Ripon Al Wasim
riponalwasim@gmail.com
LinkedIn

Course Outline

Chapter 1: Fundamentals of Testing
Chapter 2: Testing throughout the Software Development Life Cycle
Chapter 3: Static Testing
Chapter 4: Test Techniques
Chapter 5: Test Management
Chapter 6: Tool Support for Testing

Course Outline

Chapter 2: Testing throughout the Software Development Life Cycle
No. of Session: 02
Session 03: 2.1 - 2.2.4
Session 04: 2.3 - 2.4.2

Chapter 2: Testing throughout the Software Development Life Cycle

Session 03

Software Development Life cycle Models

Software Development and Software Testing

Software Development Life cycle Models in Context

Test Levels

- Component Testing
- Integration Testing
- System Testing
- Acceptance Testing

Session 04

- Test Types
  - Functional Testing
  - Non-functional Testing
  - White-box Testing
  - Change-related Testing
- Test Types and Test Levels
- Maintenance Testing
- Triggers for Maintenance
- Impact Analysis for Maintenance

Chapter 2: Testing throughout the Software Development Life Cycle

Software Development Life Cycle Models

There are several characteristics of good testing, whichever life cycle model is being used.

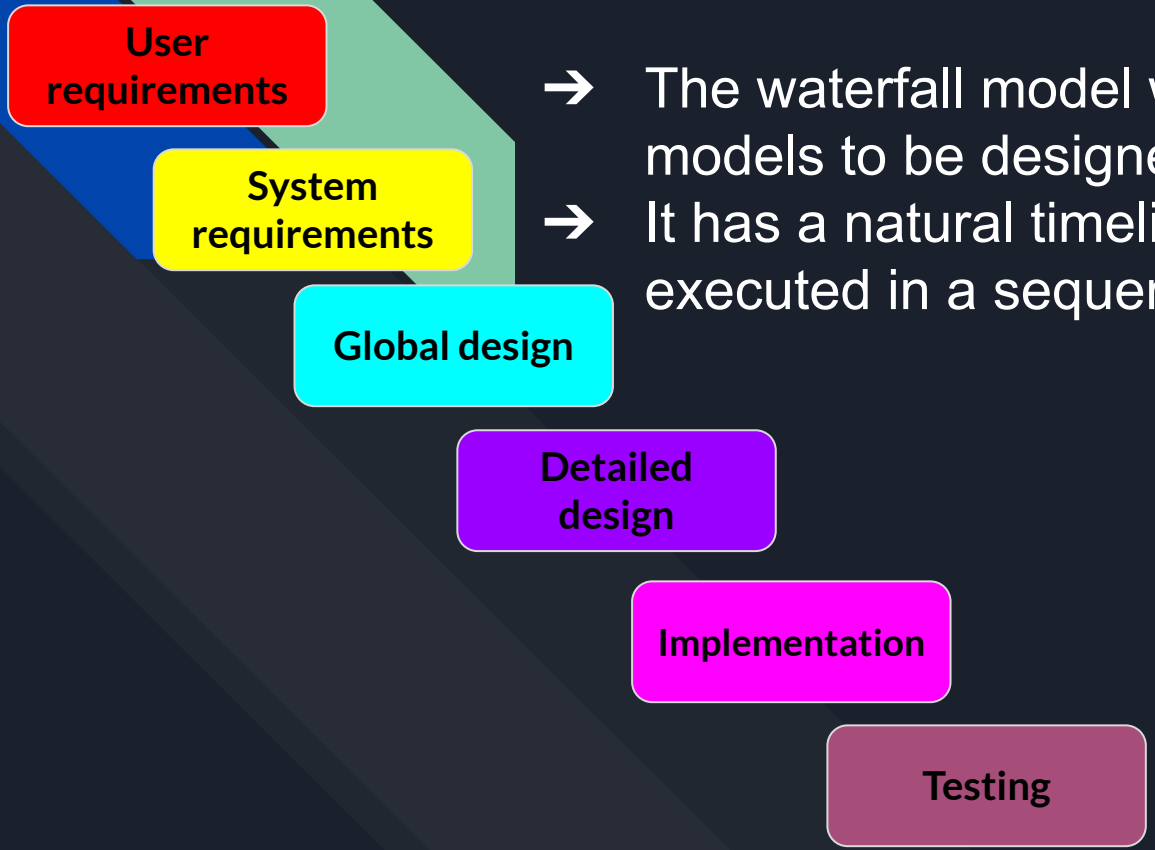
- For every development activity there is a corresponding test activity.
- Each test level has test objectives specific to that level.
- The analysis and design of tests for a given test level should begin during the corresponding software development activity.
- Testers should participate in discussions to help define and refine requirements and design. They should also be involved in reviewing work products as soon as drafts are available in the software development cycle.

Software Development Life Cycle Models

2 categories of Software Development Life Cycle

1. Sequential
2. Iterative/Incremental

SDLC Models: Sequential development models: Waterfall model



- ➔ The waterfall model was one of the earliest models to be designed.
- ➔ It has a natural timeline where tasks are executed in a sequential fashion.

Chapter 2: Testing throughout the Software Development Life Cycle

SDLC Models: Sequential development models

Waterfall model - Problem

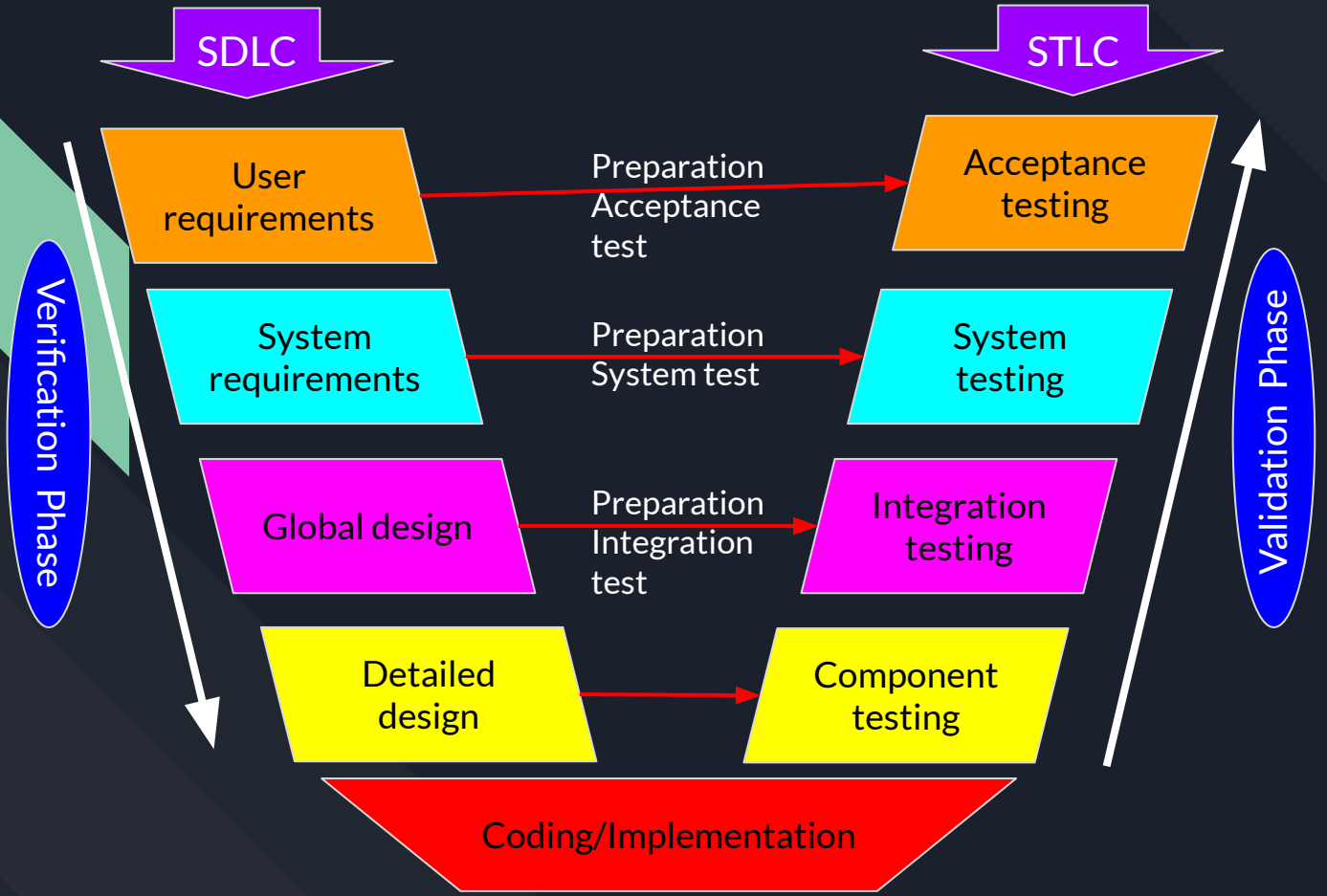
- Testing in the model starts only after implementation is done.
- Defects are detected close to the live implementation date.
- Costs of fixing a defect increase across the development life cycle.
- With this model, it is difficult to get feedback passed backwards up the waterfall and there are difficulties if we need to carry out numerous iterations for a particular phase.

