dynamism," *2010 Third International Conference on Software Testing, Verification and Validation*, pp. 127–136, 2010.

[28] D. Roest, "Automated regression testing of ajax web applications," 2010.

[29] A. Mesbah, A. Deursen, and D. Roest, "Invariant-based automatic testing of modern web applications," *Software Engineering, IEEE Transactions on*, vol. 38, pp. 1 – 1, 01 2012.

[30] E. Annon, "Strategic management: A stakeholder approach," *Boston Pitman. ISBN 0-273*, vol. 1456, 2018.

[31] L. A. Macaulay, "Requirements capture as a cooperative activity," *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pp. 174–181, 1993.

[32] T. Gilb, *Competitive engineering: a handbook for systems engineering, requirements engineering, and software engineering using Planguage.* Elsevier, 2005.

[33] A. Pouloudi, "Stakeholder analysis as a front-end to knowledge elicitation," *AI & SOCIETY*, vol. 11, pp. 122–137, 2008.

[34] S. A. Conger, *The new software engineering.* Course Technology Press, 1993.

[35] G. Kotonya and I. Sommerville, "Requirements engineering (processes and techniques) john wiley & sons ltd: England," 1998.

[36] M. Cotterell and B. Hughes, *Software project management.* International Thomson Computer Press, 1995.

[37] A. Dix, J. E. Finlay, G. D. Abowd, and R. Beale, *Human-Computer Interaction (3rd Edition).* USA: Prentice-Hall, Inc., 2003.

[38] J. McManus, "A stakeholder perspective within software engineering projects," 10 2004, pp. 880 – 884 Vol.2.

[39] S. Young, S. Mcdonald, H. Edwards, and J. Thompson, "Quality & people in the development of situationally specific methods." 01 2001, pp. 199–203.

[40] C. Pacheco and E. Tovar, "Stakeholder identification as an issue in the improvement of software requirements quality," in *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, ser. CAiSE'07. Berlin, Heidelberg: Springer-Verlag, 2007, p. 370–380.

[41] I. F. Alexander and S. Robertson, "Understanding project sociology by modeling stakeholders," *IEEE Software*, vol. 21, pp. 23–27, 2004.

[42] K. Power, "Stakeholder identification in agile software product development organizations: A model for understanding who and what really counts," *2010 Agile Conference*, pp. 87–94, 2010.

[43] O. Preiss and A. Wegmann, "Stakeholder discovery and classification based on systems science principles," *Proceedings Second Asia-Pacific Conference on Quality Software*, pp. 194–198, 2001.

[44] H. Sharp, A. Finkelstein, and G. Galal, "Stakeholder identification in the requirements engineering process," in *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*, 1999, pp. 387–391.

[45] L. Ballejos and J. Montagna, "Method for stakeholder identification in interorganizational environments," *Requir. Eng.*, vol. 13, pp. 281–297, 11 2008.

[46] J. Wan, "Research on knowledge creation in software requirement development," *Journal of Software Engineering and Applications*, vol. 03, pp. 487–494, 01 2010.

[47] G. Kotonya and I. Sommerville, “Requirements engineering with viewpoints,” *Softw. Eng. J.*, vol. 11, pp. 5–18, 1996.

[48] C. Urquhart, “Analysts and clients in organisational contexts: A conversational perspective,” *Journal of Strategic Information Systems - J STRATEGIC INFORM SYST*, vol. 10, pp. 243–262, 09 2001.

[49] R. Fuentes-Fernández, J. Gómez-Sanz, and J. Pavón, “Understanding the human context in requirements elicitation,” *Requir. Eng.*, vol. 15, pp. 267–283, 09 2010.

[50] J. Price and J. Cybulski, “Consensus making in requirements negotiation: the communication perspective,” *Australasian Journal of Information Systems; Vol 13, No 1 (2005)*, vol. 13, 01 2007.

[51] S. Ahmad, “Negotiation in the requirements elicitation and analysis process,” 04 2008, pp. 683–689.

[52] U. Erra and G. Scanniello, “Assessing communication media richness in requirements negotiation,” *Software, IET*, vol. 4, pp. 134 – 148, 05 2010.

[53] S. Hornik, H.-G. Chen, G. Klein, and J. Jiang, “Communication skills of is providers: An expectation gap analysis from three stakeholder perspectives,” *Professional Communication, IEEE Transactions on*, vol. 4, pp. 17 – 34, 04 2003.

[54] J. Karlsson, “Software requirements prioritizing,” *Proceedings of the Second International Conference on Requirements Engineering*, pp. 110–116, 1996.

[55] G. Ruhe, A. Eberlein, and D. Pfahl, “Quantitative winwin: a new method for decision support in requirements negotiation,” in *SEKE ’02*, 2002.

[56] A. Aurum and C. Wohlin, “Wohlin, c.: The fundamental nature of requirements engineering activities as a decision making process. information and software technology 45, 945-954,” *Information and Software Technology*, vol. 45, pp. 945–954, 11 2003.

