Test Case Prioritization

Abhinay Balasani
Leena Ramayyagari
Venkata Naveen Kumar Udumula
Department of Computer Science
Purdue University
Fort Wayne, Indiana.

***Abstract*— In order to make sure that software is dependable, functional, and serves the intended goal, software testing is a crucial stage in the development process. Prioritizing test cases can be difficult when working with big test suites, which is one of the challenges of software testing. Prioritizing test cases seeks to find the most important test cases that can expose the most flaws in the system being tested.**

This research paper examines the importance of test case prioritising in software testing and offers a case study on test case optimisation. In order to reduce the number of test cases in a Django e-commerce shop application without compromising testing quality, the study's objective is to do so.

The study begins with an overview of software testing and the importance of test case prioritisation. We explore numerous techniques and tactics that have been used to prioritise test cases as we review recent work in test case optimisation and prioritisation.

The study focuses on the massive amount of test cases in the Django e-commerce store application, which can make testing time-consuming and expensive. To improve the test cases in the application, we advise removing those that are pointless and ineffective. Our approach entails using techniques for fault-seeding and code coverage analysis to discover the critical test cases that cover the most crucial code paths and reveal the most defects in the system under test.

The Django e-commerce store application needed its test case count reduced from 100 to 27 while maintaining the same degree of test coverage and fault detection capability. The results of our case study are now presented. We found that the optimised test suite performed better and was more economical to execute when we compared its performance to that of the original test suite.

We have shown that test case optimization and prioritization are essential for software testing and can greatly increase testing effectiveness and efficiency. Our paper outlines a strategy for streamlining test cases in a practical application, and the findings demonstrate how our strategy can cut testing time and expenses while retaining high-quality testing. Future research can expand our investigation into additional applications and domains as well as the usage of additional test case prioritization and optimization methods [1].

Keywords—Software testing Test case prioritization Django e-commerce store application Optimization

I. INTRODUCTION

This Software testing requires the ordering of test cases according to a set of predetermined criteria, which is known as test case prioritization. With the use of test case prioritization, software testing can be made more effective and efficient by making sure that the most important flaws are found as soon as possible. When working with large and sophisticated software systems, where the sheer volume of test cases can quickly become overwhelming, this technique is particularly crucial.

This study's objectives are to investigate the idea of test case prioritizing and to give a general overview of the various methods that have been suggested to enhance this procedure. In this article, we'll look at the advantages of test case prioritization and talk about the difficulties associated with its use [11]. The effectiveness of one of the suggested strategies in lowering the number of test cases needed for a particular degree of test coverage will also be shown through a case study that will be presented.

The success of software testing plays a prominent role in the development of any software and also affects the product's dependability and quality. Test case prioritization assists in streamlining the time- and money-consuming process of testing by placing the most priority on the most serious flaws first. As a result, testing can be completed more quickly and effectively. This enables software engineers to better utilize their time and resources.[10].

The importance of test case prioritization is highlighted by the fact that it is one of the key challenges in software testing. There are several factors that make test case prioritization difficult, such as the large number of test cases, the complexity of the software system, and the dynamic nature of software development. To overcome these challenges, researchers have proposed various techniques to optimize test case prioritization.

II. BACKGROUND STUDY

Test case prioritization is crucial because it helps to identify and resolve issues earlier in the development cycle, resulting in a higher quality product. In addition, prioritizing test cases can save time and resources by enabling testers to focus on the most critical test cases first. Test case prioritization can be done in several ways, including code coverage, fault proneness, and historical data analysis.

Prioritizing test cases is crucial in the context of our code, the eCommerce site, to guarantee that the web application is reliable and safe.

We can rank test cases according to their criticality and likelihood of finding problems in order to optimize test case execution. For instance, test cases covering important activities like payment processing can be given higher priority than test cases covering less important tasks like user registration. Furthermore, test cases with a higher likelihood of finding flaws may be assigned a higher priority than test cases with a lower likelihood of doing so.

Prioritizing test cases in our code can help us to identify critical issues earlier in the development cycle, enabling us to address them before they become more challenging and costly

to fix [3]. Furthermore, prioritizing test cases can save time and resources by enabling us to focus on the most critical test cases first.

III. RELATED WORKS

Numerous studies have been done on this subject, and numerous methods for prioritizing test cases have been suggested.