Solution: The V-Model

The V-model was developed to address some of the problems experienced using the traditional waterfall approach.

Chapter 2: Testing throughout the Software Development Life Cycle

SDLC Models:
V-Model
(also called
a Verification
and
Validation
model)

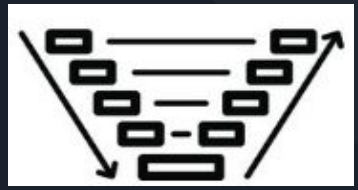


Software Development Life Cycle Models: V-Model

Verification focuses on the question
'Is the deliverable built according to the specification?'

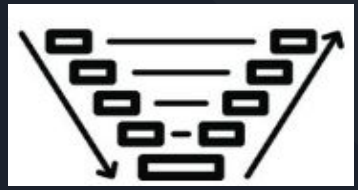
Validation focuses on the question
'Is the deliverable fit for purpose, e.g. does it provide a solution to the problem?'

Software Development Life Cycle Models: V-Model Advantages



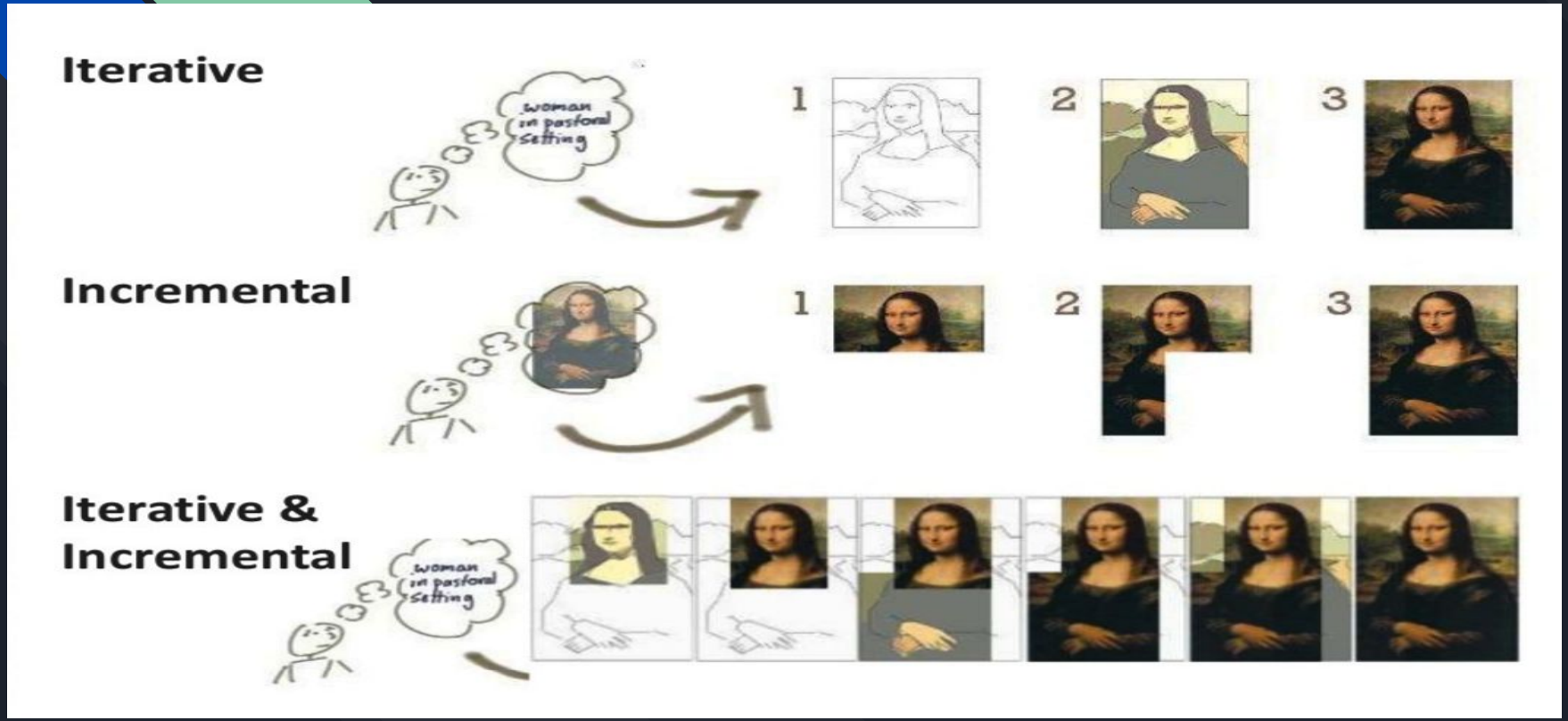
- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model.
- Each phase has specific deliverables and a review process.

Software Development Life Cycle Models: V-Model Disadvantages



- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

SDLC Models: Iterative and Incremental development models



Software Development Life Cycle Models

Iterative and Incremental development models

- Incremental: complete one piece at a time (scheduling or staging strategy). Each increment may be delivered to the customer.

- Iterative: start with a rough product and refine it, iteratively (rework strategy). Final version only delivered to the customer (although in practice, intermediate versions may be delivered to selected customers to get feedback).

Software Development Life Cycle Models

Iterative and Incremental development models

Common issues with iterative and incremental models include:

- ➔ More regression testing.
- ➔ Defects outside the scope of the iteration or increment.
- ➔ Less thorough testing.

Software Development Life Cycle Models

Example of Iterative and Incremental development models

- Rational Unified Process (RUP)
- Scrum
- Kanban
- Spiral or Prototyping

SDLC Models: Iterative and Incremental development models

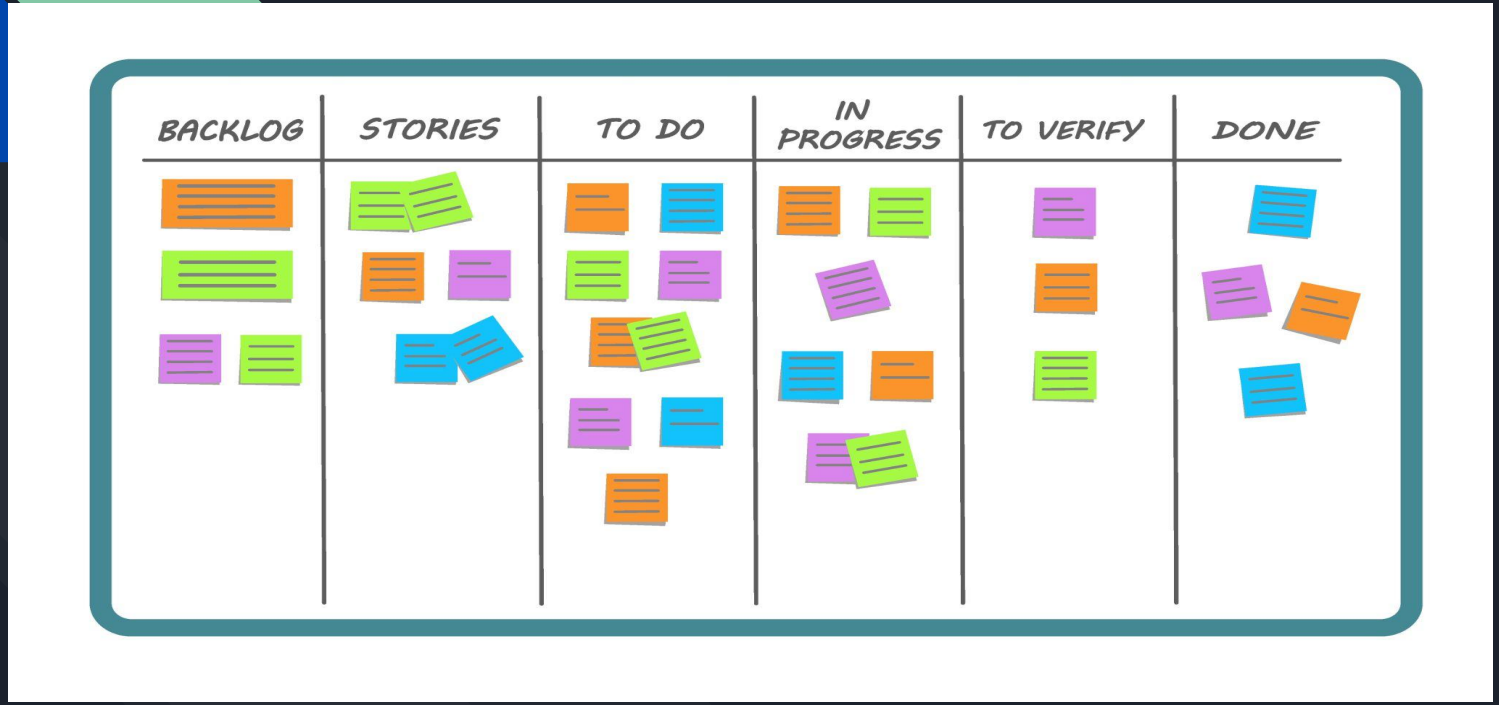
Rational Unified Process (RUP)

