Efficient Test Suite Management

M Saeed Siddik

OBJECTIVES

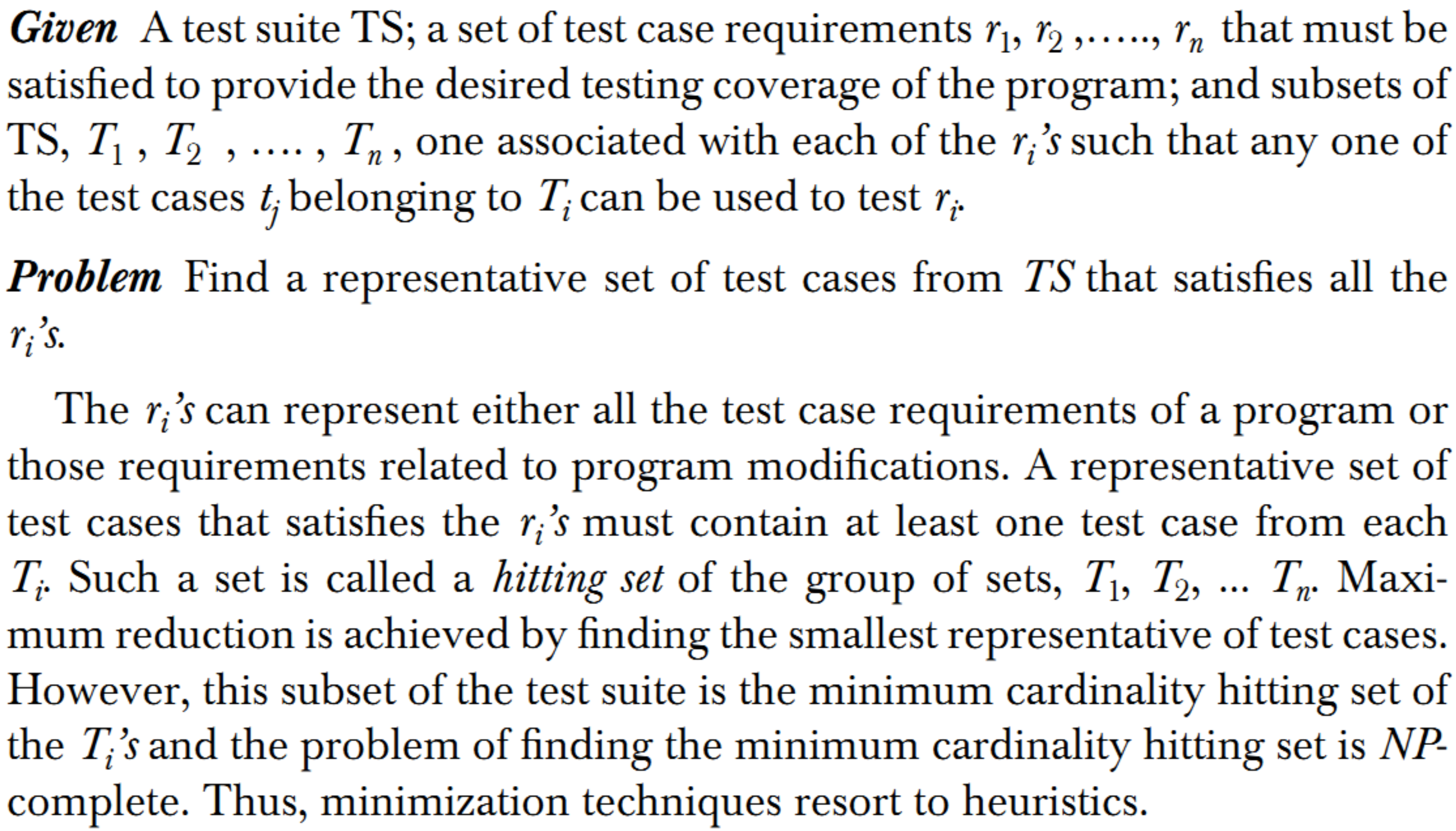
After reading this chapter, you should be able to understand:

- Why the test suite grows as the software evolves
- Why is it required to minimize the number of test cases
- Factors to be considered while minimizing a test suite
- Definition of test suite minimization problem
- Test suite prioritization as a method to reduce test cases
- Prioritization techniques

WHY DOES A TEST SUITE GROW?

- A testing criterion is a rule or collection of rules that imposes requirements on a set of test cases.
- Test engineers measure the extent to which a criterion is satisfied in terms of coverage; a test set achieves 100% coverage if it completely satisfies the criterion.
- Coverage is measured in terms of the requirements that are imposed; partial coverage is defined as the percentage of requirements that are satisfied.

