Prioritizing the Test Cases for Regression Testing

Author

Rahul Gupta[1], Vinay Yadav[2]

[1](Research Scholar/Department of CSE/UPTU, Lucknow)
[2](Asst. Professor/Department of CSE/SR Group of Institutions, Lucknow)

Abstract : *Testing is the practice of making objective judgments regarding the extent to which the system (device) meets, exceeds, or fails to meet stated objectives.*
Regression Testing is mainly executed to show the desirable functionality of existing software after doing some modifications or changes in it. It is conducted to confirm the accuracy of the functionality of modified version of code. There are various techniques of regression testing which are briefly described: Retest All, Regression Test Selection, Test Case Prioritization, Hybrid Approach and Test Suite Reduction.
Test Case Prioritization is a crucially important technique. The goal of these techniques is to reduce the expense of regression testing. There are several techniques of test case prioritization, are described, based on amount of statement coverage, branch coverage in a code and fault-exposing- potential. The objective of this research work is to propose a technique of test case prioritization which is more effective in detecting faults. This technique is an extension of additional fault- exposing- potential (FEP) prioritization. The experimental results depicts that this technique results in earlier fault detection therefore results in consumption of time in testing process.

Keywords : *Additional Fault-Exposing-Potential (FEP), STLC, Regression Test Process*

1. Introduction

The most crucial phase in the software development life cycle is maintenance phase, in which the development team is supposed to maintain the software which is delivered to the clients by them. Software maintenance results for the reasons like error corrections, enhancement of capabilities, deletion of capabilities and optimization. Now the changed or modified software needs testing known as regression testing. Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of obsolete capabilities. These modifications in the software may cause the software to work incorrectly and may also affect the other parts of the software, so to prevent this Regression testing is performed. Regression testing (RT) is used to *revalidate* the modifications of the software. Regression testing is an expensive process in which test suites are executed ensuring that no new errors have been introduced into previously tested code

2. Software Testing Life Cycle (STLC)

Software Testing is not a just a single activity. It consists of series of activities carried out methodologically to help certify your software product. These activities (stages) constitute the Software Testing Life Cycle (STLC).

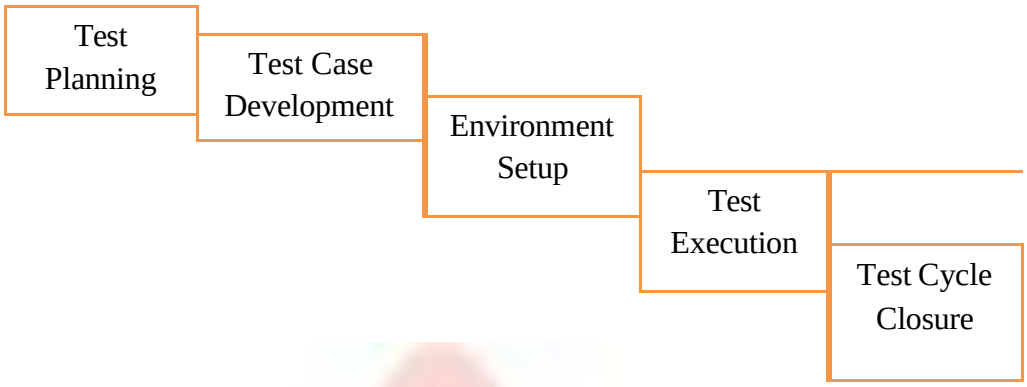

Figure 1 Software Testing Life Cycle (STLC)

Each of these stages has a definite Entry and Exit criteria , Activities & Deliverables associated with it. Here we will focus of activities and deliverables for the different stages in STLC.

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. Regression Testing is designed to address the above two purposes.

During the maintenance of a software system or as the software evolves, the regression testing is the expensive but definitely necessary task.

Regression testing is defined as "the process of retesting the modified parts of the software and ensuring that no new errors have been introduced into previously tested code".