[57] P. Carlshamre, "Release planning in market-driven software product development: Provoking an understanding," *Requir. Eng.*, vol. 7, pp. 139–151, 09 2002.

[58] L. Lehtola and M. Kauppinen, "Empirical evaluation of two requirements prioritization methods in product development projects," in *European Conference on Software Process Improvement.* Springer, 2004, pp. 161–170.

[59] F. Moisiadis, "The fundamentals of prioritising requirements," in *Proceedings of Systems Engineering Test and Evaluation Conference*, J. Andersen, Ed. Systems Engineering society of Australia, 2002, pp. 59–70, systems Engineering Test and Evaluation Conference 2002 ; Conference date: 29-10-2002 Through 30-10-2002.

[60] L. Karlsson, P. Berander, B. Regnell, and C. Wohlin, "Requirements prioritisation: an experiment on exhaustive pair-wise comparisons versus planning game partitioning," in *8th International Conference on Empirical Assessment in Software Engineering (EASE 2004) Workshop - 26th International Conference on Software Engineering.* IEE, 2004, pp. 145–154, 8th Internation Conference on Empirical Assessment in Software Engineering (EASE 2004) Workshop - 26th International Conference on Software Engineering ; Conference date: 24-05-2004 Through 25-05-2004.

[61] B. Bergman and B. Klefsjö, *Quality from customer needs to customer satisfaction.* New York: Mc-Graw Hill Book Company, 1994.

[62] J. I. McManus and G. G. Schulmeyer, *The Handbook of Software Quality Assurance.* New Jersey: Prentice Hall, 1999.

[63] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Software*, vol. 14, no. 5, pp. 67–74, 1997.

[64] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide.* New Jersey: Wiley, 1997.

[65] K. Wiegers, "First things first: prioritizing requirements," *Software Development*, vol. 7, no. 9, pp. 48–53, 1999.

[66] L. Lehtola, M. Kauppinen, and S. Kujala, "Requirements prioritization challenges in practice," 04 2004, pp. 497–508.

[67] S. Dahlstedt and A. Persson, "Requirements interdependencies - moulding the state of research into a research agenda," 06 2003.

[68] G. Ruhe, A. Eberlein, and D. Pfahl, "Trade-off analysis for requirements selection," *International Journal of Software Engineering and Knowledge Engineering*, vol. 13, pp. 345–366, 08 2003.

[69] S. Lauesen, "Software requirements-styles and techniques," 01 2002.

[70] D. Firesmith, "Prioritizing requirements," *Journal of Object Technology*, vol. 3, pp. 35–48, 09 2004.

[71] J. Karlsson, C. Wohlin, and B. Regnell, "An evaluation of methods for prioritizing software requirements," *Information and Software Technology*, vol. 39, pp. 939–947, 02 2001.

[72] P. Laurent, J. Cleland-Huang, and C. Duan, "Towards automated requirements triage," 11 2007, pp. 131–140.

[73] T. Saaty, *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*, ser. Advanced book program. McGraw-Hill International Book Company, 1980.

[74] K. Khan, "A systematic review of software requirements prioritization," 2006.

[75] V. Ahl, "An experimental comparison of five prioritization methods: investigating ease of use, accuracy and scalability," 2005.

[76] P. Berander, "Prioritization of stakeholder needs in software engineering: Understanding and evaluation," Ph.D. dissertation, Blekinge Institute of Technology, 2004.

[77] B. W. Boehm and R. Ross, "Theory-w software project management principles and examples," *IEEE Transactions on Software Engineering*, vol. 15, no. 7, pp. 902–916, 1989.

[78] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Unified Approach*. Boston: Addison-Wesley, 1999.

[79] R. Beg, Q. Abbas, and R. Verma, "An approach for requirement prioritization using b-tree," *Emerging Trends in Engineering & Technology, International Conference on*, vol. 0, pp. 1216–1221, 07 2008.

[80] P. Zave, "Classification of research efforts in requirements engineering." *ACM Comput. Surv.*, vol. 29, pp. 315–321, 12 1997.

[81] P. Berander and P. Jönsson, "Hierarchical cumulative voting (hcv) - prioritization of requirements in hierarchies," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 16, pp. 819–850, 2006.

[82] P. Tonella, A. Susi, and F. Palma, "Using interactive ga for requirements prioritization," 10 2010, pp. 57 – 66.

[83] ——, "Interactive requirements prioritization using a genetic algorithm," *Information and Software Technology*, vol. 55, pp. 173 – 187, 01 2013.

[84] P. Avesani, S. Ferrari, and A. Susi, "Case-based ranking for decision support systems," vol. 2689, 06 2003, pp. 35–49.

[85] A. Zabin, V. A. González, Y. Zou, and R. Amor, "Applications of machine learning to bim: A systematic literature review," *Advanced Engineering Informatics*, vol. 51, p. 101474, 2022.