DEFINING TEST SUITE MINIMIZATION PROBLEM



Given A test suite TS; a set of test case requirements r_1, r_2 ,….., r_n that must be satisfied to provide the desired testing coverage of the program; and subsets of TS, T_1 , T_2 , …. , 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.

Problem Find a representative set of test cases from *TS* that satisfies all the r_i*'s*.

The r_i*'s* can represent either all the test case requirements of a program or those requirements related to program modifications. 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. 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 T_i*'s* and the problem of finding the minimum cardinality hitting set is *NP*-complete. Thus, minimization techniques resort to heuristics.

TEST SUITE PRIORITIZATION

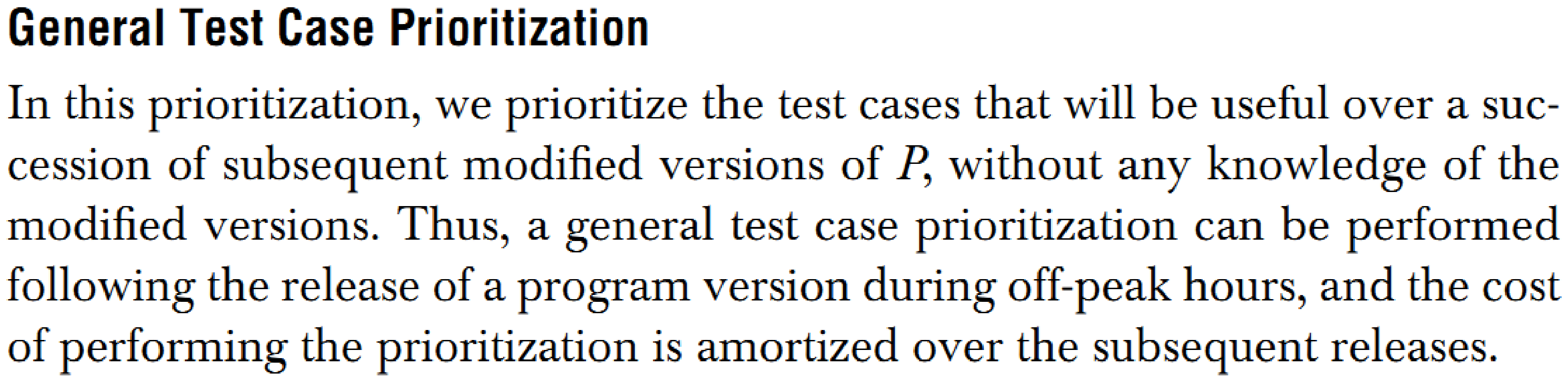
Priority 1 *The test cases must be executed*, otherwise there may be worse consequences after the release of the product. For example, if the test cases for this category are not executed, then critical bugs may appear.

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.

Priority 4 *The test case is never important, as its impact is nearly negligible.*

TYPES OF 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 versions. 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.

TYPES OF TEST CASE PRIORITIZATION

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 of P', with the knowledge of the changes that have been made.

PRIORITIZATION TECHNIQUES

Prioritization for regression test suite This category prioritizes the test suite of regression testing. Since regression testing is performed whenever there is a change in the software, we need to identify the test cases corresponding to modified and affected modules.

Prioritization for system test suite This category prioritizes the test suite of system testing. Here, the consideration is not the change in the modules. The test cases of system testing are prioritized based on several criteria: risk analysis, user feedback, fault-detection rate, etc.

COVERAGE-BASED PRIORITIZATION

- Total Statement Coverage Prioritization
  - This prioritization orders the test cases based on the total number of statements covered
- Additional Statement Coverage Prioritization
  - Total statement coverage prioritization schedules the test cases based on the total statements covered
- Total Branch Coverage Prioritization
  - The test case which will cover maximum branch outcomes will be ordered first.
- Additional Branch Coverage Prioritization
  - first selecting the test case with the maximum coverage of branch outcomes
- Total Fault-Exposing-Potential (FEP) Prioritization
  - the ability of a test case to expose a fault is called the *fault exposing potential*

RISK-BASED PRIORITIZATION

- A risk analysis table consists of the following columns:
- Probability of occurrence/fault likelihood
  - It indicates the probability of occurrence of a problem.
- Severity of impact/ failure impact
  - If the problem has occurred, how much impact does it have on the software.

Risk analysis table

- Problem ID: A unique identifier to facilitate referring to a risk factor.
- Potential problem: Brief description of the problem
- Uncertainty factor: It is the probability of occurrence of the problem.
- Probability values: are on a scale of 1 (low) to 10 (high).
- Severity of impact Severity values on a scale of 1 (low) to 10 (high).
- Risk exposure Product of probability of occurrence and severity of impact.

