www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 3 March 2015, Page No. 10972-10976

A Mutation Testing Analysis And Regression Testing

Deepti singh[1], Ankit Thakur[2], Abhishek Chaudhary[3]

[1] M.Tech (CSE) Scholar, Bhagwant University,
Sikar Road, Ajmer, Rajasthan

[2] M.Tech (CSE) Scholar, Bhagwant University,
Sikar Road, Ajmer, Rajasthan

[3] Assistant Professor (CSE), Bhagwant University,
Sikar Road, Ajmer, Rajasthan

Abstract: *Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. In this paper we focused on two main software testing –mutation testing and mutation testing. Mutation testing is a structural testing method, i.e. we use the structure of the code to guide the test process. A mutation is a small change in a program. Such small changes are intended to model low level defects that arise in the process of coding systems. Ideally mutations should model low-level defect creation. Mutation testing is a method of software testing in which program or source code is deliberately manipulated, followed by suite of testing against the mutated code. The mutations introduced to source code are designed to imitate common programming errors. A good unit test suite typically detects the program mutations and fails automatically. Mutation testing is used on many different platforms, including Java, C++, C# and Ruby. Regression testing is a type of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes such as enhancements, patches or configuration changes, have been made to them. During confirmation testing the defect got fixed and that part of the application started working as intended. But there might be a possibility that the fix may have introduced or uncovered a different defect elsewhere in the software. The way to detect these '**unexpected side-effects**' of fixes is to do regression testing. The purpose of a regression testing is to verify that modifications in the software or the environment have not caused any unintended adverse side effects and that the system still meets its requirements. Regression testing are mostly automated because in order to fix the defect the same test is carried out again and again and it will be very tedious to do it manually. Regression tests are executed whenever the software changes, either as a result of fixes or new or changed functionality.*

Keywords: *Increment model, waterfall model, Software engineering, Software process model and Software*

1. INTRODUCTION

Software testing is the process of executing software in a controlled manner. When the end product is given to the client, it should work correctly according to the specifications and requirements stated by the client. Defect in software is the variance between the actual and expected results. There are different types of testing procedures, which when conducted, help to eliminate the defects from the program. Testing is the process of gathering information by making the observations and comparing them to expectations. In our day-to-day life, when we go out, shopping any product such as vegetable, clothes, pens, etc., we do check it before purchasing them for our satisfaction and to get maximum benefits. For example, when we intend to buy a pen, we test the pen before actually purchasing it, i.e. if it's writing, does it work in extreme climatic conditions, etc. So, be it the software, hardware, or any other product, testing turns be mandatory. Testing not only means fixing the bug in the code, but also to check whether the program is behaving according to the given specifications and testing strategies. Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects). The primary function of software testing is to detect bugs in order to uncover and detect it. The scope includes execution of that code in various environments and also to examine the aspects of the code - does the software do what it is supposed to do and function according to the specifications? As we move further we come across some questions such as, "When to start testing?" and "When to stop testing?" It is recommended to start testing from the initial stages of the software development. This not only helps in rectifying the errors before the last stage, but also reduces the rework of finding bugs in the initial stages every now and then. It saves time and is cost-effective. Software testing is an

ongoing process, which is potentially endless but has to be stopped somewhere, due to the lack of time and budget. It is required to achieve maximum profit with good quality product, within the limitations of time and money. The tester has to follow some procedural way through which he can judge if he covered all the points required for testing or missed out any. To help testers carry out these day-to-day activities, a baseline has to be set, which is done in the form of checklists.

This paper focused on two main types of software testing- Mutation Testing and Regression Testing. In Section -2 and Section -3 summarizes mutation testing & regression testing and its algorithm with their advantages and disadvantages, final discussion is concluded in Section-4.

