ISTQB Foundation Testing throught the Software Development Lifecycle

A picture containing logo

Description automatically generated

Overview:

* Lifecyclye Models (SDLC)
* Test levels – a stack or level of modules to be tested
* Test types – functional, non-functional
* Maintenance testing

SDLC

* All software teams use them

Text

Description automatically generated

For a complex project with many people, you need rules

SDLC is necessary for:

Graphical user interface, text, application, email

Description automatically generated

Not all SDLC models provide the same benefits

The official SDLC definition:

Text, letter

Description automatically generated

When things get done is one of the main differences between SDLC models

There are 2 broad sdlc models: Sequential and Iterative

Graphical user interface, application

Description automatically generated

Al,l models have the same basic parts

Chart, sunburst chart

Description automatically generated

Any of these phases should begin after the last one finishes. This is the case in the waterfall model.

Sequential Models

Waterfall model

Diagram

Description automatically generated

In this model business analysists do all of the analysis, then developers and architechs design, sode, implement features, give it to testers, developpers fix them and give the next version, then maintenance as needed. As some things such as testing a and forth this model also has ssome iterative elements, but mostly sequential.

Waterfall advantage:

* Simple, easy to understand

Disadvantages:

* High cost
* Slow and inefficient waiting for others to finish
  + While developers are working testers are idle, etc.
* Any phase can become a bottleneck
* Very late feedback – an early mistake will take a long time for feedback
* Additional pressure on testers – since testers are near the end of the process they will be the most under preasure when it comes to meeting deadlines and will be held responsible for mistakes
* Testing at the end
* +++++++++, one big block to test

The V Model

* Integrates the test model throughout the development process for early testing
* It shows the test levels associated with each development phase
* Examples of test levels include component and integration testing
* Has the same sequence of development, but each layer is tested not just once all together as a whole

V-Model pros

* Includes early testing for each element in the development process

Diagram

Description automatically generated

Iterative Models

* You still do the same activities, but in small chunks
* In the original model you would design the whole thing, develop the whole thing and test the whole thing. In iterative you pass elements of the products one at a time through the cycle, this makes it more responsive
* You develop one or two features at a time instead of ten, leaves less wasted time so each person can complete their role for each piece
* Work is broken up by features or (more often) by time segments – the team estimates how long it will take to do a thing and allocates time for each feature accordingly
* The most popular models:
  + Rational Unified Process
  + Spiral
  + Agile
    - Includes Scrum and Kanban

Iterative model pros:

* Less wated time as everyone is working on something
* Less chance of bottlenecks
* More adaptable, less preasure on testers
* May deliver something usable very fast, but incomplete – more patches

RUP

* Long iterations (2-3 months)
* Delivers groupd of interrelated features

Spiral

* Experimental iteration length – can be changed or abandoned later
* Most flexible, may use other models

Scrum

* Short iterations (2 weeks, can be less)
* Less done per iteration
* Assigned roles -product owner, scrum master, etc.

Kanban

* Short iterations
* Fluid roles, flat hierarchy
* Tasks are shared by everyone, no speacialists
* No one coordinator
* Timelines evolve on an as needed basis

Further Notes:

Models are a guideline for how work proceeds, this does not always match reality

Iterative models are NOT always better than sequential – vcontext matters, a robust sequential model may be good

A combination of models may be used within the same organisation eg. You may use a V Model for the backend and a scrum for the front end

Summary:

SDLC has many models

2 Broad categories: Sequential, Iterative

VModel promotes one of the 7 principals: Test early

In iterative models you have the same activities, but in timeboxed periods to deliver a subset of the whole

Module 3.

Test levels

* Software is often broken down into multiple layers, within which are modules within which are files with the code
* You create jigsaw pieces and then assemble them so it all works
* With test levels you can test big and small – test the individual pieces, then how they interact with eachother – first two, then three, then all
* There are four test levels:
* Graphical user interface, text, application

  Description automatically generated
* Each level is associated with approaches and objectives and defects
* You MUST know the definitions for the exam

Component testing

* Software is built from multiple components with multiple layers
* You identify the smallest possible unit and test it in isolation, do that for each component

Integration testing

* You test how the units work together, you do this incrementally, testing how two units work to gether than three, then four and so on

System testing

* You do this when everything or almost everything is in place, testing the software from start to finish from the outside usually with an interface

Acceptance testing

* When you give the software to the customer, client or representative and they work with it and let you know if it meets their requirements

Common Objectives:

* Reduce risk
* Verifying functional and non-functional behaviours, build confidence in the system
* Find defects
* In integration and component testing: prevent defects from escaping to higher levels – faster and easier

A screenshot of a computer

Description automatically generated with medium confidence

Difficulty: How small does something have to be to be a unit?

