




ISTQB Foundation Level

Day 1

By Mr. Pavan



Agenda - Day 1

- Introduction to ISTQB
- Principles of testing
- Testing throughout the life-cycle
- Static techniques
- Tool support for testing



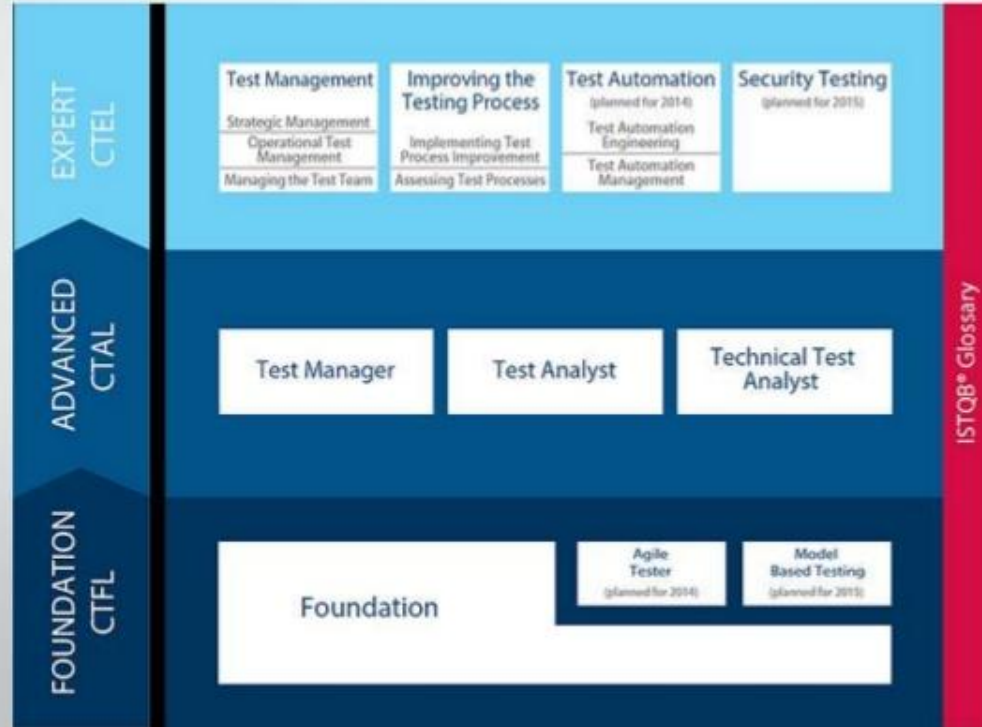
Introduction - ISTQB

What is ISTQB?

- The **International Software Testing Qualifications Board** (ISTQB) is a software testing qualification certification organization that was founded in Edinburgh in November 2002, and is a non-profit association legally registered in Belgium
- ISTQB® Certified Tester is a standardized qualification for software testers and based on a [syllabus](#), and there is a hierarchy of qualifications and guidelines for accreditation and examination

<http://www.istqb.org/about-istqb.html>

ISTQB Levels & Modules




Foundation Level Exam Structure

Exam	Number of questions per K levels				
	K1	K2	K3	K4	Total
Foundation	20	12	8		40

- a scoring of 1 point for each correct answer
- a pass mark of 65% (26 or more points)
- a duration of 60 minutes (or 75 minutes for candidates taking exams that are not in their native or local language).

What are K- Levels?

- K₁ (Remember) = The candidate should remember or recognize a term or a concept
- K₂ (Understand) = The candidate should select an explanation for a statement related to the question topic
- K₃ (Apply) = The candidate should select the correct application of a concept or technique and apply it to a given context
- K₄ (Analyze) = The candidate can separate information related to a procedure or technique into its constituent parts for better understanding and can distinguish between facts and inferences



Principles of Testing

What is Software Testing?

