Automated web testing over the last decade: A systematic literature review

Nafis Fuad Mridha*, Mahbubul Alam Joarder

1Institute of Information Technology, University of Dhaka, Dhaka, Bangladesh

*Corresponding Author email: bsse0920@iit.du.ac.bd

ABSTRACT

Web applications play a vital role in every section ranging from business to administration, making operations efficient. Reliability must be ensured to avoid loss of credibility and revenue. Testing modern web applications is very tedious due to their dynamic and asynchronous nature. Web applications evolve rapidly to serve the growing needs of the customers and manual testing is often infeasible to maintain the reliability and validity of the system due to time and resource constraints. Automated web application testing is an ongoing popular research domain due to the complexity imposed by web applications. This paper conducts a systematic literature review on the practices of automated web testing over the past decade. PRISMA framework is used to screen the most relevant research papers using Scopus and Web of Science as the database. A total of 26 papers were selected based on the inclusion criteria and quality assessment. The articles were fetched and their contributions are documented to know about current practices and the effectiveness of such approaches in minimizing human effort. Results indicate broadly two categories of automated testing, one depending on the tester’s expertise to generate test cases, and the other relying on crawlers to exploit the system dynamically.

Keywords: Automated Web Testing; Systematic Literature Review; Black Box Testing

1. INTRODUCTION

Web applications are one of the fastest-growing types of software and their impact is immense in business, education, banking, administration, etc. The major reasons for this popularity include – i) no installation cost, ii) automatic updates and access via browser, iii) reducing maintenance cost and reduced hardware dependency, and iv) universal access via internet. The reliability and stability of such systems are maintained through proper testing. However, manual testing of modern rapidly evolving systems is expensive and often not feasible due to time constraints. Testers rely on automation scripts that automate the manual browser interaction. This makes the execution of repetitive tasks easy but testers need to modify these test scripts as the systems evolve. On the other hand, automated testing approaches are used to aid testers to generate a test suite to validate the system. Automated testing is a popular research problem due to the dynamic and asynchronous nature of modern web applications. The primary challenge is to capture domain knowledge of applications and mimic the tester’s expertise to verify the application functionality.
The current representation of a web page is known as its state. Each state contains actions such as inputs, buttons, links, etc., that users interact with to achieve certain functionality. The navigation model (a state flow graph) defines the actions responsible for state transition based on application business logic. Web applications contain multiple pages representing similar functionality but different content, which makes testing all functionalities challenging in automated testing. The quality of testing models depends on the state abstraction function to avoid the exponentially increasing number of states. Through the current practice of using AJAX in web applications, the navigation model can no longer be statically defined. Actions may be available only after the completion of a certain business process. Such actions can expose states not present initially, for example, delete functionality of data models are available only after successful creation.

A systematic literature review is carried out in this paper to discover the current trends in the automated web application testing domain. The literature review was conducted based on the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) framework over the timespan of the last decade (2012-2021). Primary studies were selected that focused on black-box testing techniques for web applications to detect failures and improve code coverage through testing. Scopus and Web of Science were selected as the database to conduct the search process. The identified literature was manually inspected to find papers aligning with the goal of this study. Initially, the title, abstract, and keywords were used to compute relevance. Excel was used for data processing and filtering literature based on PRISMA guidelines. The contributions and brief methodology of the papers are reported in this paper.

Each test generation model focuses on two areas - i) action sequence selection, and ii) input generation. Tests that focus on non-overlapping functionality are prioritized to efficiently explore the state space of the system. In most cases, inputs are manually crafted based on business requirements to exploit the maximum number of failures. Automated testing can be broadly categorized into model-based and model-free approaches. This classification is based on the usage of manually defined static models (such as navigation model, UML diagram, requirement specification, domain-specific language, etc.) that represent the application behavior. In model-based automated approaches, the tester manually designs the mentioned models to incorporate the system business logic or expertise required for successful interaction.

