A MUTATION TESTING ANALYSIS AND REGRESSION TESTING

Amit Singh[1] and Khushbu babbar[2]

Department of Computer Engineering, Bhagwant University, Ajmer, Rajasthan

ABSTRACT

Software testing is a testing which conducted a test to provide information to client about the quality of the product 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 procedural testing method, i.e. we use the structure of the code to guide the test program, A mutation is a little change in a program. Such changes are applied to model low level defects that obtain in the process of coding systems. Ideally mutations should model low-level defect creation. Mutation testing is a process of testing in which code is modified then mutated code is tested against test suites. The mutations used in source code are planned to include in common programming errors. A good unit test 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. When defects are found during testing, the defect got fixed and that part of the software started working as needed. But there may be a case that the defects that fixed have introduced or uncovered a different defect in the software. The way to detect these ***unexpected bugs*** *and to fix them used regression testing. The main focus of regression testing is to verify that changes in the software or program have not made any adverse side effects and that the software still meets its need. Regression tests are done when there are any changes made on software, because of modified functions.*

KEYWORDS

Increment model, waterfall model, Software engineering, Software process model and Software.

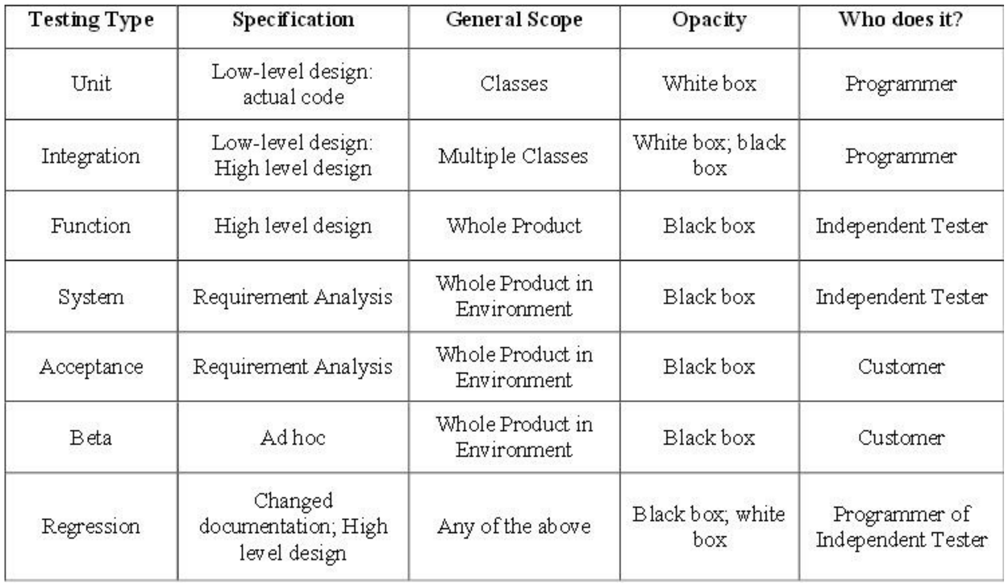
1. INTRODUCTION

Software testing is the process of executing software in a controlled manner. When the final product is delivered to the client it must be worked according to the requirements and needs of the client. The term defect in software is the changes between the actual and expected results. There are many different types of testing process, which and when conducted helps to eliminate the defects in the program. Testing is the process of gathering information by making the observations and comparing them to expectations. In our daily routine life, when we go out, for shopping any product such as vegetable, clothes, pens, etc., we check them before purchasing that its meet our requirements for our satisfaction and to get maximum gain. For example, when we need to buy a mobile phone, we test the phone before actually purchasing it, i.e. it must be worked in any condition. Testing is most important before using any software, hardware or any other product. Testing not only used to fix the error in the code, but also to check whether the program is work according to the given requirements and testing procedure. Software testing is a testing which conducted a test to provide information to client about the quality of the product 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. The main focus of software testing is to detect bugs in order to uncover and fix it. The scope of testing is to execute code in various conditions and also code work as we aspects and fulfil all the needs and produce output according to specifications. Testing must be done from the initial stages of the software development.

Software testing not only detect an error it operate the controlled condition of the software to (1) verify that it work correctly (2) to find an errors and (3) to validate the needs and requirements of user and the system work accordingly.

