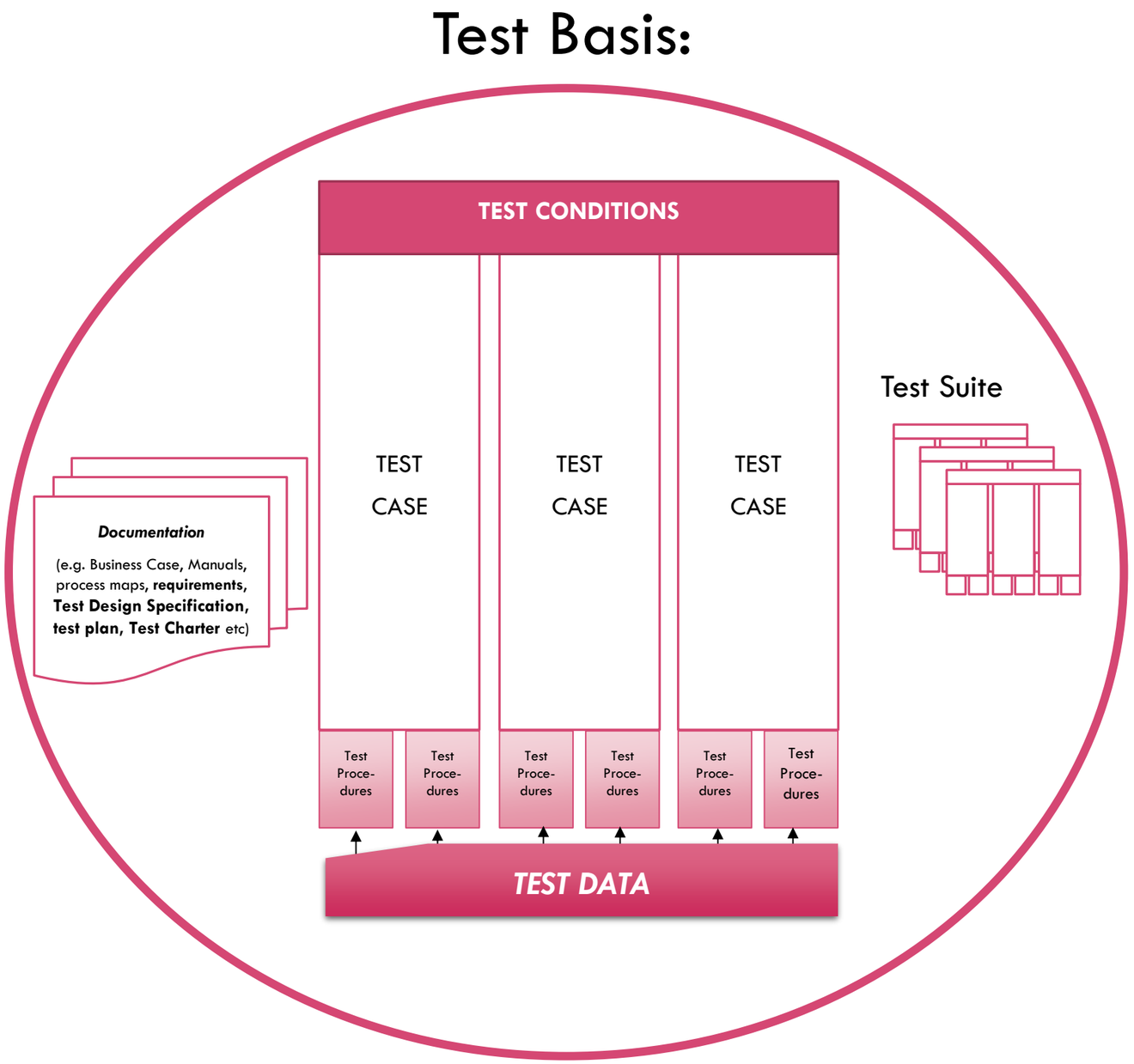
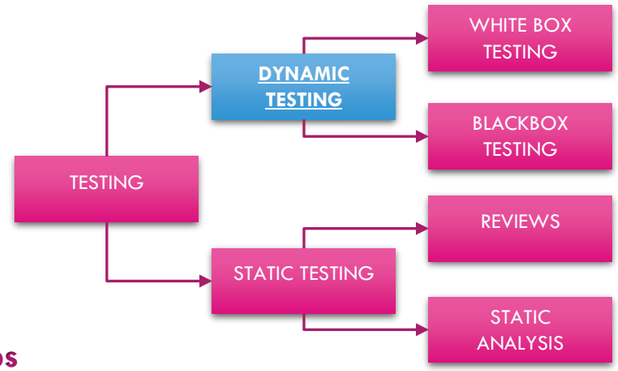
4. TEST DESIGN TECHNIQUES

A. TEST DESIGN PROCESS





i. Test Design Process Steps

ii. KEY TERMS:

Test Plan – A document describing the scope, approach, resources and schedule of intended test activities

Test Design Specification – Specifies the test conditions for a test item, the detailed test approach and identifies the associated high-level test cases

Test basis/base – The body of knowledge used as the basis for test analysis and design.

