Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2022.023423
Article

Tech Science Press

Achieving State Space Reduction in Generated Ajax Web Application State Machine

Nadeem Fakhar Malik[1,*], Aamer Nadeem[1] and Muddassar Azam Sindhu[2]

[1]Department of Computer Science, Capital University of Science and Technology, Islamabad, 45750, Pakistan
[2]Department of Computer Sciences, Quaid-i-Azam University, 45320, Pakistan
*Corresponding Author: Nadeem Fakhar Malik. Email: nadeem.fakhar@gmail.com
Received: 07 September 2021; Accepted: 10 November 2021

Abstract: The testing of Ajax (Asynchronous JavaScript and XML) web applications poses novel challenges for testers because Ajax 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. that require more demanding web-testing methods. Model based testing is amongst the effective approaches to detect faults in web applications. However, the state model generated for an Ajax application can be enormous and may be hit by state explosion problem for large number of user action based changes and for immense dynamism. Recent research has not addressed this issue comprehensively because existing techniques either apply partial reduction or compromise the effectiveness of testing. 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. Experimental results show that the framework controls the size of generated state machine without compromising the effectiveness of testing.

Keywords: State space; soft computing; Ajax web; Fuzzy C Mean; document object model; state machine

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, enhance-able, 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 test and maintain Ajax applications [9].

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 test 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.

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 [17]. 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 are 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 Mean [18] 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.

Web application testing proves to be a difficult task and the advent of Ajax applications has increased the complexity even further. Conventional web testing techniques [19–23] do not test 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.

This research proposes an approach to support the testing area in terms of Ajax applications. The technique uses soft computing based FCM to identify the most pivotal use cases and extracts a finite state machine for each use case. The approximation and learning nature of FCM makes it a good candidate to handle dynamic nature of Ajax application testing. These generated state machines can later be combined to construct an aggregated state machine of the given Ajax application.

The rest of this paper is organized as follows: Section II discusses the related work in the area of web application testing with focus on Ajax applications. Section III describes the proposed framework,

methodology, and state machine construction. Section IV gives the experimental setup and results of proposed methodology. Section V evaluates the effectiveness of the result and Section VI concludes the paper.

2 Related Work

Testing web applications through finite state machine is an effective way as compared to any other technique. Numerous notations and diagrammatic methods are used to represent web applications but finite state machines can depict their dynamic behavior very conveniently. A State machine offers an expedient method to model software behavior in a way that evades issues related with the implementation [19]. In Ajax web applications the object states change in response to user or server triggered events at run time, hence a finite state machine model can be very effectively used to show this changing behavior [9,24].

Donley et al. [25] discussed that the complexity of web-based applications has dramatically increased over the period of time. The software engineers even face issues to clearly identify the differences between web-based applications and traditional applications. They emphasized the point that in order to test web-based applications it is very important to first get a better understanding of these applications. They discussed the most important differences between traditional and web-based applications.

Arora et al. [26] discussed couple of prominent testing techniques: invariant based and state-based testing. Although these techniques remain successful regarding different domains, many issues and problems still exist, e.g., scalability issues. Issues like capturing session data, reduction of state space, advancement in FSM retrieval to automatically deduce user session-based test cases, still need answers. The authors discussed that DOM to FSM progression need a more meaningful technique. The experiments steered in this technique were able to generate test cases regarding semantically interacting sequences and evidences showed that long sequences produce more test cases with greater fault exposing potential. Finally, the authors accentuated that testing is fore mostly reliant on technology through which it is implemented and imminent techniques of testing have to adapt with assorted and vibrant nature of applications running on the web.

Marchetto et al. [24] gathered the execution trace data of the web application and then used that data to build a finite state machine. The basis of this technique is a dynamically extracted state machine for a given Ajax application, however the technique is partially dynamic and manual validation is also used for model extraction. This research states that finite state machine retrieval is mainly an unexplored area and needs improvement [24]. Dynamic identification of states, in Ajax testing, is a difficult task and requires persistent attention. Therefore, it was required to have a dynamic analysis method to construct state based model of the application. Mainly, the authors focused on detecting sets of event sequences that interact semantically, and are used to produce test cases [27]. Their perception was that the length of these sequences affect their fault revealing capability, i.e., more length more faults, and this was also validated by the conducted experiments [27–30]. The technique produces test cases that are very large in number thus resulting in state explosion and also involves events that are not related. Marchetto contributed towards minimization of test case number but only those test cases that are having asynchronous communication. Only few facets of asynchronous communication are addressed by this approach and other testing challenges like runtime changes in DOM, how to fetch these changes, and transition between different DOM states still need to be explored. Moreover, finite state machine extraction using dynamic analysis is also required.

Mesbah et al. [13,31] proposed automatic testing of Ajax user interfaces based on invariants. Main task in this work was crawling of Ajax applications using CRAWLJAX, which is a tool that simulates the real user actions on different clickable elements of application interface and infers the model from state flow graph.

Moreover, he also suggested the use of automatic invariant detection based on CRAWLJAX. Crawljax uses Levenshtein method [32] to identify the difference between two DOM instances by calculating their edit distance. The authors acknowledged in their research that for automated testing of web applications best path seeding practice is, capture and replay, which did not apply in his work. Mesbah's point of view in his work [13] suggested that invariant based testing proves to be a weak form of oracle. The authors added that dynamic state extraction can be handled best by using capture and reply tools, otherwise Ajax dynamism is big challenge for thorough testing.