RUP is a software development process framework from Rational Software Development, a division of IBM. It consists of 4 steps:

1. Inception: the initial idea, planning (for example what resources would be needed) and go/no-go decision for development, done with stakeholders.
2. Elaboration: further detailed investigation into resources, architecture and costs.
3. Construction: the software product is developed, including testing.
4. Transition: the software product is released to customers, with modifications based on user feedback.

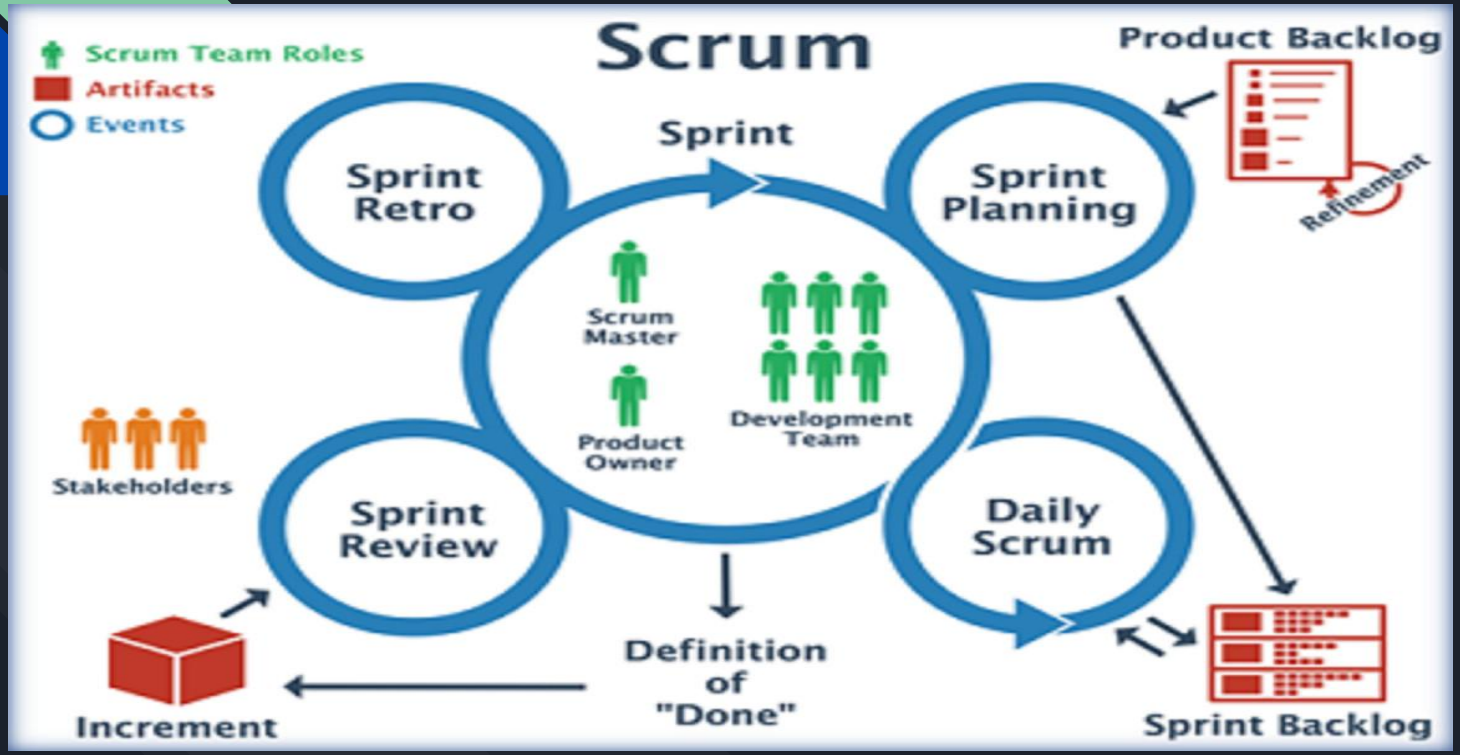
SDLC Models: Iterative and Incremental development models

Scrum: Board



SDLC Models: Iterative and Incremental development models

Scrum: Process



SDLC Models: Iterative and Incremental development models

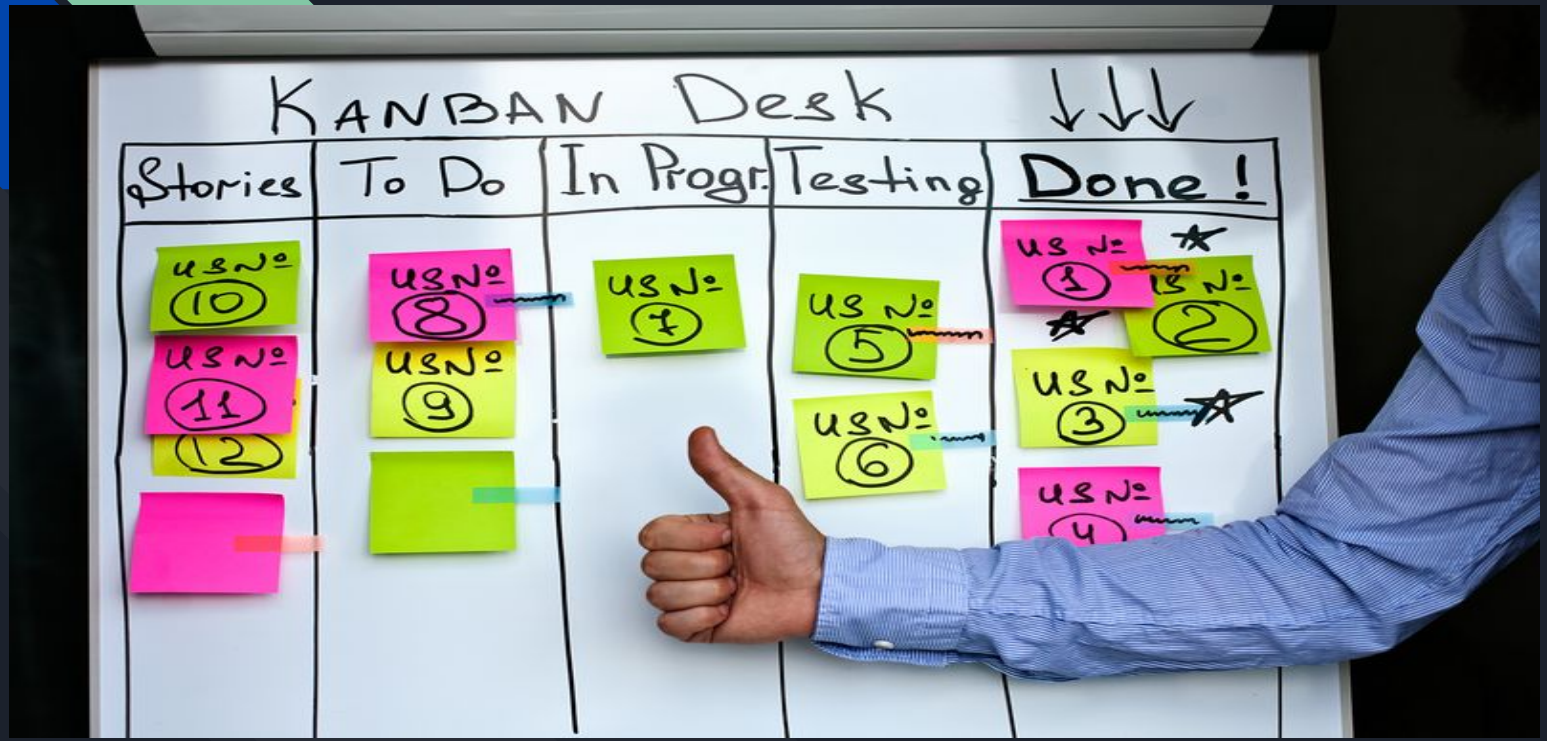
Scrum

The key roles are:

- The Product Owner represents the business, that is, stakeholders and end users.
- The Development team (which includes testers) makes its own decisions about development, that is, they are self-organizing with a high level of autonomy.
- The Scrum Master helps the development team to do their work as efficiently as possible, by interacting with other parts of the organization and dealing with problems.
  The Scrum Master is not a manager, but a facilitator for the team.

