Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2021, 8(6): 82-87

Research Article ISSN: 2394-658X

Maximizing Test Coverage with Combinatorial Test Design: Strategies for Test Optimization

Kodanda Rami Reddy Manukonda

Email id - reddy.mkr@gmail.com

ABSTRACT

In order to maximize coverage and improve test suites, this study focuses on the significance of test coverage in software development. In order to find flaws, improve software quality, and boost customer trust in the product, test coverage is essential. The three primary forms of test coverage—requirements coverage, code coverage, and fault coverage—are defined and discussed in the paper. A method for optimizing test suites by determining the bare minimum of tests required for thorough coverage is highlighted: combinatorial test design. Increasing test automation and optimizing device coverage are two tactics for maximizing test coverage. The exploration likewise covers combinatorial test suite age strategies, including FireEye, which utilizes the Initial public offering calculation to create productive test suites. In rundown, the examination features the meaning of test inclusion in programming advancement and offers important viewpoints on strategies and assets to work on the viability and proficiency of test inclusion.

Keywords: Test coverage, software development, combinatorial test design, test automation, device coverage, FireEye, IPO algorithm.

INTRODUCTION

Ensuring the quality and stability of software products is vital in the ever-evolving field of software development. Test coverage—a measure of how thoroughly a software system has been tested—is essential to reaching these objectives. Test coverage plays a critical role in finding and reducing potential flaws, boosting software quality, and building trust in the product [1].

A. Defining Test Coverage

By identifying the sections, features, or functionalities that were tested during the testing process, test coverage calculates the degree to which a software system has undergone testing. It is a statistic that helps determine any weaknesses in the testing approach and assesses how comprehensive the testing was [2]. Requirements coverage, code coverage, and fault coverage are metrics that may be used to quantify test coverage.

[1]. Needs-Based Coverage: This kind of test coverage guarantees that there are test cases for each of the criteria listed for a software system. By ensuring that all planned functionality is thoroughly tested, it lowers the possibility of missing important features.

[2]. Code Coverage: This quantifies the percentage of code that has been run through testing. It assists in evaluating the level of exercise performed on the codebase by pointing out any untested or dead code. The proportion of lines, statements, branches, or pathways that the tests cover is typically used to calculate code coverage.

[3]. Fault Coverage: This type of coverage concentrates on pinpointing the particular flaws or problems that have been found during testing. It assesses how well tests identify possible problems and contributes to raising the software's overall dependability.

B. Combinatorial Test Design

One of the most important stages of the test life cycle is test design, which establishes the amount and kind of data required for testing, as well as the amount of testing that will be done throughout execution. It also defines what tests will eventually be automated for future execution [3].

When an optimized test suite is produced, which is equivalent to the fewest tests necessary to provide the necessary test coverage, combinatorial testing as a design method becomes extremely beneficial. The rationale of the solution appeals to customers and teams alike, but adoption frequently demands a bigger transformational approach; in an environment where testing prowess is traditionally assessed by the number of tests being conducted. This strategy is thought to be disruptive in addition to being novel.

One of the most important stages of the test life cycle is test design, which establishes the amount and kind of data required for testing, as well as the amount of testing that will be done throughout execution. It also defines what tests will eventually be automated for future execution. When an optimized test suite is produced, which is equivalent to the fewest tests necessary to provide the necessary test coverage, combinatorial testing as a design method becomes extremely beneficial [4]. The rationale of the solution appeals to customers and teams alike, but adoption frequently demands a bigger transformational approach; in an environment where testing prowess is traditionally assessed by the number of tests being conducted. This strategy is thought to be disruptive in addition to being novel.

[1]. Consistency: Automated combinatorial testing methods provide consistency in the creation, execution, and assessment of test cases. This uniformity reduces the chance of human error while ensuring that testing is done methodically and consistently.

[2]. Efficiency: Combinatorial testing tools automate the creation of test cases by meticulously examining the combinations of input parameters and their values. This approach is more effective than manual testing and can quickly cover a large number of test scenarios.