Arie et al. [3] thoroughly discussed the issues and challenges faced during the analysis of modern asynchronous applications in terms of crawling. Modern applications on the web have significantly moved towards single-page model, in which DOM based user interface and interaction is maintained by JavaScript engine. This results in many analyses and understanding challenges, which modern day static analysis tools are unable to handle. These challenges include State Explosion, State Navigation, Triggering State Changes, and Unreachable States. In this paper, the authors have discovered how automated crawling can be used to address these challenges. Moreover, they identified a number of promising areas for future research which include: dynamic analysis of modern applications on the web including Benchmarking, Guided Crawling, Example-Based Crawling, Model-Based Web Application Analysis, and Cyber-Security.

Sabharwal et al. [33] 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.

Arora et al. [6] focused on testing the runtime behavior concerns of asynchronous web applications. They devised various experiments for it. They offered a process that generates a state machine to discover all on the fly spawned states and their referenced events along with DOM altering 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.

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 Proposed Approach

This section proposes a framework *StateReduceAjax* to achieve state space reduction in state machine of an Ajax web application. As already discussed, one of the significant issues in model based testing of Ajax applications is the state explosion problem. More state space under test, more this issue becomes unmanageable. The main objective behind the construction of *StateReduceAjax* is to reduce the state space.

The framework uses FCM 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.

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 U_iW for each use case. The use case based session recording calculates how frequently a use case is executed. Fuzzy C Mean takes set of U_iW 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. These pivotal use cases are required because during the execution of Ajax web applications, the possible number of events and their associated elements can be very large in number. It is almost 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 (DOM) mutations using (DOM) tools [34,35]. These (DOM) mutations are the (DOM) states and *StateReduceAjax* finally connects these (DOM) states to construct the state machine. Fig. 1 shows the complete working of the proposed solution.

In this research the proposed framework accomplishes its objectives by using Algorithm 1 shown in Tab. 1.

This algorithm initiates by taking Requirement set as input and then sends it to requirement prioritization module. Requirement prioritization module takes stakeholder set as input if required, applies requirement prioritization method 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 all the use cases in 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.

3.1 Stage I

The first stage reads system requirements from the file for further processing.

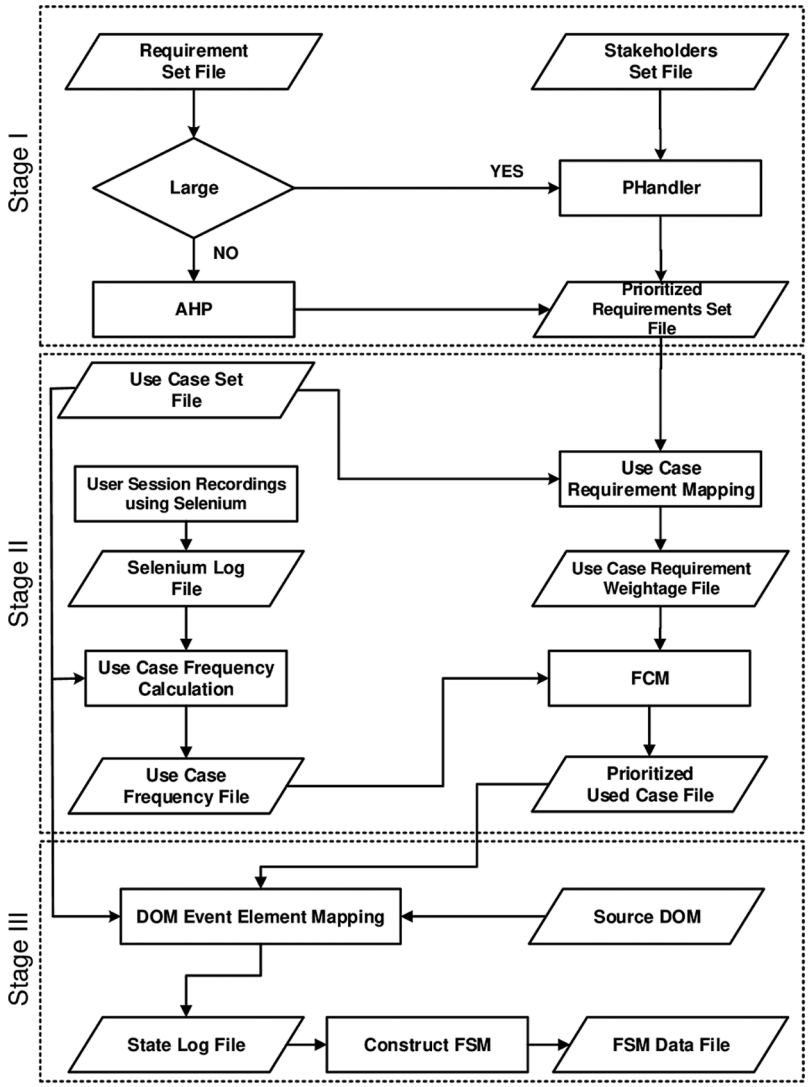

Figure 1: Proposed solution

Table 1: Algorithm I

Procedure *StateReduceAjax* ( )	
Declare	Prioritized_Requirements_Set, Prioritized_Usecase_Set, FSM_Data
Input	Requirements_Set, Stakeholder_Set
Output	Prioritized_Requirements_Set, Prioritized_Usecase_Set, FSM_Data
1	PrioritizeRequirements()
2	PrioritizeUsecases()
3	for each Prioritized_Usecase_id in Prioritized_Usecase_Set.Usecase_id
4	gererateFSM()
end_procedure	

3.1.1 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 this research applies AHP [36] to prioritize the requirements. The requirement set having large number of requirements is a candidate for PHandler [37].

In the first step, AHP reads the requirements file and formulates an n x n matrix where n is the number of candidate requirements. 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'.

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. 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. 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.