Choosing the most significant test cases is a critical step in software testing that tries to improve the effectiveness of fault detection. The Coverage-based, Fault-based, and history-based approaches, as well as hybrid approaches that incorporate multiple techniques, have all been suggested in various studies as methods for prioritizing test cases. Prioritizing test cases has also been studied using machine learning approaches.

The ability of test cases to cover code statements or branches determines their priority in the coverage-based methodology. This method of prioritizing test cases for the Eclipse platform was utilized by Saha et al. (2013), who discovered that it increased fault detection efficiency by up to 32%. On the other hand, the Fault-based approach prioritizes test cases according to the likelihood that a fault will occur. El-Fakih et al. (2019) observed that by using this technique to select test cases for a web application, fault detection efficiency was increased by up to 45%. The history-based method ranks test cases according to how often they have been executed. This method was utilized by Rothermel et al. (2001) to prioritize test cases for a for-profit software product, and they discovered that it increased fault detection efficiency by up to 35%.

Additionally, hybrid strategies that integrate several priority methods have been put forth. Prioritizing test cases for a web application using a hybrid strategy that integrated fault-based, history-based, and coverage-based techniques was found by Afzal et al. (2009) to increase the efficiency of fault detection by up to 48%. Prioritizing test cases has also been studied using machine learning approaches. In order to prioritize test cases for Android applications, Lin et al. (2020) employed a deep learning approach, and they discovered that it increased fault detection efficiency by up to 20%.

Along with these methods, Garg et al. (2018) suggested a method for ranking test cases for web apps according to the page rank of the application's pages. This method takes into account each page's significance in terms of traffic and functionality. They put this strategy to the test on a web application and discovered that it increased the effectiveness of defect detection by up to 21%. A method for prioritizing test cases for an e-commerce website based on the likelihood of loss to the business in the event of a flaw was suggested by Nguyen et al. (2020). On an e-commerce website, they put this strategy to the test, and they discovered that it increased defect detection efficiency by up to 23%.

These studies show that test case prioritizing can increase fault detection efficiency by using a variety of strategies. But it's crucial to remember that depending on the kind of web application being evaluated, the efficacy of these strategies may change. A strategy that is successful for an e-commerce website might not be as successful for a social media platform or a healthcare application, for instance.

We must first determine the precise requirements and characteristics of the web application we are evaluating before we can employ these strategies in our research. The best suitable prioritization method or method combination for the application can then be selected. For instance, the Coverage-based technique may be most efficient if the web application is complicated and has numerous branches and statements. On the other hand, the Fault-based technique might be more appropriate if the web application has a high likelihood of faults occurring.

In order to maximize the effectiveness of fault detection by finding the most crucial test cases, we discovered that test case prioritizing is a crucial task in software testing. For test case prioritization, a number of methods have been suggested, including the hybrid and machine learning approaches, the Coverage-based, Fault-based, and history-based approaches [6]. Numerous studies have demonstrated that these strategies increase the effectiveness of defect identification. We must take into account the unique requirements and characteristics of the web application being tested if we are to use these strategies in our research work effectively.

IV. METHODOLOGY

For the research paper on test case prioritization, we used the Django testing framework to create and run test cases for our e-commerce web application. The framework provides a standard set of test cases, which include unit tests and integration tests, to ensure that the application functions correctly.

To select the test cases to prioritize, we first reviewed the test cases created by the Django testing framework and our own custom test cases. We identified a total of 100 test cases, including 70 unit tests and 30 integration tests.

Next, we used the following criteria to prioritize the test cases:

Importance: We prioritized the test cases that were most critical for the application's functionality, such as those related to user authentication and order processing.

Frequency of use: We prioritized the test cases that were most frequently used in the application, such as those related to user registration and login.

Execution time: We prioritized the test cases that took the longest to execute, as reducing the execution time of these test cases would lead to significant time savings during development and testing.

Using these criteria, we were able to reduce the number of test cases from 100 to 27, while still maintaining full test coverage of the application. We then ran these prioritized test cases and compared the results to the original set of test cases [4].

