WebEV: A Dataset on the Behavior of Testers for Web Application End to End Testing

Mridha Md. Nafis Fuad
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
bsse0920@iit.du.ac.bd

Kazi Sakib
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
sakib@iit.du.ac.bd

***Abstract*—Automated End-to-End (E2E) web testing is a key component in modern rapid development to validate system functionality. However, there are no resources supporting practitioners on how diverse scenarios are tested manually. This paper presents WebEV, a dataset containing E2E test cases from open-source popular projects. Projects are selected based on - i) Cypress-based automation, ii) popularity on GitHub and iii) executability of test cases. The dataset contains information regarding each test command along with the incurred state change representation. Snapshots of the application are used to retrieve - i) the current URL of the application, ii) the screenshot and HTML text of the entire page, and iii) the screenshot and HTML text of an operated UI element. This process is done both before and after each command execution to capture the perception of testers on each state transition, i.e., extract their thought process during testing. This dataset can assist the research community to model user web interaction, predicting the tester's perception, and improving the state of automated testing approaches. Moreover, WebEV can be used to mine how automated approaches differ from real-life E2E test scenarios.**

***Index Terms*—testing dataset, web testing, web test comprehension, behavior, tester's behavior**

I. Introduction

Modern web applications provide a rich set of interaction patterns that make the testing process harder. Testers are challenged to adopt a wide range of strategies to continuously re-validate the diverse usage scenarios efficiently [1]. Despite the availability of many automated testing approaches in literature [2]–[4], their industrial satisfaction is limited [5]. Such approaches do not learn the tester's thought process to effectively validate diverse functionalities. This is due to the unavailability of datasets that encapsulate the behavior of testers during manual web testing.

A fully automated approach integrating the tester's behavior can make testing efficient and satisfactory, combining the benefits of both manual and automated testing. However, obtaining a trusted source to extract such behavioral patterns is challenging. It is prone to subjectivity in selected web **S**ystem **U**nder **T**est (**SUT**) and the tester's individual preference. Manually generated browser interaction scripts can be statically analyzed to get the tester's actions (e.g., click, input, scroll). But this fails to understand the tester's goal by capturing the web state changes due to UI interactions. Therefore, a dataset is required that represents the changes in the application state that lead testers to interact the way they do.

The absence of such a dataset is noticeable in automated web testing literature as these approaches cannot satisfy practitioners [5]. State-of-the-art approaches [3], [4] focus on maximizing code coverage metrics such as - branch coverage, state coverage, etc. rather than how testers perceive manual testing. The Android testing domain shows advancements toward this goal. Linares-Vásquez et. al. [6] surveyed open-source Android contributors regarding the current practices during manual testing. However, this study focuses on tools used and the preference of developers regarding test design, tools used, and documentation. The Rico dataset [7] contains large-scale crowdsourced usage interaction sequences from 9.7k apps. However, such studies and datasets fail to consider the tester's behavior during E2E testing. To the best of acquired knowledge, no research work captures such behaviors required to validate the diverse scenarios of SUT efficiently.

This paper presents a WebEV, a dataset encapsulating the behavior of testers during End-to-End testing extracted from GitHub. The dataset is constructed keeping in mind the real-world practice, diversity of the web ecosystem, extendibility, and automatic reconstruction. Cypress [8] is selected as the E2E testing tool due to its popularity based on npm downloads. Initially, projects having clear documentation and maintaining E2E Cypress tests are picked from GitHub. Then, these projects are filtered based on the number of collaborators, stars, etc. Tests are executed after resolving the project dependencies. Google's Puppeteer [9] library is used to extract the test commands and UI interactions from the mock browser instance. More specifically, the UI element locator operated and their interaction type such as click, input, scroll, etc. along with their values are parsed. The snapshots before and after the respective command are collected targeting the UI element locator. Moreover, the snapshots of the entire page before and after each command is also tracked.

The snapshots and interactions in WebEV can be utilized to understand the tester's usage scenarios. These patterns can improve the current state of - i) user interaction modeling, ii) tester perception guidance and iii) automated testing.