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 [38]. This method assigns a 1–10 range value to each stakeholder based on some key attributes. These attributes are significance, domain knowledge, participation level, dependency, control and level of decision-making. 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). The projRCFs are feasibility, modifiability, urgency, traceability, and testability while the reqRCFs considered are completeness, consistency, understandability, within the scope and non-redundancy. Eq. (1) uses these RCFs to determine value of each requirement (RV) in the range of 0 to 5.

$$RV = 0.35 + 0.02 \left\{ \sum_{i=1}^{5} pRCFi + \sum_{i=1}^{5} rRCFi \right\} \tag{1}$$

RCFi in Eq. (1) indicates the specific classification factor whose existence or nonexistence affects the RV of a requirement [36].

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 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.

Tab. 2 shows the algorithm for stage I. Algorithm II takes requirement set as input and the output of this module is Prioritized Requirement Set.

Table 2: Algorithm II

Procedure PrioritizeRequirements()	
Declare	LARGE = 30
Input	Requirements_Set, Stakeholder_Set
Output	Prioritized_Requirements_Set
1	if (Requirements_Set.count() >= LARGE) then
2	Prioritized_Requirements_Set = PHandler(Requirements_Set,Stakeholder_Set)
3	else
4	Prioritized_Requirements_Set = AHP(Requirements_Set)
end_procedure	

3.2 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 [39], to gather the user interaction information with reference to the system functionality. Selenium is a functional testing tool that records the user events and subsequent behavior as the user interacts with system under test.

StateReduceAjax uses these recordings to calculate the use case frequency which along with Use Case Requirement Weightage U_iW is input to Fuzzy C mean classification algorithm (FCM) [18]. (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. U_iW 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. Tab. 3 depicts the algorithm for stage II.

Table 3: Algorithm III

Procedure PrioritizeUsecases ()		
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
Input	:	Prioritized_Requirements_Set, Usecase_Set, Ajax_URL, Session_Id
Output	:	Prioritized_Usecase_Set
1		for each Usecase in Usecase_Set
2		for each Step in Usecase
3		for each Prioritized_Requirement in Prioritized_Requirements_Set
4		if (Map(Prioritized_Requirement, Step) == True)
5		Usecase_Requirement_Weight += Prioritized_Requirements_Set. Prioritized_Requirement.Weight
6		Usecase_Requirement_Weightage_Set.append([Usecase.Usecase_id, Usecase_Requirement_Weight])

(Continued)

Table 3 (continued)

Procedure PrioritizeUsecases ()	
7	Usecase_Requirement_Weight = 0
8	Selenium_Log_File = Selenium.Execute(Session_Id, Ajax_URL)
9	while !EOF (Selenium_Log_File)
10	for each Session_id in Selenium_Log_File.Session_id
11	for Usecase_id in Usecase_Set.Usecase_id
12	Intermediate_Usecase_Count = Calculate_Usecase_Count(Session_id,Usecase_id, Usecase_Set, Selenium_Log_File)
13	Intermediate_Usecase_Frequency = Usecase_Count/Selenium_Log_File.SessionTime
14	Intermediate_Usecase_Frequency_Data.append(Usecase_id, Intermediate_Usecase_Frequency)
15	for each Usecase_id in Usecase_Set
16	while !EOF() Intermediate_Usecase_Frequency_Data
17	if (Usecase_id == Intermediate_Usecase_Frequency_Data.Usecase_id)
18	Usecase_Frequency = Usecase_Frequency + Intermediate_Usecase_Frequency_Data.Intermediate_Usecase_Frequency
19	Usecase_Frequency_Data.appned(Usecase_id, Usecase_Frequency)
20	Usecase_Frequency = 0
21	for each Usecase in Usecase_Frequency_Data
22	Fuzzy_Data_File.append() = [Usecase_Frequency_Data.Usecase.Usecase_id, Usecase_Frequency_Data. Usecase_Frequency, Usecase_Requirement_Weightage_Set.Usecase_Requirement_Weight]
23	Prioritized_Usecase_Set = FCM.Execute(Fuzzy_Data_File, Number_of_Clusters)
end_procedure	

3.2.1 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 [40]. 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. In order to record multiple sessions, multiple users use the application and Selenium records the usage data 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 Tab. 8 shows the working of user session recording functionality.

3.2.2 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 [41]. 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 for testing. 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 Tab. 8 shows the working of use case frequency calculation. The output of use case frequency calculation is the use case frequency data.

3.2.3 Use Case Requirement Mapping

The next level of this stage calculates the use case requirement mapping. Every use case comprises of multiple requirements and here the solution identifies the requirements related to each use case. 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. 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 Tab. 8 shows the working of use case requirement mapping. The output of this step is Use Case Requirement Weightage Set.

3.2.4 Application of Fuzzy C Mean Clustering

The last level of Stage II implements Fuzzy C mean classification algorithm (FCM) [18] 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 [42–46]. Previously studies have used FCM and its variations in the area of software requirements prioritization [36] and pattern recognition [47]. 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 Eq. (2).

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

d is the Euclidean distance between two objects x, and y and x_i and y_i 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. Eq. (3) is used in FCM to compute membership level during iterative optimization process of the FCM.

$$U_{ij} = \frac{1}{\sum_{k=1}^{c} \left[\frac{||x_i - c_j||}{||x_i - c_k||}\right]^{\frac{2}{m-1}}} \tag{3}$$

Eq. (4) denotes the objective function used in the FCM.

$$J = \sum_{i=1}^{N} . \sum_{j=1}^{N} U_{ij}^{m} ||x_j - v_i||^2 \tag{4}$$

In Eqs. (3) and (4) U_{ij} is the membership of x_j in ith cluster, v_i is the center of the *ith* cluster, the bars ||…|| 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. 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.

