FP-International Journal of Computer Science Research (IJCSR)
Volume 2, Issue 2, Pages 83-87, June 2015

Techniques on Prioritization of Software Testing: A Review

Pratibha
Department of Computer Science and Engineering
Chandigarh University, Mohali, Punjab, India
pratibhathakur3@gmail.com

Isha Sharma
Department of Computer Science and Engineering
Chandigarh University, Mohali, Punjab, India
ishasharma211@gmail.com

ABSTRACT
Test case prioritization technique includes test cases for completion in an order that attempts to increase their effectiveness, efficiency at meeting some requirements. Various goals are possible; one necessitate rate of fault detection- a measure of how quickly detected faults within the testing process. An improved rate of fault detection during testing process can give faster outcomes on the system under test and software engineers begin correcting faults earlier. One application of prioritization techniques involves regression testing- the retesting of software changes, modifications; In this frame of references, prioritization techniques can take advantage of information collected about the previous completion of test cases to obtain test case orderings. Test Cases are treated as one of the most favorable part of software testing process. They are executive for the validation of the software under look over. Test suites are used to test modification in the coding phase during regression testing. In number of cases, the test suites are so large that executing all tests for every coding phase modification is incongruous. Testers need to prioritize the test suite so that most effective test cases are executed first. This can result in increasing the effectiveness, efficiency of testing and saving time and decreasing cost. In this paper, we introduce a new algorithm for test case prioritization that is based on the code coverage of the test cases.

Keywords
Test case prioritization, fault detection, regression testing

1. INTRODUCTION

Software testing is one of the major and primary techniques or achieving reliable software. Software testing is done to detect the error, which occur software failure. However, software testing is a time taking and expensive work [1], [2]. Software modifications many times during development and maintenance phase. Modification are done for several reasons, such as

adding new application, platform, correcting some bugs. After modifications have been made, regression tests are applied to the rectify parts of the software. These modified parts have not affected the quality of the other parts of software. According to Rothermel *et al.*, [3], "test suites can increase so large that it is cost expenditure

to execute each test case for every new source code modification". In these type of circumstances, Developers/testers are required to prioritize the test suite so that test cases that are susceptible to find undetected errors are run in the inception. Also the test cases that cover more part of the code may be completion first. Regression test selection and prioritization techniques effort to preserve the time and decrease the overall cost of regression testing. These techniques select and run only that subset of the test cases from existing test suite which is associated to the modified part of the code.

According to the IEEE Standard 1219-1998 [4], "regression testing can be associate in several modules such as unit, integration or system level testing". Most existing regression testing techniques concentrate on unit testing. Some of the techniques applied on all modules of testing [5, 6]. Regression testing is performed after justifying of modification or new functionality. Justifying that the errors are fixed and the newly added features have not occurred in problem in on-time working version of software. Testers perform functional testing when new build is available for justification. The determined of this test is to verify the changes made in the existing functionality and newly added functionality.

Test case prioritization techniques could be of advantage to growing the efficiencies of test suites in practice. Test case prioritization is a technique helps to grow the fault detection. In an empirical evaluation of regression test suite prioritization technique ordering was measured using an evaluation metric called APFD (Average Percentage Faults Detected) and PTR (Problem Tracking Report).

A software bugs refers to a detect in a system. A bugs is disagreement between the perceive performance of a system and it's individualize performance. A software failure creates when the transfer product diverge from correct service and perform unexpected behavior from

user requirements. A software fault or error may not necessarily cause a software failure. Fault detection is apprehending that a problem has created, even if you don't know the reason. Faults may be detected by a variety of quantitative or qualitative approaches. This includes many of the multivariable, model-based approaches. Fault analysis is investigating one or more root causes of problems to the point where corrective action can be taken. This is also referred to as “fault isolation”, especially when need to show the differentiation from fault detection. A "fault" or "problem does not have to be the result of a complete failure of a software product. In a process plant, root causes of non-optimal operation might be hardware failures but problems might also be caused by poor choice of operating targets, poor raw material quality or human error.

The following are several classes of software faults:-

Syntactic faults: interface faults and parameter faults called as syntactic faults.

a) Semantic faults: conflicting behavior and erroneous results called as semantic faults.
b) Service faults: QoS faults, SLA (Service Level Agreement) associated faults, and real-time vibrations are called service faults.
c) Communication / interaction faults: time out and service inapproachable is called communication or interaction faults.
d) Exemption: I/O related exemption and security-related exemption are called exemptions faults.

2. APPROACHES OF FAULT DETECTION

There are several approaches to find and isolate faults. Because each approach has their benefits and failures, topmost applications mix multiple approaches. We prominence some of the key differentiating factors between the different techniques.