II. Related Work

Web application automation testing research has gained much popularity due to the complex and rich nature of user interactions [5], [10]. More specifically, E2E testing is the most prominent type as it simulates real user scenarios to validate each diverse scenario [11]. Unfortunately, no dataset in literature represents how testers manually do E2E web testing. Instead, researchers have devised - automated testing techniques that reward coverage metrics [2]–[4], usage statistics based user behavior modeling [12], domain-specific language to generate tests [13], automatic page-object generation to encapsulate web pages [14], stable web locator generator for automation scripts [15], and so on. These efforts focus on either fully automated approaches that lack the tester's domain knowledge (maximize code metrics) or improving automation script generation that is expensive to maintain [5]. Integrating the benefits of both may improve the current practice, i.e., incorporate the tester's behavior to test the SUT labor-free.

Linares-Vasque et. al. [6] addressed this gap to improve the unsatisfactory state of Android automated testing. The study surveyed 102 open-source Android contributors about their approaches taken during manual testing. The fact that the code coverage metric is not useful to 51 respondents, draws more focus on exploring the behavior of testers during manual testing. Moreover, research efforts have been made by Pecorelli et. al [16] and Christophe et. al [17] to statically analyze the quality of open-source test cases in Android and web domains respectively. However, these studies focus on code quality and flakiness rather than interaction strategies.

In the Android automated testing domain, Humanoid [18] leverages UI interaction patterns learned from the Rico dataset [7]. Rico is a crowdsourced dataset containing app metadata, design, and UI interactions. The WevQuery dataset [19] similarly contains general user interaction patterns for the web. However, such datasets fail to consider the behavior exhibited by testers during E2E testing. Certain scenarios of the SUT may be left unexplored as users do not interact with the intention of finding defects.

III. Dataset Construction

This paper presents WebEV, a dataset representing the way testers perform manual web automated testing. Such a dataset can be utilized to make efficient test cases and automated testing more acceptable. The WebEV dataset is generated by extracting the tester's interaction sequences during End-to-End (E2E) web testing, as depicted in Fig. 1

A. Objectives

WebEV aims to mine the tester's behavior, which replicates real user scenarios to validate the SUT. The design of WebEV is driven by the following broad concerns:

Real-world practice: Practitioners do not always follow the theoretical best practices. WebEV aims to collect the tester's behavior from real-world large systems with rich collaborated development history. This ensures discovering the generalized trends and practices of testers at scale.

Diversity: Web applications are usually an amalgam of diverse frontend and backend technologies. Such technological and domain-specific complexities should not interfere with dataset construction. This in turn would make the dataset generalized and allow a richer diversity of the tester's behavior.

Extendibility: The dataset construction process should be such that it can incorporate new projects in an automated way. The test case extraction process should not be affected by newer technology or application domain.

Automatability: Manual extraction of test cases and test command from large-scale projects is labor-intensive and erroneous. Keeping automatability in mind, projects are selected that have well-maintained test cases and clear documentation regarding dependencies and test cases.

B. Source of Tester's Behavior

The initial step in constructing the dataset is to identify web testing artifacts containing interaction sequences (i.e., clicks, inputs, and hovers). Open-source projects are selected such that the scripted automation usage scenario can be extracted. E2E tests can be mined from projects having a high degree of collaborative and maintenance efforts. GitHub web projects that use Cypress [8] for E2E testing are considered as Cypress has become the de facto for automated E2E testing. Cypress runner starts a mock browser instance and tests execute in the same run loop as the SUT. Consecutively, it takes snapshots during execution, enabling time travel back to the application state on each test command. Therefore, a dynamic analysis of the project's test commands and their states can be examined irrespective of their underlying technology.