Test Condition – An aspect of the test basis that is relevant in order to achieve specific test objectives. Roughly defined as the aim/goal of a certain test.

Test Case – A set of preconditions, inputs, actions (where applicable), expected results and postconditions, developed based on test conditions.

Test Suite – A set of test cases or test procedures to be executed in a specific test cycle.

Test Data – Data created or selected to satisfy the execution preconditions and inputs to execute one or more test cases (i.e. collection of test cases grouped for test execution purposes)

Test Coverage – The degree to which specified coverage items have been determined or have been exercised by a test suite expressed as a percentage (or simply – % of test cases in test suite carried out).

Test Charter – Documentation of test activities in session-based exploratory testing.

Test Oracle – A source to determine an expected result to compare with the actual result under test. E.g. the existing system (for a benchmark), other software, user manuals or an individual's specialised knowledge, but should NOT be the code.

B. TEST DESIGN TECHNIQUES OVERVIEW

DYNAMIC TESTING TECHNIQUES

WHITE BOX TESTING

(aka Structural Testing, Glass box testing, code-based testing)

Test types

Statement Coverage
Decision Coverage

BLACK BOX TESTING

(aka Specification based testing, requirements-based testing, business requirements testing, functional testing)

Test types

Equivalence Partitioning
Boundary Value Analysis
Decision Table Testing
State Transition Testing
Use Case Testing

EXPERIENCE BASED TESTING

(can be Whitebox or Blackbox):

Error Guessing
Exploratory Testing
Checklist-Based testing

C. BLACKBOX TESTING

i. Smoke and Sanity Checks

Smoke Test – Check system health, aim is not to find a defect. Verifies critical functions E.g. does this computer turn on without going up on flames.

Sanity Test – Makes sure the bugs reported in previous builds are fixed for this release before doing a full regression test. Verifies NEW functionality.

ii. Decision Table Testing

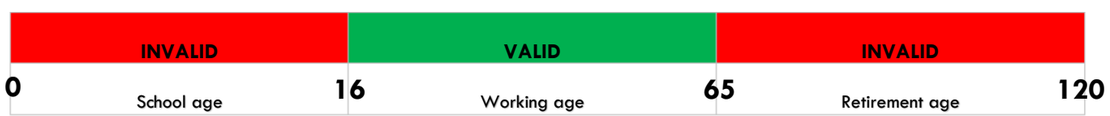
A decision table lists all the **input conditions** that can occur (as a **Boolean** value) and all the **actions** that can arise from them. Test cases can be derived from these tables. Example below:

ID	Conditions/Actions	Test Case 1	Test Case 2	Test Case 3	Test Case 4
Condition 1	Account exists	Y	Y	Y	Y
Condition 2	Pin code matched	Y	Y	N	N
Condition 3	Sufficient money in account	Y	N	Y	N
Action 1	Transfer money	X			
Action 2	Show message 'Insufficient amount'		X		
Actions 3	Block transaction			X	X

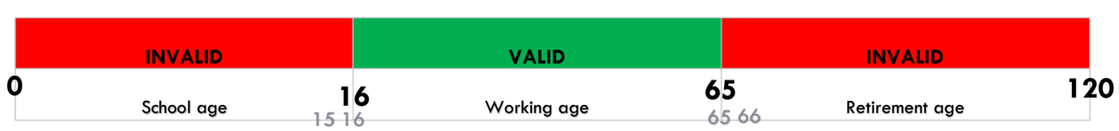
iii. Equivalence Class Partitioning/Boundary Value Analysis

Equivalence Class Partitioning (ECP) & Boundary Value Analysis (BVA) share similarities and are sometimes used in tandem for better test coverage

Equivalence Class Partitioning (ECP) – Sometimes it is impossible to test all values where there is a range in the input field (Exhaustive testing is impossible – 7 principles of testing). To overcome this, we can divide the data into **equivalence classes** and subsequently a test case can be designed for each equivalence class. This ensures we still have optimal test coverage without the need to excessive testing. The example below shows a range of people's ages who are of school age, working age and retirement age. Only those of working age are considered. They are split into partitions Only 3 test cases need to be made to test the 3 different partitions (i.e. a value between 0-15, a value between 16-64 and a value between 65-120).

