

Search-Based White-Box Fuzzing of Web Frontend Applications

Iva Kertusha

School of Economics, Innovation and Technology

Kristiania University College

Oslo, Norway

Iva.Kertusha@kristiania.no

Abstract—Frontend is the initial point of contact for users with a web application. Therefore, it is crucial to provide the best possible user experience. Testing the frontend for faults and robustness is important to ensure a smooth and engaging interaction with users. We aim to design a novel white-box automated testing approach, using evolutionary algorithms, to generate and optimize test cases for web frontend applications. We have a three-year plan during which we aim to conduct a systematic literature review, develop a frontend testing tool, and publish papers on our work. Our initial research indicates that a few frontend testing tools are already available in the literature. However, most of these tools are based on a black-box approach, and as such, cannot test an application thoroughly, reducing the possibility of detecting faults. To the best of our knowledge, no current technique or tool is based on a white-box approach.

Index Terms—frontend, whitebox, fuzzing, evolutionary algorithms, web application

I. INTRODUCTION

The evolution of web applications, from a set of static web pages to a set of interactive dynamic pages, has created a clear distinction between frontend and backend development. The frontend, also known as the client-side, focuses on what users interact with directly: the layout, design, and usability of a website or application. It is responsible for presenting content in an engaging and user-friendly manner. It utilizes technologies like HTML, CSS, and JavaScript, ensuring navigation, responsiveness across devices, and accessibility for all users [24].

Currently, in the academic literature there are a few available libraries and frameworks that enable testers to automate the *execution* of the test cases (e.g., Puppeteer [2], Playwright [1]). In addition to commercial or open-source tools, the academic literature presents several other tools focused on the *automation of test case generation* for the web frontend (e.g., Crawljax [27], QExplore [30]). These tools typically implement a black-box approach, with limited knowledge regarding the structure of the code. Although black-box testing can identify faults by providing inputs to the system and comparing outputs received with expected outputs, it may have limited code coverage. A white-box approach, on the other hand, because of its in-depth understanding of the code, can

access and examine code portions that may not be reachable through black-box methods, thereby improving code coverage, which can lead to higher chances of detecting faults.

The goal of this PhD is to first identify limitations of existing solutions related to automated testing of web frontend, and then fill the gap in the literature by designing novel techniques to tackle the limitations of the existing ones. These novel techniques will be based on white-box approaches, that will improve upon the existing state-of-the-art, which is limited to black-box approaches. We aim to utilize Search-Based Software Testing (SBST) methods [23] [26] to attempt to find the most efficient test cases. We will be using SBST because of its ability to automate and optimize the generation of comprehensive and effective test cases, while significantly reducing manual effort and associated costs. In addition to designing and implementing the technique mentioned above, our work also includes a Systematic Literature Review (SLR). A thorough SLR will help us establish a foundation for our future research. Once we have a full picture and a better understanding of the state-of-the-art of frontend testing, we will proceed with designing, implementing, and conducting experiments to evaluate our novel approaches.

II. BACKGROUND

This section first provides brief background information on web frontend, JavaScript, white-box testing, and SBST. After that, it gives a brief summary of the current frontend testing tools.

A. Web Frontend and JavaScript

The three main building blocks of a web user interface are HTML, CSS, and JavaScript. HTML is the standard markup language used to create and structure the content of web pages. CSS is a stylesheet language used to describe the presentation and design of web pages, including layout, colors, and fonts. JavaScript is a programming language that enables interactive and dynamic elements on web pages, such as animations, form validation, and user interactions. Since its introduction in 1995 [36], JavaScript has risen in popularity being used in more than 98% of all websites and web applications used in the world [12]. Based on an annual report from GitHub, JavaScript remains the most popular language of 2023 [6]. Other than JavaScript itself as a scripting language, there are

This project is a part of EAST, which is a larger project funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EAST project, grant agreement No. 864972).

also JavaScript-based libraries and frameworks (e.g., jQuery¹, AngularJS², Vue.js³, React⁴) that have gained popularity in recent years [4]. JavaScript has several characteristics that make it attractive for developers, including dynamic typing (i.e., variable types are assigned during runtime, not before), supporting event-driven programming (i.e., program flow depends on external events), handling events asynchronously (i.e., the program being able to respond to other events while running a long task), supporting first-class functions (i.e., functions being treated like other values), and being a weakly typed language (i.e., operations can be performed between different data types without explicitly converting them). However, even though such characteristics facilitate the development process, they make finding faults in Javascript-based applications more challenging.