This level starts by taking Use case requirement weightage and Use case frequency as input and then generates the most pivotal use cases as output. Both use case requirement weightage and use case frequency have got their own importance. We can have use cases having more weightage but are use less frequently used by the users and vice versa. For this very reason we cannot decide the importance of a use case merely on one factor. We need to identify the importance of a use case on the basis of relationship between both these factors.

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. 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 Tab. 8 shows the functionality of FCM. The output of FCM is Prioritized use case set.

3.3 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. 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 and HTML DOM Navigation tools. It then maps the extracted events to corresponding DOM elements. The solution 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 Tab. 4.

Table 4: Algorithm IV

```
Procedure gererateFSM()
Declare   Current_DOM, New_DOM, Element_List, State_List, transition, targetState, startState,
          State_Log_File
Input     Prioritized_Usecase_id, SourceDOM, Usecase_Set
Output    FSM_Data_File
1         Current_DOM = SourceDOM
2         while !EOF(Usecase_Set)
3                   if (Usecase_Set.Usecase_id == UseCase_id)
4                             for each Step in Usecase_Set
5                                       Event = Selenium.execute(Step)
6                                       newDOM = browser.fetchDom(Ajax_URL)
7                                       If (isDifferent(currDOM, newDOM))
8                                                 Element_List = ChangedElements(NewDOM)
9                                                 Curr_DOM = New_DOM
10                                                State_List = [Event , Element_List]
11                                                State_Log_File.append(State_List)
12                                                State_List = 0
13                                                Element_List = 0
14        if (State_Log_File not empty)
15                  StartState = SourceDOM
16                  while (! EOF (State_Log_File)
17                            transition = State_Log_File.Event
18                            targetState = State_Log_File.Element_List
19                            FSM_Data_File.append(startState, transition, targetState)
20                            StartState = targetState
21                  drawGraph(FSM_Data_File)
22                  State_Log_File = NULL
end_procedure
```

3.3.1 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 [48–51].

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 [24] 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.

Line 1 to 13 of Algorithm IV in Tab. 9 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.

3.3.2 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. 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 Tab. 9 shows the working of state machine construction which outputs FCM Data File. At the end, this file is sent to Draw Graph function for drawing the state machine.

Next section of the paper discusses the experimental results and analysis on those results.

4 Experimental Results and Analysis

This section presents the experiments conducted to prove the hypothesis stated earlier. The experiments implement *StateReduceAjax* discussed in the previous section. The first case study taken in this regard is a small Ajax based application called Ajax powered Coffee Maker [52].

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 '$n \times n$' matrix where 'n' denotes the total number of requirements. AHP then carries out all the steps as mentioned in Section 3.1.1 to generate Weighted Requirements Matrix. Tab. 5 shows 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. It uses Prioritized Requirement Set and Use Case Set to apply use case-requirements mapping. 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. 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) [18]. 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. Tab. 6 shows the Ajax Powered Coffee Maker Fuzzy Input Set taken from Use Case Requirement Weightage Data and Use Case Frequency Data respectively.

Table 5: 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	0.10	0.06	0.04	0.16	0.07	0.12	0.11	0.08	0.09	0.10	0.04	0.10	0.04	0.12	0.05	0.11	0.10	1.82	22.78
2	0.04	0.05	0.15	0.06	0.03	0.09	0.10	0.02	0.06	0.08	0.04	0.04	0.05	0.09	0.05	0.02	0.06	0.03	0.05	0.10	1.20	22.88
3	0.04	0.03	0.07	0.10	0.06	0.09	0.16	0.07	0.12	0.11	0.08	0.09	0.10	0.04	0.10	0.04	0.12	0.10	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	0.10	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	0.10	0.04	0.10	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	0.10	0.04	0.10	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	0.10	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	15.50
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	0.10	0.02	0.06	0.08	0.04	0.04	0.05	0.09	0.05	0.02	0.06	0.10	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	0.10	1.06	24.41
13	0.04	0.05	0.04	0.06	0.03	0.02	0.10	0.02	0.06	0.08	0.04	0.04	0.05	0.09	0.05	0.02	0.06	0.10	0.05	0.02	1.03	20.83
14	0.08	0.03	0.07	0.10	0.06	0.04	0.02	0.04	0.12	0.11	0.02	0.02	0.02	0.04	0.10	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	0.10	0.02	0.06	0.08	0.04	0.04	0.05	0.02	0.05	0.02	0.06	0.10	0.05	0.02	0.97	18.87
16	0.08	0.11	0.07	0.10	0.02	0.04	0.16	0.04	0.12	0.01	0.08	0.02	0.10	0.04	0.10	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	0.10	0.07	0.06	0.08	0.04	0.04	0.05	0.09	0.05	0.02	0.06	0.10	0.05	0.02	1.15	19.74
18	0.08	0.11	0.04	0.10	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	0.10	0.02	0.06	0.08	0.04	0.04	0.05	0.02	0.05	0.08	0.06	0.10	0.05	0.10	1.10	20.57
20	0.04	0.03	0.07	0.10	0.06	0.04	0.16	0.04	0.12	0.01	0.08	0.02	0.10	0.04	0.10	0.04	0.12	0.05	0.03	0.05	1.30	26.83

Table 6: 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

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. Fig. 2 shows the Fuzzy C Mean results of the Coffer Maker.

