Efficient Test Suite Management

Prof. Durga Prasad Mohapatra
Professor
Dept.of CSE, NIT Rourkela

Why do test suites grow?

- Test cases in an existing test suite can often be used to test a modified program.
- However, if the test suite is inadequate for retesting, new test cases may be developed and added to the test suite.
- There may be unnecessary test cases in the test suite including both obsolete and redundant test cases.

Why do test suites grow? (contd..)

- A change in a program causes a test case to become obsolete by removing the reason for the test case's inclusion in the test suite.
- A test case is redundant if other test cases in the test suite provide the same coverage of the program.
- Thus due to obsolete and redundant test cases, the size of a test suite continues to grow unnecessarily.

Why test suite minimization is important?

- release date of the product is near
- limited staff to execute all the test cases
- limited test equipment or unavailability of testing tools.

Advantages of minimizing a test suite

- redundant test cases will be eliminated
- reduces the cost of the project by reducing a test suite to a minimal subset.
- decreases both the overhead of maintaining the test suite and the number of test cases that must be rerun after changes are made to the software, thereby reducing the cost of regression testing.
- so it is a great practical advantage to reduce the size of test cases.

Defining test suite minimization problems

Harrold et al. have defined the problem of minimizing test suite as given below.

- **Given:** a test suite TS; a set of test case requirements $r_1, r_2, \ldots, r_n$ that must be satisfied to provide the desired testing coverage of the program; and subsets of TS: $T_1, T_2, \ldots, T_n$, one associated with each of the r_i's such that any one of the test cases t_j belonging to T_i can be used to test r_i.

Defining test suite minimization problems (contd..)

- **Problem:** find a representative set of test cases from TS that satisfies all the r_i's
  - a representative set of test cases that satisfies the r_i's must contain at least one test case from each T_i,
  - such a set is called a **hitting set** of the group of sets, $T_1, T_2, ..., T_n$.

Defining test suite minimization problems (contd..)

- The r_i's can represent either all the test case requirements of a program or those requirements related to program modifications.
- maximum reduction is achieved by finding the smallest representative of test cases,
- however this subset of the test suite is the minimum cardinality hitting set of the Ti's.
- the problem of finding the minimum cardinality hitting set is NP-complete.
- thus minimization techniques resort to heuristics.

Test Suite Prioritization

- The reduction process can be understood if the cases in a test suite are prioritized in some order.
- The purpose of prioritization is to reduce the set of test cases based on some rational, non-arbitrary criteria, while aiming to select the most appropriate tests.
- For example, the following priority categories can be determined for the test cases.

Test Suite Prioritization cont…

- ***Priority I*** *The test cases must be executed*, otherwise there may be worse consequences after the release of the product.
- For example, if the test cases for this category are not executed, then critical bugs may appear.

Test Suite Prioritization cont…

- ***Priority 2*** *The test cases may be executed, if time permits.*
- ***Priority 3*** *The test case is not important prior to the current release.*
- It may be tested shortly after the release of the current version of the software.

Test Suite Prioritization cont…

- ***Priority 4*** *The test case is never important, as it's impact is nearly negligible.*
- In the prioritization scheme, the main guideline is to ensure that low-priority test cases do not cause any severe impact on the software.
- There may be several goals of prioritization.

Test Suite Prioritization cont…

- These goals can become the basis for prioritizing the test cases. Some of them are discussed here:

1. Testers or customers may want to get some critical features tested and presented in the first version of the software.
   - Thus, the important features become the criteria for prioritizing the test cases.
   - But the consequences of not testing some low-priority features must be checked.
   - So, risk factor should be analyzed for every feature in consideration.

Test Suite Prioritization cont…

2. Prioritization can be on the basis of the functionality advertised in the market.

It becomes important to test those functionalities on a priority basis, which the company has promised to its customers.

3. The rate of fault detection of a test suite can reveal the likelihood of faults earlier.

Test Suite Prioritization cont…

4. To increase the coverage of coverable code in the system under test at a faster rate, allowing a code coverage criterion to be met earlier in the test process.

5. To increase the rate at which high-risk faults are detected by a test suite, thus locating such faults earlier in the testing process.

Test Suite Prioritization cont…

6. To increase the likelihood of revealing faults related to specific code changes, earlier in the regression testing process.

Types of Test Case Prioritization

- General Test Case Prioritization
- Version-Specific Test Case Prioritization

General Test Case Prioritization

- In this prioritization, we prioritize the test cases that will be useful over a succession of subsequent modified versions of P, without any knowledge of the modified version.
- Thus, a general test case prioritization can be performed following the release of a program version during off-peak hours, and the cost of performing the prioritization is amortized over the subsequent releases.

Version-Specific Test Case Prioritization

- Here, we prioritize the test cases such that they will be useful on a specific version P' of P.
- Version-specific prioritization is performed after a set of changes have been made to P and prior to regression testing P', with the knowledge of the changes that have been made.

