Meliorate Test Efficiency: A Survey

A.D. Shrivathsan and K.S. Ravichandran

School of Computing, SASTRA University, India

Abstract: This paper focuses on Test Case Prioritization Techniques to gain insight of it. The prioritization or ordering of test cases is highly demanded by the software testers in the industry, as their time bound is very critical. At the same time, testers want to improve efficiency without compromising on quality. The objective is to detect faults at the early stages of testing cycle. The test case prioritization technique is the widely used technique in software industry to achieve timely release of quality software product. Hence, this paper attempts to make a survey of some prominent research approaches towards test case prioritization.

Key words: System test • Regression test • Test Case Prioritization

INTRODUCTION

The full-fledged Software Testing consists of several traditional testing procedures and it consumes larger time to complete it. In the present era, it is very difficult to test all the traditional procedures because it will affect the time to deliver the product. Hence, in the recent years, many researchers have developed some modern technologies to test the software which is based on some prioritization procedure and now-a-days, it is accepted by the most of the software community.

Testing activity leads to software quality. Stopping criterion for testing is the resource / time availability. As time is a crucial factor in release of software, some mechanism has to be devised to expedite the testing process, without compromising on quality. The prevalent technique to achieve this is Test Case Prioritization.

This technique is much useful in regression testing. In every cycle of regression test, if the test cases are prioritized, most of the defects are detected at an early time. Thus, critical bugs are fixed and then software product is delivered.

The Test Case Prioritization technique has many objectives. One such objective is early detection of severe faults. Many prioritization techniques are available based on Fault Detection Capability, Requirements Coverage, Fault Exposing Potential, Code Coverage and so on.

This paper surveys the work on Test Case prioritization technique. Section 2 describes the papers under various techniques, which are selected for our literature survey. Section 3 describes the basic ideas of system test, regression test and, motivation behind test case prioritization. Section 4 surveys system test prioritization papers. Section 5 reviews regression test prioritization papers with respect to black box, white box and grey box approaches. And Section 6 concludes.

Selection of Papers: Among the various research papers, papers pertaining to system testing and regression testing are considered. As the system testing is always black box, requirements and fault-based prioritization papers are reviewed. While, in the case of regression testing, approaches such as White box, Black box and Gray box are considered. Based on these issues, papers are selected and a survey is done in this paper.

Background

System Testing: It is a black box approach where in, the system is tested end-to-end against the specifications. Generally, it is done by an exclusive testing team, whereas the unit testing is done by the developer. The developers will test the code what they have written.

Testers develop enormous amount of test cases. They test the system module wise. Accordingly the tester will pick the relevant test cases for execution. A set of test cases arranged in a specific sequence to test a module or component forms a test suite. Therefore, various test suites will exist to test the system against requirements. And, these test suite sizes will vary according to the set of requirements pertaining to a particular module or
component. As far as the system testing is concerned, the entire set of test suites need to be executed to cover all modules of the system to ascertain that the system is completely tested and the flaws are made known. This facilitates to enhance the system towards error-free software.

Regression Testing: It is a complement to system testing. Because without which the testing cycle never completes. During every test cycle the errors are corrected, which in turn may introduce new set of errors as a side effect. Also, there is always a possibility that customer may come with changed requirements prior to release, which leads to addition / deletion / changes in module, further directs to addition of test cases to cover up the modifications. Number of runs in regression testing is not fixed. The test authorities based on the resource availability, especially the time, decide it. Almost all projects in industry are facing the so called problem “Timely Release” and all development and testing team members thrive to achieve this, but without compromising on quality.

Quality is defined as Conforming to the user requirements, without any bug. Hence removing the error plays a major role in improving the software quality. And for this software testing is done. Time is confining the testing and henceforth number of runs in regression testing need to be stopped at one point in time to enact the software release.

The regression-testing scenario can be elucidated using the following diagram:

**Motivation:** Since there is no definitive stopping point in number of runs of a test case and also exhaustive testing is not possible many times due to deadlines laid down by the clients of the product, a tactical maneuverability becomes mandatory to impel the testing activity, perhaps making the software almost error free. This is really challenging for a skilled testing practitioner. There are few methodologies available and among them Test Case Prioritization is predominant. This paper is focusing on Test Case Prioritization techniques.

**System test-Prioritization:** Requirement based prioritization Hema Srikanth and Sean Banerjee [1], tried to ameliorate the test efficiency by capturing more faults. This has been achieved through the factors such as priority assigned by customer (CP), complexity to implement (IC), liable to faultiness in requirements (FP), changeability in requirements (RV)

**Fig. 1: Regression Test Scenario**

**Fig. 2: PORT Scheme**

Total Severity of Faults Detected is computed by adding upvalues of severity for the mistakes found through prioritized sequence of test cases. The prioritization is based on the factors applied over the requirements as illustrated in the Fig -2. The results suggest that the considered factors certainly improved testing efficiency.