2. METHODOLOGY

A Systematic Literature Review (SLR) on automated web application testing is done in this paper to know about the advancements in this field. SLR ensures transparency in literature review and removes bias due to the researcher’s experience or fame. The entire process can be divided into three major sections, namely - i) using an appropriate combination of keywords to query the database, ii) screening suitable literature based on concrete predefined criteria, and iii) analysing the contributions and limitations of selected literature. As web technology evolves disruptively, this literature review aims to encapsulate the research done in automated testing over the last decade (2012-2021). The PRISMA
framework was used to guide the review process in an established systematic manner. The overall process of literature selection is shown in Fig. 1.

2.1. **Research Questions**

Prior to a systematic literature review, broad research questions should be formulated that must be answered as the outcome of the review. Moreover, the query keywords are also derived from the terms used in formulating the research questions. The research questions that are used to guide this literature review are defined below:

RQ1: What are the strategies adopted in automated web application testing?

RQ2: How effective is the automated testing of web applications to reduce manual effort?

2.2. **Search Strategy**

The initial list of literature is obtained by adopting a search strategy in SLR. This phase is responsible for selecting broad keywords to find relevant articles but also restrictive those that are out of scope. Various combinations of the broad keywords “Automated testing” and “Web Application” are used so that no relevant articles are left out due to slight differences in keyword positioning. The query string is designed in a generic way such that all databases support such operations. The Scopus and Web of Science database is used as they provide completeness of search in the field of engineering. The same query string, ("Automated testing" AND "Web") OR ("Web Automated Testing") OR ("Automated Web Testing") OR (Automated AND (Testing OR Test) AND ("Web Applications" OR "Web Application")) was used in both databases to find relevant literature till 18th April 2022. The query is conducted over the title, abstract, and keywords in the databases. Based on the query parameters, limiting the search results from 2012 to 2021 resulted in 910 records (518 from Scopus and 392 from Web of Science). The records were exported in excel format for further processing in PRISMA.

2.3. **Selection Criteria**

The screening process is governed by some inclusion and exclusion criteria designed during initial planning. These criteria are developed based on the objective of the literature review. The inclusion criteria define the conditions that a research paper must satisfy to be included in the review. On the other hand, the exclusion criteria define the conditions for which a research paper is filtered out. The list of criteria considered for this literature review is mentioned below:

**Inclusion criteria:**

- Study area: studies related to automated web application testing
- Publication status: final articles published through a peer-review process
- Study type: all testing approaches defined in primary sources were considered, that is, literature reviews, comparison papers, books, etc. were neglected.
• Publication year: approaches developed in the last decade (2012-2021) were only considered. The ongoing year 2022 was not considered as there is a high risk of non-reproducible results based on the time query is made.

Exclusion criteria:
• Language: studies not published in the English language.
• Uniqueness: duplicates between and within the two database queries were removed.
• Availability: full text of research papers unavailable through DOI were excluded.

2.4. Quality Assessment
The records obtained from the two databases are merged and filtered in excel based on the inclusion criteria mentioned. After evaluating the titles, abstracts, and keywords of the records, 508 records were removed being out of scope, non-relevant, too technology-specific (for example, generating tests based on java code), or requiring source code (web application source code is rarely available). Full texts of the remaining 82 papers were
fetched to determine whether they are relevant to the literature review goal. The quality assessment of relevant documents was done based on the following factors:

- Web application testing, not visual testing or test case prioritization technique.
- Approaches considering complete testing process and not improving subsection involved in testing (such as state abstraction algorithm).
- Black box testing approaches, that is, testing approaches that do not require the availability of source code.
- Papers focusing on testing web security, vulnerability, penetration, performance, accessibility, test case reusability, and fault classification are neglected in this study.

### 2.5. Data Extraction and Reporting

After the selection process using PRISMA, a total of 26 papers were fetched and read. The summary of the contribution, methodology, and evaluation of each selected paper was recorded in excel. Finally, the author information, publication year, abstract, and contribution summary were stored in excel for reporting. The number of publications over the time span of this review is shown in Fig. 2.