B. Frontend Testing and the White-Box Testing Approach

Testing is an integral part of the software development cycle, ensuring that applications meet quality standards, functionality requirements, and user expectations [19]. Frontend testing specifically focuses on evaluating the user interface and experience of a web application, ensuring its responsiveness, usability, and compatibility across various browsers and devices [18]. By integrating frontend testing into the software development cycle, developers can detect and mitigate issues related to frontend early, leading to improved reliability, performance, and user satisfaction [31].

White-box testing focuses on examining the internal logic and structure of the code. This approach involves inspecting the codebase, including functions, variables, and control flow, to identify potential errors and ensure comprehensive test coverage (e.g., the number of lines executed) [22].

In the literature, there are a few tools available for frontend testing, that are discussed in more detail in Section II-D. However, to the best of our knowledge, most of those tools utilize black-box techniques. The preliminary list of the papers we have gathered for the SLR seems to back up this statement. We aim to tackle such a gap by developing a white-box testing tool for frontend testing, that generates more effective test cases (i.e., in terms of code coverage and fault finding) compared to existing frontend testing tools. This might pose a few challenges due to the nature of white-box testing that requires analyzing the source code.

C. Search-Based Software Testing

Evolutionary algorithms (EAs) are computational methods inspired by the principles of biological evolution, aiming to solve complex optimization problems [17] [32]. One important usage of EAs is finding optimal solutions in large search spaces. SBST involves using optimization search techniques to address automation challenges in software testing [26]. By leveraging the strengths of EAs, SBST can effectively explore the test space, optimize test case generation, and enhance the overall quality and efficiency of software testing. SBST

has been successfully implemented in many testing contexts, such as EvoSuite [21], EvoMaster [15], TestFul [16], Kex and Kex-reflection [13]. Due to the complexity of the problem at hand, EAs seem to be good candidates for finding the optimal solutions (i.e., the optimal set of inputs for testing). In this research, we are going to utilize different evolutionary algorithms (e.g., MIO [14]) for frontend testing, to identify the best algorithm for this problem.

D. Current frontend testing tools and frameworks

Currently, several frontend testing tools and frameworks allow developers and testers to write test code and ensure the quality of their web applications. Selenium [10] is the most known tool in frontend testing, providing cross-browser compatibility testing and robust automation features [33]. To quote the Selenium documentation itself “*Selenium automates browsers. That’s it*”.⁵ Other libraries/frameworks used for testing include Puppeteer⁶, Cypress⁷, Playwright⁸, TestCafe⁹, and Testim¹⁰. It is important to note that these libraries and tools are primarily used to *automate the execution* of test cases, which are still manually developed by testers. Consequently, the role of a quality assurance tester or engineer remains crucial. They are the ones responsible for writing test scripts and overseeing the entire process.

There exists a limited number of tools within academic literature on automatic test generation, and most of them are operating on the black-box principle. We have identified a few of these tools in our initial research. Earlier tools implemented to test web GUI include ATUSA [28] (2009), AutoBlackTest [25] (2012), and GUITAR [29] (2014). According to GitHub, none of these tools is actively maintained. In addition to open-source tools, Webmate [20] which originally started as an academic tool was converted into a commercial one.

Several other tools are currently available in the literature. Mesbah et al. introduced in 2008 Crawljax [27], a black-box testing tool developed in Java. QExplore is a tool proposed in [30] that leverages the Q-learning concept to prioritize certain actions during the testing process. TESTAR [34], is another black-box system-level testing tool that can be used to generate test cases for web apps, mobile apps, and desktop apps. WebExplor [37] performs end-to-end testing using Reinforcement Learning principles to explore the web application and generate action sequences. The source codes for Crawljax, QExplore, and TESTAR are available on GitHub, whereas the code for WebExplor is not publicly available.

