Welcome To
Software Testing Fundamentals Training

What is Testing?

Testing is the practice/process of making objective judgments regarding the extent to which the system/device meets, exceeds or fails to meet stated objectives.

Electronic testing

The act of applying a voltage or current to a circuit and comparing the measured value to an expected result.

Blood Tests

To measure the amount of certain substances in the blood or to count different types of blood cells.

Automotive testing

Testing full vehicles, components and systems through a series of laboratory, virtual and 'real world' assessments to ensure it is safe, reliable and compliant with safety regulations.

SUBSCRIBE

What is Software Testing?

- Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do.
- Some benefits of software testing are:
  - Preventing bugs
  - Reducing Software Development costs
  - Improving performance

Software Testing is a Process and has Objectives

Software Testing as a Process

- Software Testing is a process
- Software Testing is both Static Testing & Dynamic Testing
- Software Testing Requires Planning, Preparation, Evaluation, Reporting etc.
- You need to test Software products & related work products

Objectives of Software Testing

- Prevent defects
- Determine that Software meets product requirements
- Determine that software is fit for use
- Find failures/defects/bugs in software
- Reduce risk
- Build confidence

Testing as a process

Testing is not a standalone activity, it is a series of activities.

1. Software Testing is done in all phases of SDLC.
2. It starts with Static Testing which includes reviewing the documents like Software Requirement Documents (SRS, User Stories), Design Documents etc.
3. When working build is available then Dynamic Testing is performed to verify that the software is fit for use, meets customer needs and is reasonably defect free.

Software Testing involves both Static & Dynamic Testing

1. In initial phases of SDLC Static Testing is done.
   - For example - Technical Reviews, Walkthrough, Inspection, Static Code Analysis
2. Once the Software Build is available to test, then Dynamic Testing starts to validate that the software meets customer requirements and is fit for use.
   - Test Levels during Dynamic testing are:
     - *Unit Testing (Mostly done by developers during code development)*
     - *Integration Testing*
     - *System Testing*
     - *Acceptance Testing*

Software Testing Requires Planning

Test planning is very important part of Software Testing.

Before you can actually test software, you need to plan for:

- What you want to achieve?
- Who will do what?
- Time frame of testing
- Control the test progress
- Prepare test summary reports

Testing Requires Preparation

Once test plan has been finalized you need to do test preparation.

- Prepare Test Scenarios
- Prepare Test Cases
- Prepare Test Environment
- Prepare Test Data
- Any other required pre-requisites

Testing Requires Evaluation

During test execution you evaluate the software to make sure that it:

- Meets end user requirements
- It meets the exit criteria
- Evaluate Functional and Non-Functional aspects of software

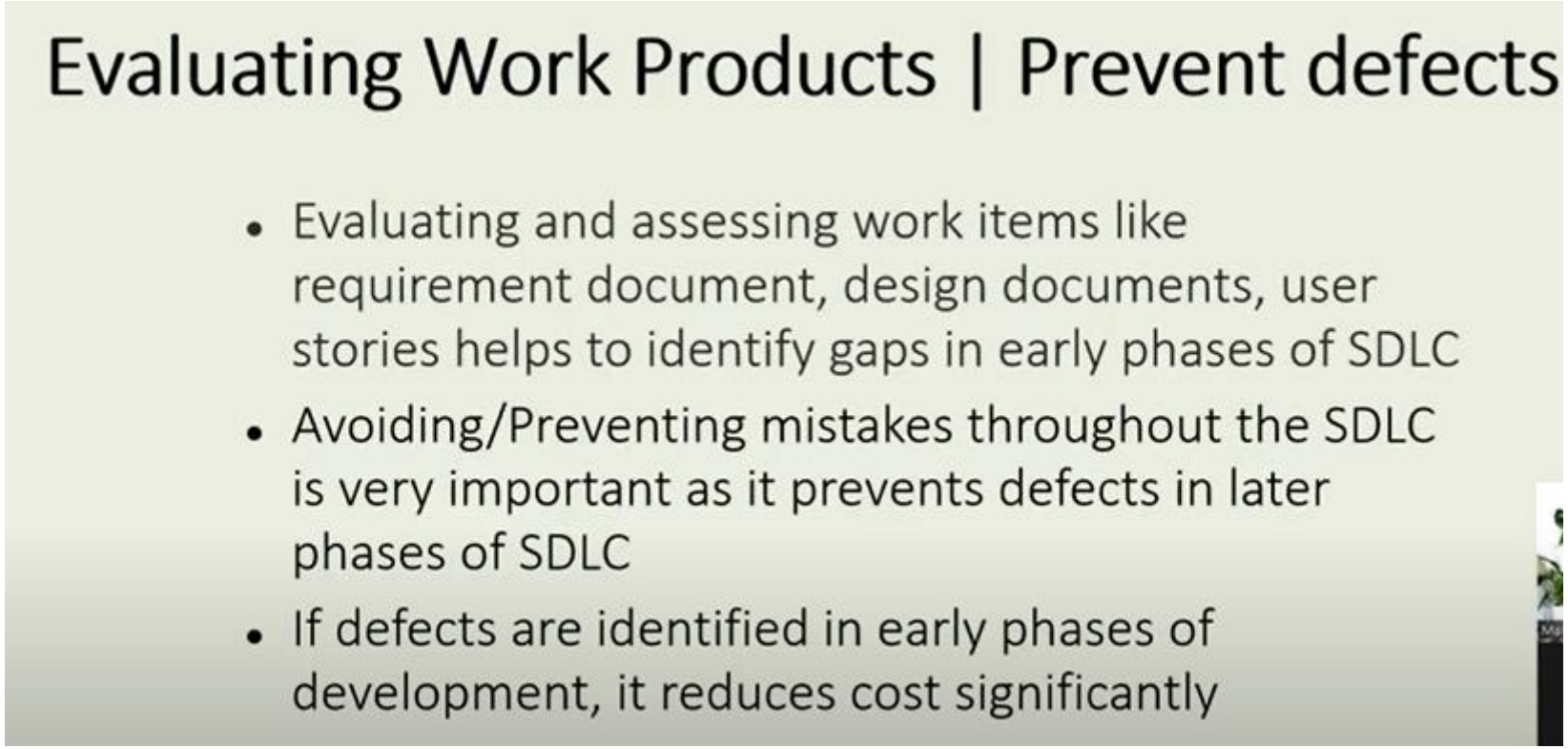
Software Products and related Work Products

Software testing is not just about testing the software code. It requires testing all the related documents like:

- Software Requirements Specification Documents (SRS), User Stories
- Design documents (HLD/LLD)
- Quick reference guide
- Training materials
- User guides
- Installation guide

and any other related work products.

Objectives for testing.



Determine that Software meets product requirements and customer needs

- Software Testing checks product against end user need and requirements of customer
- Testing ensures that all standards are met which helps stakeholders to make release decisions

Determine that software is fit for use

- Testing is done to ensure that software is fit for use for the end users who will be using the software.
- It ensures that the software does what's users expect it to do
- Meets defined standards
- Is safe

Find failures/defects/bugs in software

- Another objective of Software testing is to find the bugs/defects in software
- The focus should be to find the maximum defects during validation phase. Defects can be found during the whole SDLC process.
- The defects should be identified as early as possible in SDLC
- Fixing those defects improves the Software Quality

Reduce risk

- Software testing ensures that high risk areas of software are verified and validated properly which reduces the risk.
- Software testing ensures that the product complies with regulatory/contractual requirements and reduces risks related to contractual/regulatory obligations.

Build confidence

- Testing builds confidence in Software by ensuring that the software is of good quality and meets quality standards.
- Testing focus on defect prevention process helps in fewer errors and failures in end product.
- Testing ensures excellent product quality which increases customer satisfaction and lowers maintenance cost.

Types of Software

System Software

Windows, Linux, MAC OS, iOS, Android etc.

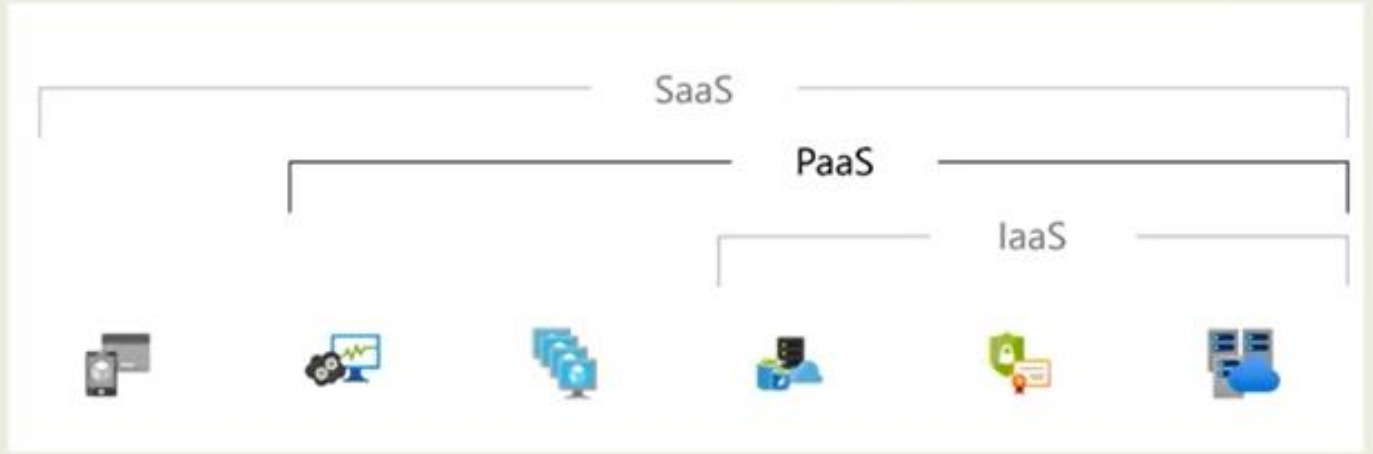
Application Software

MS Office, Google Sheets, VLC player, CRMs and BPM (Salesforce, ZOHO, Pega etc.), DB2, MS SQL, Google Meets, eCommerce websites (Amazon, eBay, Flipkart etc.), Gmail etc.

Programming software

Eclipse, Visual Studio Code, PyCharm etc.

SaaS vs PaaS vs IaaS



Software as a Service (SaaS) - Examples: JIRA, Confluence, Trello, Salesforce, Google Sheets, WebEx etc.

Platform as a Service (PaaS) – Examples: OpenShift, Google App Engine, force.com, AWS Elastic Beanstalk etc.

Infrastructure as a Service (IaaS) - Examples: Amazon Web Service (AWS), Microsoft Azure, Google Compute Eng

https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose

https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/types-of-cloud-computing

Product Based vs Project Based

Product Based Companies - Product based companies develop products for specific purpose and market it.

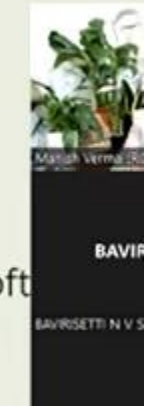
Examples – IBM Software Labs, Oracle, Atlassian, Salesforce, SAP etc.

Project Based Companies - Project based companies provide consulting and software services to other business like Banks, Utilities, Education, Governments etc.

Examples – IBM Global Services, Cognizant, Capgemini, Accenture etc.

Why is testing necessary?

- Nissan having to recall over 1 million cars due to a software defect in the airbag sensor detectors.
- Software bug caused the failure of a USD 1.2 billion military satellite launch.
- Software failures in the US cost the economy USD 1.1 trillion in assets in 2016.
  - Defects can lead to serious damages
  - A defect can cause loss of money, time, business and brand reputation
  - Testing improves software quality and gain customer confidence
  - Reduces risk of software failures in operational environment
  - Testing accelerates Software Development
  - Testing is also required as part of contractual agreement or legal requirements (For soft which have high risk associated)

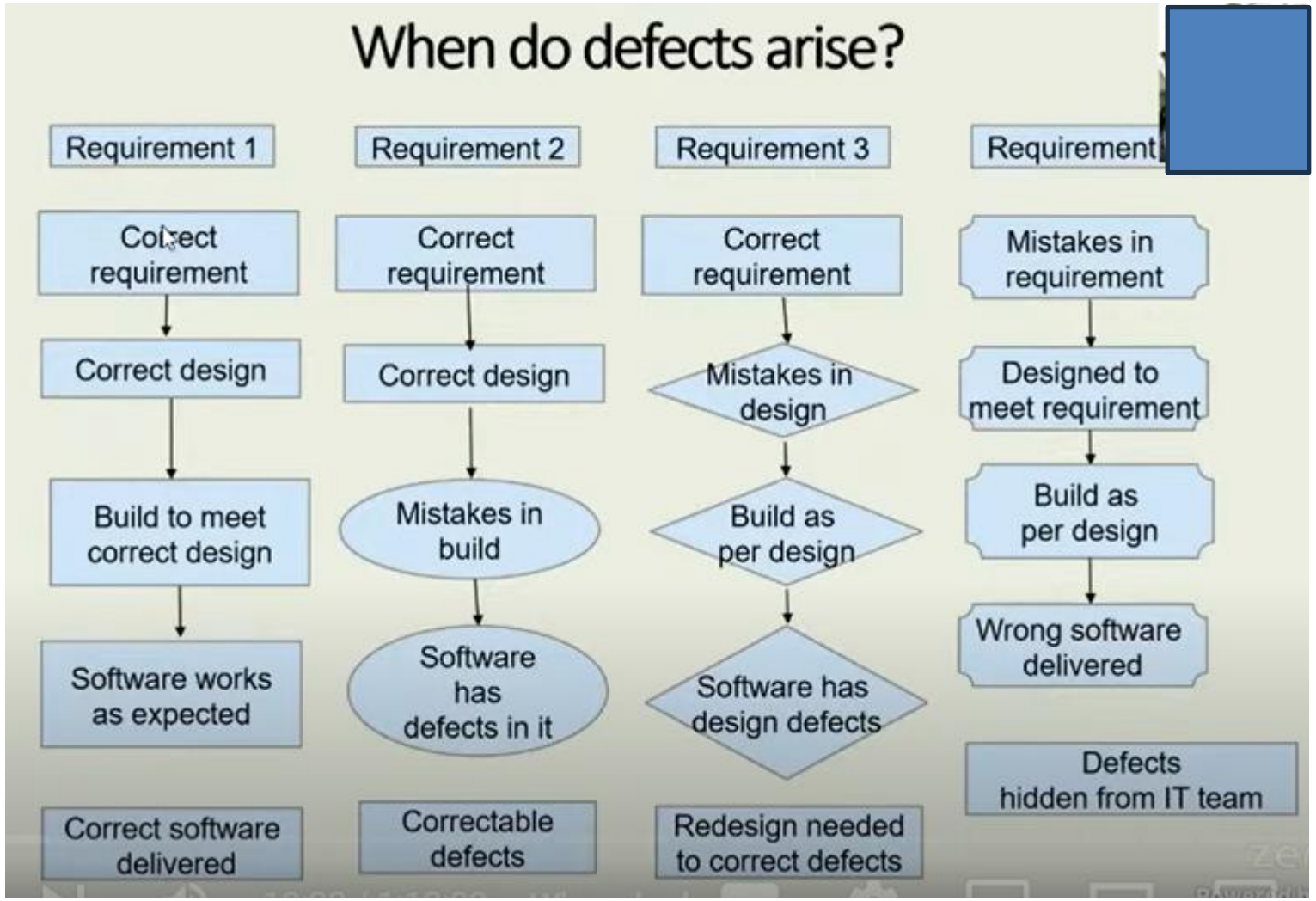


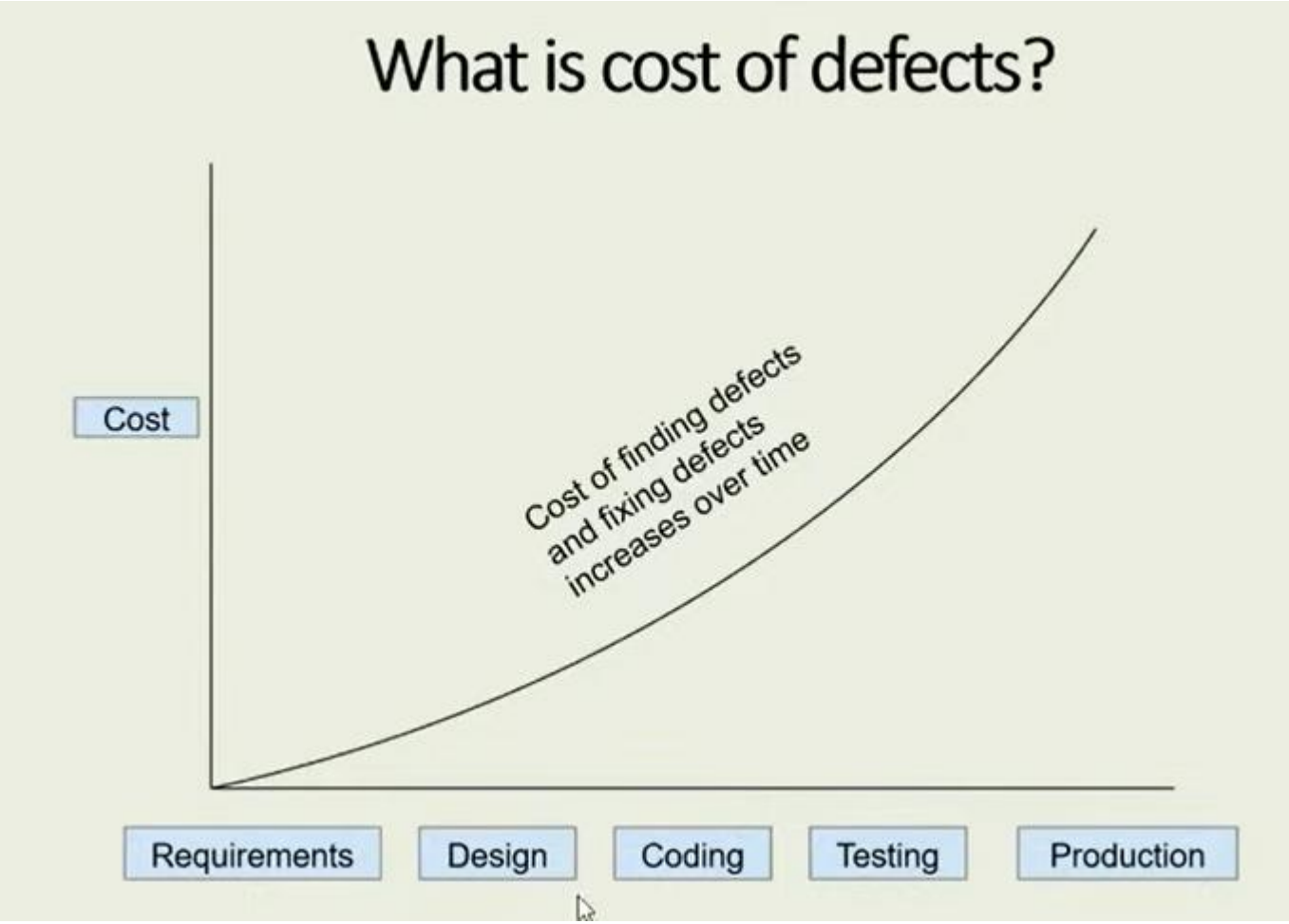
Testing is Context Dependent

- An error in your personal blog does not impact any one else so testing it is different from critical software's
- An error in a business website may put off the company as it looks unprofessional
- Net banking websites or ATMs should be thoroughly tested to maintain bank credibility
- Air traffic control system is very critical system which needs to be thoroughly tested before live deployment as human lives are dependent on it

What Causes of Software Defects?

- Unclear understanding about product/software purpose results in unclear requirements
- Miscommunication withing team during development lifecycle
- People may make mistake during requirements gathering, design, coding, testing
- Less skilled team
- External factors, like issues with third party systems


When do defects arise?
Requirement 1
Requirement 2
Requirement 3
Requirement
Correct requirement
Correct design
Build to meet correct design
Software works as expected
Correct software delivered
Correct requirement
Correct design
Mistakes in build
Software has defects in it
Correctable defects
Correct requirement
Mistakes in design
Build as per design
Software has design defects
Redesign needed to correct defects
Mistakes in requirement
Designed to meet requirement
Build as per design
Wrong software delivered
Defects hidden from IT team


What is cost of defects?
Cost
Cost of finding defects and fixing defects increases over time
Requirements
Design
Coding
Testing
Production

What is software quality?

Quality:- The degree to which a component, system or process meets specified requirements and/or user/customers needs and expectations

1. Software quality for developers and testers – It meets specifications, is technically good and has few defects.
2. Software quality for other stakeholders may be more than what is for developers – They also need value for money

Different viewpoints for software quality can be:

- Attributes of products
- Fit for use
- Good development processes
- Value for money

Testing and Quality

- Testing helps to measure software quality
- Testing provides confidence in software based on number of defects found
- Well designed tests uncover most of the defects in software and if the tests pass, it gives more confidence in software quality
- Testing helps to find defects and software quality improves when those defects are fixed

QA vs QC vs QE

QA	QC	QE
• QA is process-oriented • Focuses on preventing quality issues • Verification • Proactive strategy	• QC is product-oriented • Focuses on identifying quality issues • Validation • Reactive strategy	• Quality engineering is a process that applies rigorous quality checks to each stage of product development • QE is closely integrated with agile and DevOps processes

Seven Testing Principle

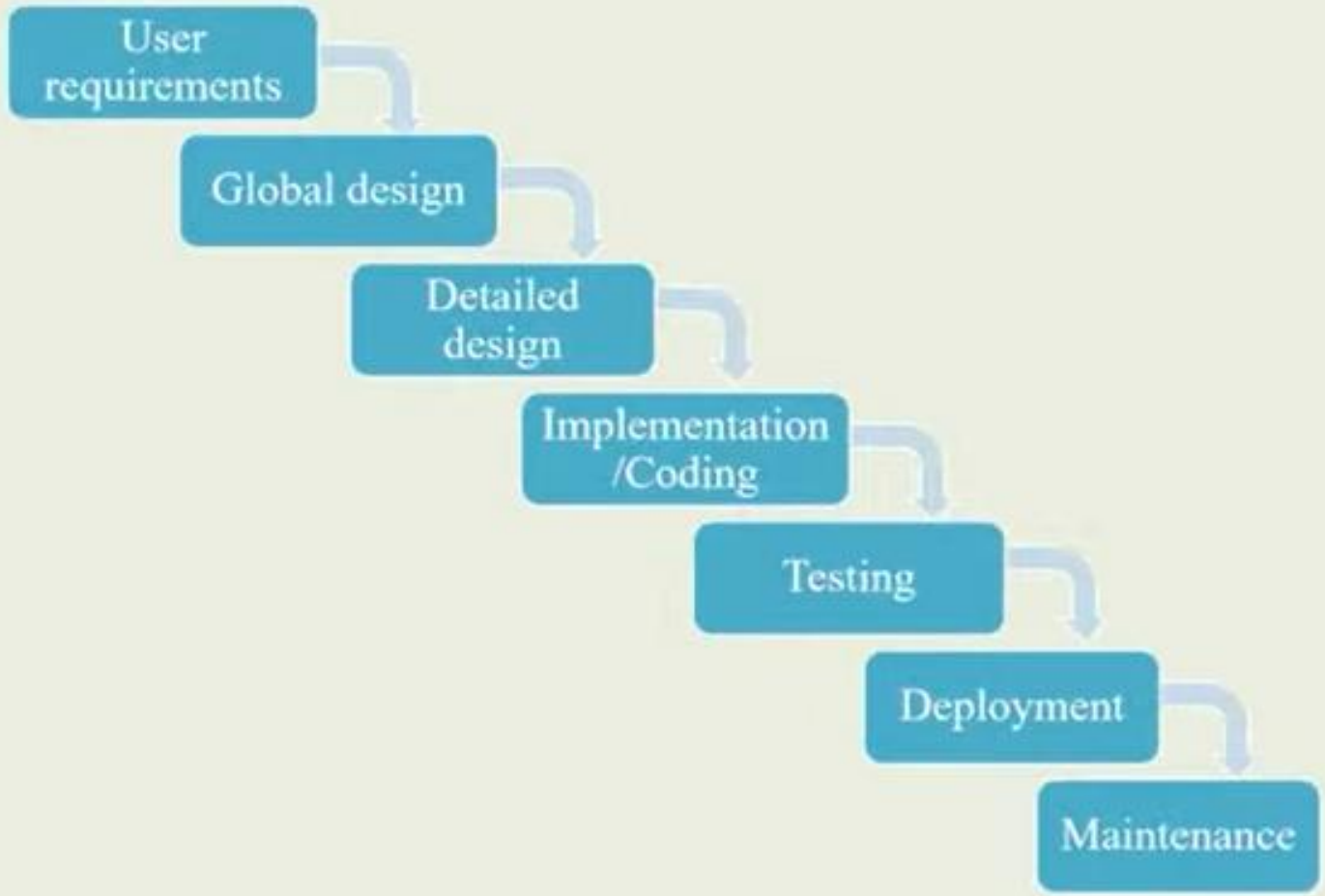
There are seven testing principles which offer general guidelines common for all testing

- Testing shows the presence of defects
- Exhaustive testing is impossible
- Early testing
- Defect clustering
- Pesticide paradox
- Testing is context dependent
- Absence-of-error fallacy

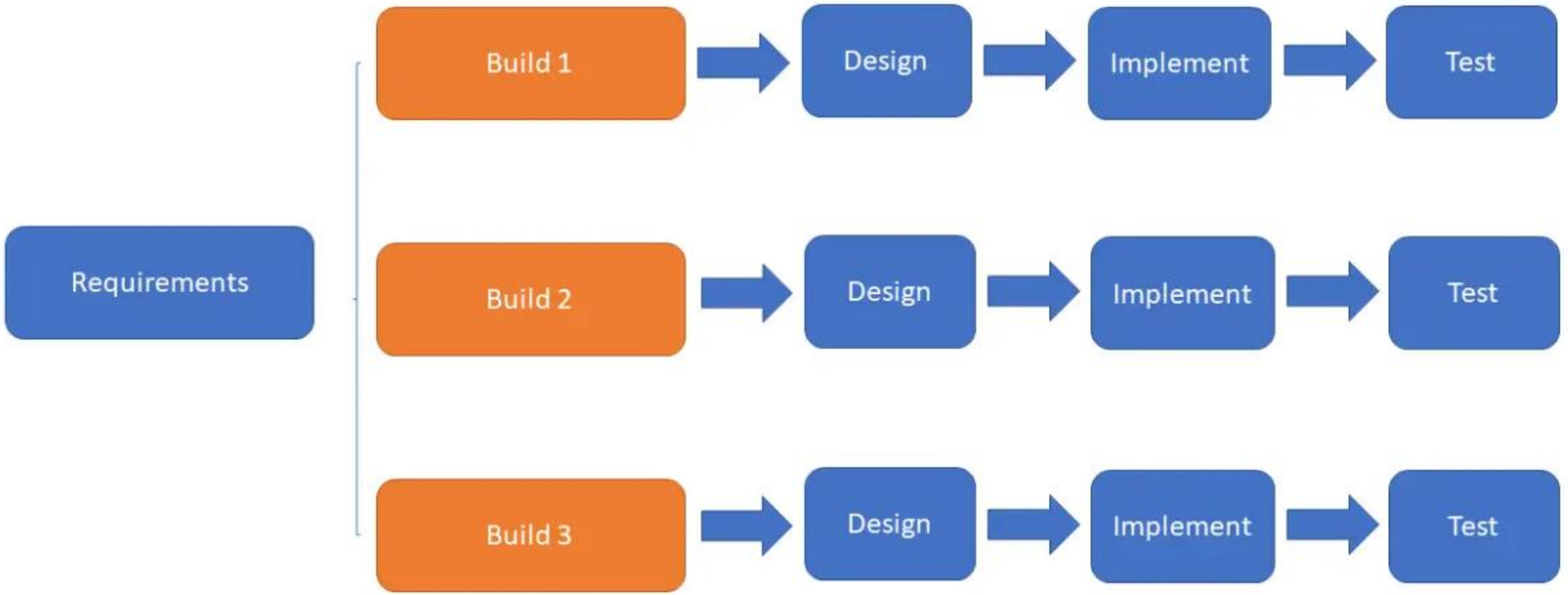
Testing throughout the software life cycle

- Testing is not a standalone activity, it does not exist in isolation
- Testing has its place in SDLC
- The selection of software development model decides how testing will be organized
- There are different test levels in SDLC

Waterfall Development Model

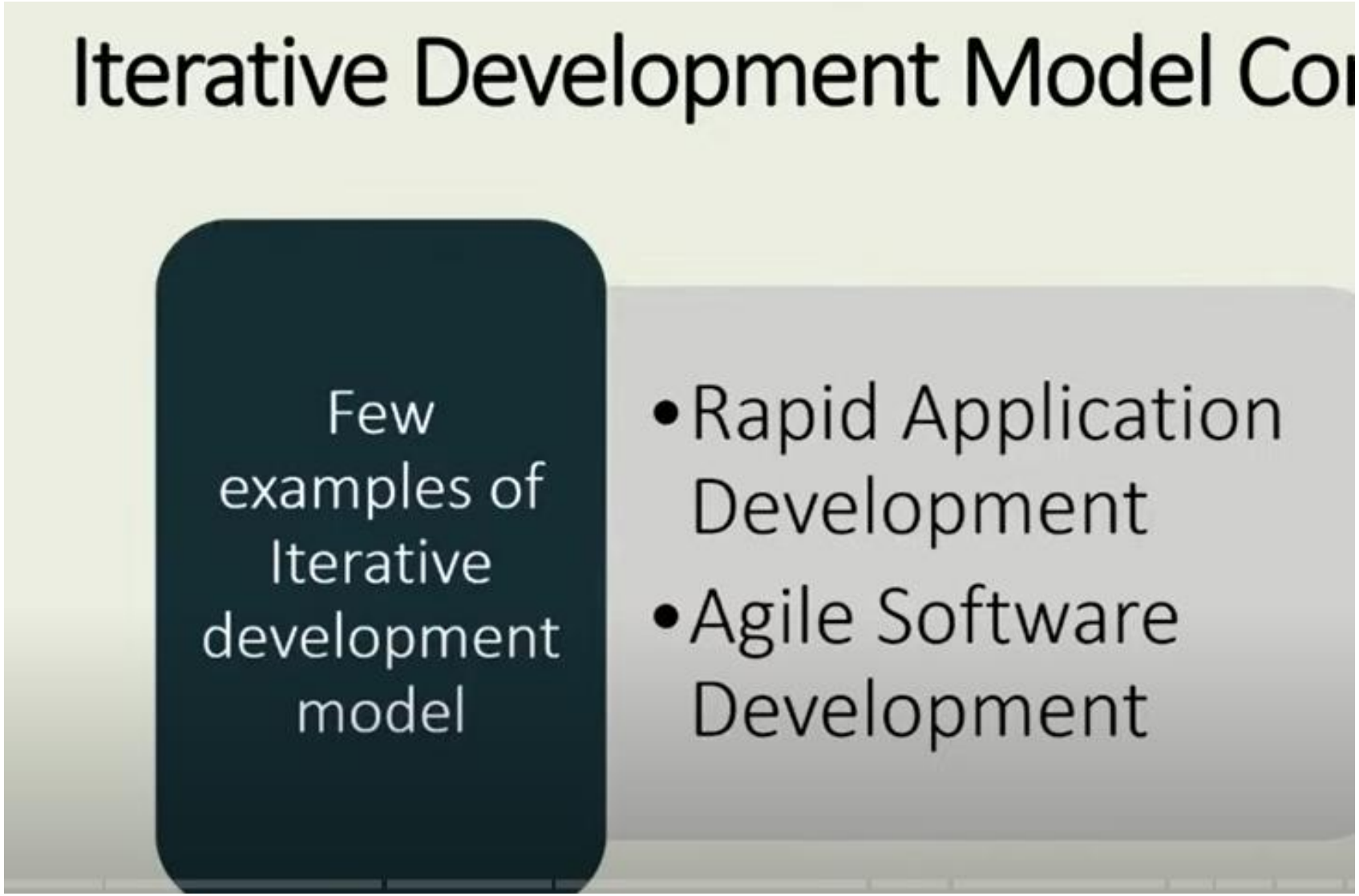


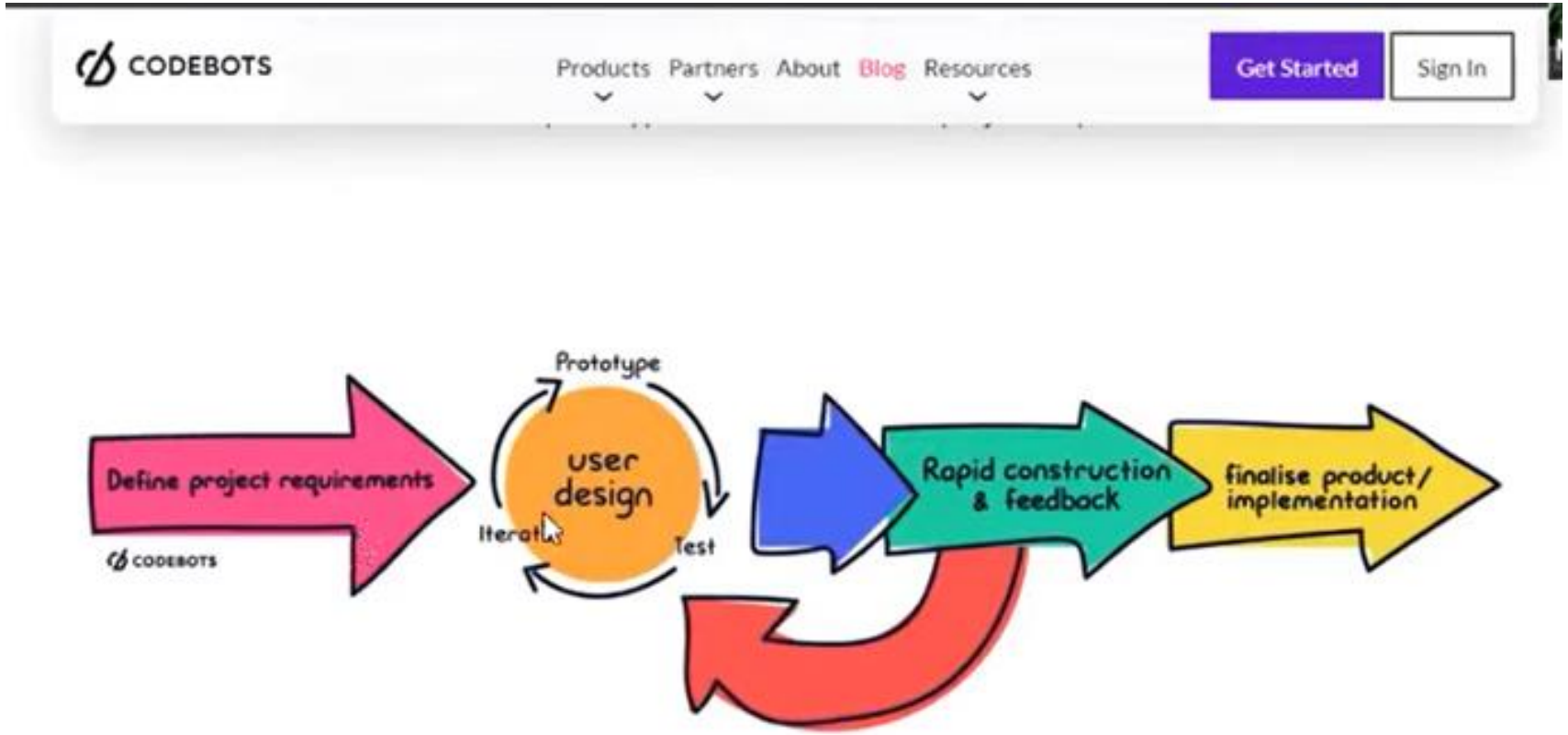
Iterative Development Model



Iterative Development Model

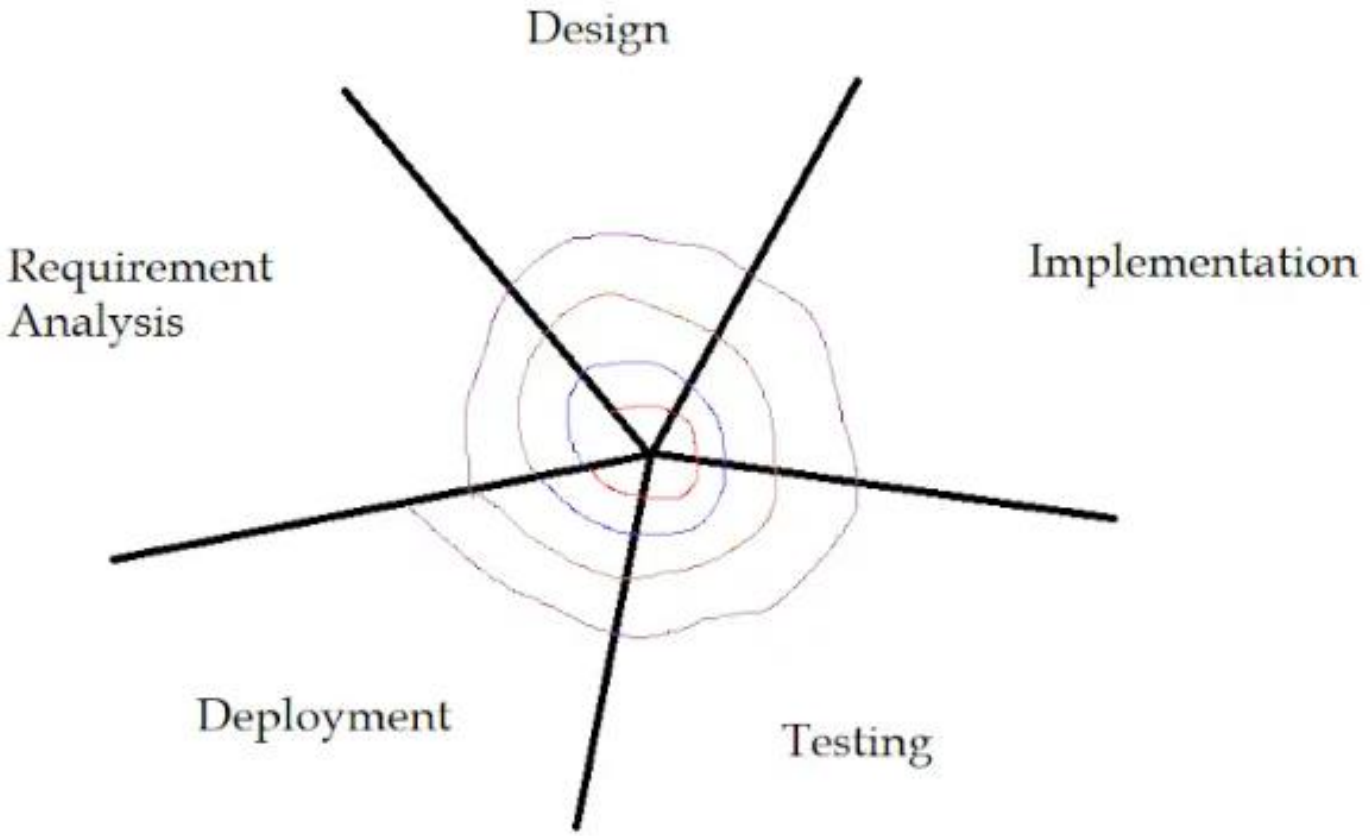
- Its a Software Development Model in which the product is designed, implemented and tested incrementally
- In each iteration, new requirements are added until the product is completely developed
- Requirements are not freeze and new requirements can be added at any point of time
- Again Testing team is not involved in the beginning phases
- Unlike Waterfall, we can see some working product in less time


Iterative Development Model Co
Few examples of Iterative development model
•Rapid Application Development
•Agile Software Development



1: Define project requirements.

Spiral Model



Spiral Model

- Combination of a waterfall model and iterative model
- Instead of taking all requirements at one, the one iteration will be conducted with basic requirements and its starts with concept creation
- Then next version of product can be achieved in next spiral and so on
- Example: MS Office 95, 97, 2000, XP, 2003, 2007, 2010, 2013 and 2016
- Spiral model is generally followed by Product Based Companies for delivering their products

V- Model

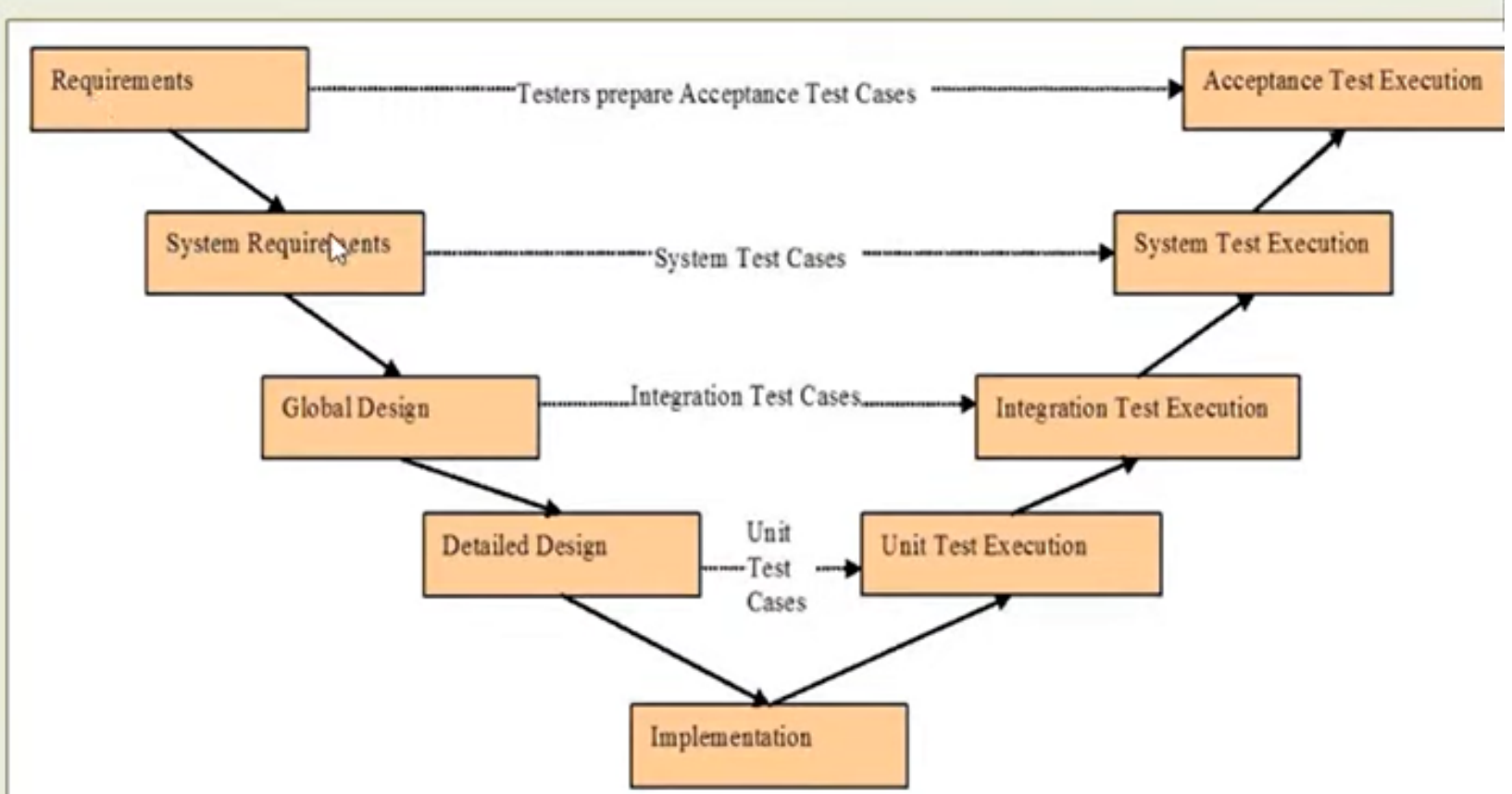
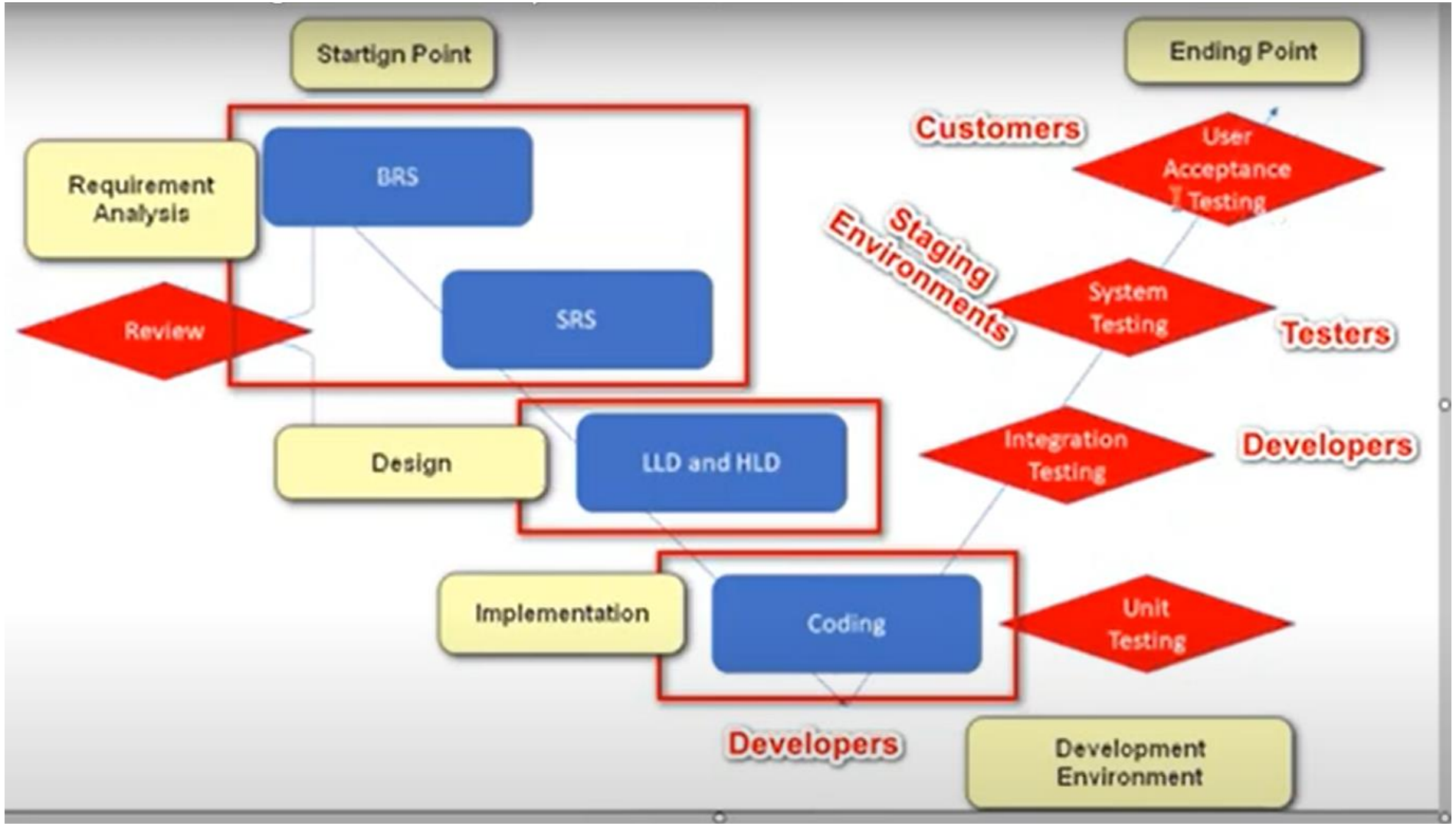


Figure 2: V-Model

V-Model


Startign Point
Ending Point
Customers
User Acceptance Testing
Requirement Analysis
BRS
Staging Environments
System Testing
Testers
Review
SRS
Design
LLD and HLD
Integration Testing
Developers
Implementation
Coding
Unit Testing
Developers
Development Environment

V-Model

- In the Phase by Phase models, we have to recheck the previous phases if we find any defects in testing which is lately introduced in the SDLC
- V Models solves this problem as Testing is introduced from the beginning itself
- Frequently changed requirements are not addressed in this Model

Verification vs. Validation

Verification	Validation
• Checks "Are we building the product right"? • Verification is the process for determining whether or not a product fulfills the requirements or specifications • Done without executing the software (All static testing techniques used) • Confirms to requirements (Producer view of quality)	• Checks "Are we building the right product"? • Validation is concerned with evaluating a product to determine if it meets the end user needs • Done by executing the software (Includes all Dynamic Testing techniques) • Fit for use (consumers view of quality)

Agile Software Development

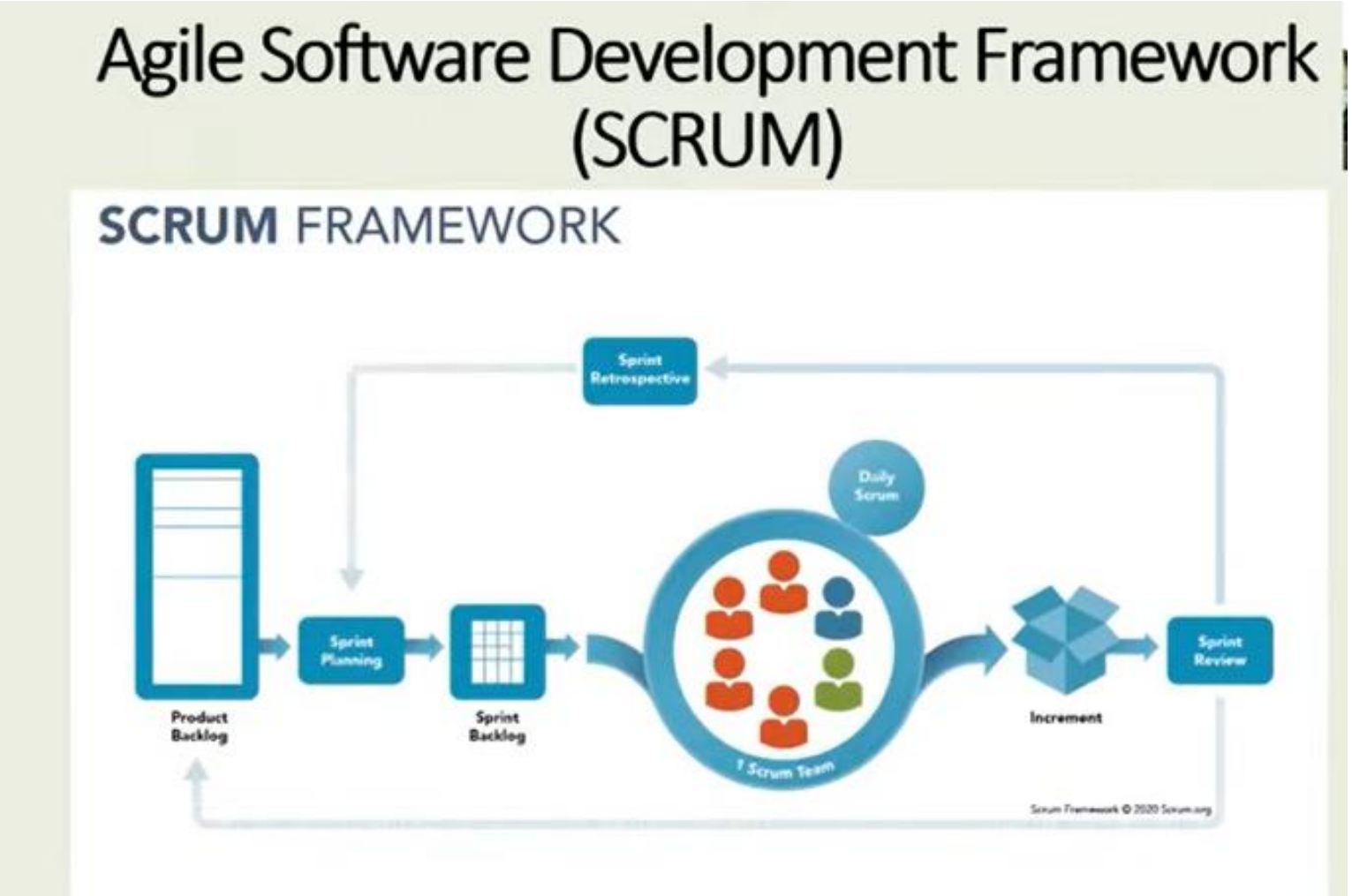
The capability of rapidly and efficiently adapting to changes is known as agility.

Agile software development is based on iterative and incremental development.

Agile methodology uses continuous stakeholder feedback to produce high quality consumable code through use cases and a series of short time-boxed iterations.

https://tech.gsa.gov/guides/user_story_example/

Agile Software Development Framework (SCRUM)



- Agile means **Quick**
- In these days, everything changing very dynamically in the market
- To accommodate such dynamic changes in the requirements, Agile Model came into the picture
- No importance to Documentation (Very less documentation is required)
- More importance to communication (So more meetings will be here)
- Requirements will be broken into small sized stories and added to the Backlog
- Prioritize the stories, estimated and added to one iteration of duration 2 weeks to 4 weeks
- In this iteration, all the Product Owners, BA, Developers, Testers and others will communicate and complete their tasks on a story by story basis in a quicker way

Fundamental Test Process

Steps in Fundamental Test Process

Fundamental test process consists of 5 steps

- Planning and Control (Analyze Requirements and Create test plan)
- Analysis and Design (WHAT and HOW to test, Come up with test scenarios, identify test data required, test env setup, create RTM)
- Implementation and Execution (Create detailed test cases and test execution)
- Evaluating exit criteria and reporting
- Test Closure Activities

Test Planning and Control

During Test planning

- Understand the goals and objectives of the customers, stakeholders, and the project
- Understand the risks which testing is intended to address.
- Set goals and objectives for testing based on goals and objectives of customers, stakeholders and project

Major tasks of Test Planning

Identify the objectives of testing based on the scope and risks of project

- Decide which components, systems or other products are in the test scope
- Decide the business, product, project and technical risks which need to be addressed
- Decide the objective of testing
  - to uncover defects,
  - to verify that the software meets requirements
  - to demonstrate if software is fit for use

Major tasks of Test Planning cont.

Determine the test approach

- How testing will be carried out?
- What test techniques will be used?
- What needs to be tested and what extent of test coverage required?
- Who is involved and when?
- Decide test deliverable's to be produced (Test cases, test data)

Implement the test policy and/or test strategy

- If the organization test policy and strategy exists then during planning ensure that testing adheres to those policy/ strategy

Major tasks of Test Planning cont.

Determine the required test resources	• Define the required resources for testing like testers, hardware and software etc.
Scheduling test analysis and design tasks, test implementation, execution and evaluation	• You need to prepare the schedule for all the tasks so that tracking can be done and progress is captured
Determining the exit criteria	• Criteria set to find out when to finish testing. The tasks that must be completed for the test level before we can exit the test phase

Test Control

After test planning we need to measure and control the progress

Test Control is the ongoing activity of comparing actual progress against the plan

Test control reports the status of test progress including any deviations from the actual plan

Test control monitors the testing throughout the project

The major tasks of test control

Measure and analyze results of reviews and testing	• Track test pass/fail percentage • Track tests remaining
Monitor and document test progress, coverage and exit criteria	• Track how many tests executed • What is the testing outcome(Number of tests passed/failed) • Risk assessment of test outcome

Test Analysis and Design

During Test Analysis and Design we build test designs and test procedures (Scripts) The major tasks of test analysis and design are:

Reviewing the test basis

- Review software requirements specification (SRS) document, design document
- Start designing black box tests using test basis
- This identifies gaps and ambiguities in specifications and prevents defects

Identifying the test conditions

- Based on the analysis of test items and specifications prepare the test conditions

Test Analysis and Design cont.

Designing the tests

- Apply test design techniques to design your tests

Evaluate testability of the requirements and system

- Make sure that all the requirements are testable

Design the test environment

- Hardware and software required to test
- Any supporting tools like test management tools etc.

Test Implementation and Execution

Take test conditions and make them into test cases

Build test environment where test execution needs to be done

Test Implementation and Execution co

Test implementation & execution have the following major tasks

Test Implementation

- Develop test cases and prioritise them
- Apply test design techniques to develop test cases
- Prioritize test cases based on the risk assessment
- Create test suites (Logical collection of tests)
- Prepare test environment

Test execution

- Execute test cases
- Record the test execution outcome with details like environment, software version
- Compare actual with expected result (Report incident – Incident is then analysed and bug is logged if its an actual defect)
- Perform re-test after defect is fixed to ensure that defect is corrected after fix

Evaluating Exit Criteria and Reporting

Exit Criteria evaluation is an activity where test execution is assessed against the defined objectives.

This should be done for each test level.

Exit criteria is set based on risk assessment for each level and exit criteria evaluation ensures that we have done enough testing to exit testing or test level

Major tasks in Evaluating Exit Criteria or Definition of Done (In Agile Dev)

Check test logs against the exit criteria specified in test plan	• Check the test execution percentage • Check the defect raised/fixed/outstanding
Assess if more tests are needed or if the exit criteria specified should be changed	• Based on exit criteria make assessment if more tests are required to fulfill exit criteria • Still some defects pending to be fixed • Project risks increased and so need to change exit criteria by consulting stakeholders
Writing a test summary report for stakeholders	• Preparing the test summary report and distributing with all stakeholders • Helps stakeholders make the release decisions about software

Test Closure Activities

Collect data from completed test activities to consolidate experience, major tasks in test closure activities are:

Check which planned deliverables have been delivered	• Test strategy/plans, test cases etc. • All incident reports have been resolved(Fixed/deferred)
Finalise and archive testware for later use	• Test cases/scripts • Test environment • Any other test infrastructure

Test Closure Activities cont.

Handover testware to the maintenance organisation

- After software release maintenance phase will start
- Maintenance organization can be a different organization other than one who developed software
- They will need the testware for maintenance changes or any bugs fixes in production environment

Evaluate the testing and analyze lessons learned for future releases and projects

- Helps to improve the whole SDLC and test process
- Improve test design and execution methodologies for reducing invalid defects

Test Levels

There are 4 test levels in Software Testing

- Component Testing or Unit Testing
- Integration Testing
- System Testing
- Acceptance testing

Component Testing/Unit Testing

Unit is the smallest testable part of the software system.

Unit testing is done to verify that the lowest independent entities in any software are working fine.

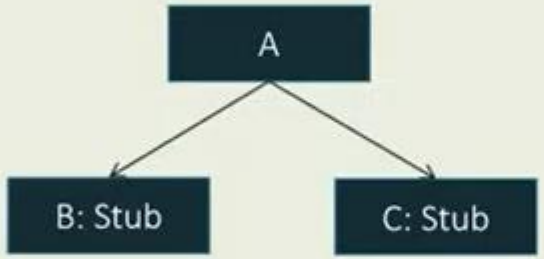
The smallest testable part is isolated from the remainder code and tested to determine whether it works correctly

STUBS and DRIVERS are used to replace the missing components in unit testing

STUBS and DRIVERS

A STUB is called from the software component

Example:



Suppose module A calls functions from Module B and C which are not ready.

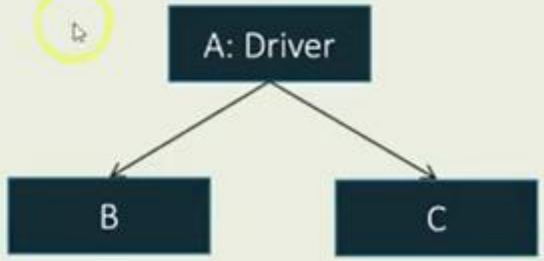
Developer will write a dummy module which simulates B and C and returns values to module A.

This dummy module code is known as stub.

STUBS and DRIVERS Cont.

A DRIVER calls a component to be tested

Example



Suppose you have modules B and C ready but module A which calls module B and C is not ready

Developer writes a dummy piece of code for module A which takes care of the call for module A

This dummy piece of code is known as driver.

Component Testing/Unit Testing

Component testing may include testing of functional & non-functional characteristics like:

- Resource behavior (e.g. memory leaks)
- Robustness testing (e.g. invalid inputs or stressful environmental conditions)
- Structural testing (e.g. branch coverage)

Component Testing/Unit Testing

- Component testing is typically done by developers
- Needs access to the code & done with the support of the development environment
- Defects are typically fixed as soon as they are found
- No formal defect logging process is followed.

Integration Testing

Integration testing tests interfaces between components

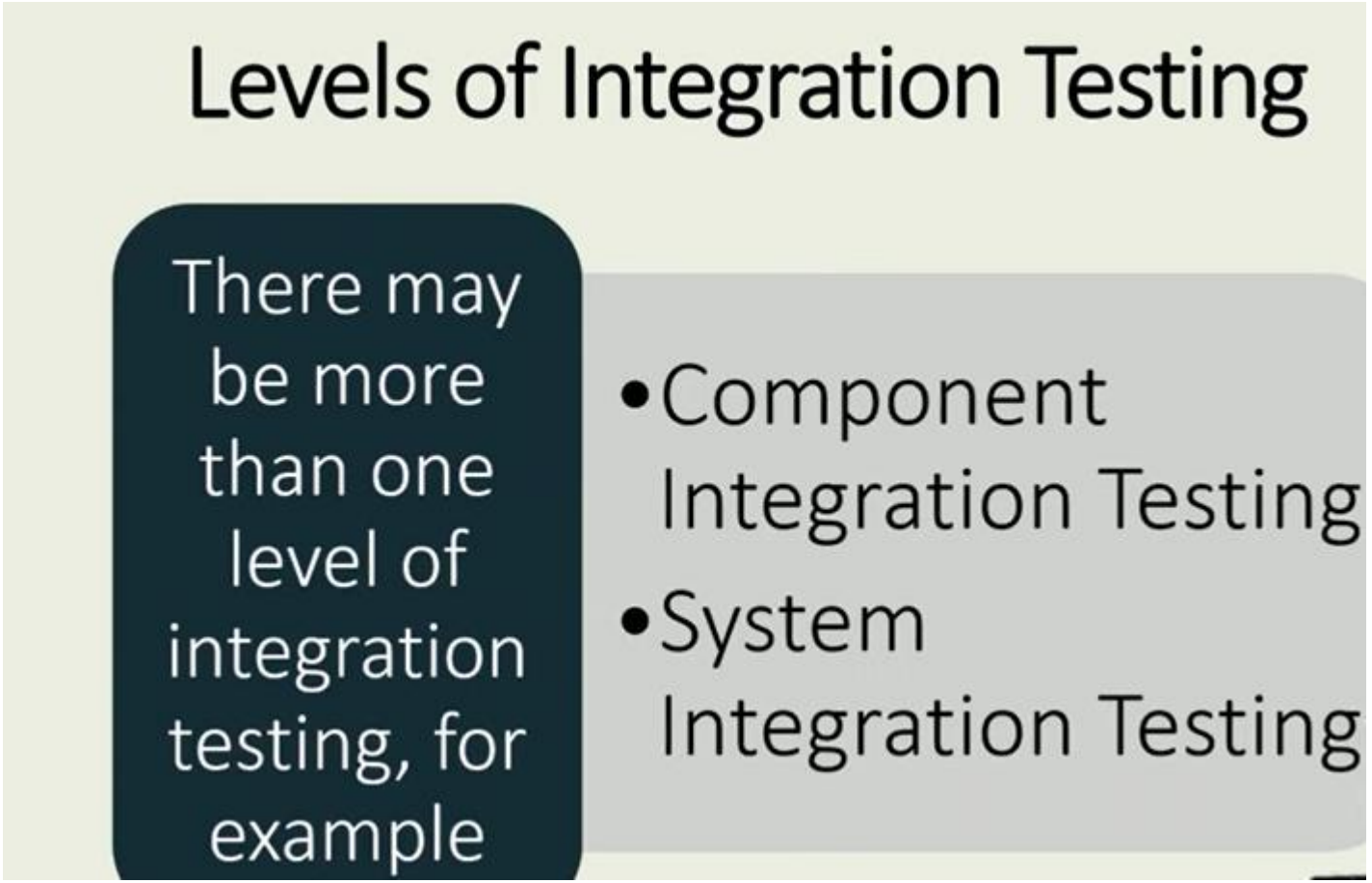
In integration testing testers concentrate only on the integration

Testers should build integration tests in the order required for most efficient integration testing

Integration testing tests interactions to different parts of a system like, system, System Hardware, Operating system, Interfaces between oth

File system, between others.

Levels of Integration Testing



Component Integration Testing

Component integration testing tests the interactions between software components

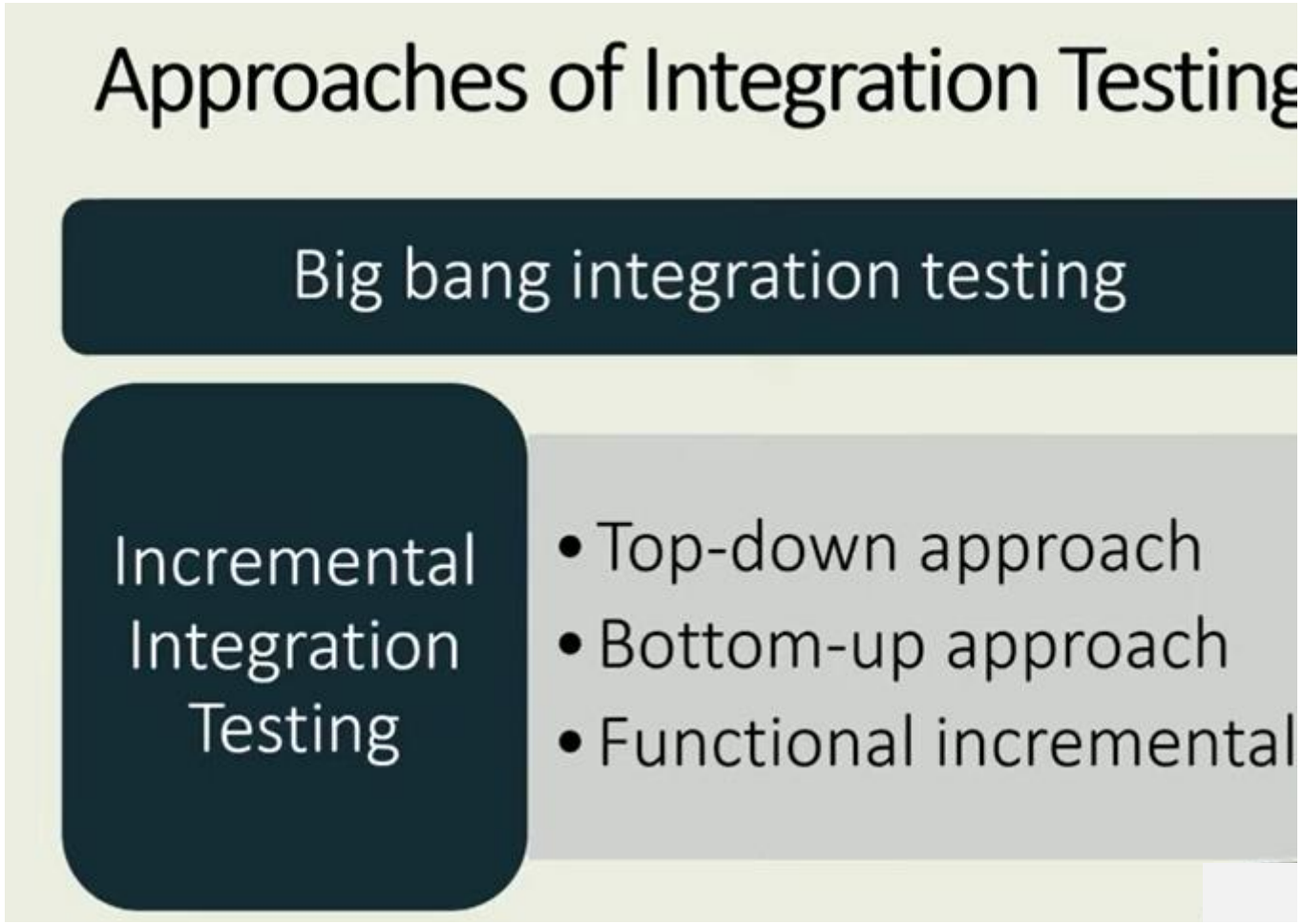
Component integration testing is done after Component /Unit testing

System Integration Testing

System integration testing tests the interactions between different systems

Done mostly after system testing is complete

Done to identify the cross-platform issues which may arise after integrating the system to other system


Approaches of Integration Testing
Big bang integration testing
Incremental Integration Testing
• Top-down approach
• Bottom-up approach
• Functional incremental

Big bang integration testing

All components or systems are integrated simultaneously, after which everything is tested as a whole

Advantage of big bang approach is that everything is finished before integration testing starts. So, STUBS and DRIVERS are not required

Disadvantage is that it is difficult to trace cause of failures due to late integration

Incremental Integration Testing

All programs are integrated one by one and testing is done after every step

Advantage is that defects are found early and easy to find root cause for those defects

Disadvantage is that it is time consuming because STUBS and DRIVERS are required

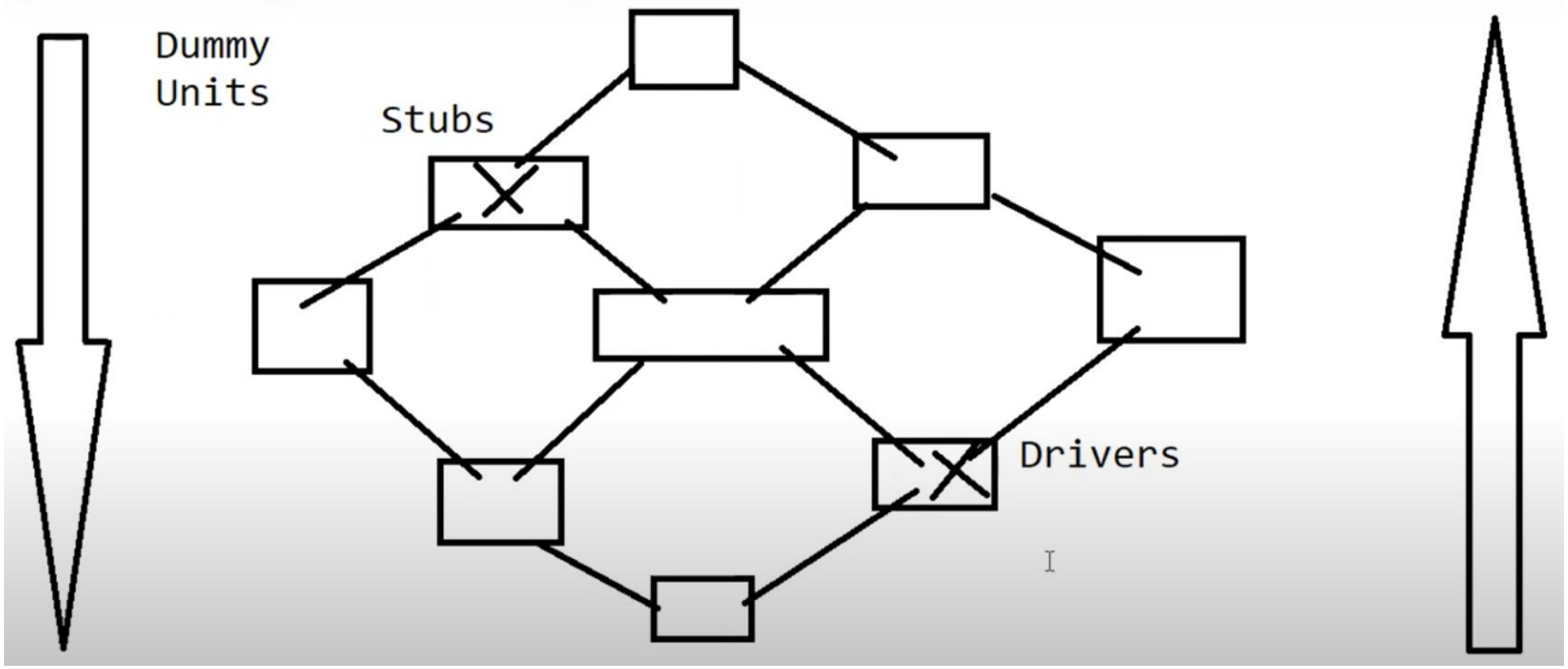
Top down approach

- Testing takes place from top to bottom
- Follows the control flow or architectural structure (e.g. It starts from the GUI or main menu).
- The top down approach utilises STUBS to replace the unfinished components
- The top level component of the hierarchy is tested first with the lower unfinished components it integrates with being replaced by STUBS
- Testing continues down the hierarchy by replacing the stubs with code

Bottom up approach

- Takes place from the bottom of the control flow upwards
- The lowest level components are tested first
- Missing components are substituted with DRIVERS
- As soon as code for other module gets ready, these drivers are replaced with the actual module.

Sandwich integration testing approach.



System Testing

Testing the behavior of the whole software/system as defined by scope is known as system testing

Purpose of system testing is to find as many defects as possible

It is often carried out by specialist testers that form an independent test team

The system test environment should correspond to the production environment as much as possible to minimize the risk of environment-specific failures not found by previous testing cycles

System Testing

System testing should investigate both functional and non-functional requirements of the system

Black box techniques (Specification – based) should be used to system test the functional requirements

White box techniques (Structure-based techniques) ma then be used to assess the thoroughness of the testing

Test cases for System Testing

System testing includes test cases based on:

- Software Requirements specifications (SRS)
- Business processes
- End user scenarios
- High level description of system behaviors
- Interactions with the operating system
- Risks involved for the system

Acceptance Testing

- Acceptance testing starts after system testing is done and all major defect are resolved
- Acceptance testing is most often responsibility of customer but other stakeholders are also involved
- The main goal in acceptance testing is to establish confidence in the system
- Test environment for acceptance testing should be in most aspects representative of production environment (as-if production)

Finding defects should not be the main focus of acceptance testing, its main focus should be to determine if the system is fit for purpose.

Acceptance Testing

Acceptance testing occurs at more than just a single test level.

For example:

- A Commercial off the shelf (COTS) software product may be acceptance tested when it is installed or integrated
- Acceptance testing of the usability of a component may be done during component testing
- Acceptance testing of a new functional enhancement may come before system testing

Forms of Acceptance Testing

Acceptance Testing can be classified into different forms like:

- User acceptance testing
- Operational acceptance testing
- Contract acceptance testing
- Compliance acceptance testing
- Alpha testing
- Beta testing

User Acceptance Testing

Focuses on functionality and validates fitness-for-use of the system

Business users and other stakeholders should be heavily involved

UAT Test environment should be very close to the production environment

Operational Acceptance Testing

Operational acceptance testing validates if the system meets operational requirements, it is also known as "Production Acceptance Testing"

Some of the tests to include in operational acceptance testing are:

- Testing backup/restore
- High availability & Disaster recovery
- User management
- Testing maintenance tasks
- Periodic checks of security vulnerabilities

Contract Acceptance Testing

It is performed against the contracts acceptance criteria

Acceptance criteria for contract acceptance testing is formally defined before signing the contract

Compliance Acceptance Testing

Compliance acceptance testing is also know and regulation acceptance testing.

Compliance acceptance testing is performed against the regulations (Government, legal, safety, medical) which must be adhered to.

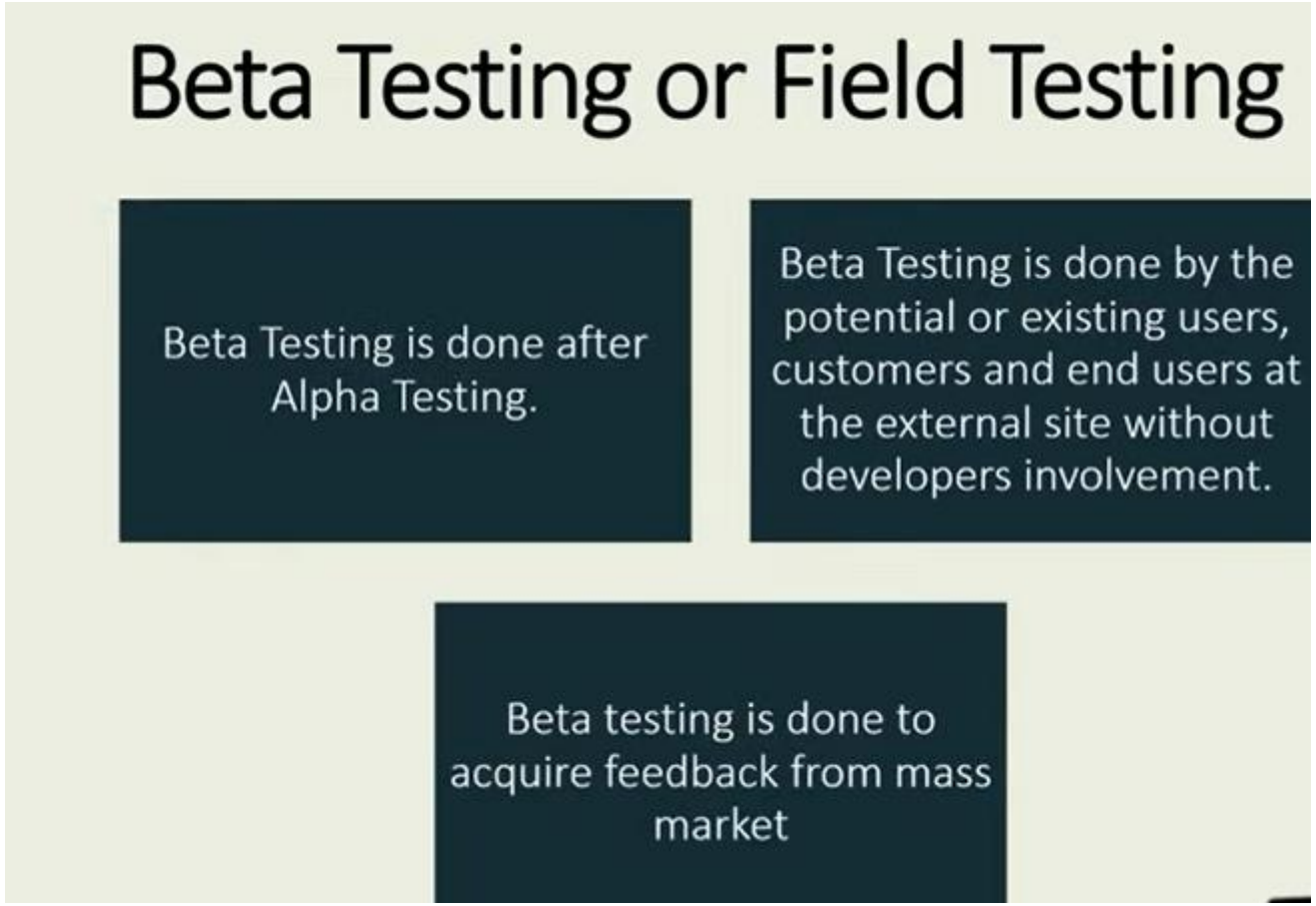
Alpha Testing

Testing the application outside of Business Owner and Development Team is known as Alpha testing.

Alpha Testing exposes the software to people and environments outside of the project team and done at the developers site.

Alpha Testing is mostly done for COTS (Commercial Off the Shelf) software to ensure internal acceptance before moving the software for beta testing.

Alpha testing provides feedback to the developers directly from the customers / potential customers


Beta Testing or Field Testing
Beta Testing is done after Alpha Testing.
Beta Testing is done by the potential or existing users, customers and end users at the external site without developers involvement.
Beta testing is done to acquire feedback from mass market

Test Types

Test type is focused on a particular test objective. There are four main test types.

- The testing of a function to be performed by the software(Functional Testing)
- Testing non-functional quality characteristic like performance, load, stress testing (non-functional testing)
- Testing the structure or architecture of the software/system (structural testing or white-box testing)
- Testing related to any defects fixes or software changes (Re-testing and regressi testing)

Testing of function (Functional Testing or Black-box testing)

Functional testing tests the function of the system or component

Functionality of the system is usually described in documents such as software requirements specifications, use cases or a functional specification

Functional testing considers the specified behaviour of the system and is often referred as black box testing

Functional Testing may be performed at all levels of testing

Perspectives of doing Functional Testing

Requirements based

- Uses functional requirements specification as basis for designing tests
- Prioritization done based on risk criteria mentioned in requirements document

Business-process based

- Uses knowledge of the business process as a basis for designing tests
- Business processes describe the scenarios involved in day to day business use of system
- Use cases are very useful basis for designing test cases

Testing of software product characteristics (Non-functional testing)

1. Non-functional testing tests the non functional attributes like reliability, efficiency, maintainability, usability etc.
2. Non functional testing is done with an aim to find how well or how fast an operation is performed by software.
3. It may be performed at all test levels
4. It tests how well the system works
5. Some of the non functional testing types are:
   - *Performance testing*
   - *Load testing*
   - *Stress testing*
   - *Usability testing*
   - *Maintainability testing*
   - *Reliability testing*
   - *Portability testing*

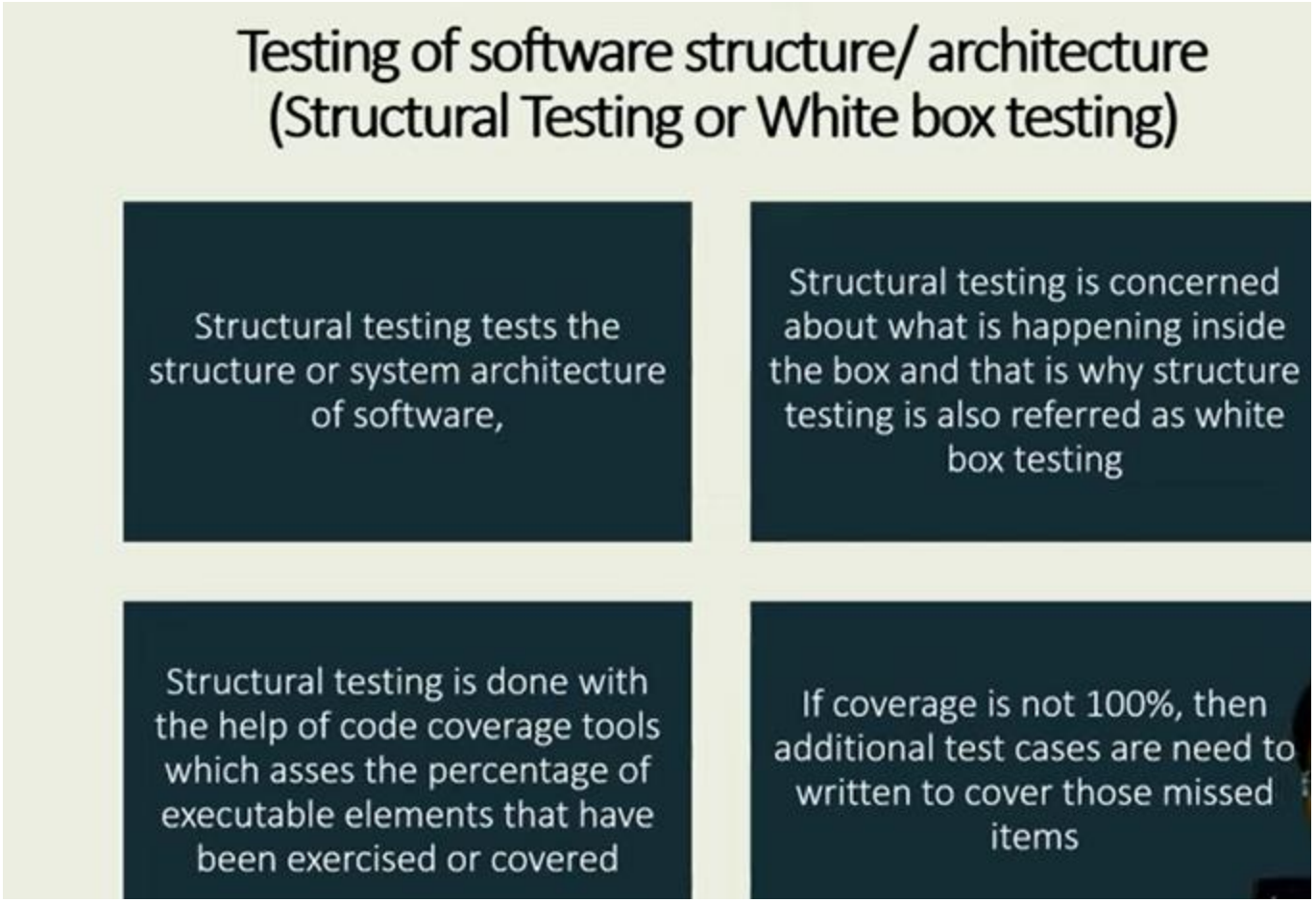
Types of Functional Testing

- Unit testing
- Integration testing
- System testing
- Acceptance testing

- Smoke testing
- Sanity testing
- Regression testing
- Re-testing
- UI testing

- GUI testing
- White-box testing
- Black-box testing

- Alpha testing
- Beta testing


Testing of software structure/ architecture
(Structural Testing or White box testing)
Structural testing tests the structure or system architecture of software,
Structural testing is concerned about what is happening inside the box and that is why structure testing is also referred as white box testing
Structural testing is done with the help of code coverage tools which asses the percentage of executable elements that have been exercised or covered
If coverage is not 100%, then additional test cases are need to written to cover those missed items

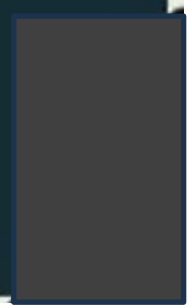
Testing related to changes (Confirmation and regression testing)

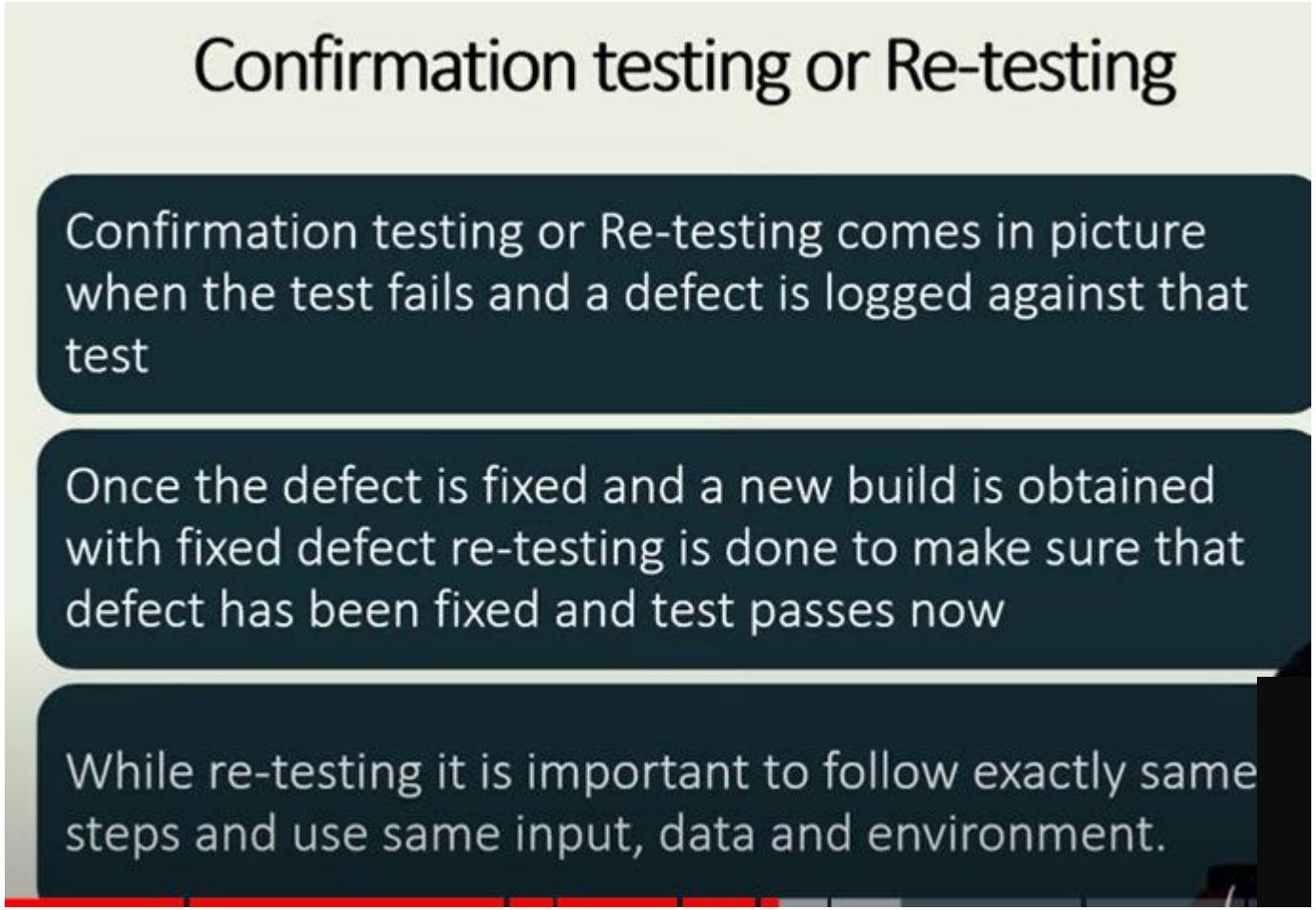
Testing related to changes tests the software whenever there are changes done in the software.

For Example:
- When you fix the defects/bugs
- When you implement new functionality in software

Testing related to changes has two types of testing

- Confirmation testing (re-testing)
- Regression testing




Confirmation testing or Re-testing
Confirmation testing or Re-testing comes in picture when the test fails and a defect is logged against that test
Once the defect is fixed and a new build is obtained with fixed defect re-testing is done to make sure that defect has been fixed and test passes now
While re-testing it is important to follow exactly same steps and use same input, data and environment.

Regression Testing

If a defect has been fixed it might have affected other areas of code. Regression testing is done to find the "unexpected side effects" of defect fixes

Purpose of regression testing is to verify that any modifications in software have not caused unintended side effects in software.

Regression tests are executed whenever the software changes either due to defect fix or addition of new functionality

All the test cases in regression test suites are executed every time the new software version is available so regression test suite is an ideal candidate for automation

Impact analysis and regression testing

Usually maintenance testing consists of two parts

- Testing any changes
- Regression testing to show that any changes done in software does not affect the rest of the software

Impact analysis is the main activity in maintenance testing

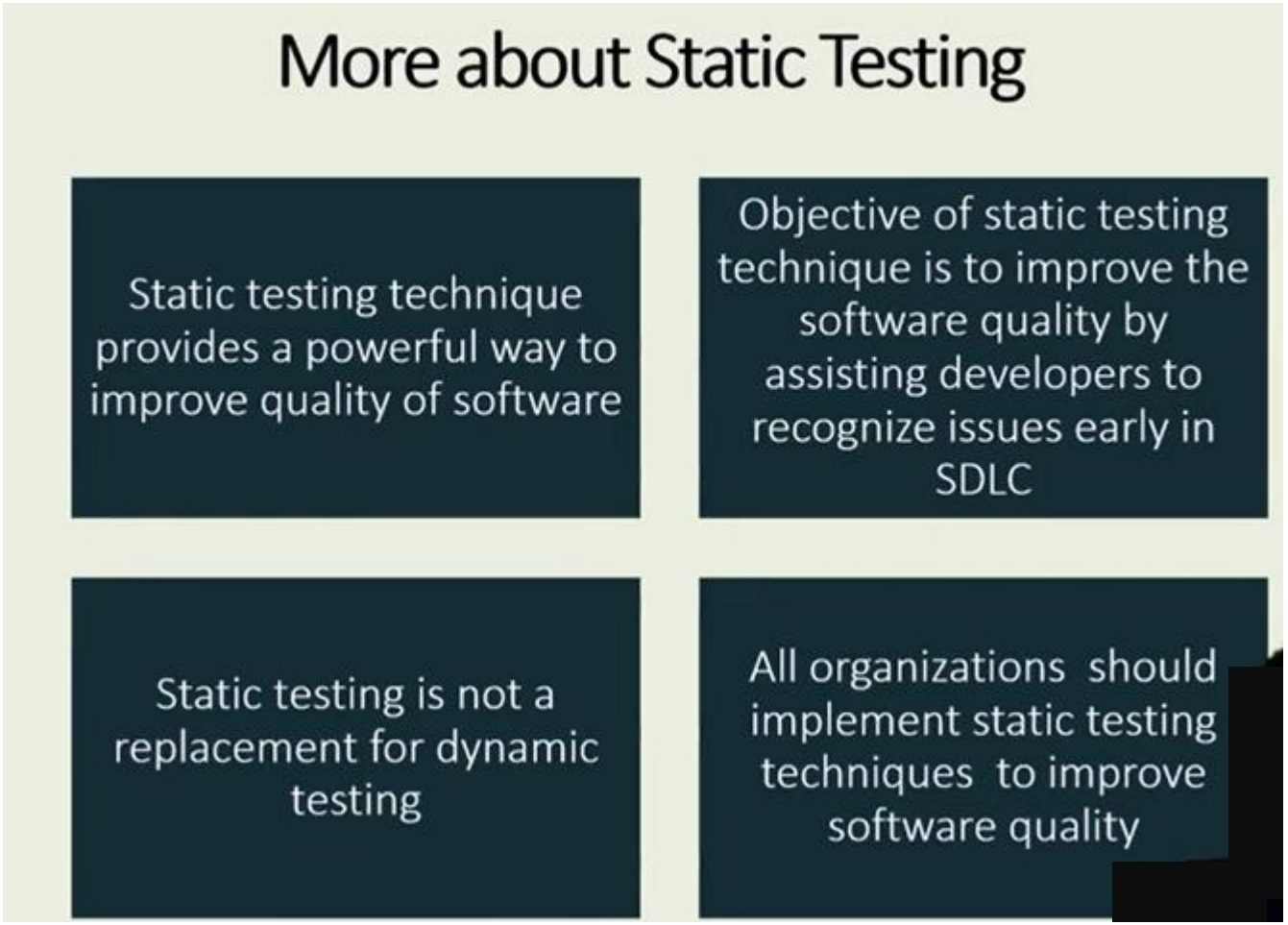
- Impact analysis is done together with stakeholders to decide what parts software might be unintentionally affected and need careful regression
- Risk analysis helps to decide where to focus regression testing

What is static testing?