![Publications over the last decade](image)

**Fig. 2.** Publications over the decade (2012-2021)

### 3. Results and Discussion

The finding obtained from the selected literature is described in this section. The contributions and brief methodology of each of the papers are presented in a tabular form below:
<table>
<thead>
<tr>
<th>Title</th>
<th>Author and Year</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic web testing using curiosity-driven reinforcement learning (Zheng et al., 2021)</td>
<td>Zheng et al. (2021)</td>
<td>WebExplor leverages model-free reinforcement learning based automated web application testing approach. A novel state abstraction algorithm is proposed that detect similar states in the system to avoid the state explosion problem. Two factors guide the Q-learning based exploration process - i) the curiosity based reward function, and ii) the gumbel-softmax function for the action selection policy. High-level DFA guidance is employed to visit the hard-to-reach areas of the system under test. The DFA activates after a defined interval if no new states are found within that period. WebExplor shows state-of-the-art results in terms of failure detection in benchmarked projects. The authors also conducted study on industrial projects and top websites based on alexa ranking.</td>
</tr>
<tr>
<td>Test Script Generation Based on Hidden Markov Models Learning From User Browsing Behaviors (Erdem et al., 2021)</td>
<td>Erdem et al. (2021)</td>
<td>This approach records human interaction with the system and generated user session by grouping URL's based on topic. The topics are manually labeled and the URL sequences are grouped based on regex starting from the root URL. This pattern of interaction is modeled as a markov model. The application under test is testing based on user navigation patterns. The authors evaluate the approach based on Bleu score to detect the similarity between generated sequences and actual user sessions. However, this work does not emphasize on failure detection.</td>
</tr>
<tr>
<td>Research and Application of Software Automated Testing Based on Directed Graph (Hu &amp; Huang, 2021)</td>
<td>Hu &amp; Huang, (2021)</td>
<td>The approach designs a graph based on adjacency list to represent the navigation model of the system under test. Depth first traversal is user to generate the test paths. The authors propose that test paths should be kept separate from test scripts such that change in the HTML elements do not affect the validity of test cases. A predefined mapping is used to determine the type of interaction for page elements. The test inputs are manually designed to ensure successful form submissions. Depth first traversal ensures diverse test path generation, reducing loss of time due to redundancy. Test code reuse is also possible due to employing Page Objects for page behavior abstraction.</td>
</tr>
<tr>
<td>Integrated Pairwise Testing based Genetic Algorithm for Test Optimization (Swathi &amp; Tiwari, 2021)</td>
<td>Swathi &amp; Tiwari (2021)</td>
<td>The proposed approach defines pairwise testing of web applications where the system navigation graph is provided to generate test action sequences. Different pairs of paths (pairs) are selected from the navigation model and their actions are combined in multiple permutations based on a genetic algorithm. Such sequences help to explore usage scenarios that the tester might miss and interaction patterns that are unlikely to be tested during manual exploration.</td>
</tr>
<tr>
<td>Codeless web testing using selenium and machine learning (Nguyen &amp; Maag, 2020)</td>
<td>Nguyen &amp; Maag, (2020)</td>
<td>A machine learning based approach is proposed that can detect web elements and perform appropriate testing operations. This work only focus on detecting search bars in web pages using support vector machine and perform testing on input fields to test the searching functionality. Cross-browser experimentation is done on popular websites to test the detection process.</td>
</tr>
<tr>
<td>Automating Test Automation (Thummalapenta et al., 2012)</td>
<td>Thummalapenta et al., 2012</td>
<td>This approach proposes a domain-specific language to automatically generate test cases that incorporate the tester's</td>
</tr>
<tr>
<td>Title</td>
<td>Author and Year</td>
<td>Contribution</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>(Thummalapenta et al., 2012)</td>
<td></td>
<td>domain knowledge. The test case generation is done based on predefined keywords that are used to define the type of interaction. A manually defined pool of inputs are used to process web forms. This approach is evaluated in terms of accuracy to generate valid test cases and reuse of generated code.</td>
</tr>
<tr>
<td>Leveraging keyword-guided exploration to build test models for web applications (Qi et al., 2019)</td>
<td>Qi et al. (2019)</td>
<td>Presents a keyword-guided exploration strategy to generate test models for web applications. To test the system efficiently within a time budget, keywords and page contents are used to group pages with similar functionality. A prototype KeyjaxTest is implemented to evaluate the testing approach in terms of statement coverage, path diversity, DOM diversity and size of test suite.</td>
