Volume 2, No. 6, June 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

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

Sahil Batra*[1], Dr. Rahul Risht[2]

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

a. When the test cases have been finished with some pass percentage.
b. When the testing budget comes to its end.
c. When Functional coverage, code coverage, the client requirements meets to certain point.
d. When bug rate drops below a prescribed level.
e. When the period of beta testing / alpha testing gets over.
f. Beta or Alpha testing period finished.
g. When Resources are availability finished.
h. Prepare defined number of test cases before test execution cycle.
i. Execution of all test cases in every testing cycle.
j. Stop testing process when all the test cases get Passed
k. Stop testing when percentage of failure in the last testing cycle is observed to be extremely low.
l. Mean Time between Failures: - recording the average operational time before the software failure.
m. Coverage metrics: - recording the percentage of number of executions during tests.
n. Defect density: - recording the defects related to size of software like the number of open bugs and their severity levels.

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: - [(Total executed test cases) / (Total test cases)]*100.
Percentage Test cases Passed: - [(Total passed test cases) / (Total executed test cases)]*100
Percentage Test cases Failed: - [(Total failed test cases) / (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 **S**eleinum**,** **J**unit**,** **E**clipse**,** **M**ozilla 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

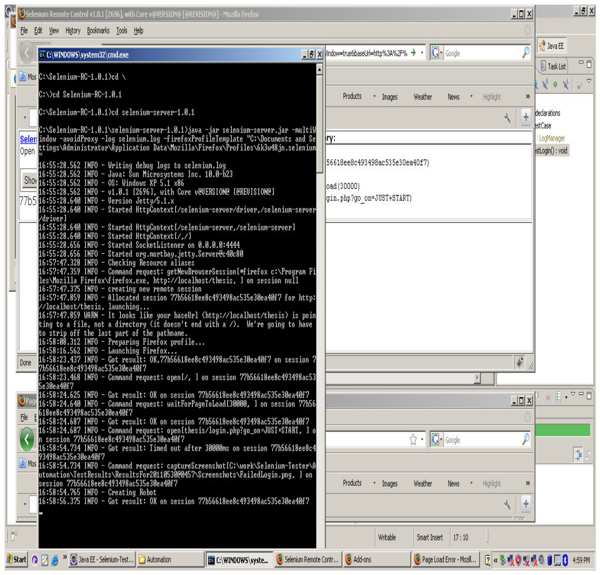
To prove the proposed scheme we can use a web based application, some 'java' base test cases. Firstly we configure Selenium-RC with Eclipse.
Selenium acts as interface between Eclipse and the Mozilla firefox. So, the first step to make the interface for mozilla firefox is to create a profile of selenium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

To prove the proposed scheme we can use a web based application, some 'java' base test cases. Firstly we configure Selenium-RC with Eclipse. General configuration of Selenium-RC with any java IDE would have following steps:
a. Start Selenium IDE.
b. Start new project in eclipse.
c. Add to project classpath selenium-java-client-driver.jar.
d. Record the test from Selenium-IDE and translate it to java code.
e. Run test in the IDE.

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 Seleinum 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 Seleinum starts (figure 1) and then an error is shown in figure 2.


Selenium Remote Control v1.0.1 [2696], with Core v@VERSION@ [@REVISION@] - Mozilla Firefox
File Edit View History Bookmarks Tools Help
C:\WINDOWS\system32\cmd.exe
Start
Java EE - Selenium-Test...
Automation
C:\WINDOWS\syste...
Selenium Remote Contr...
Add-ons
Page Load Error - Mozil...