In this research, we are going to extend an existing test generation tool, EvoMaster [15], for frontend testing. EvoMaster, primarily used for backend testing, already implements several SBST heuristics that can be reused and adapted to deal with web frontend testing. Furthermore, the content that the user interacts with in the browser depends on the backend (state of the database, manipulation of data from backend functions, etc). Consequently, integrating our proposed method

¹ <https://jquery.com/>

² <https://angularjs.org/>

³ <https://vuejs.org/>

⁴ <https://react.dev/>

⁵ <https://www.selenium.dev/>

⁶ <https://pptr.dev/>

⁷ <https://www.cypress.io/>

⁸ <https://playwright.dev/>

⁹ <https://testcafe.io/>

¹⁰ <https://www.testim.io/>

in EvoMaster would result in a tool that is able to perform exhaustive (i.e., of both frontend and backend) testing, of a given web application.

III. PROPOSED APPROACH

We aspire to design a novel white-box automated approach, using evolutionary algorithms, to generate and optimize test cases for web frontend GUI applications. To be able to achieve this, we have formulated three main objectives to accomplish throughout the duration of this PhD.

- Objective A - Systematic Literature Review
- Objective B - SBST with a focus on backend information
- Objective C - SBST with a focus on frontend information

Our methodology begins with a thorough review of the literature. Next, we will work on enabling SBST of a web app with the focus on the information received from the backend. The last and conclusive step would be enabling SBST on frontend. We will perform *experiments* to evaluate the effectiveness of our approach and analyze the data we will collect from these experiments. We will compare the performance of our tool against existing tools discussed in the literature, using as case studies web applications that will be described in Section III-C. By accomplishing the three objectives mentioned above, we will gain a comprehensive understanding of the current state of frontend testing documented in the literature. Additionally, we will develop a testing tool capable of thoroughly evaluating interface quality and verifying information received from the backend. Each objective will be discussed below in further detail.

A. Systematic Literature Review

A SLR will give us a better picture of the current methods and techniques available, and what is explored in academia regarding web frontend testing, and web testing in general. We have already started working on the Systematic Literature Review and successfully finished the first phase of selecting and compiling the list of relevant papers. The papers are collected from different databases: IEEEExplore, ACM, ScienceDirect, Wiley, WebOfScience, MIT, and Springer. The queries were initially formulated to prioritize articles on web frontend testing. The initial results amount to around 600 paper. After getting initial results, articles related to testing mobile (e.g., Android and iOS) and desktop applications are discarded. Any articles not related to web frontend testing was discarded too. The resulting list counts around 160 papers. The next step is performing forward and backward snowballing and compiling the final list of papers. Once this step is completed, we will proceed with extracting data from the papers to answer our research questions. We have formulated 15 initial research questions intended to illustrate the current state of the literature on web frontend testing. However, these questions are prone to modifications as our work on the SLR continues.

B. Automated White-Box Testing for Web Frontend

Our goal in this step is to develop a prototype to generate test cases for web frontend applications in an automated

manner and then integrate that prototype into a large-scale tool, namely EvoMaster. To achieve this goal, we established two objectives: Objective B and Objective C, as mentioned earlier. Technically, we could have started by focusing on white-box heuristics in the code running on the browser (Objective C), and then focus in tandem on the code running in the backend (Objective B). However, we have decided to proceed with Objective B, and then C, for two simple reasons. First, traditional web applications did not run JavaScript directly on the browser, but they still had code running on the backend. Hence, we will be addressing this scenario first. Secondly, EvoMaster already supports several SBST heuristics for the backend, therefore it is easier to bootstrap work on Objective B compared to Objective C.

The first step in Objective B is to analyze the code execution in the backend via bytecode instrumentation, by reusing SBST techniques. One of the main responsibilities of the backend is processing requests from the frontend, applying rules and computations, and then returning the appropriate responses. Hence, the information displayed to the user heavily depends on what is being executed, and returned by the backend. The response generated by the backend varies, from a simple JSON object to a complete HTML page (in some earlier web development techniques, the server response would be the web page itself, rather than structured data like JSON or XML). By testing the backend, we could uncover faults and errors that otherwise might not have been possible to detect by testing only the frontend.