</tr>
<tr>
<td>Automated Model-Based Test Case Generation for Web User Interfaces (WUI) From Interaction Flow Modeling Language (IFML) Models (Yousaf et al., 2019)</td>
<td>Yousaf et al. (2019)</td>
<td>Unified Modeling Language (UML) and Interaction Flow Modeling Language (IFML) are used to represent the domain knowledge and front-end web application behavior. A tool named MBUITC is devised implementing four rules to generate test cases from manually defined UML and IFML.</td>
</tr>
<tr>
<td>Ontology based test case generation for black box testing (UI Haq &amp; Qamar, 2019)</td>
<td>UI Haq &amp; Qamar (2019)</td>
<td>An approach to automatically generate test cases is proposed using manually defined ontology (formerly used in all phases in software engineering). Preconditions and postconditions of a test case are defined and a constraint solver is used to generate executable tests. Attempts to bridge the gap between manual intervention and machine driven testing. A classification model is trained to recognize web components using common abstractions in web applications. A grammar is used define hand-crafted abstract test flows including variable definitions (with values to be used in concrete tests) representing the application diverse scenarios. An LSTM is trained to map test sequences until valid test flows are generated.</td>
</tr>
<tr>
<td>Abstract flow learning for web application test generation (Santiago et al., 2018)</td>
<td>Santiago et al. (2018)</td>
<td>A tool named CRATCP, given the initial root URL of the system under test, extracts all interactable elements in the website. The validation rules associated with these elements are extracted from scripts. All possible test cases are generated from possible combinations of the obtained actions. These tests are executed automatically using randomized boundary values (obtained from the validation rules). Generates test scripts with the requirement specification as prerequisite. The textual use case diagram, textual requirement specification and HTML UI mockup are provided to generate test cases automatically. The approach is not evaluated against other approaches.</td>
</tr>
<tr>
<td>Towards the generation of end-to-end web test scripts from requirements specifications (Clerissi et al., 2017)</td>
<td>Clerissi et al. (2017)</td>
<td>Test case generation from BPMN models for automated testing of Web-based BPM applications (de Moura et al., 2017)</td>
</tr>
<tr>
<td>Automated Testing of Web Applications Using</td>
<td>Qi et al. (2017)</td>
<td>A tool ComjaxTest is proposed that employs Paiv Wise testing (pair of actions are considered as variable) approach to handle</td>
</tr>
<tr>
<td>Title</td>
<td>Author and Year</td>
<td>Contribution</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Combinatorial Strategies (Qi et al., 2017)</td>
<td></td>
<td>the complex scenario of proper submission of web forms. Constraints are defined within pair wise attribute selection using a custom language called FormDataSpec (for example, password and confirm password must have the same value). The state flow graph is generated through crawling the application and forms are tested based on pairwise value selection aligning with constraints if defined.</td>
</tr>
<tr>
<td>WebMate: Generating Test Cases for Web 2.0 (Dallmeier et al., 2013)</td>
<td></td>
<td>This paper describes the challenges in automated web2.0 testing due to its dynamic and asynchronous nature and proposed the tool WebMate. WebMate explores the system under test to generate the state flow graph and employs state abstraction to detect pages with similar functionality. Dijkstra algorithm is used to find the sequence of actions required to test a particular page. Form inputs are manually crafted to execute tests generated via traversing the generated state flow graph.</td>
</tr>
<tr>
<td>Automated testing of web applications with text input (Wu et al., 2015)</td>
<td>Wu et al. (2015)</td>
<td>An extension of Crawler focusing on remembering user profile related inputs. Keeping exploration similar to Crawler, this approach assists the input of user profile data, for example, using the same user information in login after registration.</td>
</tr>
<tr>
<td>A semantic approach for automated test oracle generation (Guo, 2016)</td>
<td>Guo, (2016)</td>
<td>The proposed approach TAO applies denotational semantics in test and oracle generation for various application domains. The initial context free grammar (CFG) and the required number of test cases are given as inputs. The CFG is converted to test cases through parsing the semantic tree and generating a sequence of actions.</td>
</tr>
<tr>
<td>Incremental web application testing using page object (Yu et al., 2015)</td>
<td>Yu et al. (2015)</td>
<td>This work first proposed the use of Page Objects in automated test scripts in modern web applications, promoting more exploration due to functionality abstraction. The initial page objects are automatically generated, representing the home page and methods described indicate separate actions within that page. Random values are used as test input to discover new pages based on feedback.</td>
</tr>
