Webology, Volume 18, Number 5, 2021
ISSN: 1735-188X
DOI: 10.29121/WEB/V18I5/39

An Progression of Prioritization Techniques for Test Cases

Surendra Shukla[1], Bhasker Pant[2], Rajesh Upadhyay[3]

[1]Department of Computer Science & Engineering,Graphic Era Deemed to be University, Dehradun, Uttarakhand India, 248002
[2]Department of Computer Science & Engineering,Graphic Era Deemed to be University, Dehradun, Uttarakhand India, 248002
[3]School of Management, Graphic Era Hill University, Dehradun, Uttarakhand India, 248002

ABSTRACT
Prioritizing test cases involves sorting them according to the same predetermined criteria, with the end goal of catching bugs as soon as feasible and reducing testing expenses. The growing complexity of software systems means more test cases must be run in order to ensure accurate validation and verification, driving up both the time and money needed to complete these processes. The goal of any prioritising method is to ensure that the most important tests are executed before the least important ones. This document provides a historical perspective on the evolution of test case prioritising approaches and procedures for component-based software systems, beginning in 1999 and continuing up to the current day.

Keywords: Regression testing, software development life cycle, component based software system, test case prioritization.

INTRODUCTION
Testing software entails seeing how well the whole thing works, or at least how well its individual parts do (s). The software development life cycle is not complete without testing (SDLC). It accounts for nearly 50% of the full SDLC timeframe. Manual or automated testing methods are available. Generally speaking, large test suites benefit more from automated testing than from human testing.

When a change is made to an existing software system, regression testing is performed. When one part of an existing software system undergoes modification, it's likely that the change has already had an impact on another part of the system. The purpose of retesting is to ensure that a bug patch does not introduce new problems. To guarantee that the updated software didn't introduce any new flaws into the existing system, it's important to run both regression testing on the changed sections of the system and new tests on the complete system. Retesting a system can be time-consuming and taxing on computing resources due to the extensive size of the test suite. Minutes, hours, days, or even a month might pass during a retesting procedure. When re-testing a system, developers often get into trouble with how they're arranging the execution of the test cases. To address this problem, test cases might be prioritised.

Webology, Volume 18, Number 5, 2021
ISSN: 1735-188X
DOI: 10.29121/WEB/V18I5/39

Using a predetermined fitness value, test cases with a higher priority are run before those with a lower one. This method is known as test case prioritisation (TCP). Retesting is a crucial part of software upkeep, but it is also quite expensive. Software testers prioritise test cases so that costly tests may run earlier in the SDLC lifecycle, mitigating the effects of this expensive phase.

The most dynamic area of software engineering is the component based software system (CBSS). Many scientists and programmers have taken an interest in it. When compared to preexisting programmes, this method is more generic.

technique based on engineering. Among the many methods used to create working software systems, the Critical Path Software System (CBSS) is an important one. The CBSS may be used in various contexts and with numerous methods. The CBSS is made up of many parts, each of which was created by a different vendor. While this does boost the system's adaptability and flexibility, it also reduces the developer's workload. The two most significant aspects of the software development life-cycle, time and money, will be wasted.

Prioritization of Test Cases

Regression testing, which seeks to test the modified software system throughout software evolution by reusing the test suites of its previous version, first proposes test case prioritisation. TCP provides a method for prioritising the execution of test cases so that defects are found at the earliest possible moment in the software development life cycle. The pace of defect detection is the primary performance target while using TCP. It is important to run test cases in a manner that increases the rate at which bugs are found and that identifies high-risk bugs at an earlier stage of the testing phase of the SDLC.

TCP Statement of the Problem might be stated simply as:
You have a programme (x) and its altered version (x'), a set of tests (t), a collection of permutations (xt), and a function (f) that maps from xt to the real numbers.
Problem: You need to look for t ′ xt where (t″) (t″ xt) (t″ t′) [f (t′) f (t″)] holds true.
Here, f is a function that, when applied to any ordering of prioritizations of t, returns an award value for that ordering, and xt is the set of all possible orderings of prioritizations of t. The concept presupposes a preference for larger reward amounts over smaller ones [2].
TCP might be used for a variety of purposes, one of which is to speed up the pace at which errors are uncovered.