The next step, Objective C, consists of implementing the novel idea we are proposing. We will develop an SBST algorithm and white-box approach to automatically generate test cases for web frontend testing. At this point, the backend is meticulously tested and we can focus on testing the code running in the frontend as well (i.e., JavaScript) in tandem with the code running in the backend (e.g., Java). The main advantage of the technique we are proposing relies on the combination of SBST and white-box. The white-box approach, having an understanding of the code and the implementation of the application under test, ensures a higher degree of testing. Via SBST we would be able to efficiently explore the test space and generate the optimum set of test cases for the application under test. This novel algorithm will be implemented in Java/Kotlin and will be added as an extension to an existing tool, EvoMaster.

JavaScript and JavaScript-based frameworks play a vital role in developing the frontend of a web application. Testing the accuracy of JavaScript functions is essential, regardless of the framework used, be it Angular, Vue, React, or others. Different frameworks implement different techniques and libraries, therefore the code a developer writes is not the same code you might see in the developer tools on the browser. In order to tackle such a challenge, we aim to develop a framework-independent testing tool, and for this purpose we are going to analyze how JavaScript is rendered in browsers.

C. Case studies and Evaluation

In [35], the authors propose a framework to evaluate different testing methods and tools. Among other elements, this framework defines a list of metrics (e.g., number of faults found, coverage reached) to measure the effectiveness, efficiency, and applicability of a tool or method. In our work, we will follow this framework to design, conduct, and evaluate our experiments in a standardized manner. For our experiments, we need a few open-source applications as case studies. We will use some case studies that are already mentioned in the literature, and a few others that might have not been tested before. Nowadays there are millions of applications on the internet that can be used as a case study, a good portion of them available in open-source repositories (e.g., GitHub). The case studies we have initially selected are NAV [5], Twitter [11], Bimrocket [3], Strapi [9], ParaBank demo [7], and Shopizer [8]. However, this initial set might be subject to change in the later stages. To conduct our experiments we will test the selected case studies with the tool we developed and the tools outlined in Section II-D. The performance of each tool will be assessed based on a set of metrics as defined in [35]. We will primarily compare the results based on code coverage and the number of detected faults.

IV. CURRENT STATUS

We have started working on this project in March 2024. At the moment we are working on the SLR. We have collected around 600 papers from all major databases, performed forward and backward snowballing, and now we are extracting data from the papers to answer our Research Questions. We have set a 3-year plan, during which we aim to publish one SLR, and at least three more papers, including a tool paper. The expected date for thesis defense is Spring 2027.

