MODEL TO IMPROVE CORRECTNESS AND QUALITY AND REDUCING TESTING TIME (MCQRTT)

Sahil Batra¹, Dr. Rahul Rishi²

¹Department of Computer Science and Engineering
The Technological Institute of Textile and Science, Bhiwani-127021, Haryana – India
sahil.batra23@gmail.com

²Department of Computer Science and Engineering
The Technological Institute of Textile and Science, Bhiwani-127021, Haryana – India
rahulrishi@rediffmail.com

Abstract: Software testing is complex and time consuming. One way to reduce the effort associated with testing is to generate test data automatically. Testing is very important part of software development. Quality is not an absolute term; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behavior of the product against a specification. Software testing process can produce several artifacts. So, we proposed a model to improve Quality and correctness and also we reduce the software testing time.

Keywords: Software test cycle, test cases, time constraints, code coverage, Quality.

INTRODUCTION

Almost 80% software fails because of not testing properly. Testing is performed by different types of strategies. Generally testing is performed on code, but if the software can be tested in the earlier phases then most of the errors can be eliminated and can be stopped from propagating to next phase. Thus there is a need to explore testing possibilities in earlier phases.[1,5] The software testing document, which consists of event, action, input, output, expected result, and actual result. Clinically defined a test case is an input and an expected result whereas other test cases described in more detail the input scenario and what results might be expected. It can occasionally be a series of steps but with one expected result or expected outcome.

PRECEDING WORK

Software testing should develop a sufficient assessment of quality, at a reasonable cost and at timely decisions to be made concerning the software. Sufficient testing means when all the necessary required testing is to be done to check the functionality of software for all possible usage scenarios. Over the years tester stops testing when following 6 conditions are satisfied: The first condition for testing is to have good quality specifications for each software lifecycle step. The second condition for testing is for all the specifications for the different integration levels to be in synchronized way. The third condition for testing is for each of the specifications to contain the complete requirements set for generating tests at that particular integration level, The fourth condition for testing is to have additive, rather than repetitive tests at the different integration stages, The fifth condition for testing is to run each of the tests at the integration stage that either yields the maximum information about the product quality, or all things being equal, it is the cheapest to run, A sixth and most obvious prerequisite to software testing is to automate the test process as much as possible. [4, 5, 6]

CRISIS AREA

“Complete Quality of software can never be fulfilled”, Tester can never be able to achieve that software is completely free from errors or Quality is achieved, but tester can maximize the test coverage by using a smart test approach to fulfill the desired software quality.

MODEL TO INCREASE CORRECTNESS AND QUALITY AND REDUCING TESTING TIME (MCQRTT)

Testing is endless process. Tester can not stop testing until all the defects are removed, it is simply impossible. At some point, tester has to stop testing and ship the software. After observing all the details of Software testing we come to a conclusion that “testing can never be considered complete”. Tester can never be proved theoretically or scientifically that our software is free from errors now. Basically testers stop testing when:

a. The planned testing deadlines are about to expire.
b. Not able to detect any errors even after execution of all the planned test Cases.

These two statements do not have any meaning and are contradictory since we can satisfy the first statement even by doing nothing while the second statement is equally meaningless since it can not ensure the quality of our test cases. Pin pointing the time when to stop testing is difficult. Most of the today’s software are so complex and run in such an Interdependent environment, that complete testing can never be done. Other important factors which are helpful in deciding when to stop the testing are:
Testing metrics helps the testers to take better and accurate decisions; like when to stop testing or when the application is ready for release, how to track testing progress & how to measure the quality of a product at a certain point in the testing cycle. The best way for tester is to have a fixed number of test cases ready well before the beginning of test execution cycle. Finally measure the testing progress by recording the total number of test cases executed using the following metrics which are quite helpful in measuring the quality of the software product.

**Percentage Completion:** - \[\frac{\text{(Total executed test cases)}}{\text{(Total test cases)}}\]*100.  
**Percentage Test cases Passed:** - \[\frac{\text{(Total passed test cases)}}{\text{(Total executed test cases)}}\]*100  
**Percentage Test cases Failed:** - \[\frac{\text{(Total failed test cases)}}{\text{(Total executed test cases)}}\]*100.

A test case is declared Failed when just one bug is found while executing it, otherwise it is considered as Passed. Practically tester’s feels that the decision of stopping testing is based on the level of the risk acceptable to the management. As testing is a never ending process tester can never assume that 100 % testing has been done, we can only minimize the risk of shipping the software to end user with some type of testing done. The risk can be measured for the small duration, low budget, low resources project; risk can be reduced simply by Risk analysis.

**IMPORTANT NECESSITIES**

For implementing the purposed scheme we use Selenium, Junit, Eclipse, Mozilla firefox, and a web-based application, to prove that the proposed scheme is correct and efficient. Selenium IDE (Integrated Development Environment) is a prototyping tool for building test scripts.

**Implementation**

**Generating Interface**

To prove the proposed scheme we can use a web based application, some ‘java’ base test cases. Firstly we configure Selenium-RC with Eclipse. Selenium acts as interface between Eclipse and the Mozilla firefox. So, the first step to make the interface for mozilla firefox is to create a profile of selenium in the mozilla firefox with the following steps:

a. Go to the start  
b. Start RUN option.  
c. Then write firefox –profilemanager in run.  
d. It opens Firefox- choose user profile window.  
e. Then click on the create profile named selenium.  
f. Then click on start firefox.  
g. Exit.

And test the web based application, and then test cases are run through eclipse in which Junit is embedded. We take a following test case to check whether login button perform accordingly or not. [13, 14]

**Outcome**

For correct test case, the Selenium starts the Mozilla firefox and open the web page of the application, on which the login page is redirected (Shown in figure 1, 2, 3). If test case executes successfully then the web page window shown in figure 3 is open after window (in figure 1 and 2). And if the test case contains an error or web application do not found on the required URL, then the Selenium starts (figure 1) and then an error is shown in figure 2.

![Figure.1 selenium interface](image)
CONCLUSION

We have proved with the help of above purposed scheme Model to increase Correctness and Quality and Reducing Testing Time (MCQRTT) that we can reduce software testing time considerably and improve the software quality and increase the correctness of the software.

REFERENCES