As a second goal, we'd want to boost the percentage of lines of code that are really used.
Thirdly, to improve the system's dependability. Prioritizing test cases aims to achieve a few different things. One of them is a higher rate of fault detection, or the discovery of errors at an earlier point in the testing process.
(2) To improve the early defect detection rate for high-risk issues.
Thirdly, we want to rapidly expand the amount of code in the system being tested that has been programmed to do tests.
Faster system reliability improvement is goal number four.
(5) To improve the likelihood of finding version-specific code modifications that cause regressions

Webology, Volume 18, Number 5, 2021
ISSN: 1735-188X
DOI: 10.29121/WEB/V18I5/39

early in the testing process.
Several techniques have been used to rank the order of the test cases:
One, TCP based on codes; two, TCP based on statements
3) TCP with a Branched Tree Topology
(4) Function based TCP
(5) TCP based on a model

LITERATURE REVIEW: AN EVOLUTION OF TEST CASE PRIORITIZATION

TCP and similar methods have been the subject of several publications during the last decade. The next portion of the article is an overview of the many different prioritising methods and associated research.
Year 1999

Prioritization of Test Cases: An Empirical Analysis

Gregg Rothermel et al. outline a number of ways for TCP and provide an empirical conclusion that quantifies how well these methods improve fault detection throughput in [1]. In the context of regression testing, an increased fault detection rate allows for early input to the system and aids debuggers in correcting faults at an earlier stage of testing. Several prioritising methods are compared and contrasted in the study, along with their respective benefits and drawbacks. Ten of TCP's methods are taken into account. Here is a list of them: Prioritizing test cases as follows: (1) not at all, (2) at random, (3) optimally, (4) with full branch coverage, (5) with enhanced branch coverage, (6) with full fault revealing potential.

There are three types of priority to consider: (9) prioritising more statement coverage, (10) prioritising entire statement coverage, and (11) prioritising prioritising exposure of further faults. Prioritizing may increase the rate at which faults are discovered, but this work also raises the issue of how different types of prioritisation addressed in the paper compare to one another in terms of this metric. We can see from the results that using these prioritising strategies makes the provided test suites much more effective at finding errors quickly. In addition, even the simplest prioritising methods had the same outcome. Year 2000

Set Regression Test Priorities

Some research questions, such as (1) whether or not prioritisation is effective when applied to modified versions of programmes, (2) the nature of the trade-offs between prioritisation techniques like fine granularity prioritisation and coarse granularity prioritisation, and (3) whether or not the inclusion of a measure of fault propensity improves the effectiveness of prioritisation techniques, were discussed by Sebastian Elbaum et al. In this work, we provided fourteen distinct TCP methods, which we divided into three categories. The first group, the control group, consists of two prioritising strategies, (1) random ordering and (2) optimum ordering, that are used as experimental control procedures. The statement level group consists of four granular methods: (3) complete statement coverage prioritising; (4) statement level aggregation; and (5) statement level reordering (4) the priority of extra statement coverage, total fault exposing potential, and further fault exposing potential. In the third category, called the function level group, eight coarse grain prioritising methods include (7) entire function coverage prioritisation (8) Prioritization of extending coverage

Webology, Volume 18, Number 5, 2021
ISSN: 1735-188X
DOI: 10.29121/WEB/V18I5/39

to other functions The possibility for prioritising at the function level is seen in (9). ten) an error that may reveal prioritising at the function level Prioritization of the Total Fault Index (12) Prioritizing fault indices further Both the total fault index (13) and the extra fault index (14) may be used to prioritise coverage in the event of a fault revealing a vulnerability. Over the course of the test suites' execution, the metric weighted average percentage of fault detected (APFD) is calculated and analysed. Higher APFD values indicate a faster rate of fault detection, with values closer to 100 indicating perfect fault detection. We utilised the APFD measure to rank the various methods, the ANOVA analysis to determine whether or not they were distinct, and the Bonferroni test to determine the nature of those differences. Dangers to the reliability of an experiment are also discussed. All all, we've covered three potential dangers: (1) internal validity threats, (2) external validity threats, and (3) construct validity risks. The results reveal that the aforementioned prioritising strategies have different problem detection rates, and they also demonstrate the implications of these findings in practise. Year 2001 Prioritization of

Regression Test Suites: Analyzing and Measuring Variation in Prioritization Methods