[3]. Resource Optimization: Combinatorial testing tools aid in the optimization of testing resources by focusing on the most crucial combinations of input parameters.

[4]. Coverage: These techniques aid in the identification of potential flaws resulting from parameter interactions by offering comprehensive coverage of all possible input parameter combinations.

[5]. Risk Reduction: By carefully testing combinations of input parameters, combinatorial testing techniques lower the likelihood of software errors caused by unforeseen interactions or dependencies between parameters.

[6]. Time Savings: Testers no longer have to spend their time manually building and managing large sets of test cases thanks to automated combinatorial testing technologies.

We use combinatorial testing tools because manual combinatorial testing on a large number of input parameters can be a laborious task due to the complex procedure involved in combinatorial testing. These tools are not only simple to use with a wide range of input parameters, but they can also generate test configurations in accordance with the constraints added to the input parameters. Online resources offer a plethora of tools for combinatorial testing. We will talk about a few of these free online tools for creating test configurations in this article.

C. Techniques for Maximize Test Coverage

[1]. Increase Test Automation

It is not possible to rely solely on manual testing to guarantee that a significant number of tests are completed by a deadline. The necessary tests for automation testing, particularly parallel testing, must be completed in a matter of days or hours. Using a cloud-based testing service that allows automation on real browsers and devices is the simplest way to conduct automated tests. With BrowserStack's cloud Selenium grid of more than 3000 browsers and actual devices, automated Selenium testing is simple [5]. The grid supports parallel testing, speeding up their builds and resulting in speedier releasesWith pre-built interfaces spanning over 20+ programming languages and frameworks, BrowserStack Automate integrates effortlessly into current CI/CD processes by providing plugins for all major CI/CD platforms.

[2]. Maximize Device Coverage

Try as many combinations of OS, browser, and device as you can. There are many versions of every OS, browser, and device type that need be taken into account. Once more, the ideal option is to select and make use of a cloud-based testing platform such as BrowserStack that provides access to both modern and antiquated hardware, operating systems, and browsers. Users have access to more than 3000 genuine devices and browsers for testing, as was indicated in the preceding section. Tests may be conducted, for instance, using Chrome on a Samsung Galaxy S20, Safari on an iPhone 14, and similar apps.

LITERATURE REVIEW

Bezney [6] The counteraction and the board of SARS-CoV-2 disease rely intensely upon continuous and dependable RT-PCR-based testing; regardless, time and assets are still habitually imperatives for dynamic contamination observation. Cleary et al. show that, both hypothetically and as tried in vitro utilizing human swab and sputum tests, considering populace level viral commonness and individual viral burdens licenses proficiency benefits upon pooled testing with least loss of awareness. Barak et al. exhibit the viability of the technique by and by pooling testing 133,816 clinic gathered patient nasopharyngeal examples, which eliminated 75% of testing reactions with a tiny bit of diminishing in responsiveness. The two outcomes show that SARS-CoV-2 testing throughput may be reliably expanded via cautiously pooling individual examples preceding testing.

Grédiac and Pierron, [7] Inside the field of trial mechanics, full-field optical estimations, for example, computerized picture relationship and the framework approach have achieved a worldview change. Despite the fact that there have been huge headways in converse recognizable proof procedures, for example, the virtual fields strategy or limited component model refreshing, the ongoing test techniques, which date back to the times of strain checks and direct factor relocation transducers, are commonly not appropriate to the abundance of data presented by these new estimation apparatuses. This work offers an outline of the writing on the turn of events and improvement of heterogeneous mechanical tests to decide material properties from full-field information, which are alluded to as Tests for Material 2.0 (MT2.0).

Khoshnevisand Fathi, [8] One of the most urgent objectives of programming advancement is reusability, especially in programming product offering (SPL) designing, which incorporates errands connected with examination, plan, execution, testing, and upkeep. Reusability should subsequently be given cautious thought in programming product offering testing as well as different undertakings. Reusability in SPL testing not set in stone and measured in a few strategies. To further develop reusability in SPL testing (SPLT), we initially present four different reusability measurements in this paper. Then, we tentatively research how two of the proposed reusability measurements can be impacted (improved) by a search-based software testing (SBST) approach for upgrading a current SPL space test suite. The two test reusability measurements — TSRR (test suite reusability regarding test requirements) and TCRR (test case reusability regarding test requirements) — that were picked for the trial and error on 20 SPL highlight models of size 5000 showed a huge improvement in enhanced arrangements when contrasted with non-upgraded arrangements.

