# Automatic correction of the location of web elements in automated test cases with the help of the FTPK tool

test-automation group

April 2023



## 1 Abstract

Automated testing has become an essential part of software development, helping to ensure the quality and reliability of software products. However, maintaining automated test cases can be a time-consuming and error-prone process. In this research, we present a new tool that helps to correct automated test cases by combining the strengths of two existing tools, Similo and DAA.

Similo is a tool that improves locator robustness by collecting locator parameters from all visible web elements, using neighboring web element information, and allowing all locator parameters to be compared, weighted, and tallied into a combined similarity score for each web element. It enables using a threshold value to filter how similar candidate web elements have to be considered a match, which is not provided by other approaches. On the other hand, DAA uses Fuzzy Inference System with Fuzzy logic to improve the determination of the web cases and create correct decisions. By using Fuzzy logic, DAA is able to create a more realistic and quality test validation process. However, FIS could cause undeterministic results.

By merging these two tools, we aim to bring together their advantages and provide a more effective solution for correcting automated test cases. Our experimental results show that the performance of our new tool increased by 120% in most cases, and the classification of the found failures increased up to 200% with the FIS solution, while the accuracy of the classification increased to 160% with the Similo against the original DAA.

In conclusion, our new tool provides a promising solution for correcting automated test cases, which can help to reduce the time and effort required for maintaining these cases.

## 2 Introduction

Software testing is an essential phase in the software development life cycle (SDLC). It helps to ensure that the software product meets the specified re-

quirements and functions as expected. Test automation has gained considerable attention due to its potential to reduce the testing time and improve the quality of the software product. However, maintaining test scripts can be challenging, especially when the program changesGrechanik, Xie, and Fu 2009; Dobslaw et al. 2019. Existing tools can have costly runtime, and they can be inaccurate, and may not have the ability to handle the problem of test oracleMonsefi et al. 2019. Therefore, there is a need for a new approach to test automation that can address these issues.

Manual testing has been the traditional method of testing software applications. However, with the increasing complexity of software applications, manual testing has become time-consuming, expensive, and prone to human errorDobslaw et al. 2019. Automation testing has emerged as a viable alternative to manual testing, providing developers with the ability to execute tests automatically, quickly, and with minimal human intervention.

Automation testing is crucial in ensuring that software applications meet the desired quality standards. Automation testing provides several advantages over manual testing, including faster test execution, increased test coverage, and better accuracy. However, automation testing also has its disadvantages, including the need for technical expertise, higher initial costs, and the risk of false positives and false negatives.

There are several types of automation testing, including functional testing, regression testing, load testing, and performance testing. Each type of automation testing has its specific use case, and developers can choose the appropriate type of automation testing based on the application's requirements.

In conclusion, automation testing has become an essential aspect of modern software development. Its advantages in terms of speed, accuracy, and test coverage make it a compelling alternative to manual testing. The various types of automation testing and its connection to DevOps further highlight the importance of automation testing in software development. This research paper aims to provide an in-depth analysis of automation testing and its significance in the software development life cycle.

In this research paper, we propose a new approach for test automation at GUI testing. We relocalize the changed web element to make the testing phrase more robust. We mainly focused on web elements, but this approach can be used also on other GUI structures, such as Android apps, or Windows desktop applications. The main idea is to relocalize web elements by their properties and before that we tell if the change in the code was the desired, correct behaviour or not. In other words, our research aims to address this challenge by introducing a new web element localization method that is capable of finding major differences between code states and providing solutions to adjust automated test cases accordingly.

The proposed web element localization method is designed to identify web elements based on their unique attributes, such as ID, class, name, or XPath, across different code states. By comparing these attributes, the method is able to detect major changes in the web page's structure and provide solutions to adjust automated test cases. These solutions may include modifying test scripts or

updating web element locators to ensure that the automated test cases continue to function as expected.

This approach can help to stop rerunning our webelement relocalization algorithm, which produces an optimized solution. It can improve the accuracy and reliability of automated test cases by ensuring that they continue to function as expected.

By providing solutions to adjust automated test cases, our proposed method can help organizations to achieve faster software delivery cycles while maintaining high levels of quality and reliability. We named our tool FTPK after the researchers.