- It can also be stated as the **process of validating and verifying** that a software program or application or product:
 - Meets the business and technical requirements that guided it's design and development
 - Works as expected
 - Can be implemented with the same characteristic.

Why is testing necessary?

- **Mistakes.** Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong –humans make mistakes all the time.
- **Bad assumptions and blind spots.** So we might make the same mistakes when we check our own work as we made when we did it. So we may not notice the flaws in what we have done.
- Ideally, someone else should check our work because another person is more likely to spot the flaws.

What is a “bug”?

- **Error:** a human action that produces an incorrect result
- **Fault:** a manifestation of an error in software
 - also known as a defect or bug
 - if executed, a fault may cause a failure
- **Failure:** deviation of the software from its expected delivery or service
 - (found defect)

Failure is an event; fault is a state of the software, caused by an error

Exhaustive testing?

- What is exhaustive testing?
 - when all the testers are exhausted ✗
 - when all the planned tests have been executed ✗
 - exercising all combinations of inputs and preconditions ✓
- How much time will exhaustive testing take?
 - infinite time ✗
 - not much time ✗
 - impractical amount of time ✓

How much testing is enough?

- it's never enough ✗
- when you have done what you planned ✗
- when your customer/user is happy ✗
- when you have proved that the system works correctly ✗
- when you are confident that the system works correctly ✗✓
- it depends on the risks for your system ✓

How much testing?

- It depends on **RISK**
 - risk of missing important faults
 - risk of incurring failure costs
 - risk of releasing untested or under-tested software
 - risk of losing credibility and market share
 - risk of missing a market window
 - risk of over-testing, ineffective testing

So little time, so much to test ..

- test time will always be limited
- use **RISK** to determine:
 - what to test first
 - what to test most
 - how thoroughly to test each item
 - what not to test (this time)
- use **RISK** to
 - allocate the time available for testing by prioritising testing ...


} i.e. where to
place emphasis

Seven Testing Principles

- Testing shows presence of defects
 - Testing reduces the probability of undiscovered defects remaining in the software but, no defects, is not a proof of correctness.
- Exhaustive testing is impossible
 - Testing everything (all combinations of inputs and preconditions) is not feasible. Instead, risk analysis and priorities should be used to focus testing efforts
- Early testing
 - To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives

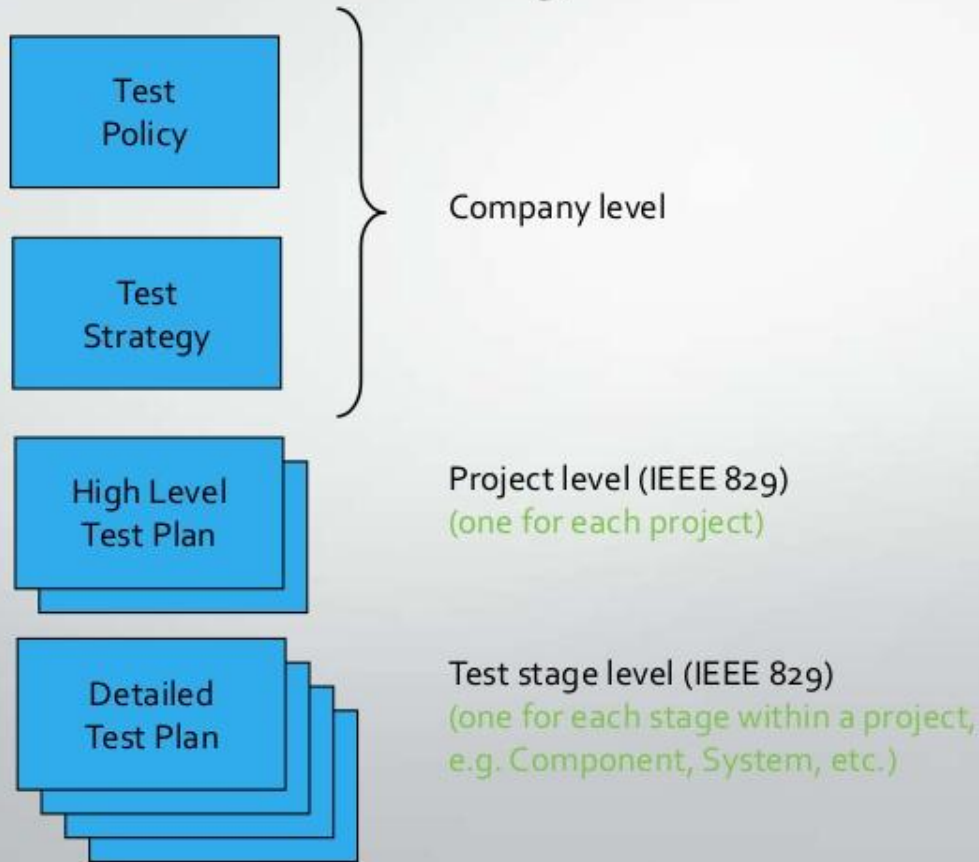
Seven Testing Principles

- Defect clustering
 - Testing effort shall be focused proportionally to the expected and later observed defect density of modules
- Pesticide Paradox
 - If the same tests are repeated over and over again, they will no longer find any new defects. To overcome this test cases need to be reviewed and revised, to exercise different parts of the software
- Absence-of-errors fallacy
 - Finding and fixing defects does not help if the system built is unusable and does not fulfill the users needs and expectations
- Testing is context dependent
 - Testing is done in different contexts. For example, safety-critical software is tested differently from an e-commerce site.



Fundamental Test Process

Test Planning - different levels



The test process

Planning (detailed level)

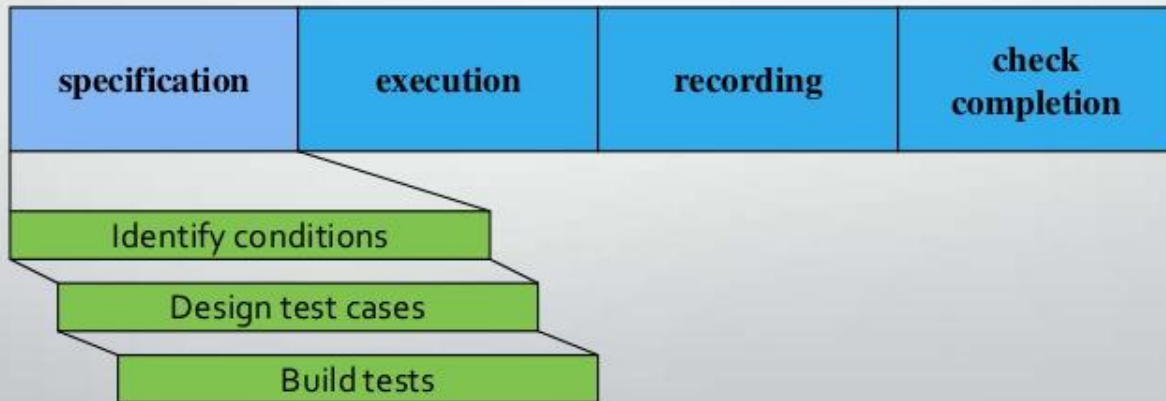


Test planning

- How the test strategy and project test plan apply to the software under test
- Document any exceptions to the test strategy
 - e.g. only one test case design technique needed for this functional area because it is less critical
- Other software needed for the tests and environment details
- Set test completion criteria

Test specification

Planning (detailed level)



Test specification

- Test specification can be broken down into three distinct tasks:
 1. **identify:** determine 'what' is to be tested (identify test conditions) and prioritise
 2. **design:** determine 'how' the 'what' is to be tested (i.e. design test cases)
 3. **build:** implement the tests (data, scripts, etc.)