C. Subject System Filtering

The goal is to narrow down peer-reviewed GitHub repositories using Cypress that have traction in the community, i.e., stars, forks, and contributors. For selection purposes, repositories using *cypress-io/cypress* or eslint-plugin-cypress as dependencies are listed using ghtopdep [20]. Merging projects having these dependencies and removing duplicates yielded 3345 projects. Similar to prior work [3], [21], using the GitHub API, projects are shortlisted if - i) #stars $\geq$ 50 to measure popularity, ii) #commits $\geq$ 50 for maturity, and iii) #contributors $\geq$ 3 to ensure project received outside collaboration. This filtering led to 1420 projects. A manual investigation was done to further filter repositories not being web applications or not actively using Cypress for E2E testing. Dependencies to run each of the projects are resolved and tests are executed using the Cypress runner. Repositories failing to execute the test cases are neglected as those did not maintain the test specifications along with code changes. Similar to literature [22], the top 100 web-based projects are considered due to time constraints, which should be a good representation of current practices. Moreover, other projects will also be gradually added to WebEV.

D. Extracting Test Cases

Cypress testing specs contain a set of test cases, which are further broken down into individual commands. The

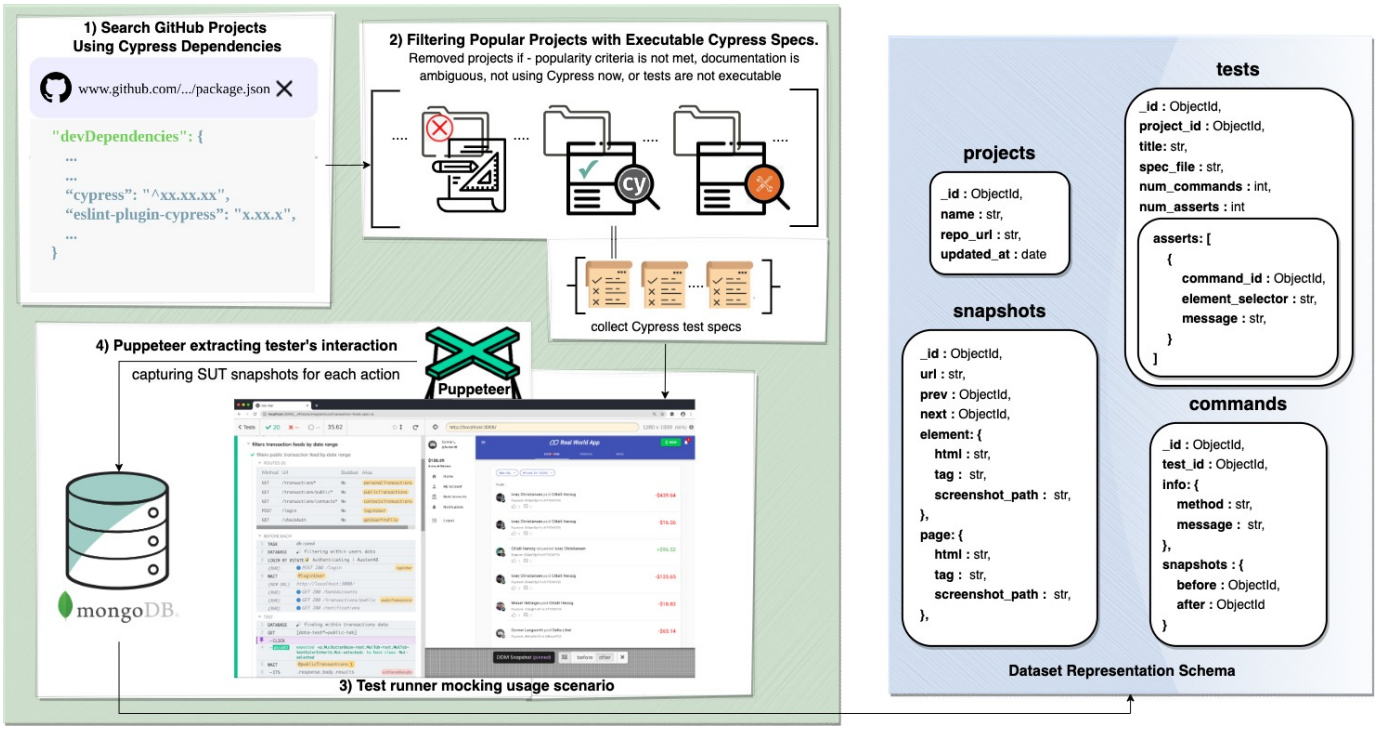

Fig. 1: Overview of approach to collect tester's usage scenarios

commands and snapshots are obtained by gaining control of the mock execution browser using Google's Puppeteer [23]. Dynamic execution is used rather than static analysis to avoid errors in resolving dependencies among files and modules. The state changes due to each command could not have been captured through static analysis. Alternatively, the commands and snapshots can be extracted from the Cypress mock browser from the commands list panel and the SUT iframe respectively. For more details on the Cypress test runner, the readers may explore the official documentation. For convenience, all projects are translated to use a specific Cypress version through the Cypress runner.