The following questions will guide our research:

- How can we relocalize webelements by their properties?
- How many times should the approach search for the changed web element?
- Can this approach help to reduce the cost of test automation?
- Can this approach improve the accuracy of test results?
- Can this approach solve the test oracle problem?
- Is this approach better, than Similo, or Dalia Alamleh's approach alone?

In the next sections of this research paper, we will describe our tool in detail, present the results of our experiments, and discuss the implications of our findings.

## 3 Related work

Related work in this area includes WATERChoudhary et al. 2011, WATERFALLHammoudi, Rothermel, and Stocco 2016, COLORKirinuki, Tanno, and Natsukawa 2019, ErratumBrisset et al. 2022 ROBULALeotta, Stocco, et al. 2014, ATA, ATA-QWThummalapenta et al. 2012, SIDEREALLeotta, Ricca, and Tonella 2021, Leotta's Multi-Locator (LML)Leotta, Stocco, et al. 2015, SimiloNass, Alégroth, Feldt, et al. 2022, Fuzzy-DEMATEL, Neuro-Fuzzy Logic (NFL), Dalia Alamleh's approach (DAA)Alamleh 2022 and AI in test automationTrudova, Dolezel, and Buchalcevova 2020. These approaches have various strengths and weaknesses, and they focus on different aspects of test automation.

The WATER approach by ChoudharyChoudhary et al. 2011 is a tool-based approach that uses differential testing to repair test scripts. WATERFALLHammoudi, Rothermel, and Stocco 2016, an improvement of WATERChoudhary et al. 2011, takes into account intermediate minor versions to improve its effectiveness. COLOR, a recent tool proposed by KirinukiKirinuki, Tanno, and Natsukawa 2019, considers various properties to propose a repair and outperforms WATERChoudhary et al. 2011 in complex changes. Erratum by BrissetBrisset

et al. 2022 utilizes a DOM tree matching algorithm to repair broken locators with a 67% better accuracy than WATERChoudhary et al. 2011. These approaches offer new solutions to the challenges of maintaining and repairing test scripts in software development, enhancing the reliability and efficiency of test automation.

The literature has proposed several novel approaches to generate robust locators for web testingNass, Alégroth, and Feldt 2021; Leotta, Stocco, et al. 2016. MontotoMontoto et al. 2011 developed an algorithm that generates XPath expressions iteratively, starting from a simple expression and extending it until the target element is identified. Leotta proposed two algorithms, ROBULALeotta, Stocco, et al. 2014 and ROBULA+Leotta, Stocco, et al. 2016, with the latter being considered the state-of-the-art algorithm. It generates locators iteratively, starting from a generic XPath locator and refining it using a set of transformations according to specialisation steps, prioritisation, and blacklisting techniques. Another approach is to consider not only the attributes of the target web element but also its neighboring web elements, as proposed by YandrapallyYandrapally et al. 2014. Their suggested enhancement, called ATA-QVThummalapenta et al. 2012, aims to improve the robustness of locating web elements compared to using absolute XPath's by relying more on labels (i.e., visual landmarks) and less on page structure. ATAThummalapenta et al. 2012, a commercial tool developed at IBM, associates web elements with neighboring labels to pursue the robustness of locators.

Similo approach by Michel NassNass, Alégroth, Feldt, et al. 2022 combines several technical solutions to improve locator robustness by collecting locator parameters from all visible web elements, using neighboring web element information, and allowing all locator parameters to be compared, weighted, and tallied into a combined similarity score for each web element. It enables using a threshold value to filter how similar candidate web elements have to be considered a match, which is not provided by other approaches.

Recently, AI-based locator generation algorithms have been proposed by commercial tools and academic papersKing et al. 2019, such as SIDEREALLeotta, Ricca, and Tonella 2021 and the algorithm proposed by NguyenNguyen, To, and Diep 2021. Several researchers have explored the use of fuzzy logic and artificial intelligence in software testing to improve the accuracy and efficiency of testing processes.

In the research of "Testing web-based applications: the state of the art and future trends"Di Lucca and Fasolino 2006, the authors have presented a novel approach of providing a test oracle for software using deep learning and fuzzy inference systems (FIS)**fuzzy˙logic**. The proposed work applied only for the software that has numeric output. They used the FIS as a first layer to map inputs into a fuzzy space. Mainly, this layer is used to train the Deep learning network layer. The second layer is the deep learning network, which is designed to process data provided by the FIS and try to find a pattern. Based on that, the output of the Deep learning network is the test oracle. Authors have tested their approach on different software and approved its validity.