[86] S. Hameed, Y. Elsheikh, and M. Azzeh, "An optimized case-based software project effort estimation using genetic algorithm," *Information and Software Technology*, vol. 153, p. 107088, 2023.

[87] A. Yan and Z. Cheng, "A review of the development and future challenges of case-based reasoning," 2023.

[88] U. Dash and A. A. Acharya, "A systematic review of test case prioritization approaches," in *Proceedings of International Conference on Advanced Computing Applications: ICACA 2021*. Springer, 2022, pp. 653–666.

[89] S. Ali, Y. Hafeez, M. Humayun, N. Jhanjhi, and D.-N. Le, "Towards aspect based requirements mining for trace retrieval of component-based software management process in globally distributed environment," *Information Technology and Management*, vol. 23, no. 3, pp. 151–165, 2022.

[90] S. Elbaum, S. Karre, and G. Rothermel, "Improving web application testing with user session data," 06 2003, pp. 49– 59.

[91] S. G. Elbaum, G. Rothermel, S. Karre, and M. Fisher, "Leveraging user-session data to support web application testing," *IEEE Transactions on Software Engineering*, vol. 31, pp. 187–202, 2005.

[92] J. Sant, A. L. Souter, and L. G. Greenwald, "An exploration of statistical models for automated test case generation," 2005.

[93] S. Sprenkle, E. Hill, S. Sampath, and L. Pollock, "Automated replay and failure detection for web applications," 01 2005, pp. 253–262.

[94] G. A. Di Lucca and A. R. Fasolino, "Testing web-based applications: The state of the art and future trends," *Information and Software Technology*, vol. 48, no. 12, pp. 1172–1186, 2006, quality Assurance and Testing of Web-Based Applications.

[95] S. Sprenkle, E. Hill, S. Sampath, and L. Pollock, "A case study of automatically creating test suites from web application field data," 01 2006, pp. 1–9.

[96] I. Alsmadi and M. Kenneth, "Using user sessions for test case generation and execution," *Lecture Notes in Engineering and Computer Science*, vol. 2168, 03 2008.

[97] T. Deenadayalan, V. Kavitha, and S. Rajarajeswari, "Examining web application by clumping and orienting user session data," *ArXiv*, vol. abs/1006.4537, 2010.

[98] M. L. Brian S. Everitt, Sabine Landau and D. Stahl, *Cluster Analysis*. New Jersey: Wiley, 2011.

[99] E. Backer and A. K. Jain, "A clustering performance measure based on fuzzy set decomposition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 1, pp. 66–75, 1981.

[100] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, p. 264–323, sep 1999. [Online]. Available: https://doi.org/10.1145/331499.331504

[101] C. Aggarwal, A. Hinneburg, and D. Keim, "On the surprising behavior of distance metrics in high dimensional space," vol. 1973, 01 2001, pp. 420–434.

[102] G. Serban and G. Cojocar, "A comparison of clustering techniques in aspect mining," *Studia Universitatis Babeş-Bolyai. Informatica*, vol. 51, 01 2006.

[103] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping Multidimensional Data*, 2006.

[104] R. Xu and D. Wunsch, "Survey of clustering algorithms," *Neural Networks, IEEE Transactions on*, vol. 16, pp. 645 – 678, 06 2005.

[105] v. kumar and H. Mittal, "Comparative study of soft computing techniques for software quality model," *International Journal of Software Engineering Research and Practices*, vol. 1, pp. 33–37, 01 2011.

[106] D. Gupta, V. Goyal, and H. Mittal, "Analysis of clustering techniques for software quality prediction," *Proceedings - 2012 2nd International Conference on Advanced Computing and Communication Technologies, ACCT 2012*, 01 2012.

[107] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, and S. Brown, "Fgka: a fast genetic k-means clustering algorithm," 01 2004, pp. 622–623.

[108] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data.* USA: Prentice-Hall, Inc., 1988.

[109] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction To Cluster Analysis*, 01 1990.

[110] J.-S. Cherng and M.-J. Lo, "A hypergraph based clustering algorithm for spatial data sets," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, ser. ICDM '01. USA: IEEE Computer Society, 2001, p. 83–90.

[111] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means clustering algorithm," *Applied Statistics*, vol. 28, no. 1, pp. 100–108, 1979. [Online]. Available: http://dx.doi.org/10.2307/2346830

[112] J. A. Hartigan, *Clustering Algorithms*, 99th ed. USA: John Wiley &; Sons, Inc., 1975.

[113] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. L. Cam and J. Neyman, Eds., vol. 1. University of California Press, 1967, pp. 281–297.

[114] G. Gan, C. Ma, and J. Wu, *Data Clustering: Theory, Algorithms, and Applications.* Society for Industrial and Applied Mathematics, 2007.

[115] J. C. Bezdek, *Pattern recognition with fuzzy objective function algorithms.* Springer Science & Business Media, 2013.