First of all, Puppeteer needs the port associated with the running mock browser instance to gain control. The port is known by querying the *launchOptions* parameter in *before:browser:launch* event, defined inside the Cypress config file (e.g., cypress.config.js). After establishing a connection, the targeted E2E spec files are run automatically by executing a click from the list of specs. The puppeteer script collects the tests after waiting until the spec has finished execution. Each test case is expanded by clicking to get the list of commands. Then, using appropriate CSS selectors, Puppeteer extracts all the commands. Additionally, Cypress keeps snapshots of SUT before and after the command is executed. Clicking on each command time travels back to that snapshot. Also, it highlights the UI element being operated on, located by the selector in *get* command. The snapshots before and after each command are used to capture - i) the current URL, ii) the screenshot and HTML text of the entire page, and iii) the non-highlighted screenshot and HTML text of the selected element.

E. Dataset Representation

WebEV and the Puppeteer script used to generate it are made publicly available [24]. The dataset is represented using mongoDB [25] due to its flexibility and wide usage in software mining research [26], [27]. It stores the textual information and file references for the screenshots, as shown in the schema design in Fig. 1. Whereas, the actual files are located in separate project-wise folders. WebEV is easily extendible as extracting usage scenarios from a new project just adds the project test command information and screenshots in the corresponding mongo collection and project folder.

IV. Use Cases

The generalized representation of test cases in WebEV is designed to support a wide range of research applications. Focusing only on Cypress-based test cases may limit the number of projects and tests, but the same methodology can easily be replicated to extend the dataset for other web testing tools. The only difference would be to monitor JS events using Puppeteer rather than extracting command logs provided by Cypress. Extending the dataset will increase the scope for more deep learning-based use cases. The generic data and representation of WebEV motivate three major categories of use cases, as discussed in subsequent subsections.

A. User Interaction Modeling

In modern web applications, users interact with the interface in a plethora of ways. Practitioners face difficulty designing how users can interact with web elements, despite catalogs of design patterns ([28]) being available [29]. Therefore, user

interaction modeling techniques can be used to train on WebEV's UI interactions. They simulate a real user functionality scenario from start to finish including the type of UI interaction (such as click, input, scroll, etc.) and the web element being operated on. The code and visual representation of the element and the entire page can be used to predict the next user interaction. Moreover, predicting the user's expected actions can optimize systems to pre-fetch resulting resources.

B. Tester Perception Guidance

Even with the existence of automated test scripts, practitioners frequently need to verify manually whether code changes reflect the evolving requirements. These manual checks can be converted to automation scripts for future use. However, the reuse of such scripts is hindered due to the modern development style of continuous integration. Moreover, preserving domain knowledge to design test cases that verify all possible usage scenarios is very difficult for evolving systems. UI interaction patterns in WebEV can help testers regarding when, which, and how diverse scenarios can be exploited, i.e., provide perceptual guidance. The traces in test cases can be mined to guide practitioners in the following ways:

Prioritize actions: Training the test traces from the large-scale projects in WebEV can model the actionable elements which should be prioritized. Such a tool can aid testers to identify elements that usually need more attention (such as web forms) or should be avoided (e.g., redundant cyclic navigation). Testers can benefit from dynamically adjusted action priority based on learned naturalness in the testing sequence. For example, the action sequence responsible for the creation of an entity can be given initially more priority rather than deletion. Afterward, the priority can be dynamically adjusted to remind exploring alternative actions.