Additionally to the target programme and TCP approaches, the unknown variation found by Sebastian Elbaum et al. in [3] indicates that there are other elements that impact the prioritising efficacy. Prioritization strategies, as described by Sebastian Elbaum's earlier work, greatly increase the pace at which errors are discovered. Even more, he clarified that the rate of problem detection is local to the software being tested. Three research questions were prompted by the aforementioned discovery: (1) whether there are factors other than the prioritisation techniques and the tested programme; (2) what type of metrics are used to represent each factor; and (3) was the inclusion of other factor affect the prioritisation techniques. These inquiries were formulated after a series of experiments were run on three factors: (1) the nature of the programme, (2) the make-up of the testing suite, and (3) the nature of the changes themselves. Two kinds of variables, dependent and independent, were employed in experiments. The application of the APFD metric is the dependent variable; the four categories of independent variables are the subject programme, the prioritising approaches, the version and updates introduced to the programme, and the features of the test suite. There are a total of eight programmes in the experiment, each of which has a "baseline" version and twenty-nine "fault" versions. There is a predefined set of test cases for each major revision of the base code. The following are short descriptions of the prioritising methods that have been chosen from those reported in past works. The seven levels are as follows: (1) full function coverage, (2) full statement coverage, (3) full function coverage, (4) full statement coverage, (5) full fault index, (6) full fault index, (7) full fault index, and (8) optimum. The major goal of this analysis is to comprehend the program's complexity, the information supplied by each variable, and the predictability of the association among all variables.

The gathered data and measurements have been subjected to a principal component analysis (PCA). Through PCA, we learned that there are six main causes of the observed differences. Regression analyses have been used to determine how one dependent variable relates to a set of independent variables. It was a two-stage process to do the regression analysis. The first stage included comparing the results of single-variable regression analysis to the APFD measure criteria; the second involved doing multiple regressions to better understand the elements that influence various kind of prioritising. New variables contribute to the development of more potent TCP procedures, as

Webology, Volume 18, Number 5, 2021
ISSN: 1735-188X
DOI: 10.29121/WEB/V18I5/39

shown by the study's findings, and the strong prediction capacity of regression analysis aids in evaluating the new ranking of test suites.

Prioritization of Test Cases with Respect to Test Effort and Error Severity

In [4] Taking into account the different test cases and fault cost of priority test cases, Sebastian Elbaum et al. provide a new statistic for assessing the fault detection rate. Prioritized test cases may differ in terms of cost and severity of defects, however these factors were not taken into account by the measure employed in the original research. However, the metric APFD quantifies the weighted average cumulative percentage of fault detection across the lifetime of test suites, keeping test cost and fault detection severity constant. To demonstrate how the new measure may be put to use in the real world, a case study was conducted. Questions about how to put the new measure to use in prioritising test cases are raised by this research. The examples have been created to show why the previous APFD measure is unacceptable owing to the inclusion of variable test costs and fault severity. By modifying the original APFD measure, APFDC was developed. The APFDC is shown as a graph in which the horizontal axis represents the percentage of total cost of the test case, and the vertical axis represents the percentage of total severity of the defect found. Five distinct cost distributions for test cases were employed in the research, as shown below. There are five types of testing expenses: (1) the cost of a single unit of testing, (2) the cost of a single random test, (3) the cost of a single normal test, (5) the cost of testing an entire software system with more than 300K lines of code, as in the case of Mozilla, an open-source web browser with a sizable team of developers and testers working on it, and so on. The report also outlined the three methods used to determine fault severity. One, a linear measure of fault severity in Mozilla, and two, an exponential measure of fault severity in Mozilla. Given this distribution, there are fifteen different permutations of test case cost and fault severity detection. However, the research only focused on nine permutations of test case price and fault detection severity.

Year 2002

Prioritizing Test Cases: An Empirical Analysis of Variables

In [5] Prioritization methods vary widely in terms of their relative usefulness over the intended application, as shown by the research of Sebastian Elbaum et al. A total of 18 accounts were prioritised and then further broken down into 3 categories. The first category, called the comparator group, includes the methods of (1) random ordering and (2) optimum ordering. Statement Group 2 includes four fine grain techniques: (3) Full Statement Coverage, (4) Additional Statement Coverage, (5) Full Fault Exposing Potential (FEP), and (6) Additional Fault Exposing Potential. Twelve coarse grain methods make up the third group, called the "function group." Seventh, comprehensiveness in terms of functions covered, and eighth, extra function coverage, (9) the total level of FEP implemented, (10) the additional level of FEP implemented, (11), the total fault index (FI), (14), the total FI implemented with FEP coverage, (15), the total DIF based, (16), the additional DIF based, (17), and the total DIF implemented with FEP, (18). Where TFi is the first test case in the new ordering T', the APFD metric is calculated as APFD = 1 - (TF1+TF2+...+TFm)/nm + 1/2n.