1. Verifications the checking the software consistency by checking the results against the requirements as already specified.
2. Error Detection: Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn’t or things don’t happen when they should.
3. Validation checks the system correctness –i.e. the process of checking and fulfilling the needs of the client.

A good test case is one that find the uncovered defects and fix them out. A successful test is one that uncovers an error which not find yet. A good test should neither be too simple nor too complex. To check if the system does what it is expected to do. Testing is done to check whether the application meets the requirements and executed in any conditions.



Testing Type	Specification	General Scope	Opacity	Who does it?
Unit	Low-level design: actual code	Classes	White box	Programmer
Integration	Low-level design: High level design	Multiple Classes	White box; black box	Programmer
Function	High level design	Whole Product	Black box	Independent Tester
System	Requirement Analysis	Whole Product in Environment	Black box	Independent Tester
Acceptance	Requirement Analysis	Whole Product in Environment	Black box	Customer
Beta	Ad hoc	Whole Product in Environment	Black box	Customer
Regression	Changed documentation; High level design	Any of the above	Black box; white box	Programmer of Independent Tester

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 testing is the process of changing, modifying or altering a program code, later retesting a suit of correct unit tests against the mutated program. With the increase in demand for software, increases the complexity of the design. As the complexity of software increases, need of testing increases and customer satisfaction also. Mutation testing is a testing to check w the software tester done the testing in a correct manner so that software works according to the needs of user.