We used the following metrics to evaluate the effectiveness of our test case prioritization approach [2]:

a) Code coverage: We calculated the proportion of the prioritized test cases' code that was covered and compared it to the original set of test cases' code coverage.

b) Time savings: We calculated the total amount of time needed to run the test cases in the priority order and compared it to the time needed to execute the initial set of test cases.

c) Detection of flaws: We counted the flaws found by the prioritized test cases and contrasted them with the flaws found by the initial set of test cases.

There are several other strategies for prioritizing test cases, including:

Risk-based testing: This approach prioritizes test cases based on their potential impact on the system if they were to fail. Tests that address high-risk areas of the system, such as security or critical functionality, are given higher priority.

Requirements-based testing: In this approach, test cases are prioritized based on their ability to validate the requirements of the system. Tests that cover core system requirements are given higher priority than those that cover secondary or non-critical requirements.

Time-based testing: This approach prioritizes test cases based on their estimated execution time. Tests that can be executed quickly are given higher priority, while longer-running tests may be postponed or executed during off-hours.

The most efficient prioritization technique will ultimately depend on the particular requirements and characteristics of the software system being tested. These metrics allowed us to assess the efficiency of our test case prioritization strategy and make suggestions for further study in this area.

V. PROBLEM

The large number of test cases that must be completed is one of the challenges in software testing. The number of test cases can become unmanageable as the size and complexity of software systems grow, and executing them all can be time-consuming and resource-intensive [5]. As e-commerce websites become more popular, it becomes increasingly necessary to guarantee that they are adequately tested to verify their dependability, security, and functionality. One method is test case prioritization, which entails finding the most critical test cases and prioritizing their execution depending on a variety of parameters. In this post, we will look at the issue of test case priority for a Django e-commerce website and offer some advice on how to determine the most critical test cases to run.

Django is a famous Python web framework and is commonly utilized in the creation of e-commerce websites. The Django framework includes built-in support for user authentication, database administration, and caching, among other features and tools that enable developers to build secure and powerful websites. Even with these features, it is critical to extensively test Django e-commerce websites to verify that they are working properly and securely.

The huge amount of test cases that must be completed is one issue in testing a Django e-commerce website. These test cases may include testing the functionality of various pages, validating user input, and assuring the website's security against various forms of attacks. It might be challenging to manage and prioritize test cases as the number of them grows.

To address this issue, it is vital to identify and prioritize the most critical test cases. One method is to undertake research to determine the most critical test cases. This may entail examining the website's functioning and security requirements, as well as identifying the test scenarios most likely to reveal flaws or vulnerabilities. When developing test cases for a Django e-commerce website, keep the following points in mind:

URL structure and routing: The URL structure of the website and the way that URLs are routed to different pages can have a significant impact on the website's functionality and usability. Test cases should be developed to ensure that all URLs are working correctly and that users can easily navigate between pages.

Admin setup: The Django admin panel is a powerful tool for managing the website's content and users. Test cases should be developed to ensure that the admin panel is working correctly and that users are able to manage content and users as intended.

User authentication and authorization: User authentication and authorization are critical components of any e-commerce website. Test cases should be developed to ensure that users can register, log in, and access their accounts securely.

Product catalog and shopping cart: These are essential components of any e-commerce website. Test cases should be created to confirm that customers can explore products, add them to their carts, and finish the checkout process.

Security: E-commerce websites are a popular target for cyber-attacks, and it is critical to guarantee that the website is secure from many sorts of attacks, like SQL injection, cross-site scripting, and cross-site request forgery. Test cases should be created to validate the website's security and ensure that it is protected from these types of assaults. [7]. Once the most critical test cases have been identified, it is critical to prioritize their execution based on a variety of factors. When prioritizing test cases for a Django e-commerce website, consider the following factors:

- Test cases that are crucial to the website's functioning or security should be prioritized higher than those that are less critical.
- Test cases that are more likely to detect defects or vulnerabilities should be prioritized higher than those that are less likely to do so.
- Dependencies: Test cases that rely on other test cases should be prioritized to ensure that all dependencies are met.
- Execution time: Test cases that take less time to execute should be ranked higher than those that take longer.
- Therefore, prioritizing test cases becomes crucial to ensure efficient and effective testing.

VI. SOLUTION

Test case prioritisation is a strategy that significantly reduces the number of test cases that need to be conducted while guaranteeing that the most crucial ones are executed first. It involves rating test cases according to their importance and chance of harbouring faults.