Year 2004

Case Prioritization Research in a JUnit Testing Environment: An Empirical Analysis Hyunsook Do

Webology, Volume 18, Number 5, 2021
ISSN: 1735-188X
DOI: 10.29121/WEB/V18I5/39

et al. extended the work of TCP to a wide variety of different language paradigms in [6]. Initial research confined the notion to the C programming language, but it was unclear whether the same principles would apply to other languages. This document presents the deployment of TCP in the Java programming language using the JUnit testing framework. Hyunsook ran a controlled experiment to see how well TCP works with Java applications using the JUnit testing framework. As a consequence, the rate at which errors in JUnit's test suite are discovered has increased dramatically. In addition, with reference to previous research on the prioritising of test cases, the study explains how the testing paradigm and language used inside the system under test have evolved.

Year 2005
Test Case Prioritization Strategies: An Experimental Study Using Mutation Errors. It has been suggested by Hyunsook Do and Gregg Rothermel in [7] that mutation defects may stand in for actual flaws. TCP approaches have been tested in a controlled trial to see whether they may increase the rate at which mutation problems and genuine defects are detected. This demonstrates the efficacy of TCP approaches in evaluating mutation problems. There is a correlation between the traits of test cases and mutant defects and the efficiency of TCP methods. In addition, the connection between mutation faults and hand seeded defects has been compared to previous data. Year 2006 An Empirical Analysis of Test Case Prioritization Methods Using Mutation Errors. The empirical research conducted that mutation fault may be used to reflect the true flaws. Additionally, it is hypothesised that the reliability of empirical results that centre on the frequency of fault detection in test suites might be compromised by the presence of manually seeded defects. To evaluate the efficacy of TCP strategies in light of mutation defects, two regulated experiments were conducted. Ranking of Test Cases Based on Expenses. A new metric called APFDc to measure the speed with which TCP faults are detected. Both the cost of running test cases and the cost of finding errors are specified as metrics. This research also explains how traditional prioritising methods complement the introduction of a cost-aware consideration. This research provides a side-by-side examination of as well as the cost-cognizant metric, to older, non-cost-cognizant methods and measures. The research in this article attempts to address a real-world problem. An Empirical Study of Sampling and Prioritization in Configuration-Aware Regression Testing

The limitations of configuration aware regression testing during software system evolution are discussed. Combinatorial interaction testing approaches are employed in this investigation to model and generate configuration samples for use in the study's regression testing. The impact of configuration on testing efficiency was studied empirically for a non-trivial evolving software system. Prioritizing the configuration was successful, and the results reveal that configuration had a considerable influence on defect detection rates. Test Case Prioritization Under Time Pressure: A Randomized Controlled Experiment Series, Year 2010 Some researchers have hypothesised that the software development process' imposition of time limits on the regression testing process might drastically alter the behaviour of TCP approaches. The results of the research indicate that the defect detection, software maintenance, and testing processes may benefit from the use of time constraints over prioritising strategies. In order to ascertain how time limitations affect the costs and advantages of prioritising approaches, a number of experiments have been carried out. Different ramifications for the system engineer who wants to retest the software system economically are shown by the

Webology, Volume 18, Number 5, 2021
ISSN: 1735-188X
DOI: 10.29121/WEB/V18I5/39

findings. Model-Based Prioritization of UML State Chart-Based Retests of the CBSS in 2011

A novel method for efficiently ranking the CBSS test cases based on regression testing. Using this method, developers create a UML state chart diagram that details the states of all CBSS components and how those states are modified by the system's various modules. Component interaction graph is a graph-like structure created from the UML diagrams (CIG). By constructing these graphs, we may better understand the connections between the various parts. The UML-generated graph and the existing test cases are both inputs to the newly suggested method. The algorithm produces a prioritised list of tests to run. Prioritizing test cases involves taking into account two different metrics: the first is the total number of database accesses, and the second is the amount of state changes made by the components as a result of interacting with the test cases. In addition, the technique improves the performance function and reduces the cost for a wide variety of java applications. Prioritization of Oracle-Related Test Cases for the Year 2012

To gauge the efficacy of testing, method for the TCP that takes into account the influence of test oracles directly. During the retesting process, the influence of the test oracle is not taken into account by the current literature. The new method retrieves the flow data from the variable assignment, and hence the test oracle, for each individual test case. The shortest route to the test oracle was then used to prioritise the cover variables. In comparison to structural and random coverage based TCP approaches, the findings demonstrate a significant improvement in the rate of fault identification.