Model based reasoning: When models of the observed system are used as a basis for fault detection and diagnosis, this is often referred to as "model based reasoning”. One of the major differences in approaches to fault detection & diagnosis is whether or not explicate models are used, and what type of models are used.

Fault signatures, pattern recognition and classifiers: Pattern recognition directly uses the observed sign of a problem and compares them to a set of known sign for each possible problem. The “pattern”, or “fault signature” can be represent as a vector (1 dimensional array) of symptoms for each defined fault.

Neural networks:-Neural networks are nonlinear, multivariable models built from a set of input/output data. They can be used as event detectors to detect events and trends. They also used as demonstrative models in model-based reasoning, or directly used as categories for identify fault signatures.

Event-oriented fault detection, diagnosis and correlation:-An event denotes a change of state of a monitored object. Alarms are examples of events. Diagnostics involving events can be significantly different than diagnostics including a fixed set of variables.

Passive system monitoring vs. active testing:-In the case of online monitoring systems, different diagnostic techniques assume routine scanning of each variable of interest. But many times, it is preferable to request non-routine tests.

Rule-based approaches and implementations:-Rule-based systems in most cases just implement other approaches discussed above, more as a program control mechanism than a different diagnostic technique.

Hybrid approaches:-Pattern recognition by itself does not require a model. However, input for construction of the signatures for the known failures may be based on models; for instance, as residuals from models of normal behavior. This general technique applies to static or dynamic models. For dynamic models, the patterns can be based on predicted measurement values vs. observed values in a Kalman filter, for example Smart signal offers products based on an empirical process model of normal operation used for fault detection, combined with other techniques for fault isolation. Pattern recognition can also be combined with models of exceptional behavior. For instance, in the case of the SMARTS In Charge product, the modeling was in terms of a fault propagation model (qualitative cause/effect model of abnormal behavior). But as part of the development process, this model was then used to automatically construct fault signatures - a form of compiled knowledge. At run time, diagnosis was based on matching observed data to the nearest fault signature. So the product at run time had the characteristics of a pattern matching solution. So, the overall methodology is often a combination of pattern recognition with a model-based method. Some tools such as GDA are flexible enough to support multiple approaches to fault detection and diagnosis, and also support the upfront filtering and event generation as well. One conclusion was that most applications required a mix of techniques for success [1].

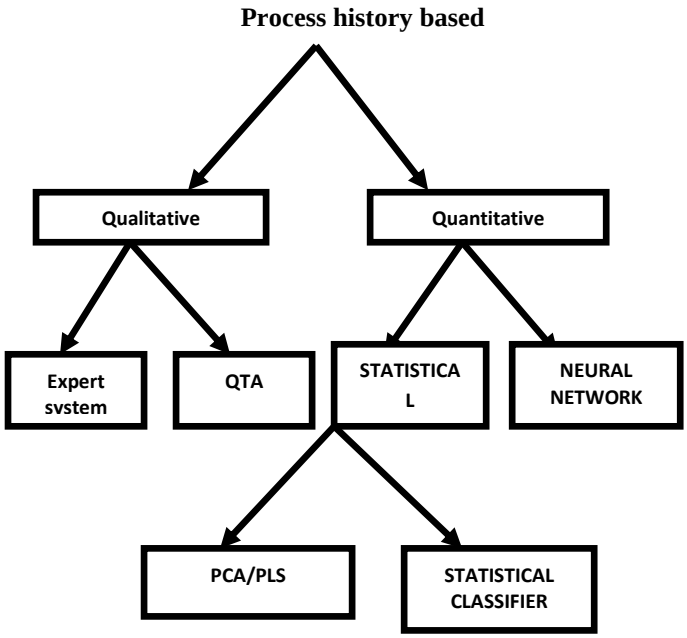

Fig 1: Architecture diagram strategy for detection of software faults

To detect the software faults, which have been generated during the development process, two different approaches may be applied:-

1. Static approach
2. Dynamic approach

1. Static approach:- Techniques guided by a static approach do not depend upon that the system is complied and may be applied at all stages of the development process. These techniques may be applied as formal reviews like investigation or automatic analyses of the code of a system or associated documents.

2. Dynamic approach:- Techniques guided by dynamic approach ensure that a program is operationally correct which mean that the system is complied with test data. On the other hand, testing is only possible when a prototype or an executable version of a program is available. Both inspections and testing are activities contributing to validation and verification.

3. SOFTWARE BASED FAULT DETECTION TECHNIQUE

3.1 Algorithm Based Fault Tolerance (ABFT)

ABFT is used for detecting, locating, and correcting faults with a software procedure. It exploits the structure of numerical operations. This approach is effective but lacks of generality. It is well suited for applications using regular structures, and therefore it is used for a limited set of problems.