<tr>
<td>Automated testing of web applications with TESTAR: Lessons learned testing the odoo tool (Almenar et al., 2016)</td>
<td>Almenar et al. (2016)</td>
<td>TESTAR is designed to explore the deeped part of a web application, focusing on the action selection policy. Experimentation revealed that employing Q-learning over random based action selection improve the performance. Q-learning is adopted by giving maximum weight to unexplored actions and decreasing the weight upon being executed.</td>
</tr>
<tr>
<td>End to end specification based test generation of web applications (Bubna, 2016)</td>
<td>Bubna, (2016)</td>
<td>Derive automated tests given UML state diagram as input. The state diagram is parsed and symbolic execution is used to derive possible combination of paths. Abstract test cases are generated by manually providing inputs where needed. Finally the abstract test cases are made concrete by replacing symbolic variable with actual ones. This approach makes test abstraction possible even if the underlying feature implementation change. Employing symbolic execution based on state diagram reduces redundancy is exploration.</td>
</tr>
<tr>
<td>Keyword driven automated testing framework for web application (Arya &amp; Verma, 2014)</td>
<td>Arya &amp; Verma (2014)</td>
<td>This approach reduces the testers effort to write test cases, instead uses a list of keywords mapped to specific implementation to generate test cases. Presenting a non technical way of generating test cases. Testers need to describe</td>
</tr>
<tr>
<td>Title</td>
<td>Author and Year</td>
<td>Contribution</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Verifying web applications: From business level specifications to automated model-based testing (Colombo et al., 2014)</td>
<td>Colombo et al. (2014)</td>
<td>Incorporate human domain knowledge from manually written selenium test scripts in a tool named Testilizer. The devised algorithm mines existing test cases to infer a model that includes (1) input data, (2) event sequences, (3) and assertions, and feeding and expanding that model through automated crawling. After generating the SFG, unexplored areas (located near the manual ones) are given focus for test case generation (action sequence + assertion). The tool CrawlScripter is an extension of Crawljax that describe a set of high level instructions to write automated acceptance tests. The instructions can be written by testers without any programming knowledge as they are close to natural language. The crawler on parsing the instructions, execute actions in that matter if found in the web application. Generates test cases automatically from finite state model (FSM). The FSM is automatically derived from manually defined UML state diagram using ArgoUML. The main contribution of this paper is representing web application state diagrams as an FSM and automatically generating test cases that prioritize diverse system behaviors. ATUSA aims to make automated web testing of AJAX based systems more reliable. Client-side invariants are used to validate the functionality of the system after performing actions in a page. Challenges imposed by AJAX based applications (for example, browser history not registered on dynamic DOM update) are tackled in this approach.</td>
</tr>
<tr>
<td>Leveraging existing tests in automated test generation for web applications (Fard et al., 2014)</td>
<td>Fard et al. (2014)</td>
<td>A domain specific language Cherkir is used to imitate the tester’s domain knowledge. The pre conditions and post conditions are defined in the test specification. The paper presented some keyword driven rules of writing the specification so that actions can be automatically derived from the system under test.</td>
</tr>
<tr>
<td>Automated acceptance testing of JavaScript web applications (Negara &amp; Stroulia, 2012)</td>
<td>Negara &amp; Stroulia (2012)</td>
<td>The tool CrawlScripter is an extension of Crawljax that describe a set of high level instructions to write automated acceptance tests. The instructions can be written by testers without any programming knowledge as they are close to natural language. The crawler on parsing the instructions, execute actions in that matter if found in the web application. Generates test cases automatically from finite state model (FSM). The FSM is automatically derived from manually defined UML state diagram using ArgoUML. The main contribution of this paper is representing web application state diagrams as an FSM and automatically generating test cases that prioritize diverse system behaviors. ATUSA aims to make automated web testing of AJAX based systems more reliable. Client-side invariants are used to validate the functionality of the system after performing actions in a page. Challenges imposed by AJAX based applications (for example, browser history not registered on dynamic DOM update) are tackled in this approach.</td>
</tr>
<tr>
<td>AGT4W: Automatic generating tests for web applications (Chen et al., 2012)</td>
<td>Chen et al. (2012)</td>
<td></td>
</tr>
<tr>
<td>Invariant-based automatic testing of modern web applications (Mesbah et al., 2011)</td>
<td>Mesban et al. (2011)</td>
<td></td>
</tr>
</tbody>
</table>