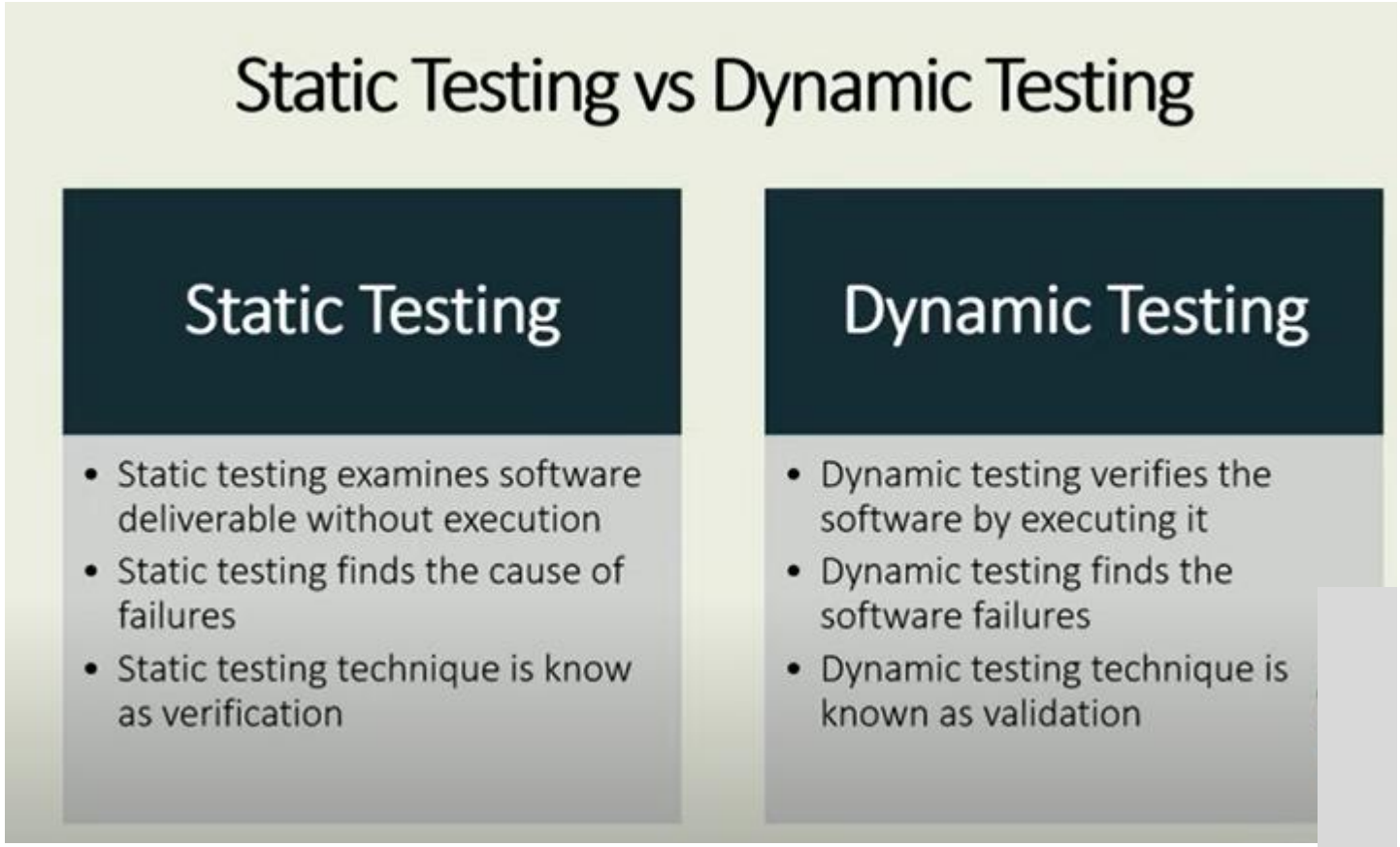
Testing of a component or system at specification or implementation level without executing the code is known as static testing (**Walkthrough, Technical Reviews, Inspection**)

During static testing software products are examined manually or with some tools but there is no execution done

Static testing helps to verify the software deliverables for which dynamic testing techniques cannot be applied (like, design document, SRS, test plan etc.)


More about Static Testing
Static testing technique provides a powerful way to improve quality of software
Objective of static testing technique is to improve the software quality by assisting developers to recognize issues early in SDLC
Static testing is not a replacement for dynamic testing
All organizations should implement static testing techniques to improve software quality

Static Testing vs Dynamic Testing



Next chapter

Introduction to Test Development Process

It is important to know what you are trying to test, inputs and expected outcome before you actually start test execution.

Let's learn about

1. Test Conditions (Documented in test design specification)
   - An item or event of a component or system that could be verified by one or more test cases, e.g., a function, transaction, feature, quality attribute, or structural element.
2. Test Procedures or Test Scenario (Documented in test procedure document)
   - A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script.
3. Test Cases (Documented in test case specification)
   - A set of input values, execution preconditions, expected results and e postconditions, developed for a particular objective or test condition, exercise a particular program path or to verify compliance with a spe requirement.

Formality of test documentation

Testing can be performed with varied level of formality

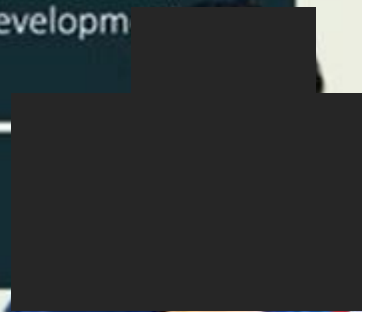
It could be very informal with minimal documentation or very formal with extensive test documentation

Right level of formality depends on context of testing

- A commercial safety-critical application needs very formal approach as compared to a family website which will be used by only few people.

Level of formality is also influenced by the organization, its culture, maturity of developm testing process etc.

In this session we will cover the formal approach of test documentation



Test Development Process

The test development process consists of 3 main phases

- Test analysis
- Test design
- Test implementation

Test Analysis - Identifying Test Conditions

The process of looking at test basis to derive test information is known as Test analysis.

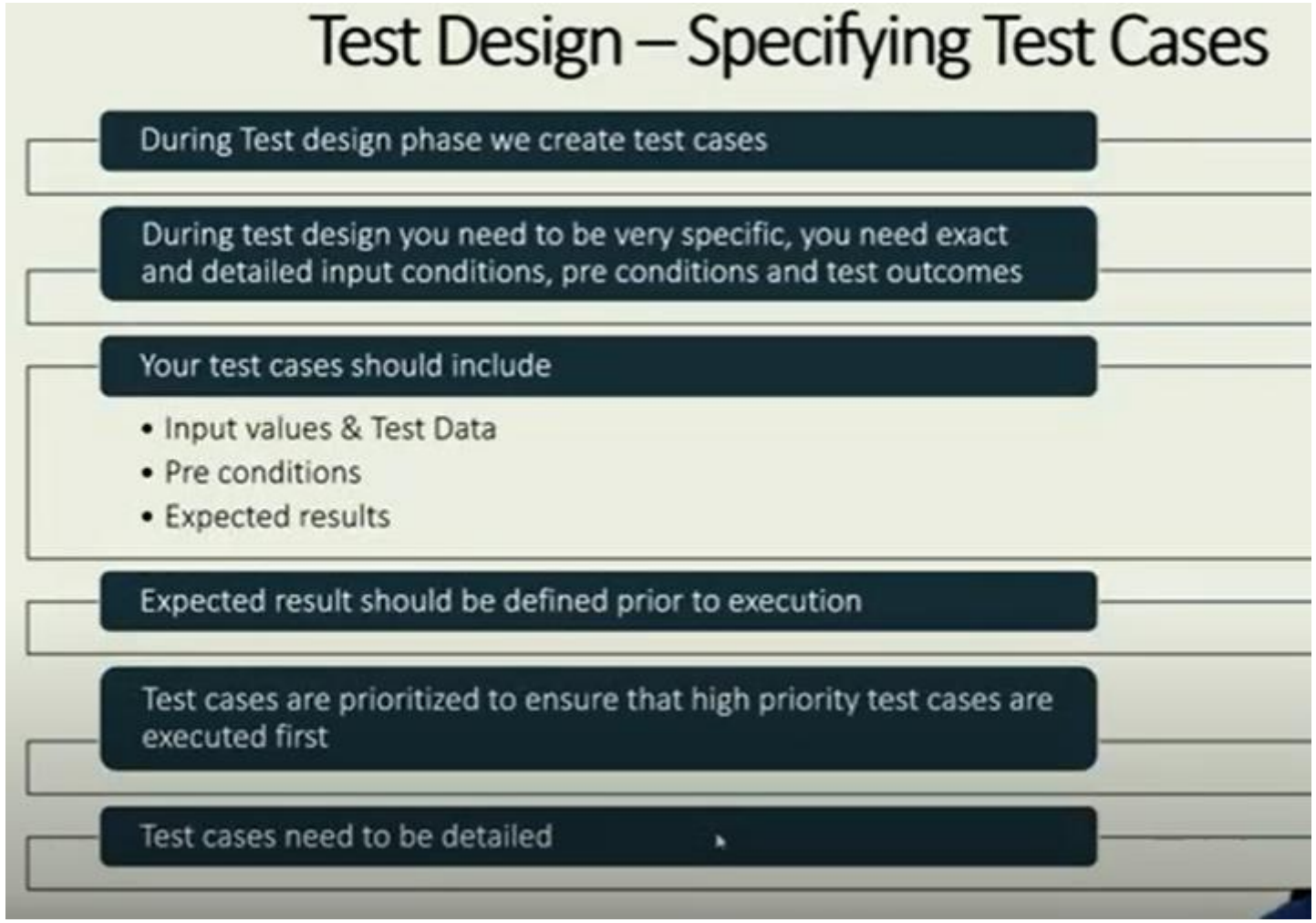
Test basis is anything like system requirement, SRS, design documents, code, business process etc.

In test analysis we find out what could be tested or test conditions, test condition is something which can be tested

The chosen test conditions depend on the test strategy or detailed test approach. For example, they might be based on risk, models of the system, likely failures, compliance requirements, expert advice

The test conditions are then converted into detailed test cases, test design techniques are u figure out good set of tests

Test conditions should be able to be linked back to the source documentation to identify traceability and Requirements Coverage


Test Design – Specifying Test Cases
During Test design phase we create test cases
During test design you need to be very specific, you need exact and detailed input conditions, pre conditions and test outcomes
Your test cases should include
• Input values & Test Data
• Pre conditions
• Expected results
Expected result should be defined prior to execution
Test cases are prioritized to ensure that high priority test cases are executed first
Test cases need to be detailed

Test Implementation – Specifying test procedures or scripts

In test implementation the test cases are grouped into test suites and ordered into proper sequence

- Functional Test Suite
- Regression Test suite

During test implementation you also specify the test procedure – Test procedure is a document which specifies the steps to be taken in running a set of tests, it is also referred as test script.

Test procedures are then formed into test execution schedule or super script which specifies which procedures are to be run when and by whom.

Test design techniques categories

There are three categories of Test design techniques

- Specification-based (Black-box) testing technique
- Structure-based (White-box) testing technique
- Experience-based testing technique

Specification-based (Black-box) testing technique

Specification based testing technique is also known as black box testing technique

This is dynamic testing technique

This is input-output driven testing technique as you are not concerned about internal software structure.

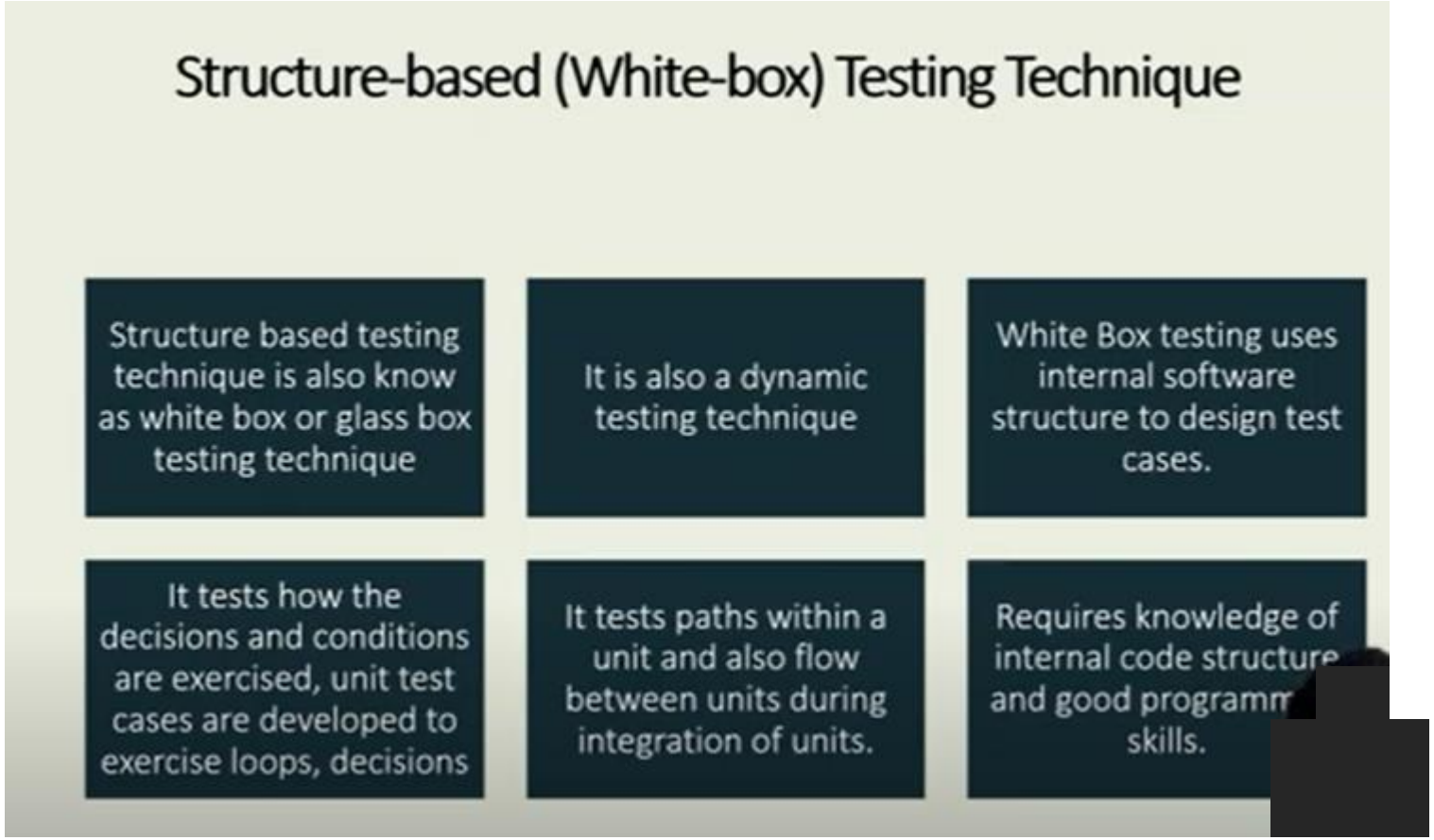
In this technique tester is concerned about what the software does not how it does

You test both functional and non functional characteristics of software using this technique

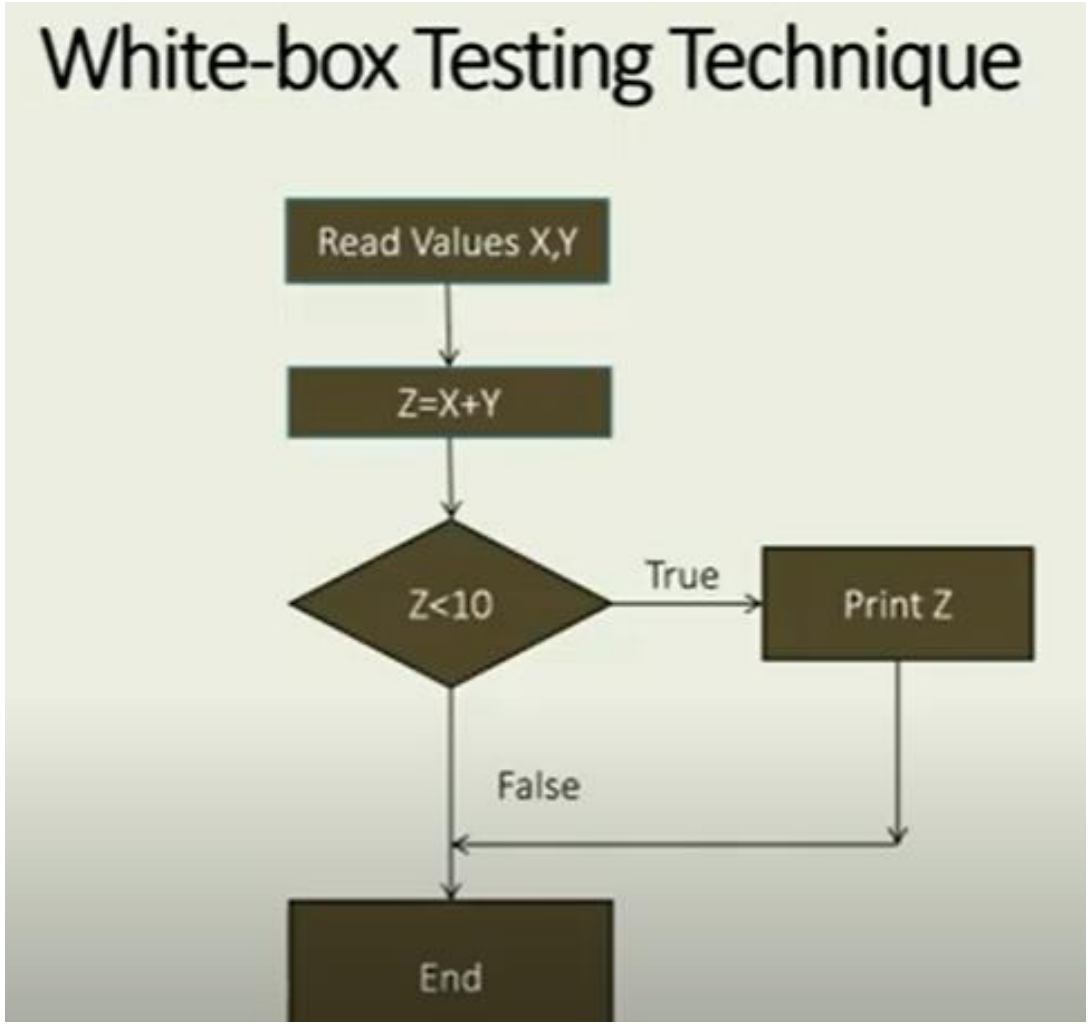
Black Box Testing Technique

If Output = Expected result, then your test passes

Structure-based (White-box) Testing Technique



White-box Testing Technique



Experience-based testing technique

In experience based testing technique knowledge of testers, developers, users and other stakeholders is the used to design test conditions and test cases

Experiences of both technical people and business people is considered

Because of the prior experience people know where things can go wrong in the software.