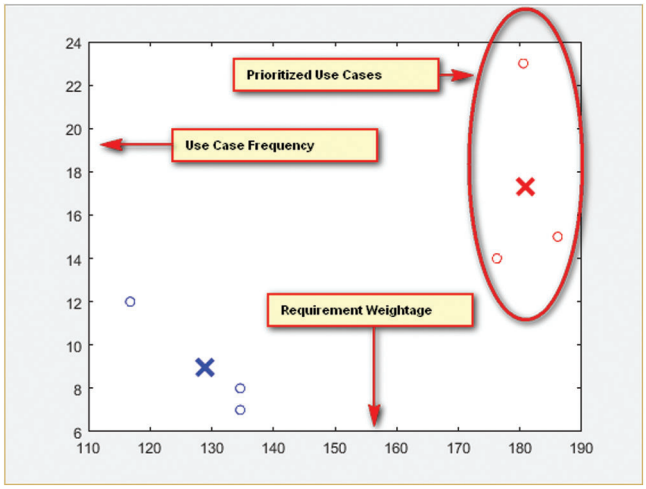

Figure 2: FCM results for Ajax powered coffee maker

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. Tab. 7 shows the "Place Order by Entering Name, Size and Beverage Type" use case steps along with DOM states.

Table 7: Use case steps with DOM states

S. no.	Use case steps	DOM state XPath
1	Enter name	//*[@id = "name"]
2	Select size	//*[@id = "controls1"]/form/p [2]
3	Select beverage	//*[@id = "controls1"]/form/p [3]
4	Order coffee	//*[@id = "btnUpdate"]
5	Coffee maker Status	//*[@id = "coffeemaker1-status"]

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 which outputs FSM data file and constructs state machine for the use case. Fig. 3 shows the state machine for use case.

Second case study is an Ajax based ToDo list [53]. 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. Tab. 8 shows the RV values of some ToDo list requirements.

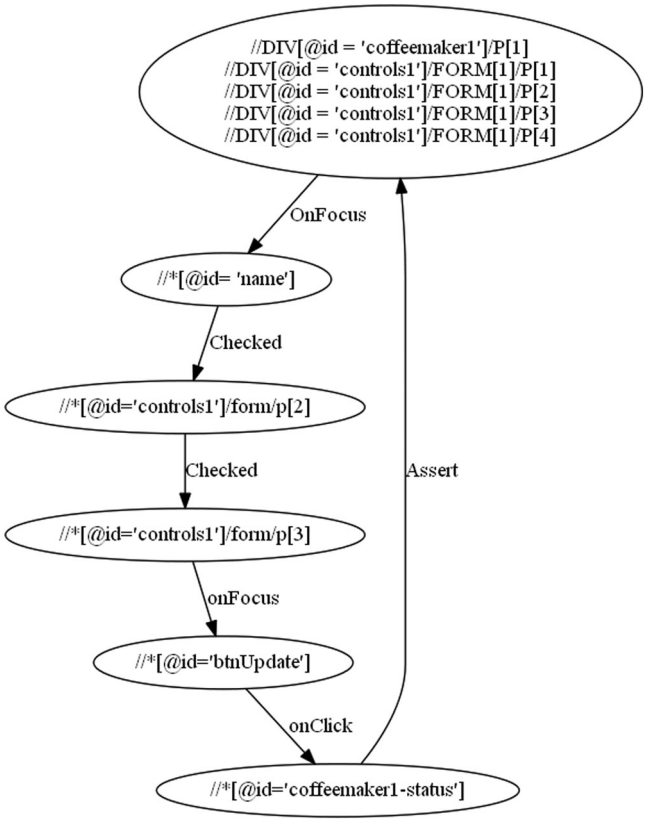

Figure 3: State machine of "place order by entering name, size and beverage type" use case

The output of PHandler is the Prioritized Requirement Set. The solution then reads ToDo list use cases from Use Case Set and uses both these sets to apply use case-requirements mapping. 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). Tab. 9 shows the ToDo List Fuzzy input Set taken from Use Case Requirement Weightage Data and Use Case Frequency Data respectively.

The output of FCM is the prioritized use case set having two clusters, i.e., pivotal and non-pivotal. Fig. 4 shows the Fuzzy C Mean results of the 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.

Table 8: 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**

Table 9: Todo List FCM input data

Use case	Weightage	Frequency	Use case	Weightage	Frequency	Use case	Weightage	Frequency	Use case	Weightage	Frequency
UC1	3.41	7	UC12	4.24	103	UC23	1.05	61	UC34	4.32	3
UC2	2.95	141	UC13	5.11	44	UC24	1.01	11	UC35	2.76	1
UC3	6.79	63	UC14	1.11	5	UC25	0.95	31	UC36	2.21	1
UC4	3.49	33	UC15	1.31	3	UC26	0.99	26	UC37	1.58	1
UC5	4.19	8	UC16	1.09	13	UC27	0.89	15	UC38	3.09	2
UC6	3.44	13	UC17	2.02	21	UC28	0.98	71	UC39	2.16	2
UC7	2.58	19	UC18	0.99	2	UC29	0.97	57	UC40	4.7	7
UC8	1.24	127	UC19	1.19	17	UC30	0.89	16	UC41	1.48	1
UC9	1.24	35	UC20	1.04	9	UC31	1.04	6	UC42	2.14	17
UC10	2.02	17	UC21	1.04	30	UC32	1.74	2			
UC11	0.98	7	UC22	1.01	54	UC33	2.51	0			

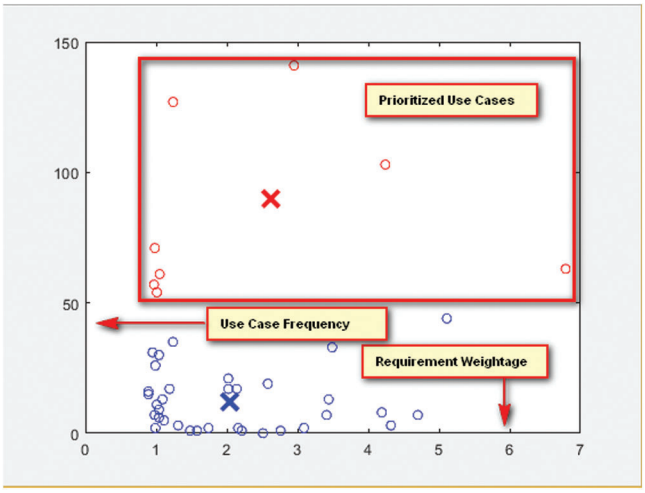

Figure 4: FCM results of Todo list

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.