[116] M.-S. Yang, Y.-J. Hu, K. C.-R. Lin, and C. C.-L. Lin, "Segmentation techniques for tissue differentiation in mri of ophthalmology using fuzzy clustering algorithms," *Magnetic Resonance Imaging*, vol. 20, no. 2, pp. 173–179, 2002.

[117] L. A. Zadeh, "Fuzzy sets," in *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh.* World Scientific, 1996, pp. 394–432.

[118] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," in *Proceedings 3rd IEEE symposium on application-specific systems and software engineering technology.* IEEE, 2000, pp. 85–90.

[119] A. Nederlof, A. Mesbah, and A. v. Deursen, “Software engineering for the web: the state of the practice,” in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 4–13.

[120] A. Marchetto, P. Tonella, and F. Ricca, “State-based testing of ajax web applications,” in *2008 1st International Conference on Software Testing, Verification, and Validation*, 2008, pp. 121–130.

[121] B. Donley and J. J. Offutt, “Web application testing challenges blaine donley and jeff offutt,” 2010.

[122] A. Arora, “Web application testing: A review on techniques, tools and state of art,” 2012.

[123] A. Marchetto and P. Tonella, “Search-based testing of ajax web applications,” in *2009 1st International Symposium on Search Based Software Engineering*, 2009, pp. 3–12.

[124] S. Sampath, V. Mihaylov, A. Souter, and L. Pollock, “A scalable approach to user-session based testing of web applications through concept analysis,” in *Proceedings. 19th International Conference on Automated Software Engineering, 2004.*, 2004, pp. 132–141.

[125] A. Marchetto and P. Tonella, “Using search-based algorithms for ajax event sequence generation during testing,” *Empirical Software Engineering*, vol. 16, pp. 103–140, 02 2011.

[126] M. Benedikt, J. Freire, and P. Godefroid, “Veriweb: Automatically testing dynamic web sites,” in *In Proceedings of 11th International World Wide Web Conference (WW W’2002.* Citeseer, 2002.

[127] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Soviet physics. Doklady*, vol. 10, pp. 707–710, 1965.

[128] Y.-F. Li, P. K. Das, and D. L. Dowe, “Two decades of web application testing—a survey of recent advances,” *Information Systems*, vol. 43, pp. 20–54, 2014.

[129] C. Duda, G. Frey, D. Kossmann, R. Matter, and C. Zhou, "Ajax crawl: Making ajax applications searchable," in *2009 IEEE 25th International Conference on Data Engineering.* IEEE, 2009, pp. 78–89.

[130] G. Frey, "Indexing ajax web applications," Master's thesis, ETH Department of Computer Science, Institute of Computational Sciences, 2007.

[131] G. Pellegrino, C. Tschürtz, E. Bodden, and C. Rossow, "jäk: Using dynamic analysis to crawl and test modern web applications," in *International Symposium on Recent Advances in Intrusion Detection.* Springer, 2015, pp. 295–316.

[132] S. Choudhary, M. E. Dincturk, G. V. Bochmann, G.-V. Jourdan, I. V. Onut, and P. Ionescu, "Solving some modeling challenges when testing rich internet applications for security," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation.* IEEE, 2012, pp. 850–857.

[133] K. Ayoub, H. Aly, and J. Walsh, "Document object model (dom) based page uniqueness detection," Jul. 16 2013, uS Patent 8,489,605.

[134] A. Moosavi, S. Hooshmand, S. Baghbanzadeh, G.-V. Jourdan, G. V. Bochmann, and I. V. Onut, "Indexing rich internet applications using components-based crawling," in *International Conference on Web Engineering.* Springer, 2014, pp. 200–217.

[135] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "Enemy of the state: A {State-Aware}{Black-Box} web vulnerability scanner," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 523–538.

[136] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "Reverse engineering finite state machines from rich internet applications," in *2008 15th Working Conference on Reverse Engineering.* IEEE, 2008, pp. 69–73.

[137] ——, "Rich internet application testing using execution trace data," in *2010 Third International Conference on Software Testing, Verification, and Validation Workshops.* IEEE, 2010, pp. 274–283.

[138] ——, "An iterative approach for the reverse engineering of rich internet application user interfaces," in *2010 Fifth International Conference on Internet and Web Applications and Services.* IEEE, 2010, pp. 401–410.

[139] S. Sabharwal, P. Bansal, and M. Aggarwal, "Article: Modeling the navigation behavior of dynamic web applications," *International Journal of Computer Applications*, vol. 65, no. 13, pp. 20–27, March 2013, full text available.

[140] O. Grumberg, E. Clarke, and D. Peled, "Model checking," 1999.

[141] J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in cesar," in *International Symposium on programming.* Springer, 1982, pp. 337–351.

[142] R. Jhala and R. Majumdar, "Software model checking," *ACM Computing Surveys (CSUR)*, vol. 41, no. 4, pp. 1–54, 2009.