A good test case

- Effective

Finds faults

- Exemplary

Represents others

- evolvable

Easy to maintain

- economic

Cheap to use

Test execution

Planning (detailed level)



Execution

- Execute prescribed test cases
 - most important ones first
 - would not execute all test cases if
 - testing only fault fixes
 - too many faults found by early test cases
 - time pressure
 - can be performed manually or automated

Test recording

Planning (detailed level)



Test recording 1

- The test record contains:
 - identities and versions (unambiguously) of
 - software under test
 - test specifications
- Follow the plan
 - mark off progress on test script
 - document actual outcomes from the test
 - capture any other ideas you have for new test cases
 - note that these records are used to establish that all test activities have been carried out as specified

Test recording 2

- Compare actual outcome with expected outcome. Log discrepancies accordingly:
 - software fault
 - test fault (e.g. expected results wrong)
 - environment or version fault
 - test run incorrectly
- Log coverage levels achieved (for measures specified as test completion criteria)
- After the fault has been fixed, repeat the required test activities (execute, design, plan)

Check test completion

Planning (detailed level)



Check test completion

- Test completion criteria were specified in the test plan
- If not met, need to repeat test activities, e.g. test specification to design more tests



Test completion criteria

- Completion or exit criteria apply to all levels of testing - to determine when to stop
 - coverage, using a measurement technique, e.g.
 - branch coverage for unit testing
 - user requirements
 - most frequently used transactions
 - faults found (e.g. versus expected)
 - cost or time

Why test?

- build confidence ✓
- prove that the software is correct ✗
- demonstrate conformance to requirements ✓
- find faults ✓
- reduce costs ✓
- show system meets user needs ✓
- assess the software quality ✓