Tab. 10 shows the use case steps with the DOM states of "Add Task to a list" use case and Fig. 5 shows its state machine.

Table 10: "Add task to a list" use case steps with DOM states

	Use case steps	DOM state XPath
1	Select list	//*[@id = "mtt_body"]/h2 //*[@id = "settings"] //*[@id = "list_id"]/a/span //*[@id = "htab_newtask"]/table //*[@id = "taskview"]/span [1] //*[@id = "tabs_buttons"]/div //*[@id = "htab_search"]/table
2	Enter task Label	//*[@id = "task"]
3	Click submit Button	//*[@id = "newtask_submit"]
4	Task added	//*[@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]

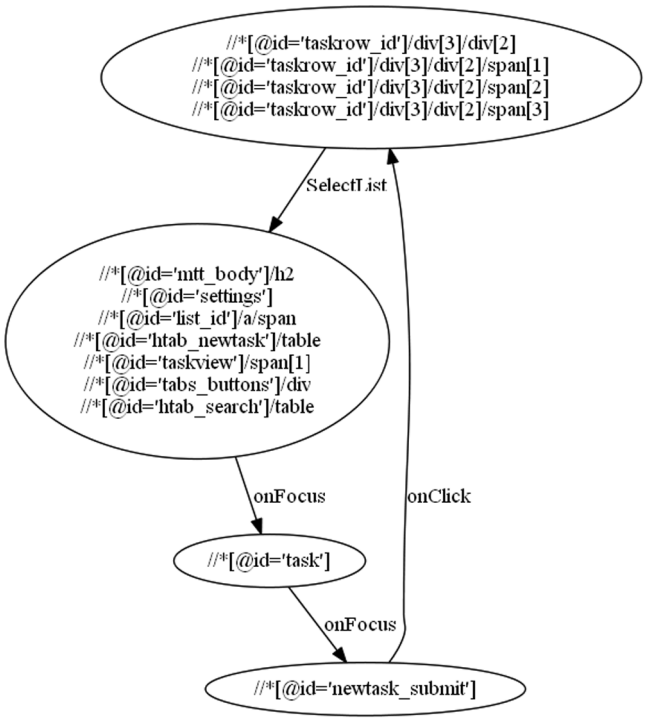

Figure 5: State machine for the "Add Task to List" use case

5 Evaluation of the Proposed Solution

In order to evaluate the proposed approach we have considered 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 technique is capable enough to uncover them. The comparison of detected and undetected seeded faults gives a confidence measure of testing [54]. 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 [55]. Almost all studies consider the faults alone as the measure of testing effectiveness without considering their frequency of execution [56–60]. This frequency of execution or the frequency profile provides the quantitative characterization of system usage [61]. Software testing based on this frequency of execution confirms that most frequently used operations are focused thus achieving maximum reliability in available testing time [61]. One important thing to consider here is the fault types to be seeded. This research uses classification of faults given in [62] and their distribution given in [63]. Grigorjev et al. [63] give a classification of faults and their distribution in the program, as shown in Tab. 11.

Table 11: 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 [64], in isolated manner [65], or by a human expert [66]. 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 [63]. Only those severity types are used that do not lead to system crashes, i.e., 3 to 5. Tab. 12 shows the severity levels as given by [63].

Table 12: 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). 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. Eqs. (5)–(7) show the definitions of Effectiveness of a test case, Effectiveness of testing and Cost respectively.

$$Effectiveness_{tc} = \frac{Frequency\ of\ \text{corresponding use}\ case}{\sum_{i=1}^{n} Frequency\ of\ \text{uc}_i} \tag{5}$$

$$Effectiveness_{High} = \sum_{i=1}^{h} Effectiveness_{tc_i} \tag{6}$$

$$Cost_{Testing} = \frac{\sum_{j=1}^{h} \{tc_j\}}{\sum_{i=1}^{n} \{tc_i\}} \tag{7}$$

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. Tab. 13 shows the frequency and effectiveness of a use case for an example scenario.

Table 13: Effectiveness of a use case

Use case/test case	Frequency	Effectiveness
UC_1/TC_1	2	0.0625
UC_2/TC_2	3	0.09375
UC_3/TC_3	10	0.3125
UC_4/TC_4	5	0.15625
UC_5/TC_5	12	0.375

FCM results show that UC_3 and UC_5 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. Eq. (8) shows the relationship between high and low priority use cases.

$$UC_{All} = UC_{High} \cup UC_{Low} \tag{8}$$

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. Eq. (9) shows this relationship.

$$Cost(UC_{High}) = \frac{|UC_{High}|}{|UC_{All}|} \tag{9}$$

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. Eq. (10) shows the effectiveness calculation.

$$Effectiveness(UC_{High}) = \frac{\sum_{j=1}^{h} \{freq(uc_j)\}_{High}}{\sum_{i=1}^{n} \{freq(uc_i)\}_{All}} \tag{10}$$

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, Eqs. (11) and (12) are used respectively.

$$Reduction_{Cost}(UC_{All}) = 1 - Cost(UC_{High}) \tag{11}$$

and

$$Reduction_{effectiveness}(UC_{All}) = 1 - Efectiveness(UC_{High}) \tag{12}$$

Tab. 14 shows the results of these calculations on the Coffee Maker and ToDo List case studies.