Yuen Tak Yu, Man Fai Lau [2] approach is based on specifications and hence ignoring the implementation technology. They used measures essentially as Average Percentage of Faults Detected (APFD), Fault Adequate Test Set Size (FATE). They established a fault class hierarchy for Boolean specifications.

Figure 3 describes the fault classes which may occur in Boolean test specifications.

**Fig. 3: Fault Class Hierarchy**

As we know the test suite consists of test cases, it becomes essential to represent the test suite as a sequence of binary values. If a test case is present represented as 1, else the value is flipped. From this representation, Unique True Point (UTP), Overlapping True Point (OTP), Near False Point (NFP), Remaining False Point (RFP) are identified. From these points, three component strategies were formed namely MUTP (U),
Fig. 3: list of fault classes

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression Negation</td>
<td>Expression or sub expression is</td>
</tr>
<tr>
<td>Fault (ENF)</td>
<td>negated</td>
</tr>
<tr>
<td>Term Negation Fault</td>
<td>A term is negated</td>
</tr>
<tr>
<td>(TNF)</td>
<td></td>
</tr>
<tr>
<td>Term Omission Fault</td>
<td>A term is omitted</td>
</tr>
<tr>
<td>(TOF)</td>
<td></td>
</tr>
<tr>
<td>Disjunctive Operator</td>
<td>An OR operator is implemented</td>
</tr>
<tr>
<td>Reference (ORF[+])</td>
<td>as AND operator</td>
</tr>
<tr>
<td>Conjunctive Operator</td>
<td>An AND operator is implemented</td>
</tr>
<tr>
<td>Reference (ORF[−])</td>
<td>as OR operator</td>
</tr>
<tr>
<td>Literal Negation Fault</td>
<td>A literal is negated</td>
</tr>
<tr>
<td>(LNF)</td>
<td></td>
</tr>
<tr>
<td>Literal Omission Fault</td>
<td>A literal is omitted</td>
</tr>
<tr>
<td>(LOF)</td>
<td></td>
</tr>
<tr>
<td>Literal Insertion Fault</td>
<td>A literal is inserted</td>
</tr>
<tr>
<td>(LIF)</td>
<td></td>
</tr>
<tr>
<td>Literal Reference Fault</td>
<td>A literal is implemented as another</td>
</tr>
<tr>
<td>(LRF)</td>
<td>literal</td>
</tr>
</tbody>
</table>

The execution orders of these points are made through permutation and hence it is given as CNU, UCN, CUN, UNC, NUC are formed. By weighing the efficiency through APFD and FATE measures, UCN prioritization is outperforming the others. Hence this method is purely based on the specification of Boolean expressions and it categorized as fault-based testing.

Regression test-Prioritization

White Box Approach: R.Krishnamoorthy and S.A.Sahaya Arul Mary [3] applied genetic algorithm to prioritize test cases. The fitness function is based on code coverage as well as time bound factor. Actually, it used code coverage of the entire test suite and incremental code coverage as parameters of the fitness function. It also included the time required to execute test cases. So; the objective is to maximize code coverage with minimal time. After choosing test cases optimally, their efficiency is to be weighed. And, it is done by employing Average percentage of Faults Detected (APFD) and proved that the genetic algorithm with these parameters remarkably improving testing efficiency.

Gregg Rothermel, Mary Jean Harrold, Roland H. Untch and Chengyun Chu [4] performed Fault Exposing Potential Prioritization. They took into account nine different methods for prioritizing test cases. The methods are no prioritization, random ordering, optimal greedy approach, statement coverage, additional statement coverage, branch coverage, additional branch coverage, total probability of Fault Exposing Potential, additional probability of Fault Exposing potential. In this regard, the given program is rewritten as mutant version, which contains faulty statements. Then the ratio of how many mutants killed by a particular test case to the full amount of mutants is calculated as FEP. When a test case does not reveal any fault in statement S then it is said to be Confidence of S, C(S). The additional confidence is statement S is defined be the difference in C(S) before and after executing a test case, which is given below.

\[ C(t) = C(s) - C(s) = (1 - C(s)) \]

In this way, by summing \( C_{add}(S) \) for every statement covered by test case t is found to be additional confidence of test case t, as shown below.

\[ C_{add}(t) = C_{add}(S_1) + C_{add}(S_2) + \ldots + FEP(S_n) \]

Now, the test cases are arranged in descending order of \( C_{add}(t) \). This prioritization method works well and its effectiveness is measured using APFD. It is found that the effectiveness is increasing as we move from top to bottom of the techniques mentioned above.