Skillet [9] There is a developing opportunity that noxious inserts, frequently alluded to as equipment Trojans, could think twice System-on-Chip (SoC) (SoC) plans because of the overall semiconductor production network. Sadly, because of the outstanding info space intricacy of current SoCs, conventional reenactment-based approval utilizing a great many test vectors isn't appropriate for recognizing secretive Trojans with profoundly exceptional trigger circumstances. To ensure solid SoCs, making powerful Trojan recognition methods is basic. Albeit certain test creating frameworks show guarantee, their versatility and discovery exactness are seriously restricted. In this exploration, we consolidate testability examination and support figuring out how to introduce a special rationale testing method for Trojan ID. This work explicitly gives three huge commitments. 1) We enormously improve the trigger inclusion over past strategies by using both controllability and recognizability examination notwithstanding the uncommonness of signs. 2) Test age time is fundamentally abbreviated while support learning is utilized, all without compromising test quality. 3) In light of exploratory outcomes, we can essentially beat cutting edge strategies concerning trigger inclusion (normal of 14.5%) and test age time (normal of 6.5 times).

Yan et al., [10] various brain network inclusion rules (e.g., model accuracy against ill-disposed attacks) have been proposed to direct the testing of brain network models, enlivened by the huge outcome of using code inclusion as direction in programming testing. It is as yet not completely self-evident, however, assuming that there is an equivalent monotonicity between brain network model inclusion and model quality, regardless of the way that various powerful examination in programming have shown a monotonic connection between code inclusion and programming quality. This audit intends to give a response to that question. Specifically, this paper researches the connection between inclusion measures and DNN model quality, the adequacy of inclusion-based retraining in contrast with past antagonistic preparation, the impacts of inclusion directed ill-disposed model age in examination with slope good based techniques, and the inner connections among inclusion rules.

TOOLS THAT GENERATE COMBINATORIAL TEST SUITES

Algebraic, greedy, or heuristic search algorithms are the three primary categories of algorithms used to build combinatorial test suites. The following is a summary of the main benefits and drawbacks of the methods used to create combinatorial test suites:

A. Algebraic approaches

[1]. It provides time-efficient constructs, but using algebraic methods to get correct results on a wide range of inputs might be challenging.

[2]. Due to their relative accuracy and time efficiency, greedy algorithms have been extensively researched for the creation of covering arrays.

[3]. In various cases up to this point, heuristic hunt — particularly while utilizing Simulated Annealing (SA) — has created the most reliable outcomes. Large numbers of the briefest test suites for different framework designs have been created by this nearby hunt method, but the test suite age process costs execution time.

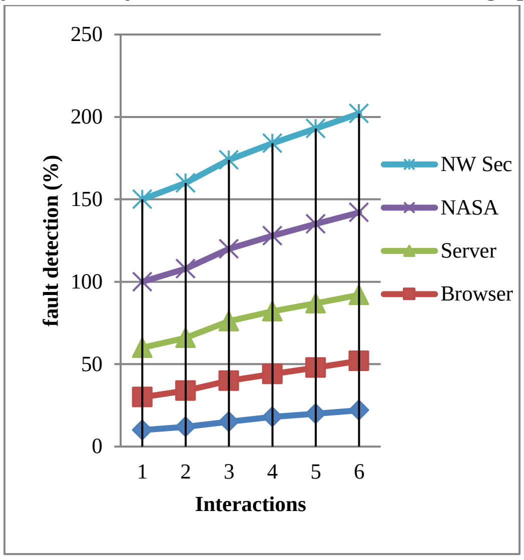


Figure 1: Interaction strengths 1–6 fault detection

For further information on these techniques, please refer to the aforementioned articles. We have also included an explanation of FireEye, a free research tool that may be used to create combinatorial test suites. [12] Initially introduced the IPO technique for pairs testing, whilelater expanded it to generic t-way combinatorial testing. The generic version of the IPO algorithm, known as IPOG, is implemented by the FireEye tool. We offer a synopsis of the algorithm as well as screenshots of the tool that applies it. As a result, the readers have a model to work from when creating their own test suites.