SDLC Models: Iterative and Incremental development models

Kanban: Board



SDLC Models: Iterative and Incremental development models

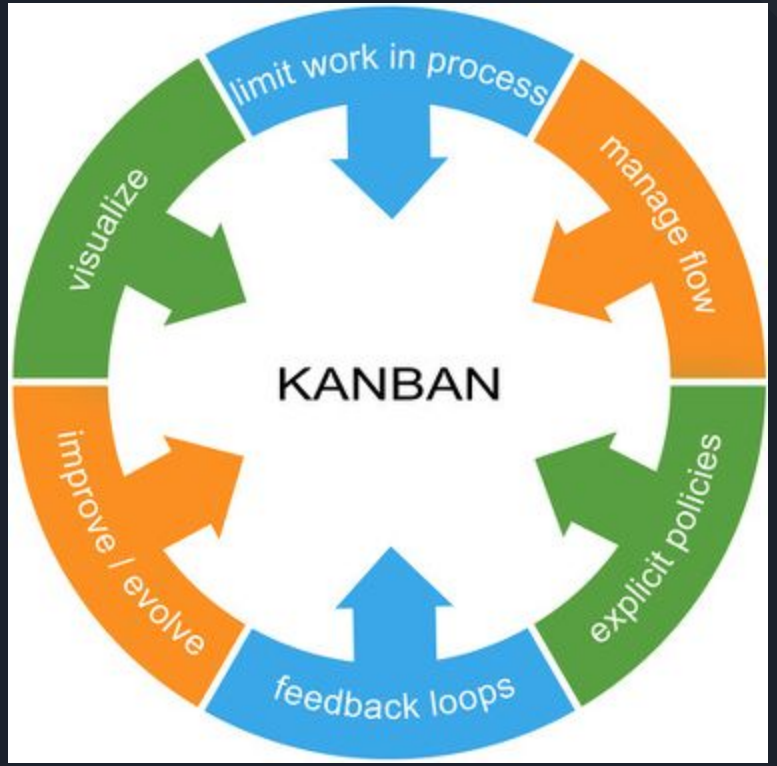
Kanban: Principles

- Start with what you're doing now
- Changes occur organically over time and shouldn't be rushed
- Respect current roles and responsibilities
- Encourage leadership from everyone

SDLC Models: Iterative and Incremental development models

Kanban: Core Practices

1. Visualize the flow of work
2. Limit WIP (Work in Progress)
3. Manage Flow
4. Make Process Policies Explicit
5. Implement Feedback Loops
6. Improve Collaboratively, Evolve Experimentally



SDLC Models: Iterative and Incremental development models

Scrum & Kanban at a glance:

- Scrum is a project management framework that helps teams structure and manage their work through a set of values, principles, and practices.

- Kanban is a project management framework that relies on visual tasks to manage workflows.

SDLC Models: Iterative and Incremental development models

A side-by-side look at Scrum & Kanban: source: https://www.coursera.org/articles/kanban-vs-scrum

Methodology	Scrum	Kanban
Roles	Scrum master, product owner, and development team	There is no role assigned to individuals
Delivery cycle	Sprint cycle lasts one to four weeks	The delivery cycle is continuous
Change policy	Generally not made during sprint	Can be incorporated any time
Artifacts	Product backlog, sprint backlog, product increments	Kanban board
Tools	Jira Software, Axosoft, VivifyScrum, Targetprocess	Jira Software, Kanbanize, SwiftKanban, Trello, Asana
Key concepts or pillars	Transparency, adaptation, inspection	Effective, efficient, predictable

SDLC Models: Iterative and Incremental development models

Spiral (or prototyping)

The Spiral model, initially proposed by Boehm [1996] is based on risk.

There are 4 steps:

1) determine objectives
2) identify risks and alternatives
3) develop and test
4) plan the next iteration

SDLC Models: Iterative and Incremental development models

Agile development

Most Agile teams use Scrum & Extreme Programming (XP)

Typical Agile teams are 5 to 9 people

The Agile manifesto consists of 4 statements describing what is valued in this way of working:

I. individuals and interactions over processes and tools
II. working software over comprehensive documentation
III. customer collaboration over contract negotiation
IV. responding to change over following a plan.

SDLC Models: Iterative and Incremental development models

Agile development: Scrum and XP

Programming (XP) as the main source of Agile development ideas.

Some characteristics of project teams using Scrum and XP are:

- The generation of business stories (a form of lightweight use cases) to define the functionality, rather than highly detailed requirements specifications.
- The incorporation of business representatives into the development process, as part of each iteration (called a sprint and typically lasting 2 to 4 weeks), providing continual feedback and to define and carry out functional acceptance testing.

SDLC Models: Iterative and Incremental development models

Agile development: Scrum and XP

- The recognition that we cannot know the future, so changes to requirements are welcomed throughout the development process, as this approach can produce a product that better meets the stakeholders' needs as their knowledge grows over time.
- The concept of shared code ownership among the developers, and the close inclusion of testers in the sprint teams.

SDLC Models: Iterative and Incremental development models Agile development

- test-driven development (TDD).
- Simplicity: building only what is necessary, not everything you can think of.
- The continuous integration and testing of the code throughout the sprint, at least once a day.

Software Development and Software Testing

- development is writing the code to create the software.
- testing is verifying if this code functions as it is expected to or not.

Software Development Life cycle Models in Context

The most suitable development model for you may be based on the following:

- The project goal.
- The type of product being developed.
- Business priorities (for example time to market).
- Identified product and project risks.

Test Levels

- Component testing (also known as unit, module and program testing)
- Integration testing
- System testing
- Acceptance testing

Test Levels

Unit/Component testing	Done by Developers	Test Individual component
Integration testing	Done by Testers	Test Integrated component
System testing	Done by Testers	Test the entire System
Acceptance testing	Done by End Users	Test the final System

Test Levels

Each test level has the following clearly identified:

- Specific test objectives for the test level.
- The test basis, the work product(s) used to derive the test conditions and test cases.
- The test object (that is, what is being tested such as an item, build, feature or system under test).
- The typical defects and failures that we are looking for at this test level.
- Specific approaches and responsibilities for this test level.

Test Levels

We will cover the followings for each Test Level:

- Objectives
- Test basis
- Test objects
- Typical defects and failures

Test Levels

Component testing - also known as unit, module and program testing

Objectives

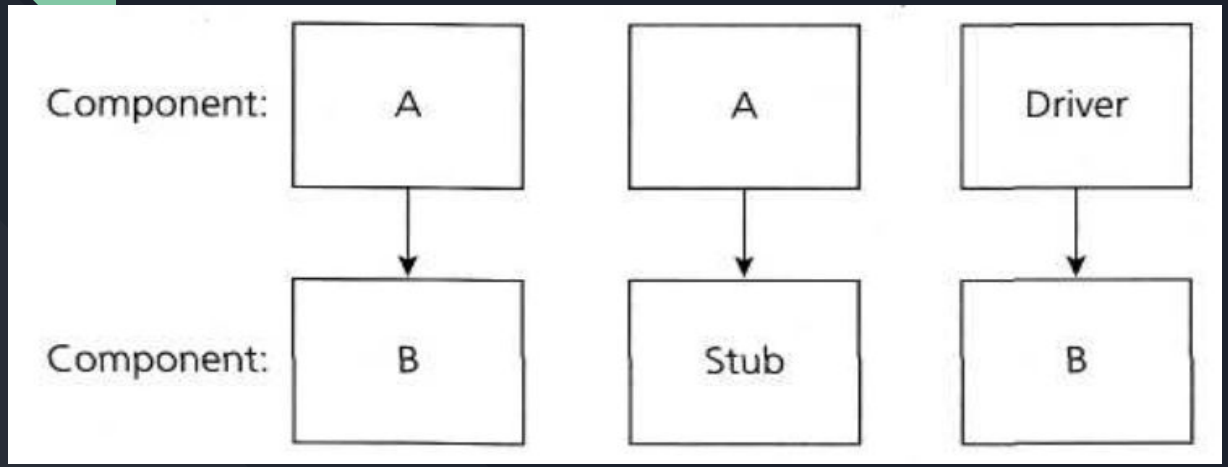
- Reducing risk (for example by testing high-risk components more extensively).
- Verifying whether or not functional and non-functional behaviours of the component are as they should be (as designed and specified).
- Building confidence in the quality of the component: this may include measuring structural coverage of the tests, giving confidence that the component has been tested as thoroughly as was planned.
- Finding defects in the component.
- Preventing defects from escaping to later testing.