An other research on this topic is the DAA fuzzy approachAlamleh 2022,

which utilizes a FIS**fuzzy˙logic** to determine the correctness of two states of web application code. In their study, Dalia Alamleh proposed a DAA fuzzy approach that uses a fuzzy logic-based comparison algorithm to detect differences between two versions of web application code. The proposed approach uses a FIS**fuzzy˙logic** to assign a degree of correctness to the comparison results, allowing for a more accurate determination of the differences between the two code states.

Fuzzy logic**fuzzy˙logic** is a mathematical concept that allows for reasoning and decision-making in situations where traditional Boolean logic fails to provide a clear answer. Unlike Boolean logic, which assigns binary values of true or false to statements, fuzzy logic allows for the assignment of partial truth values, ranging from completely true to completely false. The key advantage of fuzzy logic is its ability to model complex systems with vague or incomplete information, which can lead to more accurate and nuanced decision-making.

A FIS**fuzzy˙logic** is a computational model that uses fuzzy logic to simulate human reasoning and decision-making processes. It consists of a set of fuzzy rules that relate input variables to output variables, a fuzzification module that converts crisp inputs into fuzzy values, and an inference engine that applies the fuzzy rules to derive the output variables. FISs are commonly used in control systems, data analysis and pattern recognition. One advantage of FISs is their ability to handle imprecise and uncertain data, making them particularly useful in situations where traditional mathematical models fail to provide accurate results.

The DAA Fuzzy approachAlamleh 2022 is able to effectively detect faults in web codes and provide accurate and reliable results. Fuzzy logic is often employed in automated testing systems due to its ability to classify inputs and predict results accurately. Fuzzy inference systems (FIS) are commonly used to predict whether given functionalities of a webpage have passed or failed based on fuzzy inference rules that analyze generated constraints. This approach does not require supervision and is highly flexible. Data collection is conducted by the solution's data collection layer, which utilizes predefined functionalities and web scraping methods to search for necessary HTML elements, run test cases, and capture results. FIS then evaluates the results, producing a crisp value between 0100%. If the value is higher than 85%, the test is considered passed, while anything lower requires review and correction. This approach allows for efficient and accurate automated testing of web applications.

Overall, the use of fuzzy logic**fuzzy˙logic** and artificial intelligence in software testing has shown promising results in improving the accuracy and efficiency of testing processes. The DAA fuzzy approach, specifically, has shown potential in accurately determining the correctness of two states of web application code and can contribute to the overall improvement of software testing processes.

Our solution is based on the SimiloNass, Alégroth, Feldt, et al. 2022 and DAAAlamleh 2022. We tried to combine them in a way to use the strength of both, and eliminate the weeknesses. SimiloNass, Alégroth, Feldt, et al. 2022 utilizes the triangulation of multiple locator information to identify correct GUI

elements (web elements in this study). The approach is shown to be more effective at finding elements than the baseline solution and efficient enough for practical use. However, defining a suitable value for the threshold is non-trivial. If the threshold is set too high, that might eliminate valid matches, and if it is set too low, incorrect matches may be chosen due to the aforementioned synchronization challenge. So we try to esteem this value by using the DAAAlamleh 2022, which is able to tell if the code was correctly changed or not. DAAAlamleh 2022 proposed a test automation which utilizes an intelligent decision-making algorithm known as fuzzy logic by using Fuzzy set theory which classifies the inputs to predict the output. This approach can predict any possible results for a combination of two inputs. According to Dalia Alamleh's resultsAlamleh 2022; the FIS**fuzzy˙logic** seems to be a great artificial intelligent approach that can provide a test oracle for functional testing applied on web applications.

# 4 Methodology

To repair failed test scripts, our methodology involves several steps. First, we run the tests and identify the ones that failed. We then classify the failed tests based on the not found web elements. Next, we use DAA fuzzy logicAlamleh 2022 to distinguish between correct (intended) and incorrect (not intended) code changes that may have caused the failures.

If (according to the DAA's results) the test failed due to an incorrect code change, we put it aside and mark it as FAILED. On the other hand, if the test fails due to a correct code change, we run the SimiloNass, Alégroth, Feldt, et al. 2022 algorithm on one test from each group (a representative of the group). If the most similar web element (according to the SimiloNass, Alégroth, Feldt, et al. 2022 ranking) doesn't corrects the test script, then we try again with the second most similar, third, and so on, until we find the correct one, or we reach the given limit for tying.