### 3.1. Evaluation

RQ1: What are the strategies adopted in automated web application testing?

Based on the prerequisite required to initiate automated testing, the approaches proposed in the literature can be broadly categorized into model-based and model-free approaches. The model-based techniques rely on a manually defined representation of the application under test, most commonly the navigation model of the application is used. Besides navigation models, UML diagrams are also a popular source of application domain knowledge. Moreover, the use of specialized grammar and keyword definitions which are closer to natural language help to incorporate the tester’s expertise in generating a test suite automatically. On the other hand, model-free techniques depend on a crawler that updates the navigation model and performs actions encountered in the current state. The approaches considering random or pseudorandom interaction of actions are also termed random-based approaches. Work has also been done to adopt reinforcement learning
algorithms such as Q-learning for action selection and prioritizing unexplored actions. Model-free approaches successfully consider the dynamic nature of modern web applications but fail to incorporate the tester’s behavior. The test input can be selected from a defined input pool for successful form submission.

RQ2: How effective is the automated testing of web applications to reduce manual effort? Model-based techniques are one of the most prominent types in literature that require human-level intervention to model the web application or guide the test steps for automatic script generation. The adoption and success of such approaches depend heavily on the tester’s expertise. Therefore, these approaches optimize the testing process through test code generation and abstracting the dependency with actual code via an intermediate representation of the system. Popular practices include manually designing the navigation model, UML, business process and test step definition via specialized language, etc. On the other hand, model-free approaches crawl the dynamic states without human intervention (in some cases intervention is required for test input). The test suite generated relies on state abstraction functionality and efficient action selection for code coverage. Model-free approaches work best when resources are very limited.

4. LIMITATION

This study uses two databases—Scopus and Web of Science to query for relevant literature. Although these two databases are the most used ones in engineering work, some unique records might have been missed. The query keywords were designed without using too specific technology or specialization in automated testing to get maximum relevant results. Some papers may have been missed due to not including them in the title, abstract, or keywords section. The SLR process is designed in such a way that papers are visible only if they align with the keywords to make the process reproducible. Moreover, research on automated web application testing is in the spotlight of researches for a long time and new approaches are proposed continuously, making the tracking process difficult.

5. CONCLUSION AND FUTURE WORK

This paper presents a systematic literature review on the practices of automated web application testing using the PRISMA framework. The framework yielded 26 relevant papers over the last decade (2012-2021). The papers were extracted and read to summarize the contributions and point out the current practices in testing automation. The evaluation exposed the current practices as well as the effectiveness of such automation in terms of saving human labor. The limitations of the two categories of automated testing are also discussed, indicating work in this field to bridge the gap. Although the automated approaches make the testing process less expensive and time-consuming, they will be more effective if the tester’s knowledge is incorporated successfully.

Our plans include broadening the search space using more databases and tuning the keywords based on keywords found in relevant papers. Also, secondary studies can also be included to compare the approaches and incorporate other perspectives from grey sources.
Author Contributions:

Mridha Md. Nafis Fuad, conceptualizing the research idea, designing the study, literature review and overseeing the data collection process. Mahbubul Alam Joarder, contributed to the data analysis and interpretation of the results.

Funding:

No funding received.

Institutional Review Board Statement:

Not applicable.

Informed Consent Statement:

Not applicable.

Data Availability Statement:

Not applicable.

Acknowledgments:

None.

Conflicts of Interest:

No conflict of interest.

Reference:


