

Chapter 3: Dynamic Testing Techniques

"...The system was not fully tested to a satisfactory level of quality and resilience before full implementation on 26 October 1992."

Extract from the main conclusions of the official report into the failure of the London Ambulance Service's Computer Systems on October 26th and 27th and November 4th 1992.

3.1 OVERVIEW

This module introduces idea of a test case design technique. Broadly, these are categorized as either functional or structural testing techniques. The advantage of using these proven design methods is that they provide a more intelligent and effective means of identifying tests than a purely intuitive approach. You will be expected to know two functional techniques and two structural techniques in detail and be aware there are many other techniques that can be applied to test case design. There are many excellent books on testing techniques some of which are listed in the appendix.

3.2 OBJECTIVES

After completing this module you will:

- § Understand the difference between black box (functional) and white box (structural) testing techniques.
- § Be able to name at least three black box techniques.
- § Understand how to use equivalence partitioning and boundary value analysis to design test cases.
- § Appreciate the use of state transition testing.
- § Be able to name at least three white box techniques.
- § Understand the meaning of statement testing and branch testing.
- § Be aware that the BS7925 standard details some (but not all) of the recognized testing techniques.
- § Know that all of these testing techniques can be supplemented by error guessing.

3.3 INTRODUCTION

Each of the techniques we are about to describe has its strengths and weaknesses. A useful rule of thumb is that if you are having difficulty applying a particular technique to a testing problem then perhaps you ought to try a different technique. This section introduces the different types of testing technique and discusses the difference between them.

3.3.1 Functional test techniques

Functional test techniques are often referred to as 'black box' test techniques and the common parlance is that we are 'doing black box testing'. A functional test technique will help design test cases based on functionality of component or system under test, without necessarily having to understand underlying detail of software design. Consider functionality of system of determine test inputs and expected results.

Structural test techniques are sometime called white box text techniques hence term 'white box testing'. Glass box testing is less widely used term for structural test case design. Structural test techniques help design test cases based on internal structure and design of component or system under test. Look at code, database spec, data model, etc., to determine test inputs and expected results.

Exercise

What's the difference between Black and White box testing?

	BASED ON	USED BY	SUITABLE FOR
Black Box	Requirements Functionality	Independent tester Users	System test Acceptance test
White Box	Internal structure Code DB design	Developer Designer	Unit test Link test Sub-system test

3.4 BLACK BOX TECHNIQUES

The following list of black box techniques is from BS 7925-2. On this course we will describe and give an example of only those ones highlighted in bold:

- Ø Equivalent Partitioning.
- Ø Boundary Value Analysis
- Ø State Transition Testing
- Cause-Effect Graphing
- Ø Syntax Testing /

Equivalence partitioning (EP) is a test case design technique that is based on the premise that the inputs and outputs of a component can be partitioned into classes that, according to the component's specification, will be treated similarly by the component.. Thus the result of testing a single value from an equivalence partition is considered representative of the complete partition.

As an example consider any program that accepts days of the week and months of the year as inputs. Intuitively you would probably not expect to have to test every date of the year. You would obviously try months with 30 days (e.g. June) and months with 31 days (e.g. January) and you may even remember to try out the special case of February for both non-leap year (28 days) and leap years (29 days). Equally, looking at the days of the week you would not, depending on the application, test every day. You may test for weekdays (e.g. Tuesday) and weekends (e.g. Sunday). What you are in effect doing is deciding on equivalence classes for the set of data in question.

Not everyone will necessarily pick the same equivalence classes; there is some subjectivity involved. But the basic assumption you are making is that any value from the equivalence, class, is as good as any other when we come to design the test.

We hope that you can see how this technique can dramatically reduce the number of tests that you may have for a particular software component.

ISTQB Certification Preparation Guide: Chapter 3 – Dynamic Testing Techniques

Exercise

Consider a software component called GENERATE-GRADING, which is used by the system to calculate student's grade based on an examination mark and a coursework mark. The component is passed an exam mark (out of 75) and a coursework mark (out of 25) from which it generates a grade for the course in the range 'A' to 'D'. The grade is calculated from the overall mark, which is calculated as the sum of the exam mark and coursework marks, as follows.