Prioritization Techniques

- Prioritization techniques schedule execution of test cases in an order that attempts to increase their effectiveness at meeting some performance goal.
- Various prioritization techniques may be applied to a test suite with the aim of meeting that goal.
- Prioritization can be done at two levels:
  - Prioritization for regression test suite
  - Prioritization for system test suite

Types of Prioritization Techniques

- Coverage-based Test Case Prioritization
- Risk-Based Prioritization
- Prioritization Based on Operational Profiles
- Prioritization using Relevant Slices
- Prioritization Based on Requirements

Coverage-based Test Case Prioritization

a) Total statement Coverage Prioritization
b) Additional Statement Coverage Prioritization
c) Total Branch Coverage Prioritization
d) Additional Branch Coverage Prioritization
e) Total Fault-Exposing-Potential(FEP) Prioritization

Total statement Coverage Prioritization

- This prioritization orders the test cases based on the total number of statements covered.
- It counts the number of statements covered by the test cases and orders them in a descending order.
- If multiple test cases cover the same number of statements, then a random order may be used.
- For example, if T_1 covers 5 statements, T_2 covers 3, and T_3 covers 12 statements; then according to this prioritization, the order will be T_3, T_1, T_2.

Additional statement coverage prioritization

- Total statement coverage prioritization schedules the test cases based on the total statements covered.
- However, it will be useful if it can execute those statements as well that have not been covered yet.
- It iteratively selects a test case T_1, that yields the greatest statement coverage, and then selects a test case which covers a statement not covered by T_1.
- Repeat this process until all statements covered by at least one test case have been covered.

Additional statement coverage prioritization cont…

- For example, if we consider Table 1, according to total statement coverage criteria, the order is 2, 1, 3.
- But additional statement coverage selects test case 2 first and next, it selects test case 3, as it covers statement 4 which has not been covered by test case 2. Thus, the order according to addition coverage criteria 2, 3, 1.

Table 1. Statement coverage

Statement	Statement Coverage		
	Test Case 1	Test Case 2	Test Case 3
1	X	X	X
2	X	X	X
3		X	X
4			X
5			
6		X	
7	x	X	
8	X	X	
9	X	x	

Total branch coverage prioritization

- In this prioritization, the criterion to order is to consider condition branches in a program instead of statements.
- Thus, it is the coverage of each possible outcome of a condition in a predicate.
- The test case which will cover maximum branch outcomes will be ordered first.
- For example, in Table 2, the order will be 1, 2, 3.

Total branch coverage prioritization cont…

Table 2. Branch coverage

Branch statements	Branch Coverage		
	Test case 1	Test case 2	Test case 3
Entry to while	X	X	X
2-true	X	X	x
2-false	X		
3-true		X	
3-false	X		

Additional Branch Coverage Prioritization

- The idea is the same as in additional statement coverage of first selecting the test case with the maximum coverage of branch outcomes and
  - then, selecting the test case which covers the branch outcome not covered by the previous one.

Total Fault-Exposing-Potential (FEP) Prioritization

- Statement and branch coverage prioritization ignore a fact about test cases and faults:
  - Some bugs/faults are more easily uncovered than other faults.
  - Some test cases have the proficiency to uncover particular bugs as compared to other test cases.

Total Fault-Exposing-Potential (FEP) Prioritization

- Thus, the ability of a test case to expose a fault is called the fault exposing potential.
- It depends not only on whether test cases cover a faulty statement, but also on the probability that a fault in that statement will also cause a failure for that test case.
- To obtain an approximation of the FEP of a test case, an approach was adopted using mutation analysis.

This approach is discussed below.

- Given program P and test suite T,
- First create a set of mutants N = {n1, n2,…..nm} for P, noting which statement s_j in P contains each mutant.
- Next, for each test case t_i ɛ T, execute each mutant version n_k of P on t_i , noting whether t_i kills that mutant.
- Calculate FEP(s,t) = Mutants of s_j killed / Total number of mutants of s_j.
- To perform total FEP prioritization, given these FEP(s,t) values, calculate an *award value* for each test case ti ɛ T, by summing the FEP(sj, ti) values for all statements s_j in P. Given these award values, we prioritize test cases by sorting them in order of descending award value.

The next table shows FEP estimates of a program. Total FEP prioritization outputs the test order as (2, 3, 1).

Statement	FEP(s,t) values		
	Test case 1	Test case 2	Test case 3
1	0.5	0.5	0.3
2	0.4	0.5	0.4
3		0.01	0.4
4		1.3	
5			
6	0.3		
7	0.6		0.1
8		0.8	0.2
9			0.6
Award values	1.8	3.11	2.0

Summary

- Discussed why do test suites grow.
- Explained why test suite minimization is important.
- Explained the basics of test suite prioritization.
- Presented the different types of prioritization techniques.
- Discussed in detail the different types of coverage-based test case prioritization techniques.

References

1. Naresh Chauhan, Software Testing: Principles and Practices, (Chapter – 12), Second Edition, Oxford University Press, 2016.

Thank you