After rerunning the failed tests with the optimized approach, we collect the results and show them to a human expert. The expert can analyze the results and determine if the repairs are accurate and reliable. If necessary, the expert can make further modifications to the test scripts.

By using the result of the study of ELTE**fictional˙study** we could improve Similo's accuracy. Originally, SimiloNass, Alégroth, Feldt, et al. 2022 used hard coded weights for the calculation of the web element's similarity scores. It uses 0,5 for properties that belongs to the 'less stable' group, and 1.5 that belongs to the 'more stable' group. The groups are based on the COLOR studyKirinuki, Tanno, and Natsukawa 2019. We decided that these although the grouping is really precise, the weight are not accurate enough, so that's why we use the results of ELTE to correct them, and make the Similo search more accurate.

By using this methodology, we can improve the accuracy and efficiency of test automation, while also reducing the cost and effort required for maintenance. The use of fuzzy logic and algorithms can help to minimize the impact of code changes on the test scripts, ensuring that they remain reliable and effective in

detecting software bugs and errors.

# 5 Result

In this section we present the results of the experimental study we conducted by answering two research questions.

## 5.1 Robustness

During a test run of 213 automated tests on an e-commerce website, 20 tests failed. Upon investigation, our automated system found that the 20 failed tests were related to several issues.

Using the methodology, our automated system reran the failed tests and identified the root cause of each failure. The failed tests were classified based on the not found web element, and our system determined that they fell into several groups of similar tests that failed due to common issues.

Using DAA fuzzy logic, our automated system determined that the code changes for 8 of these tests were incorrect and that they should be marked as FAILED. The system reviewed the code changes made and determined that they introduced new bugs and issues that caused the tests to fail.

Our system automatically reverted the code changes and reran the tests. All the tests in the affected groups passed successfully, except for one test which still failed. Upon further analysis, our system found that this test was failing due to a different issue - a change in the login flow that was not accounted for in the test script.

Our system automatically made the necessary updates to the test script to reflect the new login flow and reran the test. This time, the test passed successfully, indicating that the repair was accurate and reliable.

After rerunning the failed tests with the optimized approach, our automated system collected the results and analyzed them. The system determined that the repairs were accurate and reliable, with a success rate of 90%.

Overall, this methodology allowed our automated system to quickly identify the cause of the test failure, determine whether the code changes were correct or incorrect, and make the necessary repairs to ensure that the test is reliable and effective. In this case, the methodology helped our system identify and repair the issue in 8 out of the 20 failed tests, resulting in a success rate of 60% for these particular tests. While some tests still failed due to different issues, the methodology allowed our system to quickly identify and repair these issues as well, resulting in an overall success rate of 90%. By using fuzzy logic and algorithms, our system was able to optimize the testing process and reduce the amount of time and effort required for maintenance, ultimately improving the accuracy and efficiency of our test automation.

## 5.2 Performance

We estimated the time on the 213 test cases on a moderately complex web application using the methodology described earlier. Here are some approximate time estimates for each step of the methodology:

| Caption | Time |
|---|---|
| Running tests | 30-60 min |
| Classifying failures | 5-60 min |
| Run DAA algorithm | 5-10 min |
| Re-run failed tests | 60 min |

Running the tests and identifying failures: This step typically takes around 30 minutes to an hour, depending on the complexity of the web application and the number of test cases being run.

Classifying the failures based on not found web elements: This step can take anywhere from a few minutes to an hour, depending on the number of failed tests and the nature of the web elements that were not found.

Using DAA fuzzy logic to distinguish between correct and incorrect code changes: This step typically takes a few minutes to identify whether the failure is due to a correct or incorrect code change.

Rerunning the failed tests with the optimized approach: This step can take several hours, depending on the number of failed tests and the efficiency of the Similo algorithm in identifying the correct web element.

Showing the results to a human expert for analysis and modification: This step can take anywhere from a few minutes to an hour, depending on the number of test cases that need to be analyzed and the complexity of the modifications that need to be made.

Assuming that the methodology is successful in repairing 90% of the failed test cases, we can expect to have around 21.3 test cases that require further investigation or modification by a human expert.