Test Levels

Component testing

Stubs & Drivers

- A stub is called from the software component to be tested.
- A driver calls a component to be tested.



Test Levels

Component testing

Test basis

- detailed design
- Code
- data model
- component specifications (if available).

Test Levels

Component testing

Test objects

- components themselves, units or modules
- code and data structures
- Classes
- database models

Test Levels

Component testing

Typical defects and failures

- incorrect functionality (for example not as described in a design specification)
- data flow problems
- incorrect code or logic

Test Levels

Integration testing

- Component integration testing: tests the interactions between software components and is done after component testing.
- System integration testing: tests the interactions between different systems and may be done after system testing.

Test Levels

Integration testing

Objectives

- Reducing risk, for example by testing high-risk integrations first.
- Verifying whether or not functional and non-functional behaviours of the interfaces are as they should be, as designed and specified.
- Building confidence in the quality of the interfaces.
- Finding defects in the interfaces themselves or in the components or systems being tested together.
- Preventing defects from escaping to later testing.

Test Levels

Integration testing

Test basis

- software and system design
- sequence diagrams
- interface and communication protocol specifications
- use cases
- architecture at component or system level
- Workflows
- external interface definitions

Test Levels

Integration testing

Test objects

- Subsystems
- Databases
- Infrastructure
- Interfaces
- APIs (Application Programming Interfaces)
- microservices

Test Levels

Integration testing

Typical defects and failures

- Incorrect data, missing data or incorrect data encoding.
- Incorrect sequencing or timing of interface calls.
- Interface mismatch, for example where one side sends a parameter where the value exceeds 1,000, but the other side only expects values up to 1,000.
- Failures in communication between components.
- Unhandled or improperly handled communication failures between components.
- Incorrect assumptions about the meaning, units or boundaries of the data being passed between components.

Test Levels

Integration testing

- Top-down: testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu). Components or systems are substituted by stubs.
- Bottom-up: testing takes place from the bottom of the control flow upwards. Components or systems are substituted by drivers.
- Functional incremental: integration and testing takes place on the basis of the functions or functionality, as documented in the functional specification

Test Levels

System testing

Objectives

- reducing risk
- verifying whether or not functional and non-functional behaviours of the system are as they should be (as specified)
- validating that the system is complete and will work as it should and as expected
- building confidence in the quality of the system as a whole
- finding defects
- preventing defects from escaping to later testing or to production.

Test Levels

System testing

Test basis

- software and system requirement specifications (functional and non-functional)
- risk analysis reports
- use cases
- epics and user stories
- models of system behaviour
- state diagrams
- system and user manuals

Test Levels

System testing

Test objects

- Applications
- hardware/software systems
- operating systems
- system under test (SUT)
- system configuration and configuration data

Test Levels

System testing

Typical defects and failures

- incorrect calculations.
- incorrect or unexpected system functional or non-functional behaviour.
- incorrect control and/or data flows within the system.
- failure to properly and completely carry out end-to-end functional tasks.
- failure of the system to work properly in the production environment(s).
- failure of the system to work as described in system and user manuals.

Test Levels

Acceptance testing

Objectives

- establishing confidence in the quality of the system as a whole.
- validating that the system is complete and will work as expected.
- verifying that functional and non-functional behaviours of the system are as specified.

Test Levels

Acceptance testing

Different forms of Acceptance Testing

- ❑ User Acceptance Testing (UAT)
- ❑ Operational Acceptance Testing (OAT)
- ❑ Contractual & Regulatory Acceptance Testing
  - Alpha testing
  - Beta testing

Test Levels

Acceptance testing

Test basis

- business processes.
- user or business requirements.
- regulations, legal contracts and standards.
- use cases.
- system requirements.
- system or user documentation.
- installation procedures.
- risk analysis reports.

Test Levels

Acceptance testing

Test basis: additional Test basis for OAT

- backup and restore/recovery procedures.
- disaster recovery procedures.
- non-functional requirements.
- operations documentation.
- deployment and installation instructions.
- performance targets.
- database packages.
- security standards or regulations.

Test Levels

Acceptance testing

Test objects

- system under test (SUT).
- system configuration and configuration data.
- business processes for a fully integrated system.
- recovery systems and hot sites (for business continuity and disaster recovery testing).
- operational and maintenance processes.
- forms
- reports
- existing and converted production data.

Test Levels

Acceptance testing

Typical defects and failures

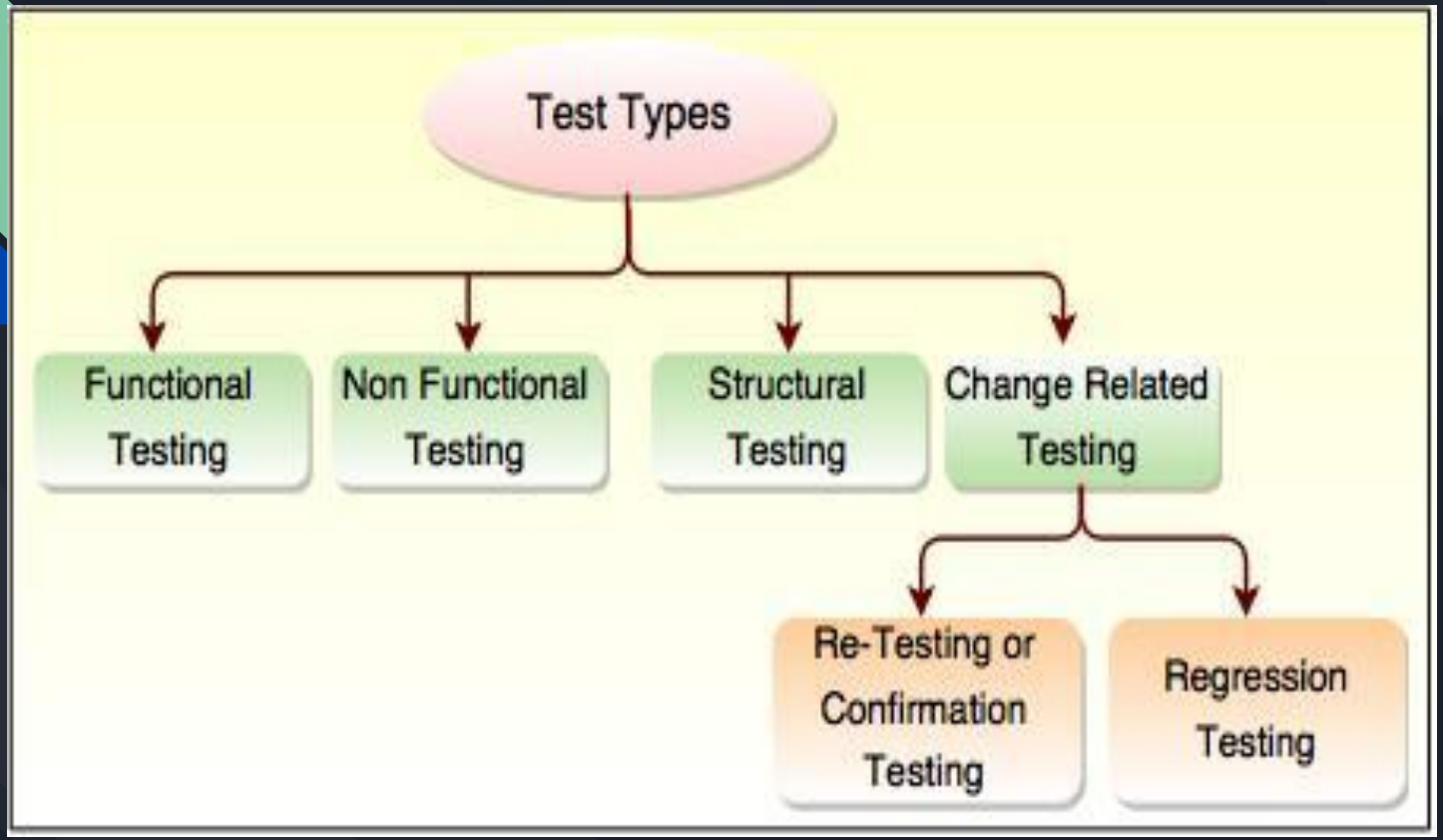
- system workflows do not meet business or user requirements.
- business rules are not implemented correctly.
- system does not satisfy contractual or regulatory requirements.
- non-functional failures such as security vulnerabilities, inadequate performance efficiency under high load, or improper operation on a supported platform.