Where to apply different categories of testing techniques?

Specification based or black box testing techniques are used at all levels of testing

- Component testing through to acceptance testing
- Requirements docs ,design docs form the basis for tests

Structure based or white box testing can also be applied at all test levels

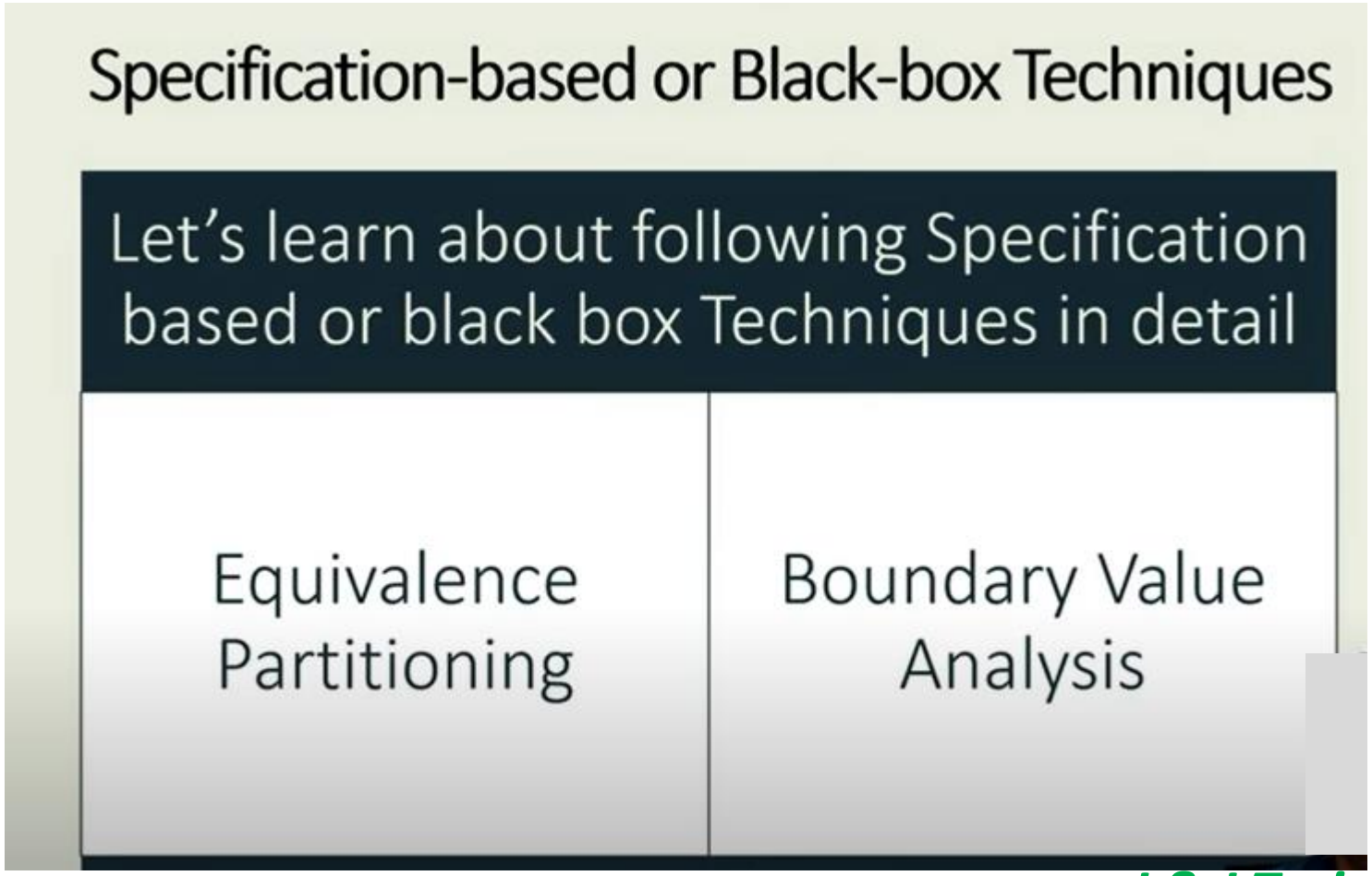
- Component testing through to acceptance testing

Experience based testing technique is used to complement black-box and white box testing technique

- This is also used when there is inadequate documentation or no documentation for designing tests
- Useful for less risk applications or under extreme time pressure
- This also leads to exploratory testing

New chapter

Test Design Techniques in Detail



J & J Training

Equivalence Partitioning

- Equivalence partitioning is a black box test design technique
- Can be applied at all levels of testing
- Idea behind EP is to divide or partition a group of test conditions into a set that is considered same by the software
- We need to test just one condition from that partition as it is assumed that all the values in that partition will be treated in same way
  - If one of the value in a partition will work then we assume all the values in that partition will work.
  - If one of the value in that partition fails then it is assumed that all the values in that partition will fail
- These conditions may not always be true however testers can use better partitions and also test some more conditions within those partitions to confirm that the selection of that partition is fine.
- Equivalence partitions are also known as equivalence class

Equivalence Partitioning Example

A store in city offers different discounts depending on the purchase amount. Minimum cart amount is $5

If purchase is in the range of $5.00 up to $50.00 - no discounts

Purchase over $50.01 and up to $200.00 - 5% discount

Purchase of $200.01 and up to $500.00 - 10% discounts

Purchase of $500.01 and up to $max - 15% discounts.

Now to test this we can identify 4 valid equivalence partitions and 2 invalid partition as shown below.

Invalid Partition	Valid Partition(No Discounts)	Valid Partition(5%)	Valid Partition(10%)	Valid Partition(15%)	Invalid partition
-$4.99	$5.00-$50	$50.01-$200	$200.01-$500	$500.01-$max	$max+$0.01

Equivalence Class Partitioning - Black Box Test Design Technique

•Software Inputs are divided into Group according to their similar behavior
•Examples:

1.Data Range: Text field accepting values from 1 to 100

1. Dividing into three partitions (Invalid - Valid - Invalid)
2. Invalid - -5
3. Valid - 75
4. Invalid – 121

2.Data Type: Text field only accepts positive number

1. Dividing into different partitions (Alphabets, Positive Numbers, Negative Numbers, Special Characters, Combination)
    1. Invalid (c) - Valid (9) - Invalid (-3) - Invalid ($) - Invalid (c9, 9$ etc)

3.Data Size: Text field accepts 10 size (Ex: Mobile Number)

1. Dividing into different partitions (Less than 10 size, Size 10, More than 10 Size)
    1. Invalid (123), Valid (1234567890), Invalid (123456789101112)

Equivalence Class Partitioning - Black Box Test Design Technique

4. **Others**: Only Credit Cards accepted for payment
    1. Dividing into different partitions (Net-banking, Credit Card, Cash on delivery, Debit Cards)
    2. Invalid - Valid - Invalid – Invalid

4. **Combinations**:
    1. Text field accepting 10 to 20 Alphabets (Range and Type)
    2. Text field accepting 10 numerical digits only (Size and Type)

Boundary Value Analysis (BVA)

A boundary value is any input or output value on the edge of an equivalence partition.

Boundary value analysis is based on testing at the boundaries between partitions

Boundary Value Analysis is a black box test design technique where test case are designed by using boundary values, BVA is used in range checking.

To apply BVA you need to take maximum and minimum values from valid partition together with first or last value respectively in each of the invalid partition adjacent to the valid partition

Boundary Value Analysis (BVA) example

Suppose you have a text field on webpage which accepts values between 1-1000, so the valid partition will be (1-1000), equivalence partitions will be like:

Invalid Partition	Valid Partition	Invalid Partition
0	1-1000	1001 and above

And the boundary values will be 1, 1000 from valid partition and 0,1001 from invalid partitions.

Boundary Value Analysis - Black Box Test Design Technique

This is an extension of the Equivalence Class Partition.

- The maximum and minimum values from the boundary values of ECP
- Edge Values - (maximum + 1, maximum and maximum -1) and (minimum + 1, minimum and minimum -1) for every ECP partition
- Generally we get defects with the boundary values, hence BVA is useful
- Example: Text fields accepts value from 1 to 100

1.Three Partitions - Less than 1, 1 to 100 and More than 100

2.First Partition

1. Min/Max - 0

3.Second Partition

1. Min - 1
2. Max - 100

4.Third Partition

1. Min/Max - 101

Why do both EP and BVA?

Testing only boundaries does not give much confidence as we are testing just extreme values of partitions

It is important to test some values in between the partitions

It is recommended that partitions should be tested separately from boundary values

If you want to be more thorough then start with valid partition testing then invalid partitions, then valid boundaries and finally invalid boundaries

Decision Tables Technique

Field/Case	Case 1	Case 2	Case 3	Case 4
Username	Valid	Valid	Invalid	Invalid
Password	Valid	Invalid	Valid	Invalid
Output	Home Page	Error Message	Error Message	Error Message

T T T
T T F
T F T
T F F
F T T
F T F
F F T
F F F

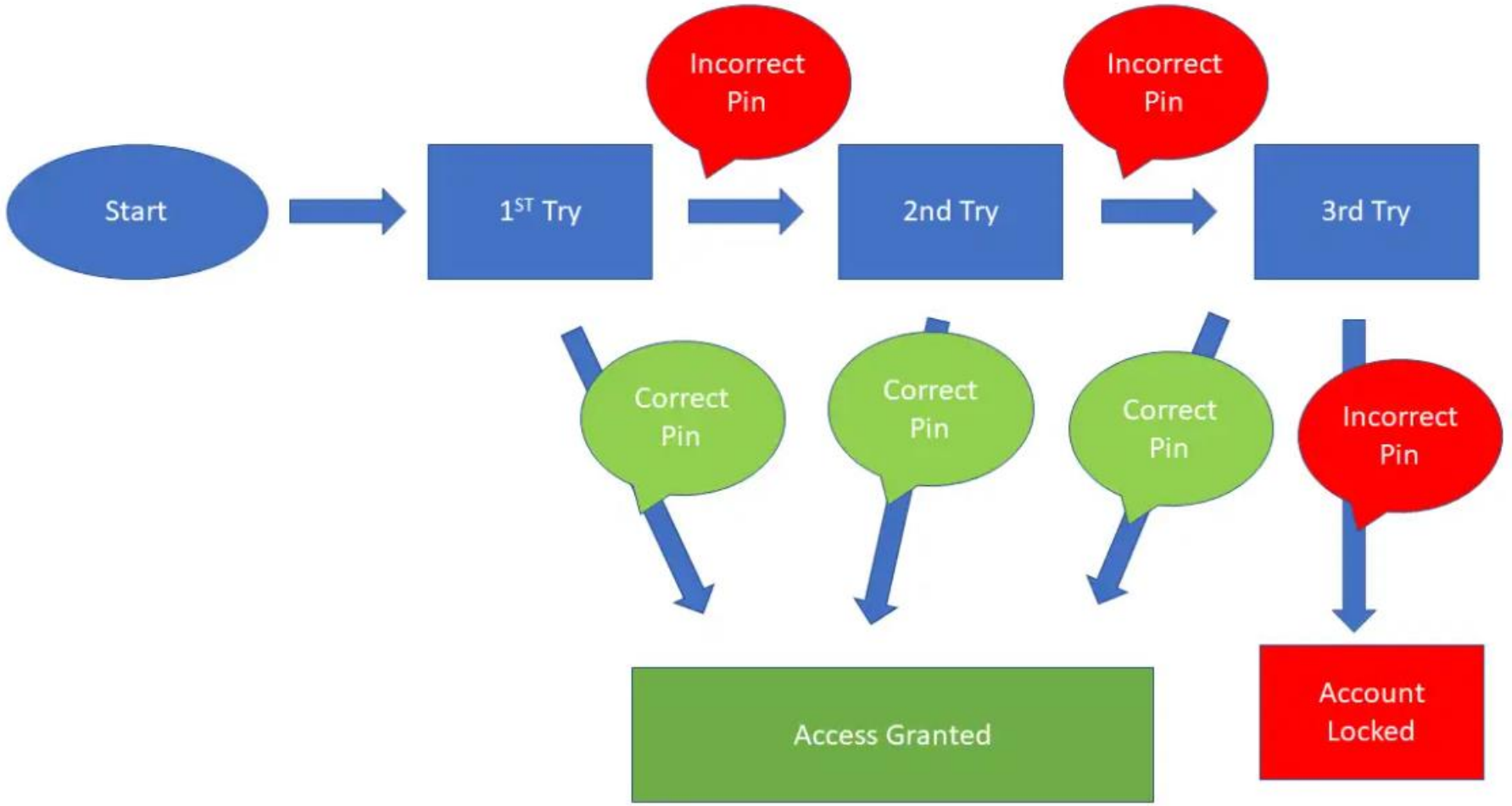
Condition /Cases	Case 1	Case2	Case3	Case 4	Case 5	Case6	Case7	Case8
Passport	Valid	Valid	Valid	Valid	Invalid	Invalid	Invalid	Invalid
Visa	Valid	Valid	Invalid	Invalid	Valid	Valid	Invalid	Invalid
Ticket	Valid	Invalid	Valid	Invalid	Valid	Invalid	Valid	Invalid
Output	Allow	Invalid Ticket	Invalid Visa	Invalid V & T	Invalid Passport	Invalid P & T	Invalid P & V	Invalid PVT

State Transition Testing

1. Used in the functionalities where the different Software States can be captured by providing different sequences of valid and invalid inputs while using

2. Example: Net Banking
    1. User Enter Invalid pin three times and the account will be locked for next 24 hours
    2. What if the User enters valid pin in the second attempt (Allowed to use Netbanking)
    3. What if the User enters valid pin in the third attempt (Allowed to use Netbanking)
    4. What if the User enters valid pin in the first attempt (Allowed to use Netbaking)

State Transition Testing



https://www.youtube.com/watch?v=MRiey1HVgQw