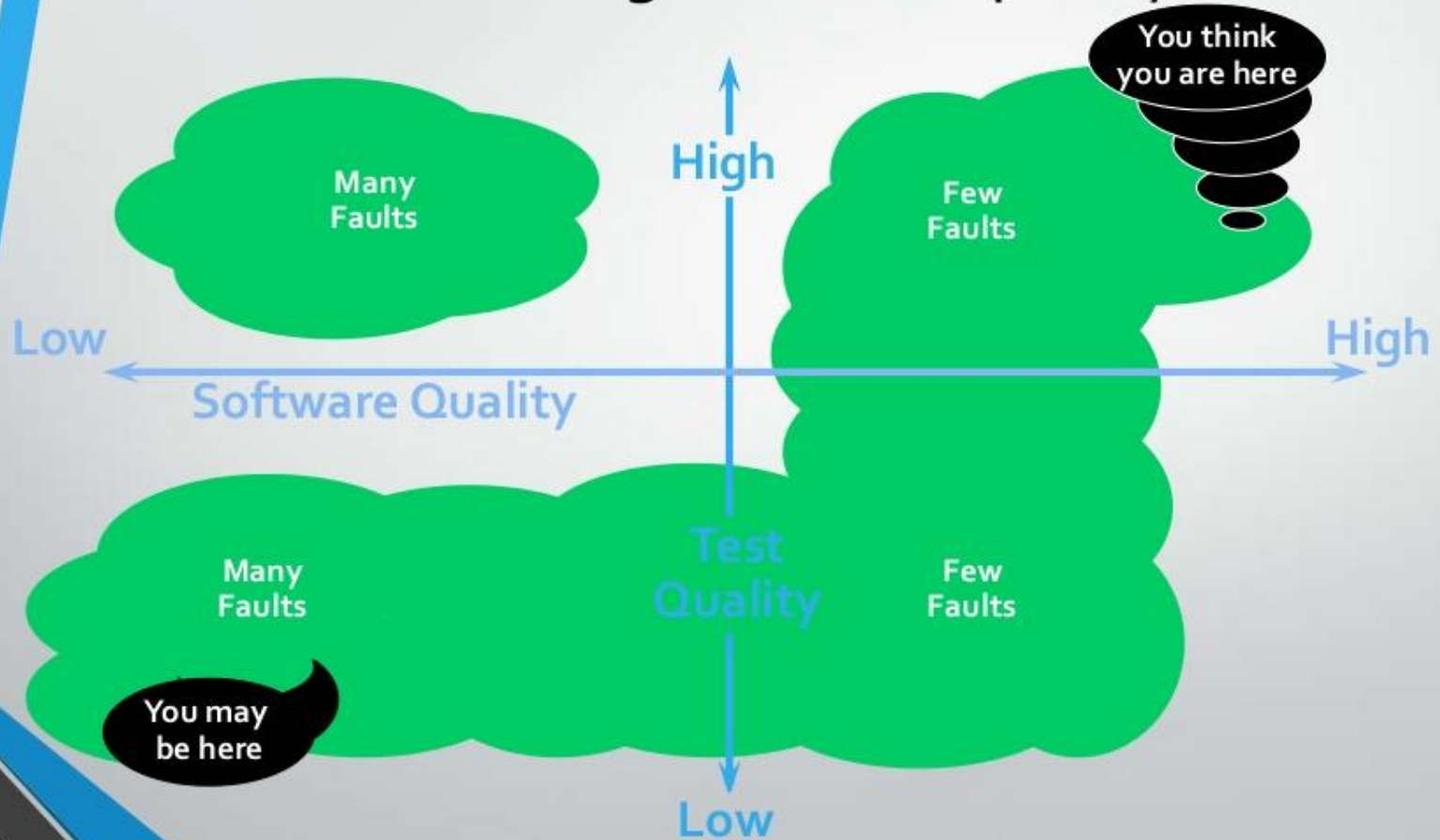
Confidence


Confidence



No faults found = confidence?

Assessing software quality





Testing throughout the life-cycle

What is software verification?

- Ensures product is designed to deliver all functionality to the customer
- Includes reviews and meetings, walkthroughs, inspection
- Answers the question: **Am I building the product right?**

What is software validation?

- Determines if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs
- Done at the end of the development process and takes place after verifications are completed
- Answers the question: **Am I building the right product?**


What is Capability Maturity Model?

- Bench-mark for measuring the maturity of an organization's software process
- Assesses an organization against a scale of five process maturity levels based on certain Key Process Areas (KPA)
- Describes the maturity of the company based upon the project the company is dealing with and the clients
- There are five maturity levels designated by the numbers 1 through 5 as shown below:
 1. Initial
 2. Managed
 3. Defined
 4. Quantitatively Managed
 5. Optimizing

CMM (Continued)

Maturity ladder





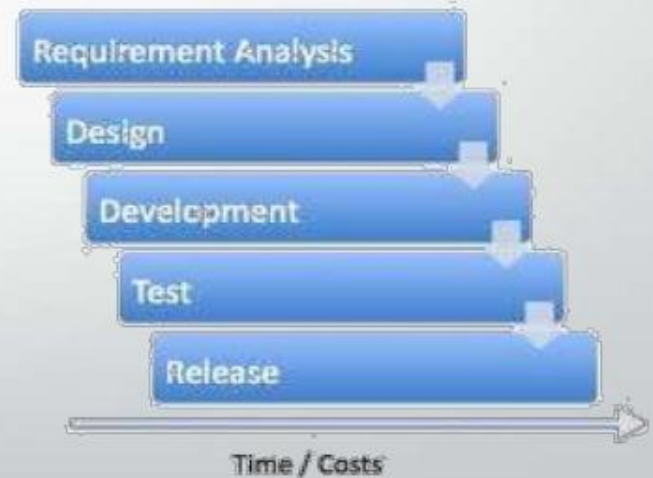
Software Development Lifecycle

Software Development Models

- Software life cycle models describe phases of the software cycle and the order in which those phases are executed
- Few Software development models are as follows:
 - Waterfall model
 - V model
 - Agile model