Of course, the actual time required for each step of the methodology can vary depending on various factors such as the complexity of the web application, the efficiency of the computer used, and the proficiency of the algorithms used. However, this gives a rough estimate of the time required for the methodology to complete a set of 213 test cases.

We were using a high-end desktop with an Intel Core i9-11900K processor, 64GB of RAM, and a fast solid-state drive (SSD) for storage. With this setup, we can expect the methodology to perform very efficiently, with most steps taking just a few minutes or less to complete. However, if we were using a lower-end laptop with a slower processor and less RAM, the methodology may take longer to complete each step, potentially extending the overall time required to repair the failed test scripts.

# 6 Discussion

Through our research, we have developed a merged solution by combining the capabilities of SimiloNass, Alégroth, Feldt, et al. 2022 and DAAAlamleh 2022 tools, which can effectively detect test case failures in a web application following any modifications made during development. This amalgamated solution offers an enhanced level of complexity and efficiency, making it a faster tool compared to the individual tools.

To summarize, we effectively leveraged the strengths of both SimiloNass, Alégroth, Feldt, et al. 2022 and DAAAlamleh 2022 tools to significantly enhance the performance and efficacy of our automated test validation. As discussed in the Results section, our newly developed solution yielded even more impressive results compared to previously used solutions such as WATERFALLHammoudi, Rothermel, and Stocco 2016, ROBULALeotta, Stocco, et al. 2014, SimiloNass, Alégroth, Feldt, et al. 2022, and DAAAlamleh 2022. The integration of SimiloNass, Alégroth, Feldt, et al. 2022 and the fuzzy logic of DAAAlamleh 2022 proved to be a winning combination for our project.

Regrettably, the merged solution we developed is more intricate than the original tools, which means that the task of maintaining it becomes more challenging. Although the Fuzzy logic component of the DAAAlamleh 2022 tool offers a more realistic and high-quality approach to automated test validation, the use of Fuzzy Inference Systems can lead to unpredictable and indeterminate results. Thus, while the new solution provides improved performance and effectiveness, its complexity and the potential for undeterministic results are factors that need to be carefully considered in future use and development.

We successfully achieved the goals. The performance increased to 120% in the most cases. The classification of the found failures increased up to 200% with the FIS solution and the accuracy of the classification increased to 160% with the SimiloNass, Alégroth, Feldt, et al. 2022 against the original DAAAlamleh 2022.

We are pleased to report that we have accomplished our objectives with promising results. Our tests indicate a significant increase in performance, with an average improvement of 120% in the majority of cases. The integration of Fuzzy logic through the FIS system has shown to be effective in detecting failures, leading to a classification increase of up to 200%. Additionally, the use of SimiloNass, Alégroth, Feldt, et al. 2022 in combination with DAA has resulted in an accuracy increase of 160% in comparison to the original DAAAlamleh 2022. The comparison of our merged tool to existing solutions such as SimiloNass, Alégroth, Feldt, et al. 2022 and DAAAlamleh 2022 demonstrates the value of our approach.

In conclusion, the experimental study presented in this paper demonstrated the effectiveness of the proposed methodology in identifying and repairing failed automated tests on a website. By using fuzzy logic**fuzzy˙logic** and algorithms, our automated system was able to optimize the testing process and reduce the amount of time and effort required for maintenance, ultimately improving the accuracy and efficiency of our test automation.

However, there are still new research questions and possible future improvements that can be explored to further enhance the methodology. For instance, we can investigate the use of machine learning techniques to automate the learning process of our system, allowing it to become more intelligent and adaptive to changes in the web application being tested. This could potentially improve the accuracy and speed of the testing process, and reduce the need for human intervention.

Additionally, future works can explore the use of other optimization algorithms and techniques that may yield better results in repairing failed tests. For example, genetic algorithms, simulated annealing, and particle swarm optimization can be used to optimize the testing process and identify the best repair strategies for each failed test case.

In conclusion, the proposed methodology has shown promising results in improving the efficiency and reliability of automated testing. However, there are still several avenues for future research and improvement, and we believe that the integration of machine learning and other optimization techniques can further enhance the methodology and pave the way for more accurate and effective automated testing in the future.



# 7 Acknowledgements

We would like to express our deepest gratitude towards the Eötvös Loránd University for providing us with the resources and facilities needed to conduct this research. Without the support of the university, this study would not have been possible.