Scenario coverage: E2E testing validates a sequence of actions in which the user follows - i) an expected usage scenario based on requirements, and ii) an unexpected path of execution that should be handled properly or blocked. Based on test scenarios taken from WebEV, testers simulate the expected user behavior of filling out the required information before submitting a form. Alongside, the alternate scenarios are also checked where the user forgot any required field or puts invalid input such as putting characters in a number field, mismatching password constraint, boundary value analysis, etc. For such scenarios, assertions are made to ensure the user must be unable to submit the form. Testers often fail to test such paths of execution due to the lack of domain knowledge, experience, or time. WebEV solves this gap as it contains the testing behaviors exhibited by testers in large-scale collaborated projects. Furthermore, WebEV can be used to devise a design-based test suggestion tool that captures the subject UI component to derive the possible interaction patterns, reducing the risk of untested and duplicate scenarios.

Assertability: Web automation scripts ensure functional correctness through assertions, such as, asserting whether a particular button is present on a page, an error message is shown on blank input, an entity created/deleted is reflected in the table, and so on. With the change in design and requirements, these assertions also have to be modified. Study has been done to learn meaningful assertions in Java-based unit testing [30], but there is no such work on E2E tests for web applications. The assertions crawled in WebEV can help practitioners understand the current practice in peer-reviewed systems. Practitioners can make queries based on SUT design to suggest related possible assertions. For example, by detecting that the SUT is on the signup page, the tool can suggest assertions like - the signup button's existence, ensure password confirmation match, redirect to the home page after registration, etc.

C. Automated Testing

The tester's behavior can be learned from WebEV to devise an automated testing tool based on action prioritization, scenario coverage, and assertability. State-of-the-art automated testing tools rely on metrics such as state coverage [4], branch coverage [3], etc. but fail to meet industrial satisfaction. For example, automated tools prefer navigating to diverse pages to increase coverage rather than submitting a partially filled web form, i.e., fulfill a complete usage scenario. Learning the interaction patterns from real-world testing scenarios can improve automated testing. Moreover, the state explosion problem can be reduced as testers avoid test scenarios representing duplicated functionality.

V. Conclusion and Future Work

This paper introduces WebEV (publicly available at [24]), an E2E testing dataset encapsulating interactions done to validate diverse usage scenarios of web applications. Although WebEV is in the initial stage, it can be easily extended for any web application using the same methodology. It requires a one-time effort of running the E2E test cases to extract the interaction scenarios. Cypress E2E tests from popular 100 open-source projects are executed to replicate how testers manually test diverse scenarios of SUT. Later using the Puppeteer library, test commands are extracted along with representations capturing state changes on each interaction. Practitioners can utilize this dataset to understand how large-scale practitioners exploit diverse scenarios. Also, this dataset will help researchers improve automated testing approaches that take into account the tester's perception.

We believe WebEV can shift the paradigm in automated web testing based on actual testers' behavior rather than relying on traditional code metrics. Our future plan includes comparing the differences between scenarios covered in WebEV and by state-of-the-art automated E2E testing approaches. This study can track the limitations of current approaches using the same projects in WebEV. Also, we plan to analyze the most prominent usage patterns and suggest scenarios based on provided web application layout.

Acknowledgment

This study is supported by the Fellowship from ICT Division, Government of Bangladesh – No.: 56.00.0000.052.33.002.22-42, date: 12.06.2022

REFERENCES

[1] J. Itkonen, M. V. Mantyla, and C. Lassenius, "How do testers do it? an exploratory study on manual testing practices," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2009, pp. 494–497.

[2] A. M. Fard and A. Mesbah, "Feedback-directed exploration of web applications to derive test models." in *ISSRE*, vol. 13, 2013, pp. 278–287.

[3] M. Biagiola, A. Stocco, F. Ricca, and P. Tonella, "Diversity-based web test generation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 142–153.

[4] Y. Zheng, Y. Liu, X. Xie, Y. Liu, L. Ma, J. Hao, and Y. Liu, "Automatic web testing using curiosity-driven reinforcement learning," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 423–435.

[5] F. Ricca, M. Leotta, and A. Stocco, "Three open problems in the context of e2e web testing and a vision: Neonate," in *Advances in Computers*. Elsevier, 2019, vol. 113, pp. 89–133.