3. Steps of Regression Testing

Let *P* be a procedure or program, let *P'* be a modified version of *P*, and let *T* be a test suite for P. A typical regression test procedures is:

Step 1: Select *T'*⊆*T*, a set of test cases to execute on *P'*.
Step 2: Test *P'* with *T'*, establishing *P'*"s correctness with respect to *T'*.
Step 3: If necessary, create T'', a set of new functional or structural test cases for *P'*.
Step 4: Test *P'* with *T''*, establishing *P'*"s correctness with respect to *T''*.
Step 5: Create *T'''*, a new test suite and test execution profile for *P'*, from t, *T'*, and *T''*

3.1. Need of Regression Test

This test is very important when there is continuous change/improvements added in the application. The new functionality should not negatively affect existing tested code

3.2. Regression Example:

Lets take an example of a program takes integer input (x,y) and should output x^y:

```
read (x, y);
if (y < 0) then
power = -y;
else
power = y;
endif
z = 1;
while (power != 0) do
z = z - y;
power = power – 1;
end while
if (y < 0) then
z = 1 / z;
endif
result = z;
write(result);
```

These two tests succeed and cover all branches:
(x=2, y=-1)
(x=1, y=0)

Now take a third test case:
(x=3, y=3)

Here is a Table 1 showing these 3 test cases with output.

Table 1 Example of Regression Testing

Test Case (x, y)	Expected Output	Actual Output
(2, -1)	½	½
(1, 0)	1	1
(3, 3)	27	-8

Test case (3, 3) fails!

Possible fix: when raising to a power, values should be multiplied, not subtracted!
Change line * to be:z= z * y;

Before modification, the program is

```
read(x, y);
if (y < 0) then
power = -y;
else
power = y;
endif
z = 1;
while (power != 0) do
* z = z - y;
power = power – 1;
endwhile
if (y < 0) then
z = 1 / z;
endif
result = z;
write(result);
```

Now after making required modification, the program is

```
read(x, y);
if (y < 0) then
power = -y;
else
power = y;
endif
z = 1;
while (power != 0) do
* z = z * y;
power = power – 1;
endwhile
if (y < 0) then
z = 1 / z;
endif
result = z;
write(result);
```

Here is a Table 2 showing output generated after modifications.

Table 2: Example of Regression Testing After Modifications

Test Case (x, y)	Expected Output	Actual Output
(2, -1)	½	-1
(1, 0)	1	1
(3, 3)	27	27

Now the test case (3, 3) now succeeds!
But, test case (2, -1) fails!
Regression testing should reveal this (correct fix: line* should be: z = z *x)

3.3. Regression Test Process (RTP)

Test revalidation refers to the task of checking which tests for P remain valid for P'. Revalidation is necessary to ensure that only tests that are applicable to P' are used during regression testing.

Test selection can be interpreted in several ways. Validated tests might be redundant in that they do not traverse any of the modified portions in P'. The identification of tests that traverse modified portions in P'. The identification of tests that traverse modified portions of P' is often referred to as test selection and sometimes as the *regression test selection* (RTS) problem.

Test minimization discards tests seemingly redundant with respect to some criteria. The purpose of minimization is to reduce the number of tests to execute for regression testing.