Thank you
so much!

Session 04

- Test Types
  - Functional Testing
  - Non-functional Testing
  - White-box Testing
  - Change-related Testing
- Test Types and Test Levels
- Maintenance Testing
- Triggers for Maintenance
- Impact Analysis for Maintenance

Test Types


Test Types
Functional Testing
Non Functional Testing
Structural Testing
Change Related Testing
Re-Testing or Confirmation Testing
Regression Testing

Functional Testing: Testing of function

A type of software testing which is used to verify the functionality of the software application, whether the function is working according to the requirement specification.

➔ The function of a system (or component) is 'what it does'.

Purpose

Functional testing mainly involves black box testing and can be done manually or using automation. The purpose of functional testing is to:

- Test each function of the application:
  ➔ providing the appropriate input -> verifying the output
- Test primary entry function: to check all the entry and exit points.
- Test flow of the GUI screen: to verify the user can navigate throughout the application.

Functional Testing

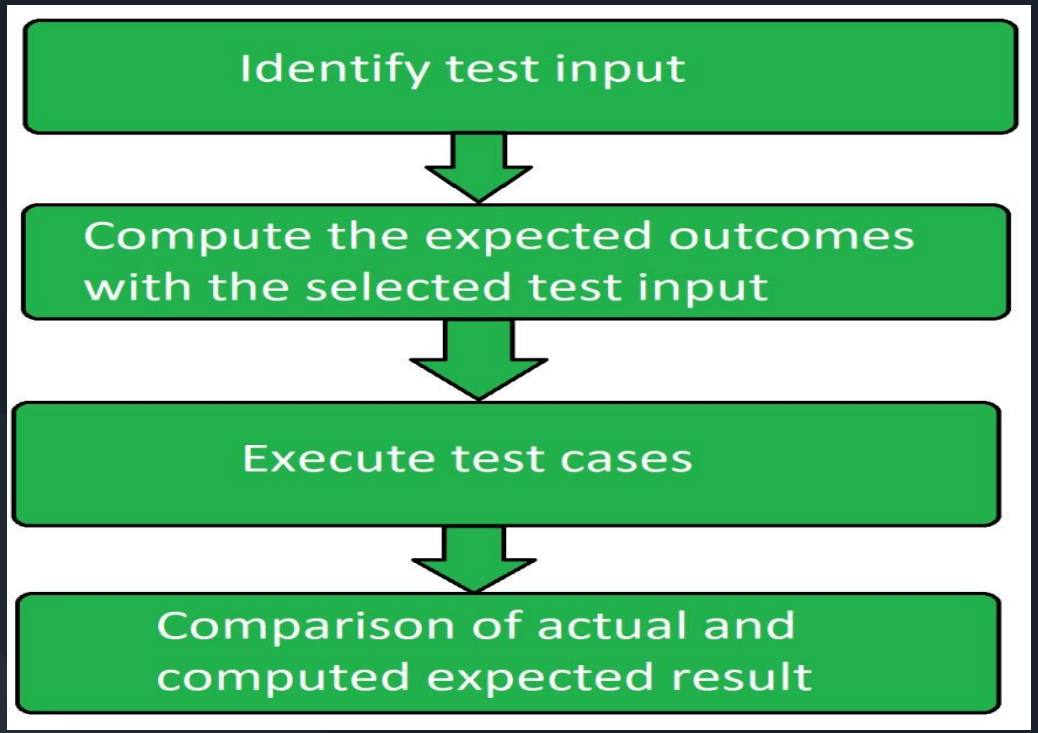
Goal

to check the functionalities. It concentrates on:

- Basic Usability: to check whether the user can freely navigate through the screens without any difficulty.
- Mainline functions: testing the main feature and functions of the application.
- Accessibility: testing the accessibility of the system for the user.
- Error Conditions: checking whether the appropriate error messages are being displayed or not in case of error conditions.