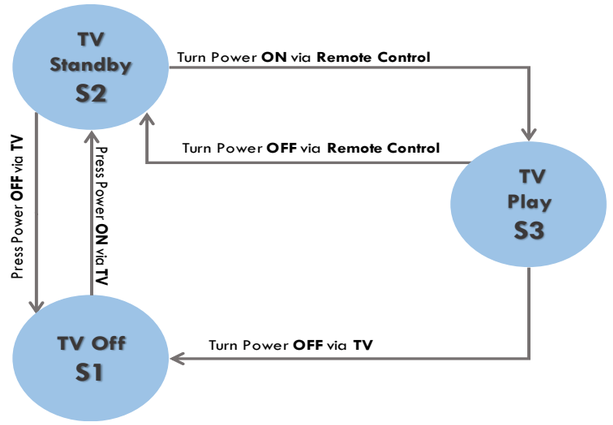


Boundary Value Analysis (BVA) – BVA explores the boundary conditions of a program as more errors occur at the boundaries. As with ECP, data is divided into equivalence classes. Each boundary has a valid boundary value and an invalid boundary value. Test cases are created based on the both valid and invalid boundary values. The figure below shows that 7 test cases (shown in grey) can be created using the BVA testing technique if we use the same data range used in the ECP example:



iv. State Transition Testing

When a system is represented as being in one state and transitions from that state to another. The transformations are determined by the rules of the system. This means we can follow these rules to create a diagram that represents the change of transitions and thus a test to see if it works. An example of turning on and off TV and putting it in standby is shown below:



States – How something exists at that time. (TV On/Off/Standby)

Transitions – The change from one state to another (The arrows on the diagram). Transitions are triggered by...

Inputs or Events – (e.g. Press Power OFF via Remote Control)

Actions – The actions that can result from a transition (able to watch TV)

Transitions can be tabulated into a **State Transition Table**:

ID	Transition 1	Transition 2	Transition 3	Transition 4	Transition 5
Start State	S1	S2	S2	S3	S3
Input	Press ON via TV	Press OFF via TV	Press ON via Remote Control	Press OFF via Remote Control	Press OFF via TV
Effect (expected outcome)	TV standby	TV off	TV play	TV standby	TV off
End State	S2	S1	S3	S2	S1

When it comes to **coverage**, the above table only show "**0-switch**" transitions (i.e. all transitions are made once). If all transitions are made twice, this I called "**1-switch**" coverage.

v. Use Case Testing

Use cases (see figure to the right) describe actions between actor's and systems. Steps by step test cases are derived from use cases for each 'Use Case' (e.g. the tasks 'Track progress' will have a step by step test case):

Actors – Whoever or Whatever expects the service from the system

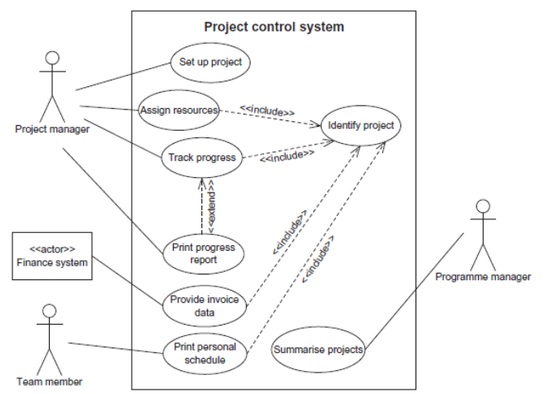
Use Cases – Shown as an oval and represents a function the system will perform as a response to a trigger from an actor

Systems Boundary – A box around all of the use cases. Helps to agree the scope of the system

Associations – The lines that link actors to use cases

EXTEND: Next step is optional

INCLUDE: Next steps MUST be included



D. WHITEBOX TESTING

i. Decision and Statement Coverage

Decision (branch) Coverage (DC)- Check if each possible branch from a decision point has been executed at least once (i.e. total number of test cases required to ensure each one of the possible branches from each decision point is executed at least once).
Decision Condition Testing – A white box test design technique in which test cases are designed to execute condition outcomes and decision outcomes.
Statement Coverage (SC) – Check if each statement in the code has been executed at least once (i.e. total number of test cases required to execute each node at least once).
Statement Testing – A white box test design technique in which test cases are designed to execute statements.

ii. Finding out the DC and SC from code & pseudocode

Pseudocode is a simplified version of regular code that can be understood by those who are not familiar with coding languages.