1. Assertions:- Assertions or the logic statements inserted at several points in the program reflect undeviating relationships between the variables of the program and they often lead to many asseveration problems as assertions are not transparent to a programmer and their operatives depends on the nature of an application and on a programmer's ability.

2. Control Flow Checking (CFC):- The basic task of CFC is to partition an application program in basic blocks or the branch-free parts of code. A deterministic signature (or number) is assigned to each block and faults are detected by comparing the run-time signature with a pre-computed one. In most CFC techniques one of the major problems is to tune the test granularity that should be used.

3. Procedure Duplication (PD):- The programmer decides to duplicate the most critical procedures and to compare the obtained results on executing the procedures on two different processors. This approach requires a programmer to decide which procedures to be duplicated and to introduce proper checking on the results. These code modifications are done manually and might introduce bug.

4. Error Detection by Duplicated Instructions (EDDI):- Computation results from master and shadow instructions are compared before writing to memory. Upon mismatch, the program jumps to a bug handler that will cause the program to restart. EDDI has high bug coverage at the cost of performance penalty due to time redundancy as introduced into the system. Since we use general purpose registers as shadow registers, more register spilling occurs with EDDI. More spilling causes more performance overhead since it increases the number of memory operations.

5. Periodic Memory Scrubbing: - This approach relies on periodic reloading of code on main memory from an immutable memory. This is effective for protecting the code segment of Operating system and application programs. Performance penalty is due to repetitive memory reading.

6. Masking Redundancy: - This approach means running an application in the presence of faults. Few processors are used to run the same program and vote to identify errors in any single processor. Errors can be masked from application software. No software rollbacks are required to fix errors.

7. Reconfiguration: - This means removing failed modules from the system. When failure occurs in a module, its effects on the remaining portion of the

system is isolated. A large number of functional modules are used, which are switched automatically to replace a failing module.

8. Replication: **-** This ensures reliability but is expensive in terms of hardware or runtime cost. The idea is to take a majority vote on a calculation replicated N times. Its software solution requires each processor to run N copies of surrounding computations and then vote on the result. This slows down the computation by at least a factor of N.

9. Restore Architecture: **-** Transient errors or soft errors are detected through time excessive in the restore architecture. The novelty of the restore architecture is the use of transient error symptoms, such as, memory protection violation and incorrect control flow etc. The tendency for these symptoms to occur quickly after a transient, coupled with a check pointing implementation in hardware to restore clean architectural state, enables a cost effective soft error detection and recovery solution.

10. Dual Modular Redundancy (DMR) & Backward-Error Recovery (BER) & Checkpoint: **-** Error is detected through differences in execution across a dual modular redundant (DMR) processor pair. DMR is a backward-error.

4. PRIORITIZED TEST SUIT EFFECTIVENESS

4.1 Average Percentage of Fault Detections (APFD) Metrics

To quantify the goal of increasing a subset of the test suite's rate of fault detection, we use a metric called APFD developed by Elbaum et al.[8] that measures the rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the percentage of faults detected during the execution of the test suite. APFD values range from 0 to 100; higher values imply faster (better) fault detection rates. APFD can be calculated as follows:

$$APFD=1-\{(Tf1+Tf2+\ldots.+Tfm)/mn\}+(1/2n)$$

Where n is the no. of test cases and m be the no. of faults.
(Tf1,….,Tfm) are the position of first test T that exposes the fault.

5. REGRESSION TESTING TECHNIQUES

There are number of available regression testing techniques. Here we are representing all these techniques in basic 3 categories as defined.

1. *Retest All:* As the name suggest in this testing technique we perform whole testing cycle again after the inclusion of new code and component and related test cases into it. Again the test cases will be generated, sequence reset etc. This type of technique is not feasible in most of time, as it requires much time and cost. But in smaller software where a small change in code impact on whole software at that time regression testing is used.

2. *Regression Test Selection:* This approach is a modification over the existing retest all approaches. In this approach instead of testing all cases a selection on the test cases is performed. To perform this selection a test cases categorization is performed. According to this rest table cases are separated from whole test cases such as the requirement based testing is generally need not to be performed again. The code based test cases and the system based test cases are selected to perform the testing process. In this technique instead of rerunning the whole test suite, we select a part of test suite to rerun if the cost of selecting a part of test suite is less than the cost of running the tests that RTS allows us to omit. RTS divides the existing test suite into (1) Reusable test cases; (2) Re-testable test cases; (3) Obsolete test cases.