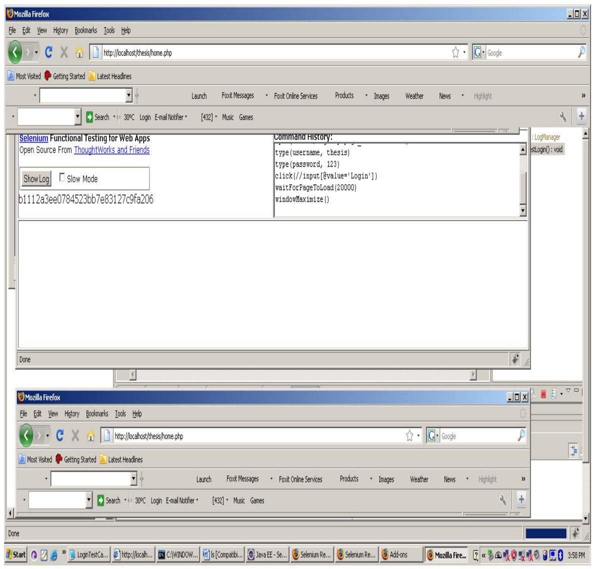
```
C:\Selenium-RC-1.0.1>cd \

C:\>cd Selenium-RC-1.0.1

C:\Selenium-RC-1.0.1>cd selenium-server-1.0.1

C:\Selenium-RC-1.0.1\selenium-server-1.0.1>java -jar selenium-server.jar -multiW
indow -avoidProxy -log selenium.log -firefoxProfileTemplate "C:\Documents and Se
ttings\Administrator\Application Data\Mozilla\Firefox\Profiles\6k3w48jn.selenium
"
16:55:28.562 INFO - Writing debug logs to selenium.log
16:55:28.562 INFO - Java: Sun Microsystems Inc. 10.0-b23
16:55:28.562 INFO - OS: Windows XP 5.1 x86
16:55:28.562 INFO - v1.0.1 [2696], with Core v@VERSION@ [@REVISION@]
16:55:28.640 INFO - Version Jetty/5.1.x
16:55:28.640 INFO - Started HttpContext[/selenium-server/driver,/selenium-server
/driver]
16:55:28.640 INFO - Started HttpContext[/selenium-server,/selenium-server]
16:55:28.640 INFO - Started HttpContext[/,/]
16:55:28.656 INFO - Started SocketListener on 0.0.0.0:4444
16:55:28.656 INFO - Started org.mortbay.jetty.Server@c40c80
16:57:47.328 INFO - Checking Resource aliases
16:57:47.359 INFO - Command request: getNewBrowserSession[*firefox c:\Program Fi
les\Mozilla Firefox\firefox.exe, http://localhost/thesis, ] on session null
16:57:47.375 INFO - creating new remote session
16:57:47.859 INFO - Allocated session 77b56618ee8c493498ac535e30ea40f7 for http:
//localhost/thesis, launching...
16:57:47.859 WARN - It looks like your baseUrl (http://localhost/thesis) is poin
ting to a file, not a directory (it doesn't end with a /).  We're going to have
to strip off the last part of the pathname.
16:58:08.312 INFO - Preparing Firefox profile...
16:58:16.562 INFO - Launching Firefox...
16:58:23.437 INFO - Got result: OK,77b56618ee8c493498ac535e30ea40f7 on session 7
7b56618ee8c493498ac535e30ea40f7
16:58:23.468 INFO - Command request: open[/, ] on session 77b56618ee8c493498ac53
5e30ea40f7
16:58:24.625 INFO - Got result: OK on session 77b56618ee8c493498ac535e30ea40f7
16:58:24.640 INFO - Command request: waitForPageToLoad[30000, ] on session 77b56
618ee8c493498ac535e30ea40f7
16:58:24.687 INFO - Got result: OK on session 77b56618ee8c493498ac535e30ea40f7
16:58:24.687 INFO - Command request: open[thesis/login.php?go_on=JUST+START, ] o
n session 77b56618ee8c493498ac535e30ea40f7
16:58:54.734 INFO - Got result: Timed out after 30000ms on session 77b56618ee8c4
93498ac535e30ea40f7
16:58:54.734 INFO - Command request: captureScreenshot[C:\work\Selenium-Tester\A
utomation\TestResults\ResultsFor20110530@0457\Screenshots\FailedLogin.png, ] on
session 77b56618ee8c493498ac535e30ea40f7
16:58:54.765 INFO - Creating Robot
16:58:56.375 INFO - Got result: OK on session 77b56618ee8c493498ac535e30ea40f7
_
```