Table 1: Percentage of errors detected at interaction strengths ranging from 1 to 6

Interaction Strength	Med Dev.	Browser	Server	NASA	NW Sec
1	66%	38%	42%	65%	17%
2	98%	78%	70%	84%	65%
3	99%	96%	84%	85%	84%
4	100%	67%	96%	98%	95%
5	100%	98%	96%	98%	65%
6	100%	99%	95%	95%	45%

AUTOMATING TESTS WITH SELENIUM AND JAVA

Test automation in programming testing alludes to utilizing specific programming (not quite the same as the item being tested) to oversee test execution and analyze expected and genuine outcomes. Test automation can add seriously testing that would be trying to do physically or robotize certain dreary yet fundamental positions in a laid-out testing process.

A. Selenium

A structure for convenient programming testing of web applications is called Selenium. Without learning a test prearranging language, Selenium offers a creating/recording device (Selenium IDE). Besides, it offers Selena, a test space explicit language, for composing tests in an assortment of notable programming dialects, like Java, C#, Sweet, Perl, PHP, Python, and Ruby. Most of current internet browsers may then be utilized to execute the tests. Selenium is accessible for Mac, Linux, and Windows. It is allowed to download and utilize, open-source programming conveyed under the Apache 2.0 permit.

B. IDE for Selenium

The integrated development environment (IDE) for Selenium testing is called Selenium IDE (Figure 1). It can record, change, and investigate tests and is executed as a Firefox Extra. Selenium Recorder was its previous name [14].

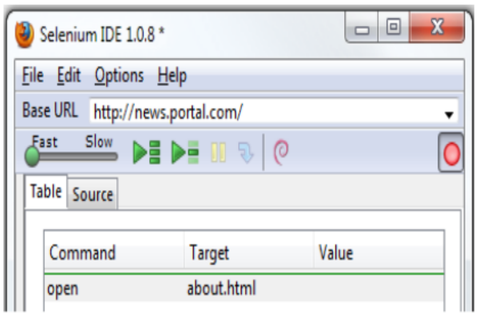
Recording. A common initial step for inexperienced users is to capture a test case of their online interactions. The record button is automatically turned on when Selenium-IDE is initially launched. To keep Selenium-IDE from beginning to record consequently, go to Choices > Choices... also, uncheck the container close to "Begin recording quickly on open" [15]. In light of your activities during recording, Selenium-IDE will consequently add directions to your test case. Generally, this will comprise of:

[1]. utilizing click or clickAndWait guidelines while clicking a connection
[2]. picking a choice from a drop-down listbox by composing the order;
[3]. embedding values
[4]. choosing radio buttons or checkboxes to execute an order

Coming up next are a few unsafe situations to be aware of:

[1]. For the composing order to record, you could have to tap on one more piece of the site
[2]. Tapping on a connection frequently starts a tick order.

Test cases might be executed on numerous spaces by utilizing the base URL. Test cases might be executed across different areas on account of the Base URL field situated at the highest point of the Selenium-IDE window. Expect that http://news.portal.com is a site, and http://beta.news.portal.com is an inside beta site. Test cases for these locales that beginning with an open assertion ought to as opposed to utilizing a flat out URL (e.g., beginning with http: or https:) as the boundary to open, utilize a relative URL. Then, by connecting the boundary of the open order to the furthest limit of the worth of Base URL, Selenium-IDE will develop an outright URL. The test case introduced underneath, for example, would be executed against http://news.portal.com/about.html (Figure 2).