Table 14: Calculation on case studies

Case study/calculations	Coffee maker	Todo list
No of use cases	06	42
Frequency of use cases	79	1102
High category use cases	03	08
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.

5.1 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 [6,29] 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 [29] 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. Tab. 15 shows the comparison amongst the approaches.

Table 15: Comparison of approaches

Factors/ approach	Arora et al. [6]	Arora et al. [29]	*StateReduceAjax using FCM*
Scalability	Low	High	High
Cost	(Difficult to implement) High	(No additional functionality or extensive modification required) High	(No additional functionality or extensive modification required) Low
Effectiveness	(Machine is first built and then reduced) Low (Effects of reduction not addressed)	(Machine is built for every session) Low (Do not guarantee state explosion avoidance)	Machine is built only for most pivotal use cases) High (All the pivot areas are tested)

6 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%.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] R. Suryansh, R. Krishna and A. Nayak, "Distributed component-based crawler for AJAX applications," in *IEEE Second Int. Conf. on Advances in Electronics, Computers and Communications ICAECC*, Bengaluru, India, pp. 1–6, 2018.

[2] K. Shah, S. Khusro and I. Ullah, "Crawling ajax-based web applications: Evolution and state-of-the-art," *Malaysian Journal of Computer Science*, vol. 31, no. 1, pp. 35–47, 2018.

[3] V. D. Arie, A. Mesbah and A. Nederlof, "Crawl-based analysis of web applications: Prospects and challenges," *Science of Computer Programming*, vol. 97, no. Part 1, pp. 173–180, 2015.

[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, B. Engin and V. D. Arie. "Crawling ajax by inferring user interface state changes," in *Eighth Int. Conf. on Web Engineering ICWE'08 IEEE*, New York, USA, pp. 122–134, 2008.

[6] A. Arora and M. Sinha, "Avoiding state explosion problem of generated AJAX web application state machine using BDD," in *Sixth Int. Conf. on Contemporary Computing IC3*, Noida, India, pp. 381–386, 2013.

[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. 47741–47755, 2020.

[8] A. Mesbah and V. D. Arie, "A component-and push-based architectural style for ajax applications," *Journal of Systems and Software*," vol. 81, no. 12, pp. 2194–2209, 2008.

[9] A. Marchetto, F. Ricca and P. Tonella, "A case study-based comparison of web testing techniques applied to AJAX web applications," *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 6, pp. 477–492, 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 (IJACI)*, vol. 10, no. 3, pp. 33–59, 2019.

[11] A. Arora and M. Anuja, "Test case generation using progressively refined genetic algorithm for ajax Web application testing," *Recent Patents on Computer Science*, vol. 11, no. 4, pp. 276–288, 2018.

[12] V. D. Arie and A. Mesbah. "Research issues in the automated testing of ajax applications," in *Int. Conf. on Current Trends in Theory and Practice of Computer Science*, Berlin, Germany, pp. 16–28, 2010.

[13] A. Mesbah and V. D. Arie, "Invariant-based automatic testing of AJAX user interfaces," in *IEEE 31st Int. Conf. on Software Engineering*, Vancouver, Canda, pp. 210–220, 2009.

[14] E. M. Clarke and O. Grumberg, "Avoiding the state explosion problem in temporal logic model checking," in *Proc. of the Sixth Annual ACM Symp. on Principles of Distributed Computing*, Vancouver, Canda, pp. 294–303, 1987.

[15] S. Kimura and E. M. Clarke, "A parallel algorithm for constructing binary decision diagrams," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, Cambridg, USA, pp. 220–223, 1990.

[16] K. Böhmer and S. Rinderle-Ma, "A systematic literature review on process model testing: Approaches, challenges, and research directions," arXiv preprint arXiv:1509.04076, 2015.

[17] D. Ibrahim, "An overview of soft computing," *Procedia Computer Science*, vol. 102, pp. 34–38, 2016.

[18] A. Ahmad and S. S. Khan, "Survey of state-of-the-art mixed data clustering algorithms," *IEEE Access*, vol. 7, pp. 31883–31902, 2019.

[19] A. A. Andrews, J. Offutt and R. T. Alexander, "Testing web applications by modeling with FSMs," *Software & Systems Modeling*, vol. 4, no. 3, pp. 326–345, 2005.

[20] 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.

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

[22] F. Ricca and P. Tonella, "Analysis and testing of web applications," in *Proc. of the 23rd Int. Conf. on Software Engineering ICSE*, Toronto, Canada, pp. 25–34, 2001.

[23] S. Sampath and S. Sprenkle, "Advances in Web application testing, 2010–2014," *Advances in Computers Elsevier*, vol. 101, pp. 155–191, 2016.

[24] A. Marchetto, P. Tonella and F. Ricca, "State-based testing of ajax web applications," in *1st Int. Conf. on Software Testing, Verification, and Validation*, Lillehammer, Norway, pp. 121–130, 2008.

[25] B. Donley and J. Offut, "Web application testing challenges," *Software Engineering*, George Mason University, Virginia, USA, 2009.

[26] A. Arora and M. Sinha, "Web application testing: A review on techniques, tools and state of art," *International Journal of Scientific & Engineering Research*, vol. 3, no. 2, pp. 1–6, 2012.

[27] A. Marchetto and P. Tonella, "Search-based testing of ajax web applications," in *1st Int. Symp. on Search Based Software Engineering IEEE*, Windsor, Canada, pp. 3–12, 2009.