PRIORITIZATION BASED ON REQUIREMENTS

- *Customer-assigned priority of requirements* Based on priority, the customer assigns a weight (on a scale of 1 to 10) to each requirement. Higher the number, higher is the priority of the requirement.
- *Requirement volatility* This is a rating based on the frequency of change of a requirement. The requirement with a higher change frequency is assigned a higher weight as compared to the stable requirements.
- *Developer-perceived implementation complexity* All the requirements are not equal on a implementation level. The developer gives more weight to a requirement which he thinks is more difficult to implement.
- *Fault proneness of requirements* This factor is identified based on the previous versions of system. If a requirement in an earlier version of the system has more bugs, i.e. it is error-prone, then this requirement in the current version is given more weight. This factor cannot be considered for a new software.

MEASURING THE EFFECTIVENESS OF A PRIORITIZED TEST SUITE

- When a prioritized test suite is prepared, how will we check its effectiveness? For this purpose For this purpose, the rate of fault-detection criterion can be taken. El- baum developed APFD (average percentage of faults detected) metric that measures the weighted average of the percentage of faults detected Percent detected faults during the execution of a test suite. Its value ranges from 0 to 100, where a higher value means a faster fault-detection rate.

APFD is calculated as given below.

$$\text{APFD} = 1 - ((\text{TF}_1 + \text{TF}_2 + \ldots\ldots\ldots + \text{TF}_m)/nm) + 1/2n$$

where TF_i is the position of the first test in test suite T that exposes fault i
m is the total number of faults exposed in the system or module under T
n is the total number of test cases in T

APFD

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
F1					X			X		
F2		X	X	X		X				
F3	X	X	X	X			X	X		
F4						X				X
F5	X		X		X	X		X		X
F6					X	X	X		X	
F7									X	
F8		X		X		X			X	X
F9	X									X
F10			X	X					X	X

Consider a program with 10 faults and a test suite of 10 test cases

If we consider the order of test suite as (T1, T2, T3, T4, T5, T6, T7, T8, T9, T10), then calculate the APFD for this program.

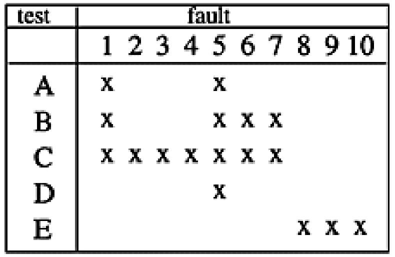
Solution

$$\text{APFD} = 1 - ((5+2+1+6+1+5+9+2+1+3)/10*10) + \frac{1}{2\times 10}$$

$$= 0.65 + 0.05$$

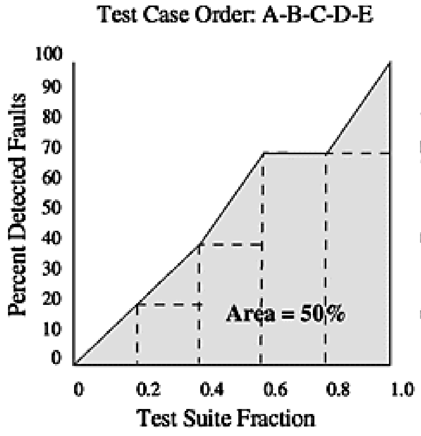
$$= 0.7$$

APFD

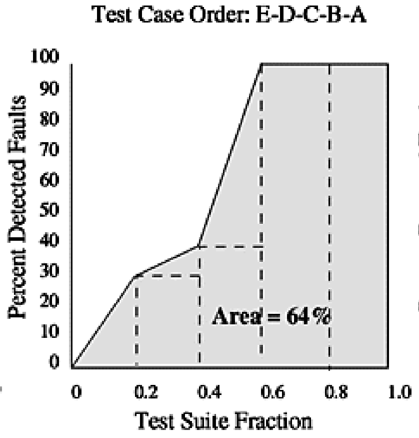


test	fault									
	1	2	3	4	5	6	7	8	9	10
A	x				x					
B	x				x	x	x			
C	x	x	x	x	x	x	x			
D					x					
E								x	x	x

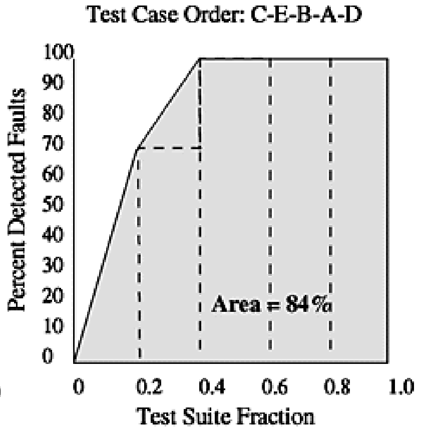
(a)



(b)



(c)



(d)

Thank You