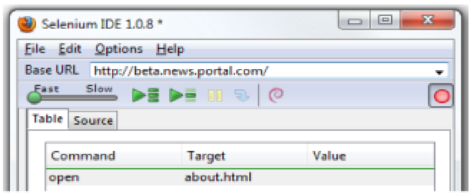


Base URL http://news.portal.com/

Command	Target	Value
open	about.html	

Figure 2: Executing Test Cases on a URL

The test case in question would be executed against http://beta.news.portal.com/about.html with a changed Base URL configuration. (Fig. 3):



Base URL http://beta.news.portal.com/

Command	Target	Value
open	about.html	

Figure 3: Executing Test Cases on another URL

Many commands in Selenium require a target. This target, which is composed of the location strategy and the location in the format "locatorType=location," identifies a component inside the web application's content. In many circumstances, the location type need not be specified. Below is an explanation of the different locator types along with examples.

CONCLUSION

To guarantee that testing procedures are reliable and comprehensive, test coverage is an essential component of software development. The study has brought attention to how crucial test coverage is for finding bugs, improving software quality, and boosting user confidence. It has been suggested that techniques like combinatorial test design are useful for optimizing test suites and reducing the total number of tests needed for thorough coverage. In order to improve test coverage efficiency, the study has also underlined the need of test automation and optimizing device coverage. It has been determined that resources such as FireEye, which implements the IPO technique, are useful for creating combinatorial test suites and enhancing test coverage.

REFERENCES

[1]. A. Mockus and N. N., "Test Coverage and Post-Verification Defects: A Multiple Case Study," in the 3rd International Symposium on Empirical Software Engineering and Measurement, 2009. ESEM 2009.

[2]. A. Rauf and S. Anwar, "Automated GUI Test Coverage Analysis Using GA," in Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations, pp. 1057-1062, 2010.

[3]. J. Takahashi, H. Kojima, et al., "Coverage Based Testing for Concurrent Software," in 28th International Conference on Distributed Computing Systems Workshops, 2008. ICDCS '08.

[4]. A. Jalote and P. Gupta, "An approach for experimentally evaluating effectiveness and efficiency of coverage criteria for software testing," in International Journal on Software Tools for Technology Transfer (STTT), vol. 10, no. 2, pp. 145-160, 2008.

[5]. M. W. Whalen, M. P. E. Heimdahl, et al., "Coverage metrics for requirements-based testing," in Proceedings of the 2006 International Symposium on Software Testing and Analysis, ISSTA 2006, pp. 25-35, 2006.

[6]. B. Cleary et al., "Using viral load and epidemic dynamics to optimize pooled testing in resource-constrained settings," Science translational medicine, vol. 13, no. 589, p. eabf1568, 2021.

[7]. F. Pierron and M. Grédiac, "Towards Material Testing 2.0. A review of test design for identification of constitutive parameters from full-field measurements," Strain, vol. 57, no. 1, p. e12370, 2021

[8]. M. Fathi and S. Khoshnevis, "Reusability Metrics in Search-Based Testing of Software Product Lines: An Experimentation," in 2021 26th International Computer Conference, Computer Society of Iran (CSICC), pp. 1-6, Mar. 2021.

[9]. Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in Proceedings of the 26th Asia and South Pacific Design Automation Conference, pp. 408-413, Jan. 2021.

[10]. S. Yan et al., "Correlations between deep neural network model coverage criteria and model quality," in Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 775-787, Nov. 2020.

[11]. J. Bergdahl et al., "Augmenting automated game testing with deep reinforcement learning," in 2020 IEEE Conference on Games (CoG), pp. 600-603, Aug. 2020.

[12]. S. Verma, M. Pant, and V. Snasel, "A comprehensive review on NSGA-II for multi-objective combinatorial optimization problems," IEEE access, vol. 9, pp. 57757-57791, 2021.

[13]. S. A. Al-Ahmed, M. Z. Shakir, and S. A. R. Zaidi, "Optimal 3D UAV base station placement by considering autonomous coverage hole detection, wireless backhaul and user demand," Journal of Communications and Networks, vol. 22, no. 6, pp. 467-475, 2020.