Figure.1 selenium interface


File Edit View History Bookmarks Tools Help
http://localhost/thesis/home.php
Selenium Functional Testing for Web Apps
Open Source From ThoughtWorks and Friends
Show Log
Slow Mode
Command History:

b1112a3ee0784523bb7e83127c9fa206

```
type(username, thesis)
type(password, 123)
click(//input[@value='Login'])
waitForPageToLoad(20000)
windowMaximize()
```

Figure.2command history

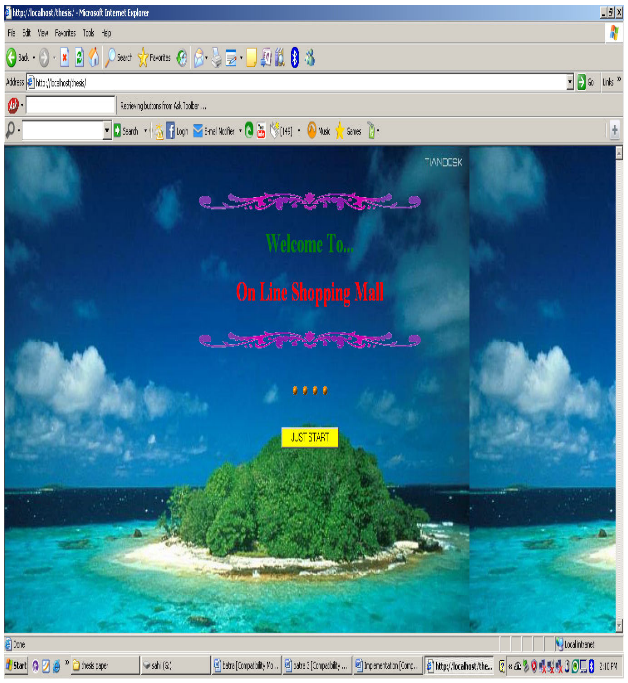


Figure.3 output

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

[1] Carl Erickson, Ralph Palmer, David Crosby, Michael Marsiglia, Micah Alles"Make Haste, not Waste: Automated System Testing" in 2004
[2] Cem Kaner" Inefficiency and Ineffectiveness of Software Testing: A Key Problem in Software Engineering" in 2004.
[3] Rex Black" Investing in Software Testing: The Cost of Software Quality" in 2003.
[4] Alan Kusinitz "Software Testing—How Much Is Enough?" in june 2003.
[5] Inspection vs. Testing in 2003.
[6] Pentti Pohjolainen"SOFTWARE TESTING TOOLS" in march 2003.
[7] B.Baudry, F.Fleurey, J.M Jezequel and Y.L.Traon "Automatic Test Cases Optimization using a Bacterological Adaption Model: Application to .NET Components" Published in IEEE 2002, pp.253-256
[8] Harish V. Kantamneni Sanjay R. Pillai Yashwant K. Malaiya "Structurally Guided Black Box Testing" in 2002
[9] Gerry Gaffney "Conducting a Walkthrough" in 2002.
[10] Thom Garrett" Useful Automated Software Testing Metrics" in 2000.
[11] Tom Chen, Mehmet Sahinoglu, Anneliese von Mayrhauser, Amjad Hajjar"How Much Testing is Enough? Applying Stopping Rules to Behavioral Model Testing" in 2001.
[12] David Banks, William Dashiell, Leonard Gallagher, Charles Hagwood, Raghu Kacker, Lynne Rosenthal "Software Testing by Statistical Methods" in 2001
[13]www.google.com
[14]http://www.wikipedia.org/