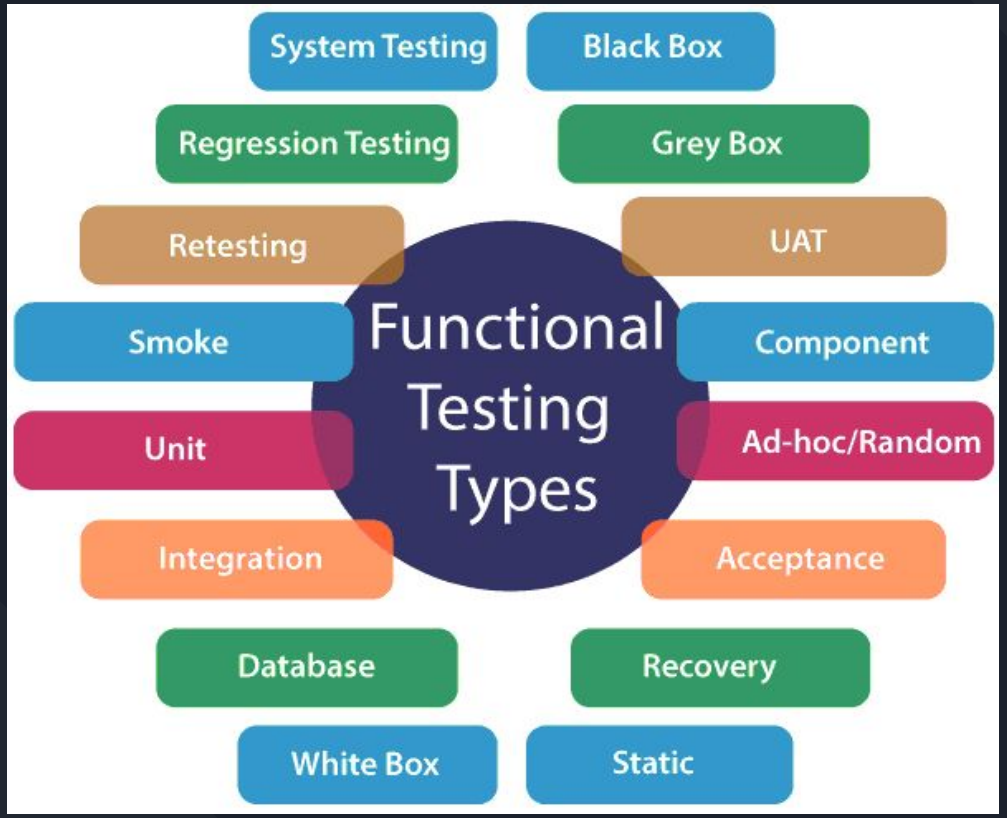
Functional Testing

Functional Testing Process



Functional Testing

Types of functional testing



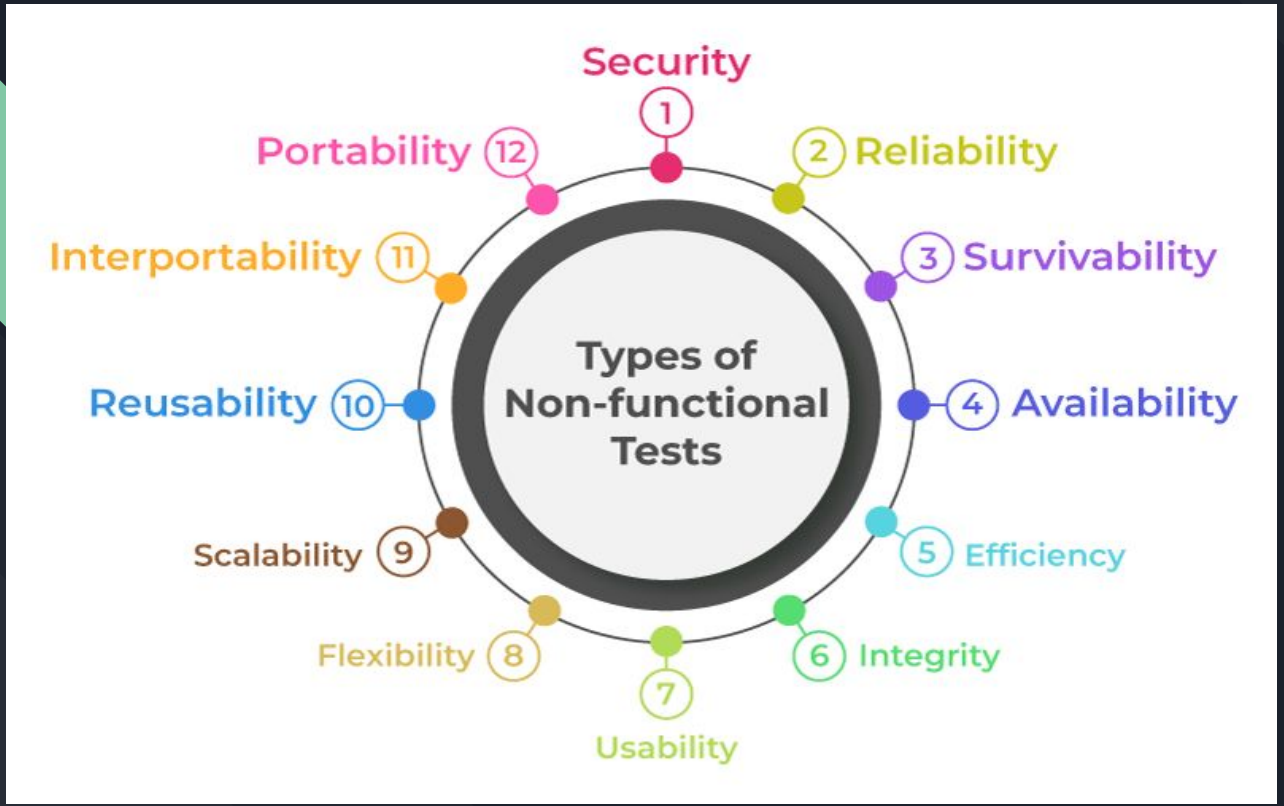
Non-functional Testing Testing of the quality characteristics or non-functional attributes of the system.

Objectives

- **Increased usability**: To increase usability, efficiency, maintainability, and portability of the product.
- **Reduction in production risk**: To help in the reduction of production risk related to non-functional aspects of the product.
- **Cost Reduction**: To help in the reduction of costs related to non-functional aspects of the product.
- **Optimize installation**: To optimize the installation, execution, and monitoring way of the product.
- **Collect metrics**: To collect and produce measurements and metrics for internal research and development.
- **Enhance knowledge of product**: To improve and enhance knowledge of the product behavior and technologies in use.

Non-functional Testing

Non-functional Testing Parameters



Structural testing It is also known as structure-based, glass-box, clear-box, and white box testing.

- A type of software testing that tests the internal code structure of the software.
- It verifies what happens inside the system
- It can occur at any test level

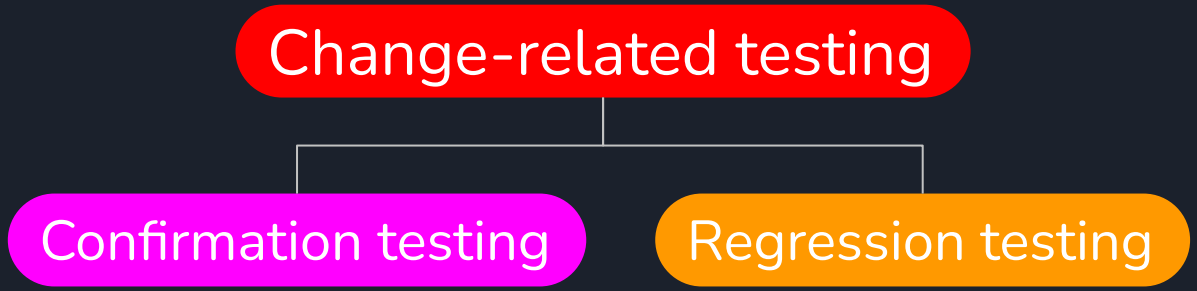
Structural testing

Characteristics of Structural testing

- It requires an understanding of the software's internal code. Hence, members of the developer team can perform who understand how the software was built.
- This testing is focused on how the system performs tasks rather than how users perceive it or how it functions.
- Better coverage as it thoroughly tests the entire code, and defects are quickly fixed.

Change-related testing

2 types: Confirmation testing (re-testing) and Regression testing



Types of changes

- Defect Fix
- Change of environment
- Change of functionality
- New functionally added

Change-related testing Confirmation testing (re-testing)

Confirmation testing is a type of software testing technique used by testers to check if the previously posted bugs are rectified or not in the system or its components.

When to do Confirmation Testing? /1

- ➔ Right after a bug fix: Once a bug has been reported and fixed, confirmation tests should be run to check that the bug has actually been resolved.

Change-related testing Confirmation testing (re-testing)

When to do Confirmation Testing? /2

- Right before regression tests: Regression tests verify that code changes have not affected existing software functionalities. So, for regression to go smoothly, confirmation tests should be run beforehand. That way, the regression tests won't be disrupted by existing bugs.

- Right after bugs have been rejected: If a bug has been reported, and is then rejected by the dev team, these tests must be run to reproduce the bug....and report it again in greater detail.

Change-related testing Regression testing

- Regression Testing is a type of testing in the software development cycle that runs after every change to ensure that the change introduces no unintended breaks.

- We can also say it is nothing but a full or partial selection of already executed test cases that are re-executed to ensure existing functionalities work fine.