[6] M. Linares-Vásquez, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk, "How do developers test android applications?" in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 613–622.

[7] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the 30th annual ACM symposium on user interface software and technology*, 2017, pp. 845–854.

[8] "cypress-io/cypress: Fast, easy and reliable testing for anything that runs in a browser." https://github.com/cypress-io/cypress, (Accessed on 02/07/2023).

[9] Google, "Puppeteer — tools for web developers," 2010. [Online]. Available: https://developers.google.com/web/tools/puppeteer

[10] F. Ricca and A. Stocco, "Web test automation: Insights from the grey literature," in *International Conference on Current Trends in Theory and Practice of Informatics*. Springer, 2021, pp. 472–485.

[11] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Approaches and tools for automated end-to-end web testing," in *Advances in Computers*. Elsevier, 2016, vol. 101, pp. 193–237.

[12] S. E. Sprenkle, L. L. Pollock, and L. M. Simko, "Configuring effective navigation models and abstract test cases for web applications by analysing user behaviour," *Software Testing, Verification and Reliability*, vol. 23, no. 6, pp. 439–464, 2013.

[13] S. Thummalapenta, S. Sinha, N. Singhania, and S. Chandra, "Automating test automation," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 881–891.

[14] A. Stocco, M. Leotta, F. Ricca, and P. Tonella, "Apogen: automatic page object generator for web testing," *Software Quality Journal*, vol. 25, no. 3, pp. 1007–1039, 2017.

[15] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "Robula+: An algorithm for generating robust xpath locators for web testing," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 177–204, 2016.

[16] F. Pecorelli, G. Catolino, F. Ferrucci, A. De Lucia, and F. Palomba, "Testing of mobile applications in the wild: A large-scale empirical study on android apps," in *Proceedings of the 28th international conference on program comprehension*, 2020, pp. 296–307.

[17] L. Christophe, R. Stevens, C. De Roover, and W. De Meuter, "Prevalence and maintenance of automated functional tests for web applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 141–150.

[18] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Humanoid: A deep learning-based approach to automated black-box android app testing," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1070–1073.

[19] A. Apaolaza and M. Vigo, "Wevquery: testing hypotheses about web interaction patterns," *Proceedings of the ACM on Human-Computer Interaction*, vol. 1, no. EICS, pp. 1–17, 2017.

[20] "github-tooling/ghtopdep: Cli tool for sorting dependents repo by stars," https://github.com/github-tooling/ghtopdep, (Accessed on 02/07/2023).

[21] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th international conference on Software engineering*, 2014, pp. 356–366.

[22] G. Kudrjavets, N. Nagappan, and A. Rastogi, "Do small code changes merge faster? a multi-language empirical investigation," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 537–548.

[23] "Puppeteer — puppeteer," https://pptr.dev/, (Accessed on 02/07/2023).

[24] "Webev-dataset - github," https://github.com/fuadmmnf/WebEV-Dataset.git, (Accessed on 03/08/2023).

[25] "The mongodb database - github," https://github.com/mongodb/mongo, (Accessed on 02/07/2023).

[26] K. Lyu, K. Noda, and T. Kobayashia, "Sdexplorer: a generic toolkit for smoothly exploring massive-scale sequence diagram," in *Proceedings of the 26th Conference on Program Comprehension*, 2018, pp. 380–384.

[27] G. Beniamini, S. Gingichashvili, A. K. Orbach, and D. G. Feitelson, "Meaningful identifier names: The case of single-letter variables," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 45–54.

[28] "Interaction design pattern library - welie.com," http://www.welie.com/patterns/, (Accessed on 02/05/2023).

[29] V. Gkantouna, A. Tsakalidis, and G. Tzimas, "Mining interaction patterns in the design of web applications for improving user experience," in *Proceedings of the 27th ACM conference on hypertext and social media*, 2016, pp. 219–224.

[30] C. Watson, M. Tufano, K. Moran, G. Bavota, and D. Poshyvanyk, "On learning meaningful assert statements for unit test cases," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1398–1409.