[28] S. Sampath, V. Mihaylov, A. Souter and L. Pollock, "A scalable approach to user-session based testing of web applications through concept analysis," in *19th Int. Conf. on Automated Software Engineering*, Linz, Austria, pp. 132–141, 2004.

[29] A. Arora and M. Sinha, "A sustianable approach to automate user session based state machine generation for AJAX web applications," *Journal of Theoretical and Applied Information Technology*, vol. 53, no. 3, pp. 401–419, 2013.

[30] A. Marchetto and P. Tonella, "Using search-based algorithms for ajax event sequence generation during testing," *Empirical Software Engineering*, vol. 16, no. 1, pp. 103–140, 2011.

[31] A. Mesbah, V. D. Arie and S. Lenselink, "Crawling ajax-based web applications through dynamic analysis of user interface state changes," *ACM Transactions on the Web TWEB*, vol. 6, no. 1, pp. 1–30, 2012.

[32] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[33] S. Sabharwal, P. Bansal and M. Aggarwal, "Modeling the navigation behavior of dynamic Web applications," *International Journal of Computer Applications*, vol. 65, no. 13, pp. 20–27, 2013.

[34] K. Dzwinel, DOM Listener Extension, 2018. [Online]. Available: https://github.com/kdzwinel/DOMListenerExtension.

[35] H. Kokila, HTML DOM Navigation. 2016. [Online]. Available: https://chrome.google.com/webstore/detail/html-dom-navigation/eimpgjcahblfpdgiknmbmglcafegimil?hl=en.

[36] 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.

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

[38] 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, pp. 359–371, 2013.

[39] T. Tanaka, S. Nomura, H. Niibori, T. Nakao, L. Shiyingxue *et al.*, "Selenium based testing systems for analytical data generation of website user behavior," in *IEEE Int. Conf. on Software Testing, Verification and Validation Workshops ICSTW*, Porto, Portugal, pp. 216–221, 2020.

[40] A. Holmes and M. Kellogg, "Automating functional tests using selenium," in *AGILE 2006*, Washington, USA, pp. 270–275, 2006.

[41] J. D. Musa, "The operational profile," in *Reliability and Maintenance of Complex Systems*, 1st ed., vol. 154. Berlin, Germany: Springer, pp. 333–344, 1996.

[42] R. Suganya and R. Shanthi, "Fuzzy c-means algorithm-A review," *International Journal of Scientific and Research Publications*, vol. 2, no. 11, pp. 1–3, 2012.

[43] O. M. Jafar and R. Sivakumar, "A comparative study of hard and fuzzy data clustering algorithms with cluster validity indices," in *Proc. of Int. Conf. on Emerging Research in Computing, Information, Communication and Applications*, Bangalore, India, pp. 775–782, 2013.

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

[45] 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.

[46] 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*, vol. 2, pp. 133–149, 2015.

[47] 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*, vol. 8, no. 2, pp. 248–255, 1986.

[48] S. Park and G. Kwon, "Avoidance of state explosion using dependency analysis in model checking control flow model," in *Int. Conf. on Computational Science and its Applications*, Berlin, Germany, pp. 905–911, 2006.

[49] A. Valmari, *The State Explosion Problem, Lectures on Petri Nets I: Basic Models, LNCS*, Vol. 1491, Berlin, Germany: Springer, 1998. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-65306-6_21.

[50] E. M. Clarke, O. Grumberg and A. Peleg, *Model Checking*, Cambridge, MA, USA: The MIT Press, 1999. [Online]. Available: https://mitpress.mit.edu/books/model-checking.

[51] R. Zhao, C. Chen, W. Wang and J. Guo, "Automatic model completion for Web applications," in *Int. Conf. on Web Engineering*, Cham, Switzerland, pp. 207–227, 2020.

[52] B. McLaughlin, *Head Rush Ajax*, Sebastopol, CA, USA: O'Reilly Media Inc., 2006. [Online]. Available: https://www.oreilly.com/library/view/head-rush-ajax/0596102259/.

[53] M. Pozdeev, myTinyTodo. 2019. [Online]. Available: https://www.mytinytodo.net/.

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

[55] M. R. Lyu, *Handbook of Software Reliability Engineering*, New York, NY, USA: Computing McGrah Hill, 1996. [Online]. Available: http://www.cse.cuhk.edu.hk/~lyu/book/reliability/.

[56] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proc. of the 36th Int. Conf. on Software Engineering*, Hyderabad, India, pp. 435–445, 2014.

[57] P. S. Kochhar, F. Thung and L. O. David. "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems," in *IEEE 22nd Int. Conf. on Software Analysis, Evolution, and Reengineering SANER*, Montreal, Canada, pp. 560–564, 2015.

[58] M. Staats, G. Gay, M. W. Whalen and M. P. E. Heimdahl, "On the danger of coverage directed test case generation," in *Int. Conf. on Fundamental Approaches to Software Engineering*, Berlin, Germany, pp. 409–424, 2012.

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

[60] W. E. Wong, J. R. Horgan, S. London and A. Mathur, "Effect of test set size and block coverage on the fault detection effectiveness," in *Proc. of 1994 IEEE Int. Symp. on Software Reliability Engineering*, Monterey, CA, USA, pp. 230–238, 1994.

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

[62] W. E. Howden, "Reliability of the path analysis testing strategy," *IEEE Transactions on Software Engineering*, vol. 3, pp. 208–215, 1976.

[63] F. Grigorjev, N. Lascano and J. L. Staude, "A fault seeding experience," in *Simposio Argentino de Ingenieria de Software ASSE*, Beunasiris, Argentina, pp. 1–14, 2003.

[64] 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.

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

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