3. *Test Case Prioritization:* All the test cases used in a testing approach or the sequence are not alike. It means each kind of test cases have there on values called the basic prioritization of the test cases. Generally the prioritization process is defined on the bases of state space diagram of the cases. The test cases that exist on initial stage of the test cases or the development process have the lower priority and the test cases that affect the whole system or tested repeatedly over the whole process having the higher priority. Besides this the prioritization process is further divided in number of sub techniques to assign the priorities.

a) *The easiest type of assigning* *priorities* ***is*** *the random prioritization* but in most of the cases it does perform the complete justification with the test cases selection. Because of this such type of technique is never recommended to generate the test cases.

b) *Optimal ordering:* In which the test cases are prioritized to optimize rate of fault detection. As faults are determined by respective test cases and we have programs with known faults, so test cases can be prioritized optimally. It is one of the dynamic prioritization approach in which decision is affected because of types of occurred faults and there frequency.

c) *Total statement coverage prioritization:* in which test cases are prioritized in terms of total number of statements by sorting them in order of coverage achieved. If test cases are having same number of statements they can be ordered pseudo randomly.

d) *Additional statement coverage prioritization:* which is similar to total coverage prioritization, but depends upon feedback about coverage attained to focus on statements not yet covered. This technique greedily selects a test case that has the greatest statement coverage and then iterates until all statements are covered by at least one test case. The moment all statements are covered the remaining test cases undergo Additional statement coverage prioritization by resetting all statements to "not covered".

Types of Testing:

1. Unit testing: It is performed on a single unit or group of units that are somehow related to each other in any manner. It is performed by the programmer itself after the development of a unit or group of units. The model used in this testing is "white box".

***2) Integration testing*:** It is performed on combined components of product. As sometimes it happens that components work correctly before they are combined and produce errors when the combination of components takes place. The model used in this testing is "white box" and "black box".

3) *Functional testing:* It is performed to check the functionality of the system that whether the system is performing its function what it is intended to perform. The model used in it is "black box".

4) *System testing:* It is performed on a complete or full system also the software is put under different environments. It can only be performed after the complete implementation of the system. Stress testing, performance testing and usability testing are performed on it in order to check the system: under stress (load), performance (speed and accuracy) and usability (user friendliness) respectively. The model used in this testing is "black box".

5) *Acceptance testing:* In this type of testing customer checks for output that whether the system is able to meet his requirements. Also customer checks whether the purpose of the system is resolved or not. The model used in it is "black box".

Regression (retest) testing: It is performed when any module, unit, component or system is modified in order to change the functionality or for some other reasons. Regression testing ensures that any modification done in specified components will not lead to any discrepancy in the actual output of the system. It also ensures that no other modules are being affected due to performed modifications. In this first of all the modules are tested which have been modifies after they give correct results is done on the whole system It maintains the quality of the system to many efforts are required in this type of testing. Also cost associated with this is high comparatively [9].

6. CONCLUSION

There are different methods and techniques used to detect faults in software system but each technique has some disadvantages. We mainly focus on the software regression testing. There are different kind of regression testing each type has unique functionality in software system. We have reviewed a number of paper and concluded there are many technique to prioritize the test cases. But no one technique is gave effective results. The technique gave higher priority to least important test cases and lower priority to most important test cases. Thus, to overcome these issues we will propose an algorithm which will provide efficient and effective results.

REFERENCES

[1] Dr. Velur Rajappa, Arun Biradar, Satanik Panda "Efficient software test case generation Using Genetic algorithm based Graph theory" International conference on emerging trends in Engineering and Technology, pp. 298--303, IEEE (2008).

[2] Praveen Ranjan Srivastava and Tai-hoon Kim " Application of Genetic algorithm in software testing", International Journal of software Engineering and its Applications, vol.3, No.4, pp. 87--96 (2009).

[3] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Prioritizing test cases for regression testing", Software Engineering, IEEE Transactions, vol. 27, no. 10, (2001), pp. 929-948.

[4] IEEE standard for software maintenance. IEEE Std 1219-1998, ix, vol. 15, no. 19, (1998) October.

[5] A. P. Mathur, "Foundations of Software Testing: For VTU. (1st ed.)", Pearson Education India, (2010).

[6] M. Khoury, "Cost-Effective Regression Testing", (2006).

[7] Greg Stanely and associates "A guide to fault detection and diagnosis" , 2010-2013.

[8] S. Elbaum, A. Malishevsky, and G. Rothermel.(2000), "Prioritizing test cases for regression testing".

[9] Quart-ul-an-farooq, Mohammad Zohaib Z. Iqbal, Zafar I Malik and Matthias Riebish, "A Model Based Regression Testing Approach For Evolving Software Systems With Flexible Tool Support".