From pseudocode control flows can be (more easily) created (see figure on the right-hand side) and from this, the number of test cases for total SC and DC for the code under test can be calculated.

Example of code (not pseudocode) and the corresponding control flow are in the figure on the right-hand side.

Examples of a string of pseudocode and the number of test cases for complete DC and SC is shown below. Drawing out a control flow diagram will help understand why the SC are the DC are what they are:

Example 1:

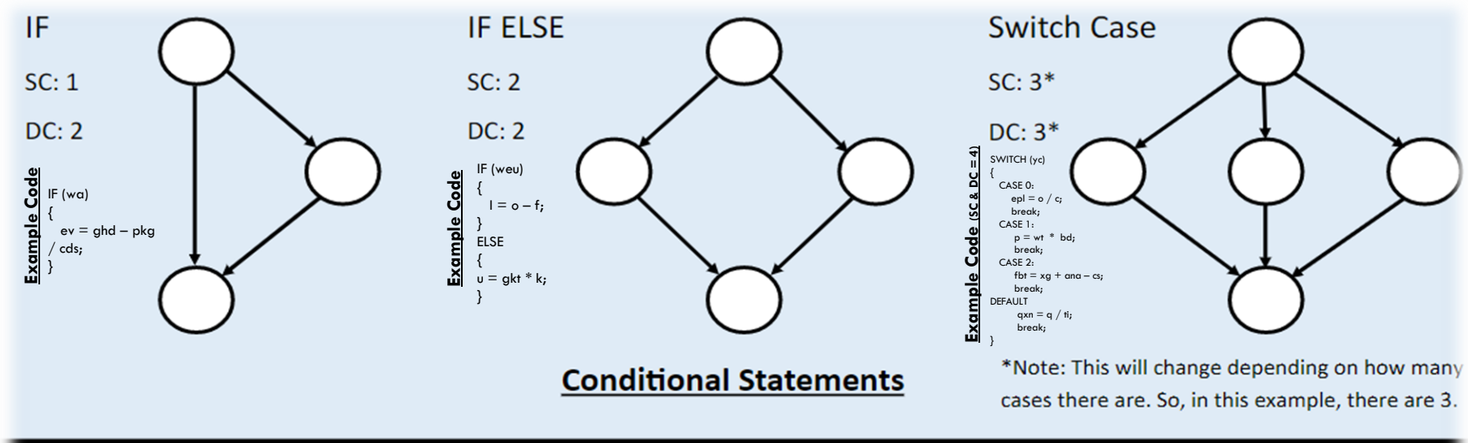
```
Read P
Read Q
IF P+Q > 100
THEN
Print “Large”
ENDIF
If P > 50 THEN
Print “P Large”
ENDIF
```

SC = 1
DC = 2

Example 2:

```
READ A
READ B
READ C
IF C>A THEN
    IF C>B THEN
        PRINT "C must be smaller bla bla.."
    ELSE
        PRINT "Proceed to next stage"
    ENDIF
ELSE
```

SC = 2
DC = 2



E. EXPERIENCED BASED TESTING

i. Error Guessing

Error Guessing is a test design technique where the experience of the tester is used to anticipate what defects might be present in the component or system under test as a result of errors made, and to design tests specifically to expose them.

Defect and Failure lists - can help identify areas that are susceptible to problems
Fault Attack – This approach is to list possible errors and design tests around that list.

ii. Exploratory Testing

An informal and reactive test design technique where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests. Avoids ‘Pesticide Paradox’.
Advantages:
It doesn’t require much preparation
Useful when there is time pressure, lack of resources or inadequate specs
Testers report a large proportion of bugs via this method.
Disadvantages:
There is no review of test planning, an experienced user of the system may not be an experienced tester.
Testers have to remember the exact steps they took to create a defect – otherwise reproduction may be difficult.

iii. Checklist-based testing

Testers design, implement, and execute tests to cover test conditions found in a checklist.

As part of analysis, testers create a new checklist or expand an existing checklist, but testers may also use an existing checklist without modification. Such checklists can be built based on experience
Checklists can be created to support various test types, including **functional** and **non-functional testing**.
In the *absence* of detailed test cases, checklist-based testing can provide **guidelines** and a degree of consistency.
Have to remember the exact steps they took to create a defect – otherwise reproduction may be difficult.