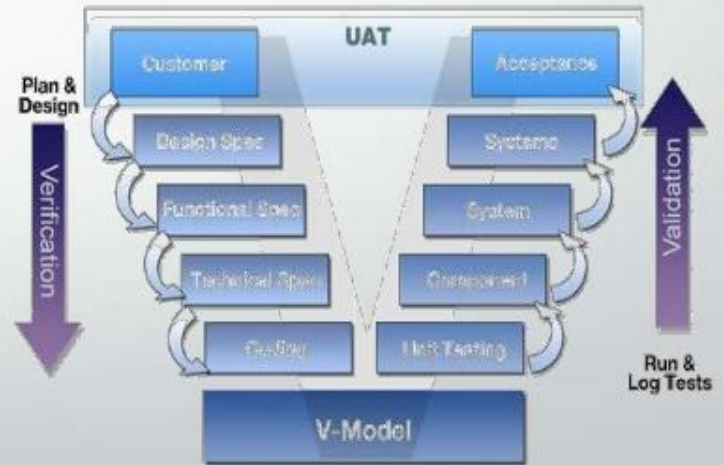
Waterfall Model

- Most common and classic of life cycle models, also referred to as a linear-sequential life cycle model
- Each phase must be completed in its entirety before the next phase can begin
- Review takes place at end of each phase to determine if the project is on the right path and whether to continue or discard the project



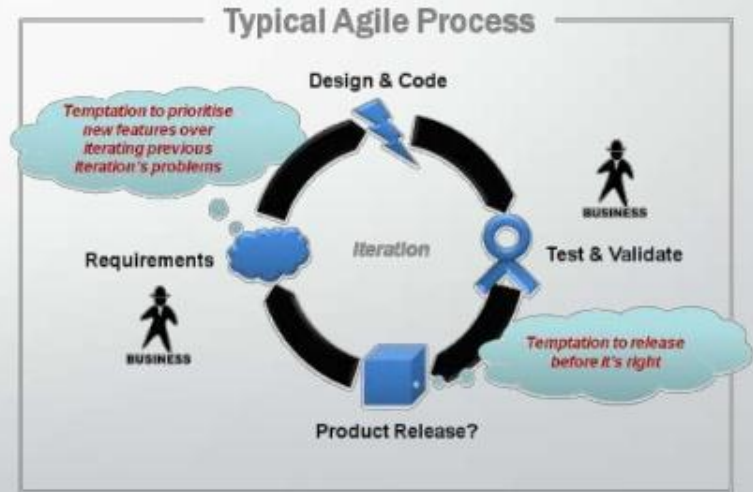
V- Model


- Means Verification and Validation model
- V-Shaped life cycle is a sequential path of execution of processes
- Testing of the product is planned in parallel with a corresponding phase of development



Agile Model

- Builds are provided in iterations
- Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing
- End of the iteration a working product is displayed to the customer and important stakeholders





Testing Levels

Testing Levels

- Identify missing areas and prevent overlap and repetition between the development life cycle phases
- Various levels of testing are:
 - Unit testing
 - Component testing
 - Integration testing
 - System testing
 - Acceptance testing
 - Alpha testing
 - Beta testing

Types of Testing

- **Blackbox Testing**

Testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

- **Whitebox Testing**

White box testing is a testing technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

Black box testing is often used for validation and white box testing is often used for verification.

Types of Testing

- **Unit Testing**

Testing of an individual unit or group of related units. It falls under the class of white box testing

- **Integration Testing**

Testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested. It may fall under both white box testing and black box testing

- **Functional Testing**

Testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing

- **System Testing**

Testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing

Types of Testing

- **Acceptance Testing**

Testing is often done by the customer to ensure that the delivered product meets the requirements and works as the customer expected. It falls under the class of black box testing.