2. MUTATION TESTING

Mutation test is one of those wonderful tests that allow you to assess the tests. Mutation test involves deliberately altering, modifying or changing a program code, later rerunning a suit of correct unit tests against the mutated program. As the demand for software grows, so is the complexity of the design. The more complex a software is, the higher will be the testing needs, quality assurance and customer satisfaction. Although testing is, an integral part of the software development process the issue is still on what is sufficient or adequate testing still open. Mutation testing is one way to verify if the software tester performed the testing in a proper manner. Through mutation testing, it can be determined, if the set of testing methods used in the developing process were appropriate and adequate to ensure product quality. To determine the correctness of a testing program, you will need to observe the behavior for each test case. If there are no detected faults, then the program is correct or it passes the test. A test case is the set of input values to the program under test and the corresponding output values. Mutation testing is a method or strategy to check the effectiveness or accuracy of a testing program to detect the fault in the system or program under test. The test is called so because mutants of the software are created and run with the test cases. Experts also call mutation testing as fault-based testing strategy because the mutants are created by introducing a single fault into the original program. You can set any number of mutants into the system. There are other versions of mutation testing, but they rely on the same method – to mutate the original software. Some of the other versions are weak mutation, interface mutation and specification based mutation. Mutation testing is not a new method, developers know about it since the late 1970s. However, researchers have been using it in the educational institutions than in the industrial software domain. Mutation testing is a simple but ingenious method used to validate source code correctness and the testing process. The concept was first coined by Richard Lipton in 1971, and there has been a surge of interest since that time. Mutation testing's working mechanism is simple and straightforward. A piece of source code encompassing all unit tests is selected. After verifying all positive testing for a given source code, a mutation is introduced into the program. The degree of mutation applied to a given code block may vary. A common mutation testing implementation involves replacing a logical operator with its inverse. For example, operator "! =" is used in place of "= =." In some cases, mutation involves rearranging lines to change the execution order or even deleting a few lines of code. Complex mutation testing levels may result in compilation errors. Once a program is modified, a suite of unit tests are executed against the mutated code. The mutated code passes or fails the unit test, depending on the testing quality. A well-written unit test must detect mutated code errors, resulting in failure. A unit test that fails to detect code errors may require a rewrite.

2.1 Mutation Testing Example

Mutation testing is done by selecting a set of mutation operators and then applying them to the source program one at a time for each applicable piece of the source code. The result of applying one mutation operator to the program is called a mutant. If the test suite is able to detect the change (i.e. one of the tests fails), then the mutant is said to be killed.Let's take example to find a maximum between two numbers:

```
function MAX(M<N:INTEGER)
  return INTEGER is
begin
   if M>N then
     return M;
  else
     return N;
  end if:
end MAX;
```

2.1.1First test data set--M=1, N=2

1. The original function returns 2

2. Five mutants: replace">" operator in if statements by (>,<,<=or=)

3. Executing each mutant:

Mutants	Outputs	Comparison
if M>=N then	**2**	**live**
if M<N then	**1**	**dead**
if M<=N then	**1**	**dead**
if M=N then	**2**	**live**

if M< >N then	**1**	**dead**

4. Adding test data M=2, N=1 will eliminate the latter live mutant, but the former live mutant remains live because it is equivalent to the original function. No test data can eliminate it.

2.2 Mutation Testing Algorithm

The following steps rely on the traditional mutation process:

1. Generate program test cases.
2. Run each test case against the original program.
    a. If the output is incorrect, the program must be modified and re-tested.
    b. If the output is correct go to the next step ...
3. Construct mutants using a tool like Mothra
4. Mutation testing will empower software testing process with trust and fidelity.
5. Execute each test case against each alive mutant.
    a. If the output of the mutant differs from the output of the original program, the mutant is considered incorrect and is killed.
6. Two kinds of mutants survive:
    a. Functionally equivalent to the original program: Cannot be killed.
    b. Killable: Test cases are insufficient to kill the mutant. New test cases must be created.
7. The mutation score for a set of test cases is the percentage of non-equivalent mutants killed by the test data.
8. Mutation Score = 100 * D / (N - E)
    a. D = Dead mutants
    b. N = Number of mutants
    c. E = Number of equivalent mutants
9. A set of test cases is mutation adequate if its mutation score is 100%.

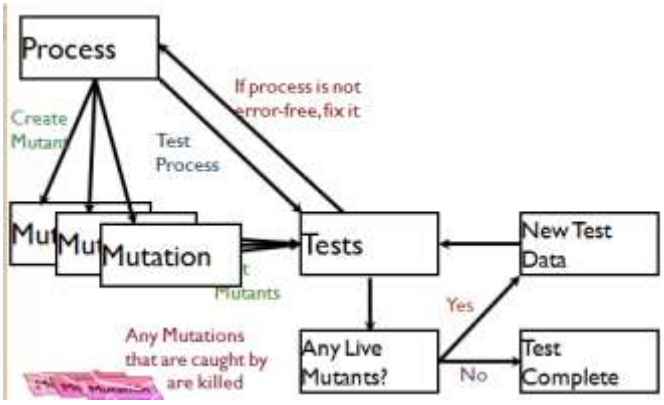

Fig 2. Process of Mutation Testing

2.3 Advantages and Disadvantages

2.3.1 Advantages:

1. Program code fault identification
2. Effective test case development
3. Detection of loopholes in test data
4. Improved software program quality
5. Elimination of code ambiguity