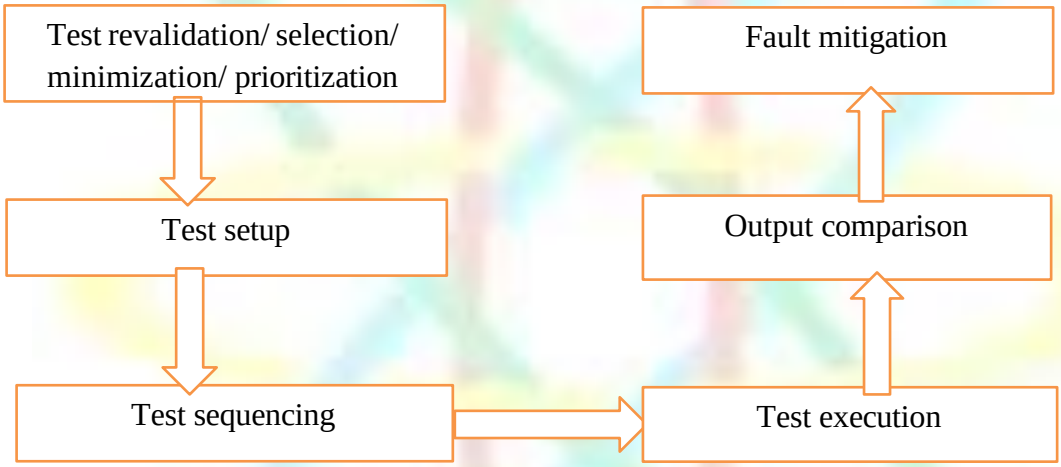

Figure 2 Regression Test Process (RTS)

A regression test process is exhibited above. The process assumes that *P'* is available for regression testing. There is usually a long series of tasks that lead to *P'* from *P*.

3.4. Goals of Prioritization

- There are many possible goals of prioritization, including the following:
- Testers may wish to increase the rate of fault detection of a test suite - that is, the likelihood of revealing faults earlier in a run of regression tests using that test suite.
- Testers may wish to increase the coverage of coverable code in the system under test at a faster rate, thus allowing a code coverage criterion to be met earlier in the test process.
- Testers may wish to increase their confidence in the reliability of the system under test at a faster rate.
- Testers may wish to increase the rate at which high risk faults are detected by a test suite, thus locating such faults earlier in the testing process.
- Testers may wish to increase the likelihood of revealing faults related to specific code changes earlier in the regression testing process.

3.5. Benefits of Prioritization

- An improved rate of fault detection during regression testing can let software engineers begin their debugging activities earlier than might otherwise be possible.
- It speeds up the release of the software.
- An improved rate of fault detection can also provide faster feedback on the system under test.
- It also provide earlier evidence when quality goals have not been met, thus allowing strategic decisions about release schedules to be made earlier than might otherwise be possible.
- In a testing situation in which the amount of testing time that will be available is uncertain, prioritization can increase the likelihood that, whenever the testing process is terminated, testing resources will have been spent more cost-effectively in relation to potential fault detection than they might otherwise have been.

4. Proposed Algorithm

Extended additional fault exposing potential (FEP) prioritization technique is the result of some modifications in additional fault exposing potential (FEP) prioritization. As in additional fault exposing potential (FEP), a term called *confidence* was used, in a very same way we also used this term. In this proposed technique, we use *C(s)*as a randomly generated value in our implementation. Let *s* denotes statement, *t* denotes test case and FEP (s, t) denotes faults in *s* covered by test case *t*, C(s) denotes confidence before execution of t and C'(s) denotes new confidence after execution of *t*.Therefore, after this change in the value of *C(s)*, equation here for the additional confidence in statement *s* becomes:

$$C\ addi\ (s) = (1 - rand\ (C(s))).\ FEP\ (s, t)$$

We can say that *C addi* (t) can be defined as the additional confidence gained from executing *t*on *P* program. It can be calculated by summing up the value of *C addi* (s) of all the statements *s* covered by *t*.

4.1. Algorithm

We here present a step by step algorithm for our proposed technique.
Input:Test cases t1, t2, . . ., tn, Test suite T and generated mutants.
Output:Prioritized test suite T'.

Process:

1. Begin
2. Set T' empty
3. For each test case t € T do
4. Calculate C addi
5. End for
6. Sort T in descending order of Caddi value
7. End

4.2. Techniques of Prioritization:

4.2.1. No Prioritization

In this case no techniques are implemented and is been used as a untreated test suit and it serves as a control.

4.2.2. Random Prioritization

This is applied to have an additional control in studies where the test cases are ordered randomly in the test suite.

4.2.3. Optimal Prioritization

In this technique known faults are been used so that its results can be used to measure the effects of other prioritization techniques which are to be used.

4.2.4. Total statement coverage prioritization

This technique instrument the program with any test cases and finds out that which statements were covered by the test cases; then these test cases can be prioritized on the bases of number of statements they covered. If more than one tests case covers equal number of statements then we have use some additional rules or we can order them randomly.