Prioritization of Multiple Test Cases Using a Modular Approach

Current approaches to TCP are more expensive and time-consuming than necessary, but they are also less dependable and less effective. Prioritizing the test cases that correspond to each module in the first stage, and then recombining the separate test suits in the second stage to further prioritise the whole programme, is a key part of the novel approach that we suggest. In addition, the total test case periodization strategy is contrasted with this one, and its efficacy in catching errors is confirmed.

The two well-known TCP techniques, the complete TCP strategy and the extra TCP strategy. Depending on the value of a parameter called x, the basic model and the extended model both provide a spectrum of evasive strategies, from the comprehensive to the supplementary. Differentiated values of x for the various testing strategies have been derived using four distinct heuristics. The empirical research was conducted on 19 iterations of the four tested Java programmes. Comparing the performance of the basic and extended models using method coverage and statement coverage, the findings suggest that the differentiated value of x may be more effective. To Prioritize Test Cases Across Multiple Environments. It provide a unified TCP strategy that incorporates the full TCP technique and the supplemental TCP strategy. There are two models, the basic and the extended, that make up the unified TCP strategy. By combining these two models into a single TCP strategy, a spectrum of TCP approaches is generated, with the bounds between the two extremes being set by the parameter z. An empirical analysis was conducted on the 40 C objects and 28 Java objects to assess the methodology's efficacy. Multiple prioritising methods generated from the two models with z value were shown to be comparable with the original TCP method and

Webology, Volume 18, Number 5, 2021
ISSN: 1735-188X
DOI: 10.29121/WEB/V18I5/39

even outperformed it in certain cases.

CONCLUSION

In this study, we examine the state-of-the-art approaches to regression TCP, including those that make use of code coverage, components, and UML state chart diagrams. The purpose of this study is to assess the performance of TCP methods use two metrics, APFD and APFDC. Prioritization methods that were formerly reserved for C dialects are presented in this article, along with their evolution from C to other languages and frameworks, such as Java under the JUnit framework. As a result, it's safe to say that several methods exist for effectively prioritising test

REFERENCES

1. Luo, Q., Moran, K., Zhang, L., & Poshyvanyk, D. (2018). How do static and dynamic test case prioritization techniques perform on modern software systems? An extensive study on GitHub projects. IEEE Transactions on Software Engineering, 45(11), 1054-1080.
2. Bajaj, A., & Sangwan, O. P. (2019). A systematic literature review of test case prioritization using genetic algorithms. IEEE Access, 7, 126355-126375.
3. Khatibsyarbini, M., Isa, M. A., Jawawi, D. N., & Tumeng, R. (2018). Test case prioritization approaches in regression testing: A systematic literature review. Information and Software Technology, 93, 74-93.
4. Di Nucci, D., Panichella, A., Zaidman, A., & De Lucia, A. (2018). A test case prioritization genetic algorithm guided by the hypervolume indicator. IEEE Transactions on Software Engineering, 46(6), 674-696.
5. Luo, Q., Moran, K., Poshyvanyk, D., & Di Penta, M. (2018, September). Assessing test case prioritization on real faults and mutants. In 2018 IEEE international conference on software maintenance and evolution (ICSME) (pp. 240-251). IEEE.
6. Almarashdeh, I. B. R. A. H. I. M., BOUZKRAOUI, H., AZOUAOUI, A., YOUSSEF, H., NIHARMINE, L., RAHMAN, A., ... & MURIMO, B. M. (2018). An overview of technology evolution: Investigating the factors influencing non-bitcoins users to adopt bitcoins as online payment transaction method. Journal of Theoretical and Applied Information Technology, 96(13), 3984-3993.
7. Azizi, M., & Do, H. (2018, April). A collaborative filtering recommender system for test case prioritization in web applications. In Proceedings of the 33rd annual ACM symposium on applied computing (pp. 1560-1567).