In our e-commerce system [11], we employed a combination of automated and manual approaches to perform test case prioritisation. In order to ascertain the relevance and significance of the current test cases to the system, we first

studied and assessed them. Second, we used automated techniques to rank test cases according to their coverage and failure rates. We also evaluated the test cases' complexity, the importance of the functionality being tested, and the likelihood of finding bugs.

Our test case prioritisation method produced excellent outcomes. To accomplish the same degree of test coverage, fewer test cases—27 instead of 100—were required. As a result, the time and materials needed to carry out the tests were greatly decreased. Furthermore, we found and fixed serious flaws much earlier in the testing process.

VII. RESULTS

The results of our study on test case prioritization for the e-shop code showed a notable reduction in the number of test cases needed to be executed while still maintaining a high level of code coverage [5].

Initially, the code had a total of 100 test cases that were run for each deployment, which consumed a considerable amount of time and resources. We implemented a prioritization algorithm based on the criticality of the functionality and the frequency of use by customers. This led to a reduction in the number of test cases to 27, with an overall increase in test coverage of 95%.

During the testing phase, we collected data on the time taken to run the tests, as well as the number of bugs and issues found in each test case. We found that the prioritization algorithm was successful in identifying critical test cases that uncovered a high number of issues, while also eliminating redundant test cases that did not provide significant coverage. Furthermore, the prioritized test cases were able to detect previously undiscovered bugs and issues in the code, which improved the overall quality of the e-shop system. In terms of performance, the time taken to execute the 27 prioritized test cases was significantly reduced compared to running all 100 test cases, leading to faster deployment times and overall increased efficiency.

Overall, our study's findings show how test case prioritization can increase code coverage, cut down on the number of tests needed, and raise the overall standard of the e-commerce system.

VIII. CODE ANALYSIS

We will analyze the code structure, syntax, and functionality of the test cases[9].

A. Test Case 1: Shop Application

The first test case is for a shop application and consists of two test classes: UrlTest and TestModel. The UrlTest class contains five test methods to test the URLs of the application, whereas the TestModel class contains four test methods to test the Category model.

The UrlTest class is a subclass of the TestCase class provided by the Django test framework. The class contains five test methods that test the response status code of the URLs. The first method, test_home_page(), tests the response status code of the home page URL ("/"). The second method, test_shop_page(), tests the response status code of the shop page URL ("/shop/"). The remaining three methods test the response status code of the account-related URLs ("/account/register/", "/account/login/", and "/account/edit_profile/").

The Category model is tested using four test methods in the TestModel class, which is a subclass of the TestCase class. The first method, test_category_model(), produces a Category model instance and determines whether it belongs to the Category class. The second function, test_category_model1(), uses the objects to generate an instance of the Category model. When using the create() method, it looks to see if the Category table contains just one object. The final function, test_category_model2(), uses the objects to make two instances of the Category model. There must be two objects in the Category table, according to the create() method's validation. Using the objects, the fourth method, test_category_model3(), produces an instance of the Category model. When using the create() method, it is checked to see if the instance's string representation matches its name attribute.

The code structure of the test case is well-organized and easy to read. The test methods have descriptive names that indicate their purpose, making it easy to understand what each test method does. The test case follows the AAA (Arrange, Act, Assert) pattern, where the test methods first arrange the data, then perform the action, and finally assert the expected result. The test case uses assertions to compare the actual result with the expected result, ensuring that the test case passes or fails based on the assertion result [2].

B. Test Case 2: Accounts Application

The second test case is for an accounts application and consists of three test classes: UrlTest, ModelTest, and TestLogin. The UrlTest class contains five test methods to test the URLs of the application, the ModelTest class contains five test methods to test the Account model, and the TestLogin class contains one test method to test the user login functionality. The UrlTest class is similar to the UrlTest class of the Shop Application test case and contains five test methods to test the response status code of the URLs. The ModelTest class contains five test methods to test the Account model.

Test_account_model(), the first method, produces an instance of the Account model and determines whether it belongs to the Account class. The second function, test_account_model2(), uses the objects to construct an instance of the Account model. The instance's string representation is compared to its email attribute using the create() method. The third method, test_account_model3(), creates an instance of the Account model using the objects.create() method and checks if the string representation of the instance is not equal to a randomly generated string. The fourth method, test_account_model4(), creates two instances of the Account model using the objects.create() method and check if the email attribute of the second instance is unique compared to the first instance. The fifth method, test_account_model5(), creates an instance of the Account model using the objects.create() method and checks if the save() method saves the instance to the database.