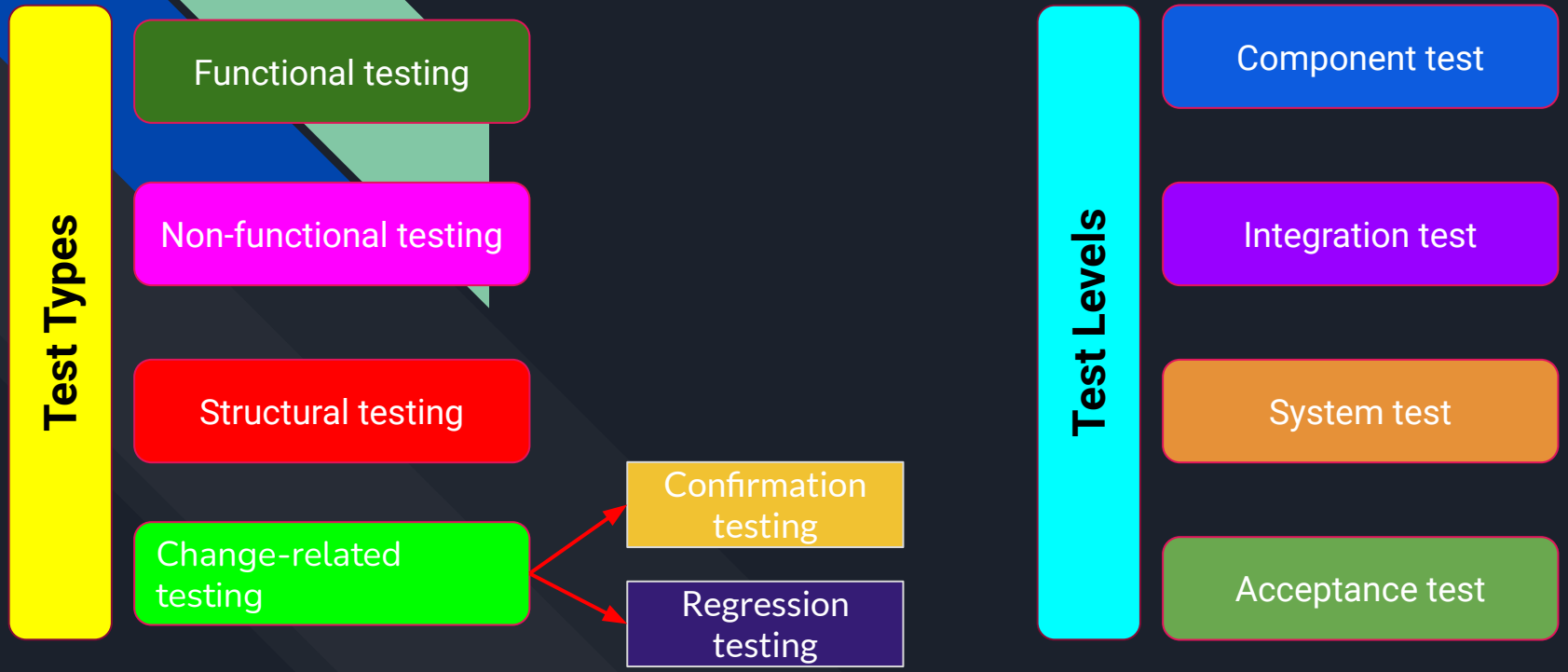
Change-related testing

Regression testing

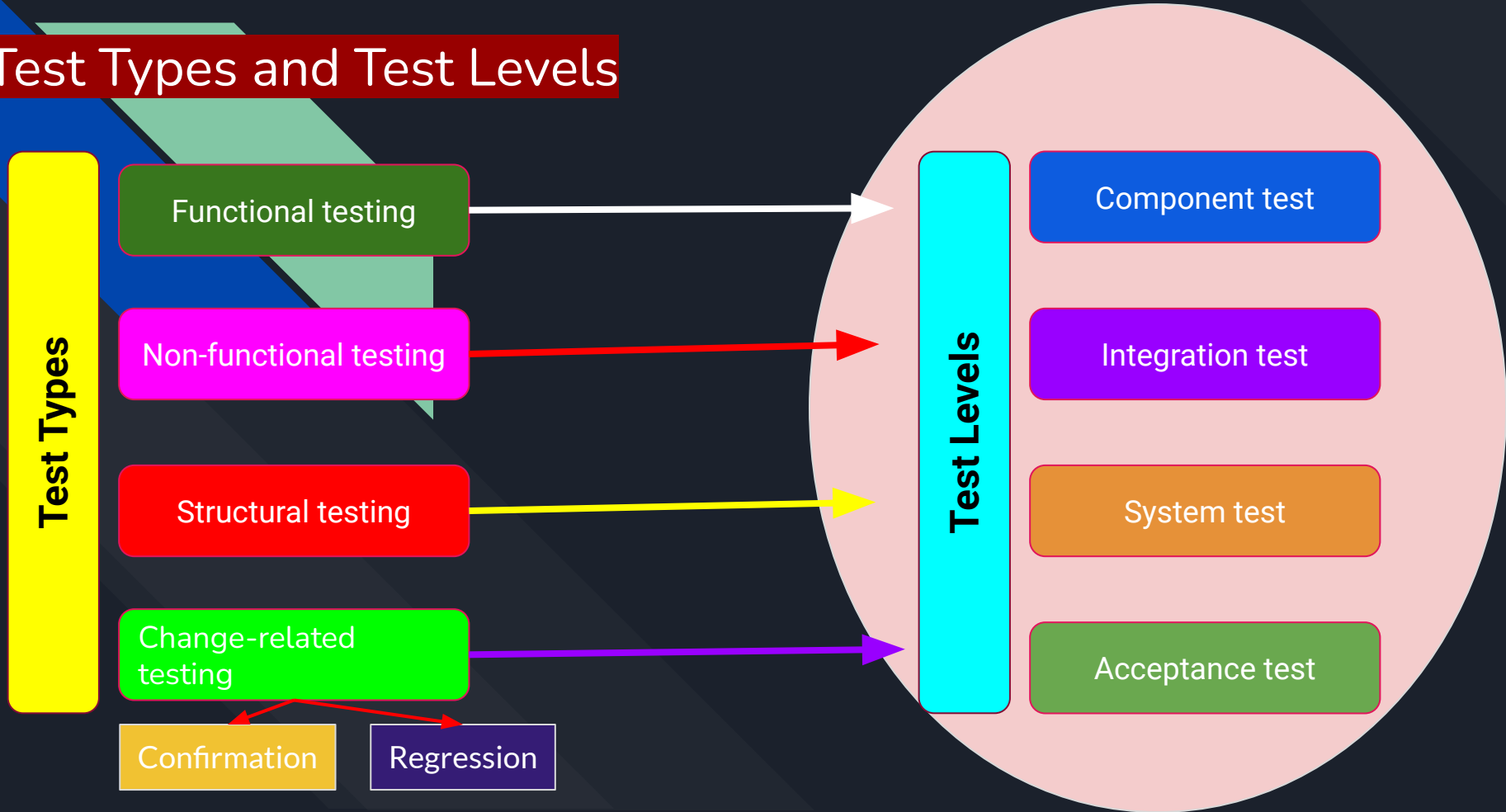
- The purpose of regression testing is to verify that modifications in the software or the environment have not caused unintended adverse side effects and that the system still meets its requirements.

Regression test is the ideal candidate for automation.

Test Types and Test Levels



Test Types and Test Levels



Test Types and Test Levels

Functional tests at each test level

Example: banking/financial application

- Component testing: how the component should calculate compound interest.
- Component integration testing: how account information from the user interface is passed to the business logic.
- System testing: how account holders can apply for a line of credit.
- System integration testing: how the system uses an external microservice to check an account holder's credit score.
- Acceptance testing: how the banker handles approving or declining a credit application.

Test Types and Test Levels

Non-functional tests at each test level

Example: banking/financial application

- Component testing: the time or number of CPU cycles to perform a complex interest calculation.
- Component integration testing: checking for buffer overflow (a security flaw) from data passed from the user interface to the business logic.
- System testing: portability tests on whether the presentation layer works on supported browsers and mobile devices.
- System integration testing: reliability tests to evaluate robustness if the credit score microservice does not respond. score microservice does not respond.
- Acceptance testing: usability tests for accessibility of the banker's credit processing interface for people with disabilities.

Test Types and Test Levels

White-box tests at each test level

Example: banking/financial application

- Component testing: achieve 100% statement and decision coverage for all components performing financial calculations.
- Component integration testing: coverage of how each screen in the browser interface passes data to the next screen and to the business logic.
- System testing: coverage of sequences of web pages that can occur during a credit line application.
- System integration testing: coverage of all possible inquiry types sent to the credit score microservice.
- Acceptance testing: coverage of all supported financial data file structures and value ranges for bank-to-bank transfers.

Test Types and Test Levels

Change-related tests at each test level

Example: banking/financial application

- Component testing: automated regression tests for each component are included in the continuous integration framework and pipeline.
- Component integration testing: confirmation tests for interface-related defects are activated as the fixes are checked into the code repository.
- System testing: all tests for a given workflow are re-executed if any screen changes.
- System integration testing: as part of continuous deployment of the credit scoring microservice, automated tests of the interactions of the application with the microservice are re-executed.
- Acceptance testing: all previously failed tests are re-executed after defects found in acceptance testing are fixed.

Maintenance Testing

Once deployed, a system is often in service for years or even decades. During this time the system and its operational environment is often corrected, changed or extended. Testing that is executed during this life cycle phase is called 'maintenance testing'.

Quick question: Are Maintenance testing and Maintainability testing the same?

Ans: No, Maintenance testing is different from Maintainability testing.
Maintainability testing defines how easy it is to maintain the system.

Maintenance Testing

The scope of maintenance testing depends on several factors, which influence the test types and test levels. The factors are:

- Degree of risk of the change, for example a self-contained change is a lower risk than a change to a part of the system that communicates with other systems.
- The size of the existing system, for example a small system would need less regression testing than a larger system.
- The size of the change, which affects the amount of testing of the changes that would be needed. The amount of regression testing is more related to the size of the system than the size of the change.

Triggers for Maintenance

Maintenance testing is done on an existing operational system. There are 3 possible triggers for maintenance testing:

- ➔ Modifications
- ➔ Migration
- ➔ Retirement

Triggers for Maintenance

- Modifications
  - planned enhancement changes (for example release-based)
  - corrective and emergency changes
  - changes of environment, such as planned operating system or database upgrades, or patches to newly exposed or discovered vulnerabilities of the operating system and upgrades of COTS software.
  - Modifications may also be of hardware or devices

Triggers for Maintenance

- Migration
  - from one platform to another
  - operational testing of the new environment, as well as the changed software
  - migrating your own files and applications

Triggers for Maintenance

- Retirement
  - The testing of data migration or archiving, if long data-retention periods are required.
  - Testing of restore or retrieve procedures after archiving may also be needed

Impact Analysis for Maintenance and Regression testing

Usually maintenance testing will consist of 2 parts:

1. testing the changes
2. regression tests to show that the rest of the system has not been affected by the maintenance work.

maintenance testing includes

- extensive regression testing to parts of the system that have not been changed
- impact analysis - a major and important activity within maintenance testing

Impact Analysis for Maintenance and Regression testing

There are a number of factors that make impact analysis more difficult:

- Specifications are out of date or missing (for example business requirements, user stories, architecture diagrams).
- Test cases are not documented or are out of date.
- Bi-directional traceability between tests and the test basis has not been maintained.
- Tool support is weak or non-existent.
- The people involved do not have domain and/or system knowledge.
- The maintainability of the software has not been taken into enough consideration during development.

Impact analysis can be very useful in making maintenance testing more efficient, but if it is not, or cannot be, done well, then the risks of making the change are greatly increased.


thank you so much