5. Result

Depicts the performance graph comparing the proposed prioritization technique to additional Fault-Exposing-Potential (FEP) prioritization technique. The above graph shows that this new technique detected greater number of mutants than already existing technique of prioritization. Therefore the fault detection is improved and it takes less time to detect more mutants in comparison to other techniques.

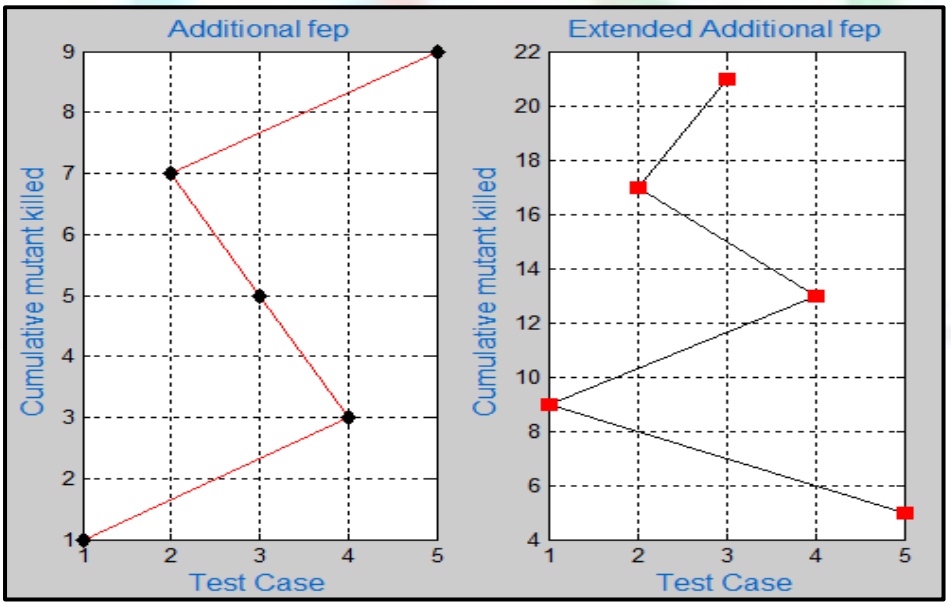

Figure 3 Performance Graph for The Comparison of Two Techniques

6. Conclusion

The purpose of this research work is to introduce the techniques of regression testing and the criteria of prioritization technique. Regression Testing is retesting the unchanged parts of a program in order to ensure that:

(a) Despite the changes, the existing unchanged part of a program continues to function as desired;
(b) Its revision does not produce faults. Regression Testing improves correctness and locate errors. It is a costly process in software development. So, there is a need to minimize cost.

Prioritization is one of a most important method of regression testing. It arranges the test cases present in a test suite based on different techniques of prioritization. So that tester can run only test cases which have higher priority. It results in consumption of cost and time. Therefore it is one of a most beneficial technique of regression testing. A new technique is introduced which makes some modifications made in additional fault- exposing- potential (FEP) prioritization. It is designed aiming at minimizing the time, effort and cost in software testing process.

References

[1]. A.J. Offutt, A. Lee, G. Rothermel, R. Untch, and C. Zapf, "An Experimental Determination of Sufficient Mutation Operators," *ACM Trans. Software Eng. and Methodology*, vol. 5, no. 2, pp. 99±118, Apr. 1996.
[2]. D. Binkley, "Semantics Guided Regression Test Cost Reduction," *IEEE Trans. Software Eng.*, vol. 23, no. 8, pp. 498±516, Aug. 1997.
[3]. Dennis Jeffrey, "Regression Testing", September 18, 2006
[4]. G. Rothermel and M. J. Harrold, Analyzing Regression Test Selection Techniques, *IEEE Transactions on Software Engineering*, v.22, no. 8, August 1996, pg. 529-551.
[5]. G. Rothermel and M.J. Harrold, "A Safe, Efficient Regression Test Selection Technique," ACM Trans. Software Eng. and Methodology, vol. 6, no. 2, pp. 173±210, Apr. 1997.