[143] C. Tian, S. Liu, and Z. Duan, "Abstract model checking with sofl hierarchy," in *International Workshop on Structured Object-Oriented Formal Language and Method.* Springer, 2012, pp. 71–86.

[144] R. Pelánek, "Fighting state space explosion: Review and evaluation," in *International Workshop on Formal Methods for Industrial Critical Systems.* Springer, 2008, pp. 37–52.

[145] D. Bošnački, "A light-weight algorithm for model checking with symmetry reduction and weak fairness," in *International SPIN Workshop on Model Checking of Software.* Springer, 2003, pp. 89–103.

[146] E. A. Emerson and T. Wahl, "Dynamic symmetry reduction," in *International conference on tools and algorithms for the construction and analysis of systems.* Springer, 2005, pp. 382–396.

[147] R. Iosif, "Symmetry reduction criteria for software model checking," in *International SPIN Workshop on Model Checking of Software.* Springer, 2002, pp. 22–41.

[148] C. Norris Ip, D. L. Dill *et al.*, "Better verification through symmetry," *Formal methods in system design*, vol. 9, no. 1, pp. 41–75, 1996.

[149] A. P. Sistla and P. Godefroid, "Symmetry and reduced symmetry in model checking?" in *International Conference on Computer Aided Verification.* Springer, 2001, pp. 91–103.

[150] T. Wahl, "Adaptive symmetry reduction," in *International Conference on Computer Aided Verification.* Springer, 2007, pp. 393–405.

[151] J.-C. Fernandez, M. Bozga, and L. Ghirvu, "State space reduction based on live variables analysis," *Science of Computer Programming*, vol. 47, no. 2-3, pp. 203–220, 2003.

[152] J. P. Self and E. G. Mercer, "On-the-fly dynamic dead variable analysis," in *International SPIN Workshop on Model Checking of Software.* Springer, 2007, pp. 113–130.

[153] M. B. Dwyer, J. Hatcliff, M. Hoosier, V. Ranganath, T. Wallentine *et al.*, "Evaluating the effectiveness of slicing for model reduction of concurrent object-oriented programs," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2006, pp. 73–89.

[154] J. Hatcliff, M. B. Dwyer, and H. Zheng, "Slicing software for model construction," *Higher-order and symbolic computation*, vol. 13, no. 4, pp. 315–353, 2000.

[155] Y. Dong and C. Ramakrishnan, "An optimizing compiler for efficient model checking," in *Formal Methods for Protocol Engineering and Distributed Systems.* Springer, 1999, pp. 241–256.

[156] R. Kurshan, V. Levin, and H. Yenigün, "Compressing transitions for model checking," in *International conference on computer aided verification.* Springer, 2002, pp. 569–582.

[157] P. Godefroid, *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem.* Springer, 1996.

[158] G. Gueta, C. Flanagan, E. Yahav, and M. Sagiv, "Cartesian partial-order reduction," in *International SPIN Workshop on Model Checking of Software.* Springer, 2007, pp. 95–112.

[159] G. J. Holzmann and D. Peled, "An improvement in formal verification," in *Formal Description Techniques VII.* Springer, 1995, pp. 197–211.

[160] D. Peled, "Combining partial order reductions with on-the-fly model-checking," in *International Conference on Computer Aided Verification.* Springer, 1994, pp. 377–390.

[161] W. Penczek, M. Szreter, R. Gerth, and R. Kuiper, "Improving partial order reductions for universal branching time properties," *Fundamenta Informaticae*, vol. 43, no. 1-4, pp. 245–267, 2000.

[162] S. Blom and J. v. d. Pol, "State space reduction by proving confluence," in *International Conference on Computer Aided Verification.* Springer, 2002, pp. 596–609.

[163] K. Ozdemir and H. Ural, "Protocol validation by simultaneous reachability analysis," *Computer Communications*, vol. 21, no. 6, pp. 591–591, 1998.

[164] J.-P. Krimm and L. Mounier, "Compositional state space generation from lotos programs," in *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 1997, pp. 239–258.

[165] C. Flanagan and S. Qadeer, "Thread-modular model checking," in *International SPIN Workshop on Model Checking of Software.* Springer, 2003, pp. 213–224.

[166] O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 843–871, 1994.

[167] A. Pnueli, "In transition from global to modular temporal reasoning about programs," in *Logics and models of concurrent systems.* Springer, 1985, pp. 123–144.

[168] J. Geldenhuys, P. De Villiers, and J. Rushby, “Runtime efficient state compaction in spin,” in *International SPIN Workshop on Model Checking of Software.* Springer, 1999, pp. 12–21.

[169] J. Geldenhuys and A. Valmari, “A nearly memory-optimal data structure for sets and mappings,” in *International SPIN Workshop on Model Checking of Software.* Springer, 2003, pp. 136–150.

[170] J.-C. Grégoire, “State space compression in spin with getss,” in *Proc. Second SPIN Workshop, Rutgers Univ.* Citeseer, 1996.