Zhenyu Chen, Baowen Xu, Bin Luo, Chen Zhang and Dongjiang You [11] considered seven different prioritization techniques namely randomized, total statement coverage, additional statement coverage, total statement coverage with time, additional statement coverage with time, time aware total statement coverage through integer linear programming, time aware additional statement coverage with integer linear programming. In all these methods, the effectiveness is measured using APSC, and APFD, which are given below:
Out of the experiments and measurements they found that time aware ordering is improving performance significantly.

Dennis Jeffrey, Neelam Gupta [8] worked on slices of the program. They identified relevant slices of test cases. It is the set of statements selected from the program which are data or control dependent. This is found by following execution trace of a test case. They identified that making changes in branching statements and changes to the statements, which are not part of slice, also may influence the change of output. Hence they formulated heuristic, which considers regular statements, output-influencing statements, potentially output-affecting statements executed by test case. Also, all these statements after modification also considered. Accordingly, test cases are given due weights. Then the effectiveness is found to be significantly increasing and the metric used is APFD.

Alexey G. Malishevsky, Gregg Rothermel, Sebastian Elbaum, Satya Kanduri [12] wanted to select a better prioritization technique. So, they identified five different techniques, which are as follows: Random, Total function coverage, additional function coverage, binary different function coverage and additional binary different function coverage. To select the best method, they combined the said techniques in pairs and ended up in combination as follows:

Total vs. random, additional vs. total, total-difference vs. total, total-difference vs. random, total-difference vs. additional, additional-difference vs. total, additional-difference vs. additional, additional-difference vs. total-difference, additional vs. random, additional-difference vs. random. And each of the pair’s APFD values are compared with respect to cost benefit threshold. Upon which, the best one is selected.

Sejun Kim and Jongmoon Baik [13] used Fault localization technique for prioritising. The idea is, when a test case covers more statements, it will detect more bugs. But in subsequent runs, it may not find more bugs, as the defects are getting removed. They used TARANTULA technique for fault localization. It uses the following metrics

\[ \text{Suspiciousness}(s) = \frac{\% \text{failed}(s)}{\% \text{failed}(s) + \% \text{passed}(s)} \]

\[ \text{Confidence}(s) = \max(\% \text{failed}(s), \% \text{passed}(s)) \]

Then they followed fault aware prioritization technique as follows: All the test cases ordered based on their code coverage capability. Then to each of them a score is computed as

\[ \text{Ordering Score}_i = \frac{n + 1 - i}{n} \]

Then for each test case, the reciprocal of its suspiciousness is computed and it is multiplied with the ordering score. Based on the resultant value, an ordering scheme is found and it is found that it gives better effectiveness.

Hoyeon Ryu, Hyuncheol Park, Jongmoon Baik [19] used Cost as well as Fault severity of test cases to prioritise. The weight factors are computed as

\[ wC_i = \frac{C_i}{C_i + FS_i} \]

\[ wFS_i = \frac{FS_i}{C_i + FS_i} \]

The test case’s Historical Value is computed as

\[ HV_{(t,i)} = \left[ 100 + \min \left( C_{(t,i)}, C_{(t,n)} \right) - C_{(t,n)} \right] \times wC_{i-1} + FS_{(t,i-1)} \times wFS_{i-1} \]

Upon this value, Test cases are ordered to improve effectiveness.

Arvinder Kaur and Shubhra Goyal [20] applied genetic algorithm to prioritise. And the fitness function used was code coverage.

Wang Jun, Zhuang Yan, Jianyun Chen [21] applied genetic algorithm to prioritise. And the fitness function used was block coverage.

**Black Box Approach:** Yu-Chi Huang, Chin-Yu Huang and Kuan-Li Peng [5] proposed genetic algorithm approach to prioritize test cases. It uses history information like test cost, faults detected, severity of faults detected.

It uses a measure called Cost cognizant Average Percentage of Faults Detected (APFD) to measure efficiency of this technique. Also, APFD serves as fitness function too.
The methodology is shown in Fig-3. Most of the other test case prioritization schemes assume test case execution cost and fault severities are equal among all test cases. Hence this methodology supersedes other techniques thereby improving testing efficiency significantly.

Chin-Yu Huang, Jun-Ru Chang and Yung-Hsin Chang [9] applied test case prioritization for GUI applications. It never considered code, rather looked at events occurring at the application. They assigned weights for the events as per the following classification: Equally assigning weights, weights assigned based on the proneness of faults, assigning weights randomly. All these are possible because of the Event Flow Graph. Having assigned weights to events, the test cases relevant to the events are arranged in a particular order, either based on ascending or descending order of events. And the performance is measured using APFD metric.