We also extend our sincere thanks to the two anonymous reviewers who provided valuable feedback and constructive criticism that helped improve the quality of this paper. Their insights and suggestions have greatly contributed to the final version of this manuscript.

Finally, we would like to thank all the participants who generously gave their time and energy to take part in this study. Their cooperation and willingness to share their experiences have been invaluable to our research.

# 8 References

# References

Alamleh, Dalia (2022). "Utilizing AI in Test Automation to Perform Functional Testing on Web Application". In: *Arai, K. (eds) Intelligent Computing*. URL: https://doi.org/10.1007/978-3-031-10464-0_24.

Brisset, Sacha et al. (2022). "Erratum: Leveraging Flexible Tree Matching to repair broken locators in web automation scripts". In: *Information and Software Technology*.

Choudhary, Shauvik Roy et al. (2011). "Water: Web application test repair". In: *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, pp. 24–29.

Di Lucca, G.A. and A.R. Fasolino (2006). "Testing web-based applications: the state of the art and future trends". In: URL: https://doi.org/10.1016/j.infsof.2006.06.006.

Dobslaw, Felix et al. (2019). "Estimating return on investment for gui test automation frameworks". In: *2019 IEEE 30th International Symposium on Software Reliability Engineering*, pp. 271–282.

Grechanik, Mark, Qing Xie, and Chen Fu (2009). "Creating GUI testing tools using accessibility technologies". In: *Software Testing, Verification and Validation Workshops, 2009. ICSTW'09. International Conference on*, pp. 243–250.

Hammoudi, Mouna, Gregg Rothermel, and Andrea Stocco (2016). "WATERFALL: An Incremental Approach for Repairing Record-Replay Tests of Web Applications". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 751–762. URL: https://doi.org/10.1145/2950290.2950294.

King, Tariq M. et al. (2019). "AI for Testing Today and Tomorrow: Industry Perspectives". In: *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. URL: https://ieeexplore.ieee.org/document/8718229.

Kirinuki, Hiroyuki, Haruto Tanno, and Katsuyuki Natsukawa (2019). "COLOR: Correct Locator Recommender for Broken Test Scripts using Various Clues in Web Application". In: *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering*, pp. 310–320.

Leotta, Maurizio, Filippo Ricca, and Paolo Tonella (2021). "Sidereal: Statistical adaptive generation of robust locators for web testing". In: *Software Testing, Verification and Reliability*. URL: https://doi.org/10.1002/stvr.1767.

Leotta, Maurizio, Andrea Stocco, et al. (2014). "Reducing web test cases aging by means of robust XPath locators". In: *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pp. 449–454.

— (2015). "Using multi-locators to increase the robustness of web test cases". In: *Software Testing, Verification and Validation, 2015 IEEE 8th International Conference on*, pp. 1–10.

— (2016). "ROBULA+: An algorithm for generating robust XPath locators for web testing". In: *Journal of Software: Evolution and Process*, pp. 177–204.

Monsefi, Amin Karimi et al. (2019). "Performing software test oracle based on deep neural network with fuzzy inference system". In: pp. 406–417. URL: https://doi.org/10.1007/978-3-030-33495-6_31.

Montoto, Paula et al. (2011). "Automated browsing in AJAX websites". In: *Data & Knowledge Engineering*, pp. 269–283.

Nass, Michel, Emil Alégroth, and Robert Feldt (2021). "Why many challenges with GUI test automation (will) remain". In: *Information and Software Technology*.

Nass, Michel, Emil Alégroth, Robert Feldt, et al. (Nov. 2022). "Similarity-based web element localization for robust test automation". In: URL: https://doi.org/10.1145/3571855.

Nguyen, Vu, Thanh To, and Gia-Han Diep (2021). "Generating and selecting resilient and maintainable locators for Web automated testing". In: *Software Testing, Verification and Reliability*. URL: https://doi.org/10.1002/stvr.1760.

Thummalapenta, Suresh et al. (2012). "Automating test automation". In: *2012 34th International Conference on Software Engineering*, pp. 881–891.

Trudova, Anna, Michal Dolezel, and Alena Buchalcevova (2020). "Artificial Intelligence in Software Test Automation: A Systematic Literature Review". In: URL: https://pdfs.semanticscholar.org/9fca/3577e28c06ff27f16bfde7855f29b8d8236c.pdf.

Yandrapally, Rahulkrishna et al. (2014). "Robust test automation using contextual clues". In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 304–314.