OVERALL MARK	GRADE
Greater than or equal to 70	A
Greater than or equal to 50 but less than 70	B
Greater than or equal to 30 but less than 50	C
Less than 30	D

ISTQB Certification Preparation Guide: Chapter 3 – Dynamic Testing Techniques

Exercise

Identify valid partitions: _____

Identify invalid partitions: _____

Identify output partitions: _____

Now derive some test cases based on input exam mark partition:

TEST CASE	1	2	3
Input (exam mark)			
Input (coursework)			
Total mark (calculated)			
Partition tested			
Expected output			

ISTQB Certification Preparation Guide: Chapter 3 – Dynamic Testing Techniques

The standard goes on to derive test cases from the other partitions but in this course we will not have time to do this completely.

Boundary Value Analysis is based on the following premise. Firstly, the inputs and outputs of a component can be partitioned into classes that, according to the component's specification, will be treated similarly by the component and, secondly, that developers are prone to making errors in their treatment of the boundaries of these classes. Thus test cases are generated to exercise these boundaries.

Exercise

Look at the boundaries for input exam mark. Choose values on each boundary and one either side of it in order to generate a set of test cases.

TEST CASE	1	2	3	4	5	6
Input (exam mark)						
Input (coursework)						
Boundary tested						
Expected output						

Again the standard continues to describe boundary tests for all of the other valid boundaries.

3.5 White Box Techniques

The following list of white box techniques is from BS79252. On this course we will describe and give an example of only those ones highlighted in bold.

- Ø Statement testing
- Ø Branch Decision Testing.
- Ø Data Flow Testing.
- Branch Condition Testing. .
- Ø Branch Condition Combination Testing
- Ø Modified Condition Decision Testing.
- Ø LCSAJ Testing.
- Ø Random Testing.

Statement testing is a structural technique based on decomposition of a software component into constituent statements. Statements are identified as executable or non-executable. For each test case, you must specify inputs to component, identification of statement(s) to be executed and expected outcome of test case.

Example of program with 4 statements:

```
X=INPUT;  
Y=4;  
IF X>Y THEN  
Message= "Limited exceeded"  
  
ELSE  
  
    Message= "No problems reported" END IF  
Z=0
```

Branch testing requires model of source code, which identifies decisions and decision outcomes. A decision is an executable statement, which may transfer control to another statement depending upon logic of decision statement. Typical decisions are found in loops and selections. Each possible transfer of control is a decision outcome. For each test case you must specify inputs to component, identification of decision outcomes to be executed by test case and expected outcome of test case.

3.5.1 Comparison of white box techniques from "Software Testing in the Real World"

	Statement Coverage	Decision Coverage	Condition Coverage	Decision/Condition Coverage	Multiple condition coverage
Each statement is executed at least once	Y	Y	Y	Y	Y
Each decision takes an all possible outcomes	N	Y	N	Y	implicit
Each condition in a decision takes on all possible outcome at least once	N	N	Y	Y	implicit
All possible combinations of condition outcomes in each decision occur at least once	N	N	N	N	Y

Each column in this figure represents a distinct method of white-box testing, and each row (1-4) defines a different test characteristics. For a given method (column), "Y" in a given row means that test characteristic is required for method. "N" signifies no requirement. "Implicit" means test characteristic is achieved implicitly by other requirements of method. (@ 1993, 1994 Software Development Technologies) reproduced with permission.

ISTQB Certification Preparation Guide: Chapter 3 – Dynamic Testing Techniques

3.7 Summary

In module three you have learnt that applying a formal, recognized testing technique in a systematic way is the most effective approach to finding errors in software. In particular you can now

- § Explain the difference between black box (functional) and white box (structural) testing techniques.
- § Name at least three black box techniques.
- § *Use* equivalence partitioning and boundary value analysis to design test cases.
- § Recognize a state transition testing technique.
- § Name at least three white box techniques.
- § Understand what the meaning of statement testing and branch testing.
- § *Use* the standard BS7925 to find out more about testing techniques.
- § Know when to apply error-guessing techniques to supplement the formal techniques.