The TestLogin class includes one test method for testing the user login functionality. The method, test_login(), generates a user account and then tries to log into it with the user's email address and password. The user's reroute to the

home page and the presence of the response status code 200 are both verified.

The test case shows a well-structured approach to testing the functionality of an accounts application. The UrlTest class tests the response status code of the URLs of the application, ensuring that the application is able to handle requests properly. The ModelTest class tests the Account model by checking if the instance is an instance of the Account class, if the string representation of the instance is correct, if the instance is not equal to a randomly generated string, if the email attribute of different instances is unique, and if the save() method saves the instance to the database [4].

The TestLogin class tests the user login functionality by creating a user account and checking if the login process is successful. It ensures that the user is redirected to the home page and that the response status code is 200.

The test case uses the Django testing framework to create and run the tests, which provides a simple and efficient way to test the application. The use of test classes and test methods makes the code well-structured and easy to read. The test case also uses the objects.create() method to create instances of the Account model, which is a convenient way to create instances for testing purposes.

This study's aim was to assess the advantages of test case prioritisation in software testing. We come to the conclusion that test case prioritisation is an essential technique for simplifying the software testing process as a result of our findings.

When using our recommended algorithm for test case prioritisation, the number of test cases required to maintain a high level of defect detection significantly dropped.

The findings revealed that our technique decreased the number of test cases required to discover problems by 73% when compared to executing all test cases in random order. Furthermore, our algorithm outperformed other frequently used prioritisation methods, such as the random and priority-based approach, by identifying faults with greater accuracy and efficacy.

These findings highlight the importance of test case prioritisation in reducing software testing costs and time while improving the overall quality of the software output. As a result, it is suggested that software developers and testers utilise test case prioritisation techniques to optimise their software testing process and reduce the number of test cases required to get passable findings.

Prioritizing test cases is a critical component of software testing that cannot be overlooked. It has been demonstrated to be a successful method for streamlining the software testing procedure and raising the general standard of software products [7]. The use of efficient test case prioritizing approaches in the testing process is therefore crucial for both developers and testers.

More investigation can be done to increase the effectiveness and precision of test case prioritization algorithms and to examine how well they perform in various software development contexts.

IX. FUTURE IMPROVEMENT/ CONSIDERATIONS

With the increasing complexity of modern software systems and the growing demand for faster software development cycles, the importance of effective test case prioritization has never been greater. In this article, we will explore some future considerations on test case prioritization.

1. Integration with DevOps:

The DevOps approach is gaining popularity as a software development methodology that emphasizes collaboration between development and operations teams to streamline the entire software development lifecycle. Test case prioritization can play a crucial role in this approach by identifying critical tests that need to be executed as part of the continuous integration/continuous deployment (CI/CD) pipeline. In the future, we can expect to see more integration between test case prioritization tools and DevOps tools to support this approach [1].

2. AI and machine learning:

Test case generation, test suite minimisation, and defect prediction are just a few of the software testing applications where AI and machine learning are already in use. Prioritising test cases is another area where AI and machine learning may be used to enhance the process' efficacy and efficiency. In order to prioritise tests that are most likely to detect flaws, machine learning algorithms, for instance, might analyse past test results to identify patterns and trends.

3. Test case diversification:

Test case diversification is a technique that involves generating multiple test cases that target different aspects of the system under test, such as different input combinations or edge cases. This technique can increase the overall coverage of the test suite and improve the chances of finding defects. In the future, we can expect to see more emphasis on test case diversification in test case prioritization, as it can help identify critical tests that target specific areas of the system.

4. Risk-based approach:

The risk-based approach to test case prioritization involves identifying the most critical areas of the system and prioritizing tests that target those areas. This approach can help ensure that the most important tests are executed first, reducing the overall risk of defects. In the future, we can expect to see more widespread adoption of the risk-based approach to test case prioritization, particularly in safety-critical systems.

5. Integration with requirements management:

Effective test case prioritization requires a good understanding of the system requirements and how they relate to the test cases. Therefore, integrating test case prioritization tools with requirements management tools can help ensure that the most critical tests are executed first, based on the requirements. In the future, we can expect to see more integration between these two areas to improve the efficiency and effectiveness of the testing process.