[171] G. J. Holzmann, P. Godefroid, and D. Pirottin, “Coverage preserving reduction strategies for reachability analysis,” in *Protocol Specification, Testing and Verification, XII.* Elsevier, 1992, pp. 349–363.

[172] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, “Efficient verification of real-time systems: Compact data structure and state-space reduction,” in *Proceedings Real-Time Systems Symposium.* IEEE, 1997, pp. 14–24.

[173] B. Parreaux, “Difference compression in spin,” in *SPIN*, vol. 56, 1998.

[174] W. Visser and H. Barringer, “Memory efficient state storage in spin,” in *Proceedings of the 2nd SPIN Workshop*, vol. 21. American Mathematical Society Providence, RI, 1996.

[175] G. J. Holzmann, “State compression in spin: Recursive indexing and compression training runs,” in *Proceedings of third international Spin workshop*, 1997.

[176] G. J. Holzmann and A. Puri, “A minimized automaton representation of reachable states,” *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 3, pp. 270–278, 1999.

[177] J. Geldenhuys, “State caching reconsidered,” in *International SPIN Workshop on Model Checking of Software.* Springer, 2004, pp. 23–38.

[178] P. Godefroid, G. J. Holzmann, and D. Pirottin, "State space caching revisited," in *International Conference on Computer Aided Verification.* Springer, 1992, pp. 178–191.

[179] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. Venturini Zilli, "Exploiting transition locality in automatic verification of finite-state concurrent systems," *International Journal on Software Tools for Technology Transfer*, vol. 6, no. 4, pp. 320–341, 2004.

[180] G. Behrmann, K. G. Larsen, and R. Pelánek, "To store or not to store," in *International Conference on Computer Aided Verification.* Springer, 2003, pp. 433–445.

[181] S. Christensen, L. M. Kristensen, and T. Mailund, "A sweep-line method for state space exploration," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2001, pp. 450–464.

[182] T. Mailund and M. Westergaard, "Obtaining memory-efficient reachability graph representations using the sweep-line method," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2004, pp. 177–191.

[183] K. Schmidt, "Automated generation of a progress measure for the sweep-line method," *International Journal on Software Tools for Technology Transfer*, vol. 8, no. 3, pp. 195–203, 2006.

[184] A. Groce and W. Visser, "Heuristics for model checking java programs," *STTT*, vol. 6, pp. 260–276, 08 2004.

[185] A. Kuehlmann, K. McMillan, and R. Brayton, "Probabilistic state space search," in *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051)*, 1999, pp. 574–579.

[186] K. Qian and A. Nymeyer, "Guided invariant model checking based on abstraction and symbolic pattern databases," in *International Conference*

on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2004, pp. 497–511.

[187] P. Godefroid and S. Khurshid, “Exploring very large state spaces using genetic algorithms,” *International Journal on Software Tools for Technology Transfer*, vol. 6, no. 2, pp. 117–127, 2004.

[188] P. Haslum, “Model checking by random walk,” 1999.

[189] R. Pelánek, T. Hanžl, I. Černá, and L. Brim, “Enhancing random walk state space exploration,” in *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, 2005, pp. 98–105.

[190] G. J. Holzmann, “Algorithms for automated protocol verification,” *AT&T technical journal*, vol. 69, no. 1, pp. 32–44, 1990.

[191] M. D. Jones and J. Sorber, “Parallel search for ltl violations,” *International Journal on Software Tools for Technology Transfer*, vol. 7, no. 1, pp. 31–42, 2005.

[192] F. J. Lin, P. Chu, and M. T. Liu, “Protocol verification using reachability analysis: the state space explosion problem and relief strategies,” in *Proceedings of the ACM workshop on Frontiers in computer communications technology*, 1987, pp. 126–135.

[193] M. Mihail and C. H. Papadimitriou, “On the random walk method for protocol testing,” in *International Conference on Computer Aided Verification.* Springer, 1994, pp. 132–141.

[194] G. J. Holzmann, “An analysis of bitstate hashing,” *Formal methods in system design*, vol. 13, no. 3, pp. 289–307, 1998.

[195] P. C. Dillinger and P. Manolios, “Bloom filters in probabilistic verification,” in *International Conference on Formal Methods in Computer-Aided Design.* Springer, 2004, pp. 367–381.

[196] ——, “Fast and accurate bitstate verification for spin,” in *International SPIN Workshop on Model Checking of Software*. Springer, 2004, pp. 57–75.

[197] P. Rawat and A. N. Mahajan, “Reactjs: A modern web development framework,” *International Journal of Innovative Science and Research Technology*, vol. 5, no. 11, pp. 698–702, 2020.

[198] A. Bhalla, S. Garg, and P. Singh, “Present day web-development using reactjs,” *International Research Journal of Engineering and Technology*, vol. 7, no. 05, 2020.

[199] H. J. S. Sitio, I. Christovita, R. K. Ahmad, and Y. Setiawan, “Web-based application development for training data management using reactjs,” *Indonesian Journal of Multidisciplinary Science*, vol. 2, no. 6, pp. 2573–2588, 2023.