REFERENCES

- [1] Playwright Docs. <https://playwright.dev/docs/intro>. [Online; accessed 18-June-2024].
- [2] What is Puppeteer. <https://pptr.dev/guides/what-is-puppeteer>. [Online; accessed 18-June-2024].
- [3] Bimrocket. <https://github.com/bimrocket/bimrocket>. [Online; accessed 25-May-2024].
- [4] Most used web frameworks among developers worldwide, as of 2023. <https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web>. [Online; accessed 25-May-2024].
- [5] NAV. <https://www.nav.no/en/>. [Online; accessed 25-May-2024].
- [6] Octoverse: The state of open source and rise of AI in 2023. <https://github.blog/2023-11-08-the-state-of-open-source-and-ai/>. [Online; accessed 25-May-2024].
- [7] ParaBank Demo from Parasoft. <https://github.com/parasoft/parabank/>. [Online; accessed 12-May-2024].
- [8] Shopizer- Java open source e-commerce software. <https://github.com/shopizer-ecommerce/shopizer>. [Online; accessed 12-May-2024].
- [9] Strapi.io. <https://github.com/strapi/strapi>. [Online; accessed 25-May-2024].
- [10] The Selenium Browser Automation Project. <https://www.selenium.dev/documentation/>. [Online; accessed 2-May-2024].
- [11] Twitter Opensource Website. <https://github.com/twitter/opensource-website>. [Online; accessed 25-May-2024].
- [12] WebTechnology Surveys. <https://w3techs.com/technologies/details/cp-javascript>. [Online; accessed 27-May-2024].
- [13] ABDULLIN, A., AKHIN, M., AND BELYAEV, M. Kex at the 2022 sbst tool competition. In *2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST)* (2022), pp. 35–36.
- [14] ARCURI, A. Many Independent Objective (MIO) Algorithm for Test Suite Generation. In *International Symposium on Search Based Software Engineering (SSBSE)* (2017), pp. 3–17.
- [15] ARCURI, A., GALEOTTI, J. P., MARCULESCU, B., AND ZHANG, M. Evomaster: A search-based system test generation tool. *Journal of Open Source Software* 6, 57 (2021), 2153.
- [16] BARESI, L., LANZI, P. L., AND MIRAZ, M. Testful: an evolutionary test approach for java. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)* (2010), pp. 185–194.
- [17] BARTZ-BEIELSTEIN, T., BRANKE, J., MEHNEN, J., AND MERSMANN, O. Evolutionary algorithms.
- [18] BOARD, I. S. T. Q. Worldwide software testing practices report, 2018.
- [19] CRAIG, R. D., AND JASKIEL, S. P. *Systematic software testing*. Artech House, 2002.
- [20] DALLMEIER, V., BURGER, M., ORTH, T., AND ZELLER, A. Webmate: a tool for testing web 2.0 applications. In *Proceedings of the Workshop on JavaScript Tools* (2012), pp. 11–15.
- [21] FRASER, G., AND ARCURI, A. EvoSuite: automatic generation for object-oriented software. In *ACM Symposium on the Foundations of Software Engineering (FSE)* (2011), pp. 416–419.
- [22] KHAN, M. E., AND KHAN, F. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications* 3, 6 (2012).
- [23] KHARI, M., AND KUMAR, P. An extensive evaluation of search-based software testing: a review. *Soft Computing* 23 (2019), 1933–1946.
- [24] LINDLEY, C. Frontend developer handbook 2017. *Frontend masters* (2019).
- [25] MARIANI, L., PEZZE, M., RIGANELLI, O., AND SANTORO, M. Autoblacktest: Automatic black-box testing of interactive applications. In *2012 IEEE fifth international conference on software testing, verification and validation* (2012), IEEE, pp. 81–90.
- [26] MCMINN, P. Search-based software testing: Past, present and future. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops* (2011), IEEE, pp. 153–163.
- [27] MESBAH, A., BOZDAG, E., AND VAN DEURSEN, A. Crawling ajax by inferring user interface state changes. In *Web Engineering, 2008. ICWE'08. Eighth International Conference on* (2008), IEEE, pp. 122–134.
- [28] MESBAH, A., AND VAN DEURSEN, A. Invariant-based automatic testing of ajax user interfaces. In *2009 IEEE 31st International Conference on Software Engineering* (2009), IEEE, pp. 210–220.
- [29] NGUYEN, B. N., ROBBINS, B., BANERJEE, I., AND MEMON, A. Guitar: an innovative tool for automated testing of gui-driven software. *Automated software engineering* 21 (2014), 65–105.
- [30] SHERIN, S., MUQEET, A., KHAN, M. U., AND IQBAL, M. Z. Qexplore: An exploration strategy for dynamic web applications using guided search. *Journal of Systems and Software* 195 (2023), 111512.
- [31] SINGH, S. K., AND SINGH, A. *Software testing*. Vandana Publications, 2012.
- [32] SLOWIK, A., AND KWASNICKA, H. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications* 32 (2020), 12363–12379.
- [33] THOORIQOH, H. A., ANNISA, T. N., AND YUHANA, U. L. Selenium framework for web automation testing: A systematic literature review. *Jurnal Ilmiah Teknologi Informasi* 19, 2 (2021), 65–76.
- [34] VOS, T. E., AHO, P., PASTOR RICOS, F., RODRIGUEZ-VALDES, O., AND MULDER, A. testar–scriptless testing through graphical user interface. *Software Testing, Verification and Reliability* 31, 3 (2021), e1771.
- [35] VOS, T. E., MARÍN, B., ESCALONA, M. J., AND MARCHETTO, A. A methodological framework for evaluating software testing techniques and tools. In *2012 12th international conference on quality software* (2012), IEEE, pp. 230–239.
- [36] WIRFS-BROCK, A., AND EICH, B. Javascript: the first 20 years. *Proceedings of the ACM on Programming Languages* 4, HOPL (2020), 1–189.
- [37] ZHENG, Y., LIU, Y., XIE, X., LIU, Y., MA, L., HAO, J., AND LIU, Y. Automatic web testing using curiosity-driven reinforcement learning. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (2021), IEEE, pp. 423–435.