- **Regression Testing**

Testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not imposing other modules to produce unexpected results. It falls under the class of black box testing.

- **Beta Testing**

Testing which is done by end users, a team outside development, or publicly releasing full pre-version of the product which is known as beta version. It falls under the class of black box testing

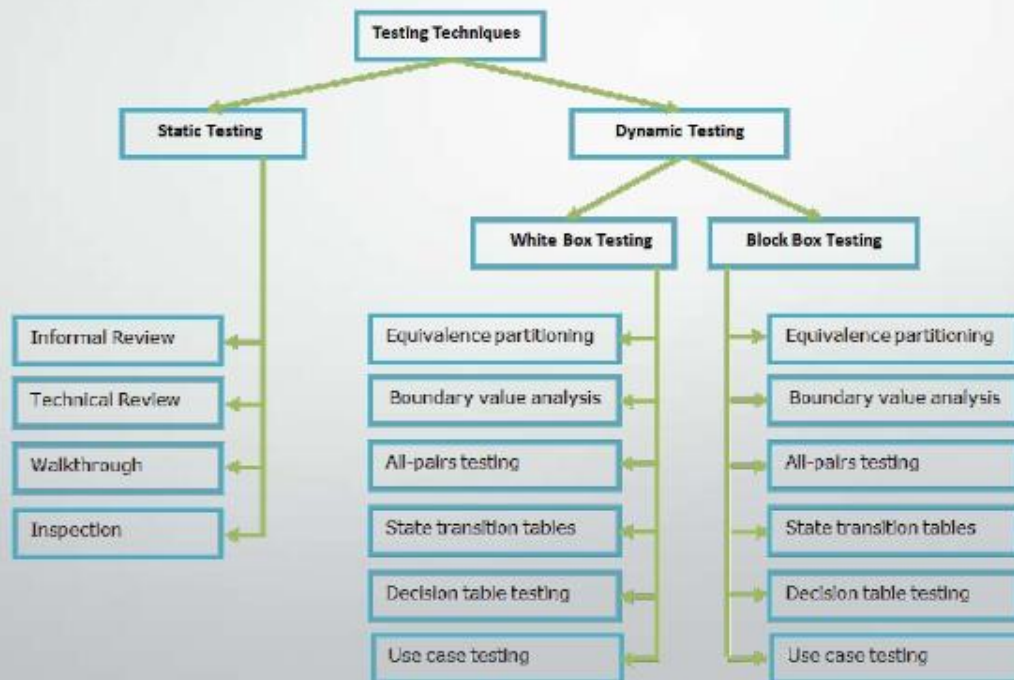


Static Techniques

Test Design Technique

- Test Design is creating a set of inputs for given software that will provide a set of expected outputs
- Ensures that the system is working good enough and it can be released with as few problems for the average user
- Two main categories of Test Design Techniques are:
 - Static Techniques
 - Dynamic Techniques

Testing Technique




Static Testing Techniques

- **Informal Review** is
 - applied many times during the early stages of the life cycle of the document
 - goal is to improve the quality of the document
 - informal reviews are **not documented**
- **Technical Review**
 - less formal review
 - performed as a peer review without management participation
 - Led by moderator
 - inform participants about the technical content of the document

Static Testing Techniques

- **Walkthrough** is
 - not a formal process/review
 - led by the author
 - aim is to gain feedback about the technical quality or content of the document; and/or to familiarize the audience with the content
- **Inspection**
 - most formal review
 - trained individuals look for defects using a well defined process
 - led by moderator
 - defects found are documented in a logging list or issue log
 - formal follow-up is carried out by the moderator applying exit criteria



Tool Support for Testing

Types of software testing tools

- Tools are grouped by the testing activities or areas that are supported by a set of tools
- Example a 'test management' tool may provide support for managing testing (progress monitoring), configuration management of testware, incident management, and requirements management and traceability



Questions??