[200] M. F. S. Lazuardy and D. Anggraini, “Modern front end web architectures with react. js and next. js,” *Research Journal of Advanced Engineering and Science*, vol. 7, no. 1, pp. 132–141, 2022.

[201] F. Ferreira and M. T. Valente, “Detecting code smells in react-based web apps,” *Information and Software Technology*, vol. 155, p. 107111, 2023.

[202] K. Dzwinel. (2018) Dom listener extension. [Accessed 29-Nov-2021]. [Online]. Available: https://github.com/kdzwinel/DOMListenerExtension

[203] hkokila. (2016) Html dom navigation. [Accessed 29-Nov-2021]. [Online]. Available: https://chrome.google.com/webstore/detail/html-dom-navigation/eimpgjcahblfpdgiknmbmglcafegimil?hl=en

[204] A. Khaira and R. Dwivedi, “A state of the art review of analytical hierarchy process,” *Materials Today: Proceedings*, vol. 5, no. 2, pp. 4029–4035, 2018.

[205] D. Dalalah, F. AL-Oqla, and M. Hayajneh, “Application of the analytic hierarchy process (ahp) in multi-criteria analysis of the selection of cranes,” vol. 4, pp. 567–578, 04 2009.

[206] F. Hujainah, R. B. A. Bakar, B. Al-Haimi, and M. A. Abdulgabber, "Investigation of stakeholder analysis in requirement prioritization techniques," *Advanced Science Letters*, vol. 24, no. 10, pp. 7227–7231, 2018.

[207] F. Hujainah, R. B. A. Bakar, M. A. Abdulgabber, and K. Z. Zamli, "Software requirements prioritisation: a systematic literature review on significance, stakeholders, techniques and challenges," *IEEE Access*, vol. 6, pp. 71 497–71 523, 2018.

[208] F. Hujainah, R. B. A. Bakar, B. Al-Haimi, and M. A. Abdulgabber, "Stakeholder quantification and prioritisation research: A systematic literature review," *Information and Software Technology*, vol. 102, pp. 85–99, 2018.

[209] M. Sadiq and V. S. Devi, "A rough-set based approach for the prioritization of software requirements," *International Journal of Information Technology*, vol. 14, no. 1, pp. 447–457, 2022.

[210] M. S. Jahan, F. Azam, M. W. Anwar, A. Amjad, and K. Ayub, "A novel approach for software requirement prioritization," in *2019 7th International Conference in Software Engineering Research and Innovation (CONISOFT).* IEEE Computer Society, 2019, pp. 1–7.

[211] M. Yaseen, A. Mustapha, and N. Ibrahim, "An approach for managing large-sized software requirements during prioritization," in *2018 IEEE Conference on Open Systems (ICOS).* IEEE, 2018, pp. 98–103.

[212] I. Ibriwesh, S.-B. Ho, and I. Chai, "Overcoming scalability issues in analytic hierarchy process with redccahp: An empirical investigation," *Arabian Journal for Science and Engineering*, vol. 43, 04 2018.

[213] A. Perini, A. Susi, F. Ricca, and C. Bazzanella, "An empirical study to compare the accuracy of ahp and cbranking techniques for requirements prioritization," in *2007 Fifth International Workshop on Comparative Evaluation in Requirements Engineering*, 2007, pp. 23–35.

[214] M. Ramzan, M. A. Jaffar, and A. A. Shahid, "Value based intelligent requirement prioritization (virp): expert driven fuzzy logic based prioritization

technique," *International Journal Of Innovative Computing, Information And Control*, vol. 7, no. 3, pp. 1017–1038, 2011.

[215] M. I. Babar, M. Ghazali, D. N. Jawawi, S. M. Shamsuddin, and N. Ibrahim, "Phandler: an expert system for a scalable software requirements prioritization process," *Knowledge-Based Systems*, vol. 84, pp. 179–202, 2015.

[216] M. I. Babar, M. GHAZALI, and D. N. JAWAWI, "Software quality enhancement for value based systems through stakeholders quantification." *Journal of Theoretical & Applied Information Technology*, vol. 55, no. 3, 2013.

[217] T. Tanaka, H. Niibori, L. Shiyingxue, S. Nomura, T. Nakao, and K. Tsuda, "Selenium based testing systems for analytical data generation of website user behavior," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, pp. 216–221.

[218] R. Teodoro, "Evaluating selenium for automated testing." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2017.

[219] A. Holmes and M. Kellogg, "Automating functional tests using selenium," in *AGILE 2006 (AGILE'06)*. IEEE, 2006, pp. 6–pp.

[220] J. D. Musa, "The operational profile," in *Reliability and Maintenance of Complex Systems*. Springer, 1996, pp. 333–344.

[221] R. Suganya and R. Shanthi, "Fuzzy c-means algorithm-a review," *International Journal of Scientific and Research Publications*, vol. 2, no. 11, p. 1, 2012.