2.3.2 Disadvantages

1. Difficult implementation of complex mutations
2. Expensive and time-consuming
3. Requires skilled testers with programming knowledge

2.4 MUTATION OPERATORS

A variety of mutation operators were explored by researchers. Here are some examples of mutation operators for imperative languages:

1. Statement deletion.
2. Replace each boolean subexpression with *true* and *false*.
3. Replace each arithmetic operation with another one, e.g. + with *, - and /.

4. Replace each boolean relation with another one, e.g. > with >=, == and <=.
5. Replace each variable with another variable declared in the same scope (variable types should be the same).

3. REGRESSION TESTING

Regression means errors that occur due to some action or activities in a system. In IT world a "regression" means the return of a bugRegression testing is a style of testing that focuses on retesting after changes are made. In traditional regression testing, we reuse the same tests (the regression tests). In risk-oriented regression testing, we test the same areas as before, but we use different (increasingly complex) tests. Traditional regression tests are often partially automated. Regression test optimization techniques reduce the cost of regression testing by selecting a subset of an existing test suite to use in retesting a modified program. Over the years, numerous regression test optimization techniques have been described in the literature. Empirical studies of the techniques suggest that they can indeed benefit testers, but so far, few studies have empirically compared different techniques. In this paper, the results of a comparative empirical study of different regression test optimization techniques are represented. Regression testing attempts to mitigate two risks:

- A change that was intended to fix a bug failed.
- Some change had a side effect, unfixing an old bug or introducing a new bug

Regression means retesting the unchanged parts of the application. Test cases are re-executed in order to check whether previous functionality of application is working fine and new changes have not introduced any new bugs. This test can be performed on a new build when there is significant change in original functionality or even a single bug fix. This is the method of verification. Verifying that the bugs are fixed and the newly added features have not created in problem in previous working version of software. Testers perform functional testing when new build is available for verification. The intend of this test is to verify the changes made in the existing functionality and newly added functionality. When this test is done tester should verify if the existing functionality is working as expected and new changes have not introduced any defect in functionality that was working before this change. Regression test should be the part of release cycle and must be considered in test estimation. Regression testing is usually performed after verification of changes or new functionality. But this is not the case always. For the release taking months to complete, regression tests must be incorporated in the daily test cycle. For weekly releases regression tests can be performed when functional testing is over for the changes. Regression testing is initiated when programmer fix any bug or add new code for new functionality to the system. There can be many dependencies in newly added and existing functionality. It is a quality measure to check that new code complies with old code and unmodified code is not getting affected. Most of the time testing team has task to check the last minute changes in the system. In such situation testing only affected application area in necessary to complete the testing process in time with covering all major system aspects.

4.1 Regression Testing Example

Example of regression testing with its process is explained below:

For Example there are three Modules in the Project named Admin Module, Personal Information, and Employment Module and suppose bug occurs in the Admin Module like on Admin Module existing User is not able to login with valid login credentials so this is the bug. Now Testing team sends the above - mentioned Bug to the Development team to fix it and when development team fixes the Bug and hand over to Testing team than testing team checks that fixed bug does not affect the remaining functionality of the other modules (Admin, PI, Employment) and also the functionality of the same module (Admin) so this is known as the process of regression testing done by Software Testers.

1. Regression testing is required when there is a-
2. Change in requirements and code is modified
3. New feature is added to the software.
4. Defect fixing
5. Performance issue fix

4.2 Regression Testing Steps

Let P be a procedure or program, let P' be a modified version of P and let T be a test suite for P. Regression testing consists of reusing T the new test cases are needed to effectively test code or functionality added to or changed in producing A typical regression test proceeds as follows:

1. Select T' subset of T, a set of test cases to execute on P'.
2. Test P' with T'. Establish P' correctness with respect to T'.
3. If necessary, create T", a set of new functional or structural test cases for P'.
4. Test P' with T", establish to P"s correctness with respect to T".
5. Create T"', a new test suit and test history for P', from T, T',T"