Krishnamurthy and S.A.Mary [14,15] made complement to PORT scheme [1] by means of adding some additional factors, especially towards regression testing. The added factors are Fault Impact, Completeness and traceability of requirements. From all the factors they computed Requirement Factor Value as

\[ RFV_i = \frac{\sum_{j=1}^{6} FactorValue_j}{6} \]

Test Cases weights are assigned based on RFV value and coverage of requirements. The effectiveness is measured using Total Severity of Faults Detected, Average Severity of Faults Detected, Average Test Effort Index and Total Test Effort Index. Also, they included cost of requirements while finding RFV and it increased effectiveness.

Sangeeta Sabharwal, Chayanika Sharma and Ritu Sibal [16-18] applied genetic algorithm to state chart diagram and activity diagram to prioritise. And in these diagrams fitness value is computed based on the weight of each node. The weight is calculated based on path it covers, fan-in, and fan-out values. In [17], they converted source code into flow graph and then applied this technique.

Sujata, Mohit Kumar, Dr. Varun Kumar [22] applied genetic algorithm to prioritise. And the fitness function is based on requirements coverage and fault severity.

**Gray Box Approach:** Alireza Khalilian, Mohammad Abdollahi Azgomi and Yalda Fazlalizadeh [6] developed a methodology that is based on selection probability. It includes both system testing and regression testing. In system testing; priorities are made based on code coverage. But during regression testing, it uses historical data especially the fault detection effectiveness during every run. Also it uses \( h_k \) a counter which shows how many times a particular test case is executed. All these put together acts as a probability for selecting a test case during every run. And in fact for regression testing it uses black box testing.

\[ PR_i = \frac{\% \text{ of Code Coverage}}{n} \]

where, \( n \) refers to different coverage criteria

\[ Pr_i = (\alpha h_k + \beta PR_{i-1})/k, 0 \leq \alpha, \beta < 1, k \geq 1 \]

\[ \alpha = \left(1 - \left(\frac{fC_k + 1}{eC_k + 1}\right)^2\right)^{\frac{1}{hk}} \]

\[ \beta = \left(\frac{fC_k + 1}{eC_k + 1}\right)^x \]

\[ x = \begin{cases} 1 & \text{The test case has revealed some faults} \\ 2 & \text{The test case has not revealed any fault} \end{cases} \]

And this methodology uses a prominent measure namely Average Percentage of Faults Detected (APFD). It proved that this technique significantly improves test efficiency by means of prioritizing.
Md. Mahfuzul Islam, Giuseppe Scanniello, Angelo Susi and Alessandro Marchetto [7] used genetic algorithm to prioritize test cases. The fitness function of this approach uses three objectives namely Weighted Code Coverage, Weighted Requirements Coverage and execution time of a test case. The computation of weighted code coverage is given below

\[
WCCov(t) = \sum_{s \in \text{Statements}} w_c \begin{cases} 
1 & s \in \text{Tester Relevant}_c \\
0.5 & s \in \text{Partial Tester Relevant}_c \\
0 & s \in \text{non Relevant}_c
\end{cases}
\]

The computation of requirement coverage is as given below

\[
WRCov(t) = \sum_{r \in \text{Requirements}} w_r \begin{cases} 
1 & s \in \text{Tester Relevant}_r \\
0.5 & s \in \text{Partial Tester Relevant}_r \\
0 & s \in \text{non Relevant}_r
\end{cases}
\]

Traceability Links are made between requirements to coding and to the Test Cases. Later from these links Information were retrieved to calculate the above.

The effectiveness is measured using APFD metric. The sensitivity is measured using the following three metrics: APFD_{all} is to assess injected faults. APFD_{nbf} for the purpose of weighing sever faults. APFD_{abf} is to weigh the faults pertaining to requirements. The diversity is measured using the Test Ordering Diversity (TOD) metric, which is given below

\[
TOD(Ord_{1s}, Ord_{2s}) = \left( \frac{Ld(Ord_{1s}, Ord_{2s})}{n} \right) \times 100
\]

where, \( Ld \) is minimum number of edit operations performed to change the ordering of a test suite.

This methodology gave remarkable improvements in testing efficiency with java applications. Lijun Mei, Robert G. Merkel, T.H. Tse and W.K. Chan [10] made an attempt to web service testing. So, they relied on XML manipulation. When a service is gotten, its code is not accessible for testing. Hence, they utilized WSDL documents and interactive messages. They prioritized test cases based on WSDL tag coverage and WSDL tag occurrences ad used APFD as measure. And they embarked into testing of dynamic environments.

CONCLUSION

This paper provides both survey and detailed analysis of various test case prioritization schemes. This paper covers both system testing and regression testing aspects. Also it concerns of white box and black box approaches. Though it is a survey paper it gives detailed insight of various methodologies, which provides empirical evidences also. Therefore it gives a much widening idea to researchers who work in the field of test case prioritization.

REFERENCES