[222] O. M. Jafar and R. Sivakumar, "A comparative study of hard and fuzzy data clustering algorithms with cluster validity indices," in *Proceedings of international conference on emerging research in computing, information, communication and applications*, 2013, pp. 775–782.

[223] N. Grover, "A study of various fuzzy clustering algorithms," *International Journal of Engineering Research*, vol. 3, no. 3, pp. 177–181, 2014.

[224] T. Singh and M. Mahajan, “Performance comparison of fuzzy c means with respect to other clustering algorithm,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 5, pp. 89–93, 2014.

[225] J. Nayak, B. Naik, and H. Behera, “Fuzzy c-means (fcm) clustering algorithm: a decade review from 2000 to 2014,” *Computational intelligence in data mining-volume 2*, pp. 133–149, 2015.

[226] R. L. Cannon, J. V. Dave, and J. C. Bezdek, “Efficient implementation of the fuzzy c-means clustering algorithms,” *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 248–255, 1986.

[227] S. Park and G. Kwon, “Avoidance of state explosion using dependency analysis in model checking control flow model,” in *International Conference on Computational Science and Its Applications.* Springer, 2006, pp. 905–911.

[228] A. Valmari, “The state explosion problem, lectures on petri nets i: Basic models, lncs tutorials, lncs 1491,” 1998.

[229] E. Clarke, O. Grumberg, and D. Peled, “Model checking the mit press,” *Cambridge, Massachusetts, London, UK*, 1999.

[230] R. Zhao, C. Chen, W. Wang, and J. Guo, “Automatic model completion for web applications,” in *International Conference on Web Engineering.* Springer, 2020, pp. 207–227.

[231] B. McLaughlin, G. Pollice, and D. West, *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D.* ” O’Reilly Media, Inc.”, 2006.

[232] M. Pozdeev, “myTinyTodo,” https://www.mytinytodo.net/, 2019, [Online; accessed 29-Nov-2021].

[233] A. Arora and M. Sinha, “Applying variable chromosome length genetic algorithm for testing dynamism of web application,” in *2013 International Conference on Recent Trends in Information Technology (ICRTIT).* IEEE, 2013, pp. 539–545.

[234] ——, "Dynamic content testing of web application using user session based state testing," 2013.

[235] A. Marchetto, P. Tonella, and F. Ricca, "Reajax: a reverse engineering tool for ajax web applications," *IET software*, vol. 6, no. 1, pp. 33–49, 2012.

[236] R. Pars, L. Moroney, and J. Grieb, "Using server controls in asp. net ajax," *Foundations of ASP. NET AJAX*, pp. 109–129, 2007.

[237] S. L. Pfleeger and J. M. Atlee, *Software engineering: theory and practice.* Pearson Education India, 1998.

[238] M. R. Lyu *et al.*, *Handbook of software reliability engineering.* IEEE computer society press CA, 1996, vol. 222.

[239] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 435–445.

[240] P. S. Kochhar, F. Thung, and D. Lo, "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems," in *2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER).* IEEE, 2015, pp. 560–564.

[241] M. Staats, G. Gay, M. Whalen, and M. Heimdahl, "On the danger of coverage directed test case generation," in *International Conference on Fundamental Approaches to Software Engineering.* Springer, 2012, pp. 409–424.

[242] Y. Wei, B. Meyer, and M. Oriol, "Is branch coverage a good measure of testing effectiveness?" in *Empirical Software Engineering and Verification.* Springer, 2010, pp. 194–212.

[243] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set size and block coverage on the fault detection effectiveness," in *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering.* IEEE, 1994, pp. 230–238.

[244] J. D. Musa, "Operational profiles in software-reliability engineering," *IEEE software*, vol. 10, no. 2, pp. 14–32, 1993.

[245] W. E. Howden, "Reliability of the path analysis testing strategy," *IEEE Transactions on Software Engineering*, no. 3, pp. 208–215, 1976.

[246] F. Grigorjev, N. Lascano, and J. L. Staude, "A fault seeding experience," in *Simposio Argentino de Ingenieria de Software (ASSE 2003).* Citeseer, 2003.

[247] A. J. Offutt and J. H. Hayes, "A semantic model of program faults," *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 3, pp. 195–200, 1996.

[248] K. H. T. Wah, "Fault coupling in finite bijective functions," *Software Testing, Verification and Reliability*, vol. 5, no. 1, pp. 3–47, 1995.

[249] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[250] G. Kaur and K. Bahl, "Software reliability, metrics, reliability improvement using agile process," *International Journal of Innovative Science, Engineering & Technology*, vol. 1, no. 3, pp. 143–147, 2014.

[251] G. Viswanathan and J. Prabhu, "Survey of methodologies for quantifying software reliability," *International Journal of Internet Technology and Secured Transactions*, vol. 10, no. 5, pp. 565–575, 2020.