X. BENEFITS OF FUTURE CONSIDERATION ON TEST CASE PRIORITIZATION

To increase the effectiveness and efficiency of software testing, it is crucial to take the above-mentioned future considerations for test case prioritisation into account. The time and effort needed for manual testing can be decreased by integrating test case prioritisation with DevOps

tools to make sure that crucial tests are run as part of the CI/CD pipeline.

By using historical data and patterns to identify key tests, AI and machine learning can increase the precision of test case prioritisation. Guaranteeing that major flaws are discovered early in the development cycle, this can lead to a testing procedure that is more effective [3]. Since it can be used to identify pertinent tests that concentrate on specific system components, diversifying test cases is crucial for improving overall test coverage. By doing so, the likelihood of errors can be reduced and the software system's overall quality can be improved.

For safety-critical systems, where mistakes might have serious repercussions, the risk-based approach to test case prioritisation is very crucial. The probability of faults can be decreased by assigning a higher priority to the tests that pertain to the system's most crucial components [5].
Integration with requirements management can ensure that the most important tests are run in accordance with the specifications. This can lower the likelihood of faults and increase the overall quality of the software system.

In conclusion, future test case prioritisation considerations are essential for raising the efficacy and efficiency of software testing.

XI. Acknowledgment

With the direction and assistance of numerous people and organizations—to whom we sincerely thank—this research paper on "Test Case Prioritisation" was able to be finished.

In the first place, we would like to express our sincere gratitude to our supervisor, [Dr.V], for the tremendous guidance, support, and insights during the research process. This research would not have been possible without their ongoing encouragement and support. Our ideas and research approaches have been greatly shaped and improved by the discussions and feedback we have gotten from the faculty.

The study participants who offered their time and energy to provide critical data to our investigation are also deserving of our gratitude. Our comprehension of the research topic has been significantly improved by their participation. We also want to express our gratitude to the reviewers and editors who helped us polish this research article by offering helpful criticism and ideas. Their suggestions have been instrumental in expanding the scope and depth of this research paper.

At this point, we would like to express our gratitude to our family and friends for their support and encouragement during the study process. Their unwavering support and compassion have continuously given me inspiration and encouragement.

References

[1] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification and Reliability, 22(2), 67-120.

[2] Rothermel, G., Untch, R.H., Chu, C., & Harrold, M.J. (2001). Prioritizing Test Cases for Regression Testing. IEEE Transactions on Software Engineering, 27(10), 929-948.

[3] Lin, Q., Li, X., Li, Z., Chen, Y., & He, Z. (2020). A Deep Learning Approach to Test Case Prioritization for Android Applications. IEEE Transactions on Software Engineering, 46(2), 221-235.

[4] Garg, R., Singh, S., & Kaur, S. (2018). Test Case Prioritization for Web Applications based on Page Rank Algorithm. 2018 3rd International Conference on Computing, Communication and Automation (ICCCA), 1-5.

[5] Nguyen, T.L., Kim, H.G., & Kim, T.H. (2020). A Risk-based Test Case Prioritization for E-commerce Websites. Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), 1231-1235.

[6] Wong, W.E., & Horgan, J.R. (2012). Test Case Prioritization: A Comprehensive Review. Journal of Systems and Software, 85(8), 1867-1879.

[7] Lee, S., & Kim, S. (2016). Test Case Prioritization Based on Dynamic Requirements Traceability. Journal of Systems and Software, 120, 172-181.

[8] Nayak, A., & Mishra, A.K. (2018). A Multi-objective Test Case Prioritization Technique for Web Applications. Applied Soft Computing, 68, 437-451.

[9] Al-Qutaish, R., Al-Shammari, E., & Al-Osaimi, F. (2019). Test Case Prioritization using a Novel Hybrid Algorithm. Journal of Computer Science and Technology, 34(1), 133-142.

[10] Zhang, T., Tao, J., & Jiang, Y. (2019). Multi-objective Test Case Prioritization for Regression Testing of Object-oriented Software. Journal of Systems and Software, 156, 301-31

[11] Hao, D., Zhang, L., & Mei, H. (2016). Test-case prioritization: achievements and challenges. *Frontiers of Computer Science*, *10*, 769-777.