Through mutation testing, we can find whether the set of testing methods used in the development process were good and correct to ensure final product quality. Correctness of the testing schedule can be determined by checking the behavior for each test conditions. If there are no defects detected, then the program is appropriate or it passes the test. For testing the program we take set of input values as a test condition so that it works according to needs and the corresponding output values according to the specifications. Mutation testing is the test to check the consistency and accuracy of a testing program to find the bugs in the program by using different test conditions. Mutation testing is a fault-based testing strategy because the faults are created by introducing a single error into the original program. There are many versions of mutation testing like weak mutation, interface mutation and specification based mutation. The method was first introduced by Richard Lipton in 1971, and there has been a huge interest since that time. Mutation testing's working method is simple to use. In mutation testing we replace 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. If the program is modified then test suite are executed against the code. The mutated code passes or fails the unit test, depending on the testing condition. Then a 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 the test in which operators get replaced with their reverse in 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;
```

2.1.1 First 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:
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.

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**

2.2 Mutation Testing Algorithm

The following steps rely on the traditional mutation process:

1. Generate program test part.
2. Test part should run against the original program.
    a. If the output is incorrect, the program must be changed 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. Each test part execute 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 parts 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. The mutation score is 100%, if set of test parts of mutation is correct

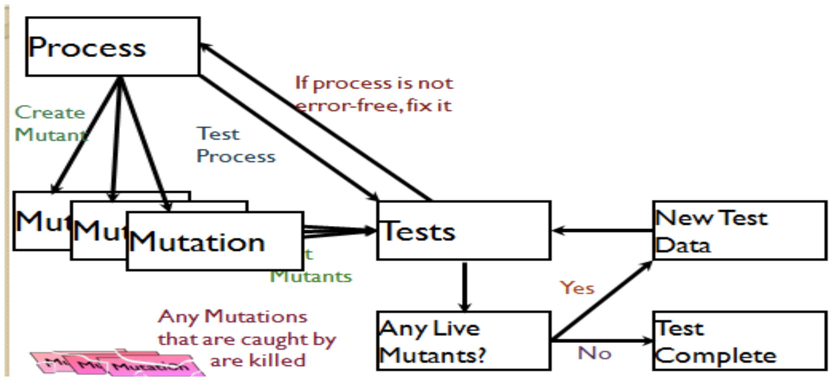


Fig 2. Process of Mutation Testing

2.3 Advantages and disadvantages

2.3.1 Advantages:

Program code fault identification

1. Effective test case development
2. Detection of loopholes in test data
3. Improved software program quality
4. 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. Mutation operators examples used for testing:

1. Statement deletion.
2. Change each Boolean expression with true and false.
3. Change each arithmetic operator with any other + with *, - and /.
4. Change each Boolean relation with any other > with >=, == and <=.
5. Change each variable with another variable declared in the same part (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 sector a "regression" means the return of an error or bug. Regression testing is a type of testing that emphasis on retesting at every change is made. Traditional regression tests are often partially automated. The numbers of regression testing have been developed for the optimization. The studies of the techniques suggest that regression testing can provide benefit to testers. Regression testing attempts to mitigate two risks:

- Changes that was planned to fix an error failed.
- Sometime modification made some problems like introducing a new bug

Regression testing is the testing of test cases to check whether the old function of application is working correctly and modification have not made any new bugs. This test can be applied on new techniques when there is an important change in original functions or even a single bug fix. Regression is the process of verification. It verifies that errors are fixed and the newly added functions will not create any problem in previous working program or software. Testers perform functional testing when new part of program is available for verification. The main focus of this test is to verify the changes made in the existing functions of program and new added functions. When test gets completed the software tester verify if the existing functions of previous software is working as expected and new changes have not introduced any defect or bugs in function as that was working before this changes. Regression testing is usually performed after verification of changes or new functionality. The release of regression tests takes months to complete, regression testing must be applied in the daily test cycle. Regression testing is applied when tester fix any error or add new code for any new functions to the software. Regression testing compiles new code with the previous code to provide the quality to the client and also it does not affect the previous code. Testing team also check the changes done in last minutes in the system. In that condition testing only affected application area in necessary condition to complete the testing process on time with taking all major system changes and aspects.

3.1 Regression Testing Example

Example of regression testing with its process is explained below:
Let's take example of college in which there are three main parts named Administration part, Personal Information, and Faculty. Suppose error occurs in the Administration part, existing User

is not able to login with valid login details then error get occurred. Then Testing team pass the bugs to the Development team to correct it and when development team fixes the Bug and again give it to testing team than they checks that bugs get fixed and does not change the remaining functions of other parts and also the functions of the same part. This process is known as regression testing done by Software Testers.

1. Regression testing is required when there is an error in program.
2. Change in requirements and code is modified.
3. New feature is added to the software.
4. Bugs get removed and Defect gets fixed.

3.2 Regression Testing Algorithm

Let's take P be a procedure or program, let P' be a improved 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 new functions 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 needed, let's take T'' which is a set of new function 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''

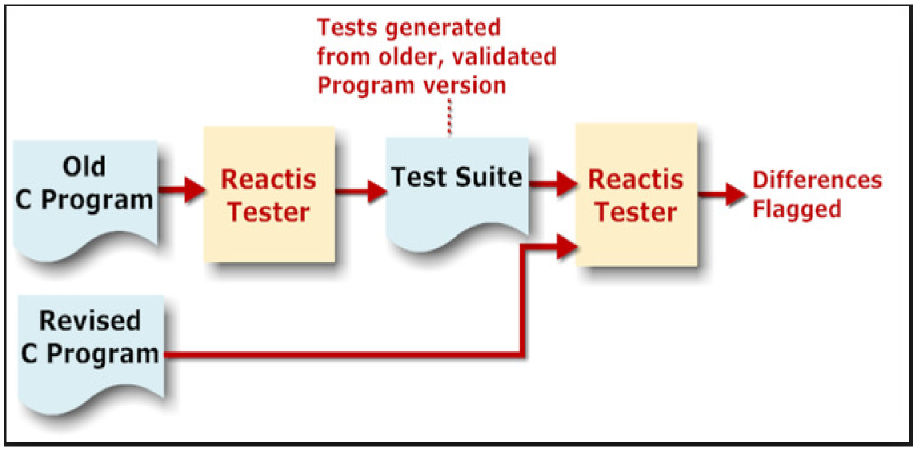


Fig.3- Regression Testing process

4. CONCLUSIONS

Software testing is a method of executing a program for finding the software bugs or errors. There are many types of software testing but in this paper we focused on two main type of testing and the working of Mutation testing and Regression Testing. Mutation testing is a process of producing small errors in the program to check that test suite picks them up. If test suite doesn't pick up the any error then it's lacking at somewhere then we have to add more test cases. In other words, mutation testing is the testing which tests the test suite rather than application. Regression testing is a test suite that is supposed to test as much function of program as possible. The idea is to make a changes in program as required for fixing bug or for new functions, and regression testing will try to catch any problems (or regressions) with any changes this is called regression since the vast majority of tests have been added due to old bugs and if any problem found, we have regressed to a old state (in which the problem once again exists).In other words, regression testing tests the program.

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.