# Definitions

## Test development process

* Test development process represents a sequence of events for developing a test that varies for different programs.
* General procedures:
  + Determine the purpose of testing
  + Build test specification
  + Create test items
  + Review the test items
  + Evaluate the quality of the items

## Test Condition

* An **item** or **event** that **could be verified** by one or more test cases

## Test Case

* Developed to cover a certain test **objective(s)** or test **condition(s).**
* Consists of:
  + Input **values**
  + Execution **preconditions**
  + Expected **results**
  + Execution **postconditions**

## Test Procedure Specification

* Specifies the sequence of actions for the execution of a test

## Test script

* The sequence of actions when tests are run using a test execution tool

## Test execution schedule

* Consists of various test procedures and automated test scripts
* Defines the **order** in which tests are **executed**
* Takes into account **factors** like:
  + **Regression** tests
  + **Prioritization**
  + Technical and logical **dependencies**

1. **Test Design Techniques**

* **Static Techniques -** Testing Without Executing   
  the Program
* It checks work products to find errors in early stage

#### Review

* **Informal review**
* **Walkthrough**
* **Technical review**
* **Inspection**
* **Peer review**

#### Static Analysis

* **Dynamic Techniques**

#### Black-box

* + A way to derive and select test conditions, test cases, or test data
  + Based on an analysis of the test basis **documentation**
  + Also called **specification-based** or **behavioral** techniques
  + Tests are based on the way the system is **supposed to work**
  + Black-box testing does not use any information regarding the **internal structure** of the component or system to be tested
  + The **code** of the tested object is not considered

#### Functional

* What the system does?
* (e.g. Unit, Sanity, Integration, Regression)

#### Non functional

* How the system does what it does?
* (e.g. Load, Performance, Stress, Security, Installation, Recovery)

#### Black-box Techniques

* + **Equivalence Partitioning**
  + **Boundary Value Analysis**
  + **Use Case Testing**
  + **Decision Table Testing**
  + **State Transition Testing**
  + **Pairwise Testing**
  + **Classification Trees Testing**

#### White-box

* + A way to derive test cases based on analysis of the code if it works as expected
  + Design test cases that
    - Exercise independent paths within a module or unit
    - Exercise logical decision //both true and false side
    - Execute loops at their boundaries
    - Exercise internal data structures
  + Also called **structural** or **glass box** testing
  + White-box testing uses the internal structure of the component or system
    - The code of the tested object is considered
  + Testing can be commenced at an earlier stage. It’s applicable to:
    - **Unit** testing
    - **Integration** testing
    - **System** testing

#### White-box techniques

* + - **Statement Coverage**
* Test every possible statement in the code at least once
  + - **Branch / Decision Coverage**
      * Tests to ensure that all branches are tested at least once
    - **Path Coverage**
      * Tests all the paths of the program

#### Experience-based

* Based on a person's knowledge, experience, skills and intuition
* Basic experience-based techniques
* **Exploratory testing**
* **Error guessing**

#### Defect-based

## Static tests

* Do not involve running (executing) the test object

## Dynamic tests

* Involve running (executing) the test object

## Choosing Test Techniques

* The choice of test techniques to be used depends on a number of factors:
* Type of the system
* Regulatory standards
* Customer or contractual requirements
* Level of risk
* Type of risk
* Test objective
* Documentation available
* Knowledge of the testers
* Time and budget
* Development lifecycle
* Use case models
* Previous experience with types of defects found

## Equivalence Partitioning

* Equivalence partitioning is about testing various groups that we expect the system to handle the same way
* Exhibiting **similar behavior** for every single member of an equivalence partition
* Test cases are designed to **cover each partition at least once**
* Equivalence partitioning aims **reducing the total** **number of test cases** to a feasible count
* **Equivalent classes (partitions)** are **portions** of an input or output domain
* The behavior of a component or system is assumed to be **the same for every member** of a partition class, based on the specification
* The operation of equivalence partitioning is performed by **splitting a set** (domain) into two or more **disjoint sets**
* **Valid** **equivalence classes**
* Describe valid situations
* The system should **handle them normally**
* **Invalid equivalence classes**
* Describe invalid situations
* The system should **reject** them
* Or at least **escalate to the user** for correction or exception handling
* **Multiple invalids should not be combined** in a single test
* The presence of one invalid value might **mask the incorrect handling** of another invalid value

## Boundary Value Analysis