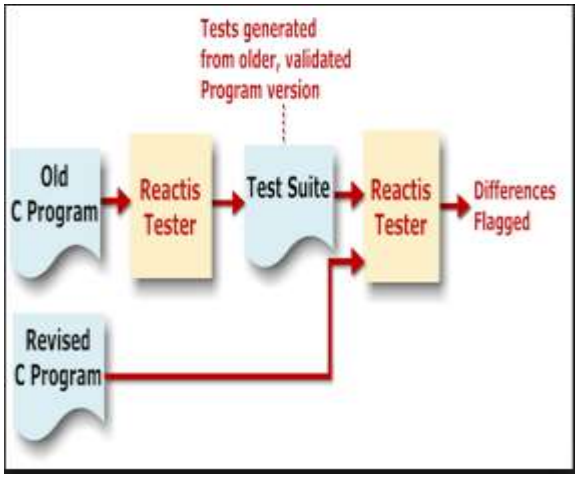

Fig.3- Regression Testing process

5. CONCLUSIONS

Software testing is a process of executing a program or application with the intent of finding the software bugs. We have many types of software testing but in this report we focused on two main type of testing and also the working of - Mutation testing and Regression Testing. Mutation testing is actually introducing small errors (called mutations) into your application (errors that are not supposed to fix bugs or provide new functionality) to see if your test suite picks them up. The idea is that, if your test suite doesn't pick up the mutations, it is deficient and should have more test cases added. In other words, mutation testing tests your test suite rather than your application. Regression testing is actually a test suite that is supposed to test as much functionality of your application as possible. The idea is to make a change to your application as required for bug fixing or new functionality, and regression testing will hopefully catch any problems (or regressions) with your changes. It's called regression since the vast majority of tests have been added due to previous bugs hence, if they find a problem, you have regressed to a previous state (in which the problem once again exists).In other words, regression testing tests our application.

REFERENCES

[1] A. S. Namin, J. H. Andrews, and D. J. Murdoch(2008), *Sufficient mutation operators for measuring test effectiveness*. In *Proc. ICSE*, pages 351–360.

[2] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G.Sayward(1979*)." Mutation analysis*". Technical report,G eorgia Institute of Technology.

[3] D. Schuler and A. Zeller(2009) "*Javalanche: Efficient mutation testing for Java*" In *Proc. FSE*, pages 297–298.

[4] J. H. Andrews, L. C. Briand, and Y. Labiche(2005)."*Is mutation an appropriate tool for testing experiments*?" In *Proc. ISE*, pages 402–411.

[5] L. Zhang, S. S. Hou, J. J. Hu, T. Xie, and H. Mei(2010) " *Is operator-based mutant selection superior to random mutant selection?"* In *Proc. ICSE*, pages 435–444.

[6] M. B. Dwyer, and G. Rothermel.(2009)" *Regression model checking."* In *Proc. ICSM*, pages 115–124.

[7] M. J. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings,S. Sinha, S. A. Spoon, and A. Gujarathi (2001) "*Regression test selection for java software*." In *Proc. OOPSLA*, pages 312–326.

[8] M. Woodward and K. Halewood.(1988) "*From weak to strong, dead oralive? an analysis of some mutation testing issues*". In *Proc.* of the Second Workshop on Software Testing, Verification, and Analysis,pages 152–158.

[9] P.Ammann and J.Offutt."*Introduction to Software Testing*." Cambridge University Press.

[10] P. E. Black, V. Okun, and Y. Yesha(2001), "*Mutation of Model Checker Specifications for Test Generation and Evaluation*," in Proceedings of the 1st Workshop on Mutation Analysis (MUTATION'00), published in book form, as Mutation Testing for the New Century. San Jose,California, pp. 14–20.

[11] P. G. Frankl, S. N. Weiss, and C. Hu(1997) *"All-uses vs mutation testing:An experimental comparison of effectiveness*" *JSS*, 38(3):235–253.

[12] R. Abraham and M. Erwig(2009), *"Mutation Operators for Spreadsheets*,"IEEE Transactions on Software engineering, vol. 35, no. 1, pp. 94–10.

[13] R. H. Carver(1993),"*Mutation-Based Testing of Concurrent Programs*," in Proceedings of the IEEE International Test Conference on Designing, Testing, and Diagnostics, Baltimore, Maryland, 17-21,pp. 845–853.

[14] W. E. Howde(1982)" *Weak mutation testing and completeness of test sets*".*IEEE TSE*, pages 371–379.