Anything you can write a unit test for?

You may have many functions in one file, so you can call the whole file a unit, or everyt5hing that performs a single function

Testers tend to have bigger units since they see the whole more than the parts

Test basis

For developers: code, tech design, datamodel – static testing or dynamic (automated unit tests)

For Testers: Component spect, the formal requirements, how a thing should happen

Integration testing

* However many units you have if you test how they interact it is an integration test
* If multiple layers then integration test
* Testing if two or more units work together regardless of how you define unit

Two Sub-categories for integration testing

* Component integration – you have multiple components test that they integrate without issue
* System Integration – Focuses on integration between systems, say between the internal system of one bank and the internal system of another
* Given the broad def, test objects can be a numer of things
* Graphical user interface, text, application

  Description automatically generated
* You can use the documentation for any of these test objects as a test basis for your scenarios
* Text

  Description automatically generated
* How much testing is done by testers and developers depends on what the project requires – if tech skills heavy devs will do more

System Testing

* Full integration testing of all modules
* Focvuses on the behaviour of the system as a whole
* Your starting point should be from outside the system not inside like with component testing. You should interact with the system only through the interfaces as though you know nothing about the internals
* You should pay attention to the internal flows. Say with a banking app you are transferring savings, that is one flow. If you are getting a mortgage, that is another flow and so on. Some flows may include combinations of flows.
* You need to cover as many flows as possible in the time allowed

Acceptance testing

* Objective is to validate complete working software
* Focus on who does the testing and why
* A means for customers to approve delivery
* Part of closing a deal between customer and supplier – MUST have clear requirements
* Many companies hire independent testers to carry out acceptance testing and find the defects before accepting it – this is called  **User Acceptance Testing (UAT) – Can users meet their needs with this?**
* **Operational Acceptance Testing (OAT)** - focuses on necessary operations like:
* Graphical user interface, text

  Description automatically generated
* These things are often not taken into consideration until needed and can then cause issues

Acceptance testing sub-categories:

* + Alpha testing
    - Happens at the developer’s site
  + Beta Testing
    - Happens at the site of the customer, can be a software granted to a small group of selected users for testing
    - The main point is to use the infrastructure (hardware and software) of the end users (to avoid ‘it works on my machine’, you can’t predict everything)
    - Often done for commercial off-the-shelf software (COTS)
  + Both carried out by independent testers or potential customers, not the testers of the development team

Summary:

* Multiple test levels allow you to test different aspects of a software indifferent ways and it is faster and easier to fix problems at lower levels
* Component testing: To test units in isolation, usually done by devs
* Unit/Component/Module testing all mean the same thing (for the ISTQB)
* Integration testing: Tests multiple units across layers and units, checks the behaviour between units, not the units themselves
* System testing: Similar to integration testing, most pieces already integrate, should be tesxted as a whole from ‘outside the box’, covers all possible paths
* Acceptance – verifies the whole system works correctly and meets the needs of the customer

Module 4.

Comparing Test Types

Overview:

* Graphical user interface, text

  Description automatically generated
* Two categories:
  + Functional vs. Non-functional
* Black vs, white box testing
* Change related testing
* Test types and test levels

Test types

* Does this feature work correctly and as expected?
* Functional:
  + Unit testing
  + Integration
  + System
  + User Interface
  + Here Acceptance testing is considered to include system and UI testing

Functional Coverage:

* Graphical user interface, text

  Description automatically generated
* Allows you to quantify the testing process and express your level of confidence in your system
* So if you had a system doing 10 things and you have thoroughly tested and fixed 8 of them then you would have 80% test coverage
* The caveat is that this is expressed as a percentage of the number of tests that you have manages to come up with, not the number of features you have tested. So the number of tests you come up with per feature and how many of those tests were passed – high test coverage does NOT guarantee quality since it depends on how good you are at coming up with tests
* There are techniques for measuring and defining the coverage of code:
  + Statement coverage
  + Decision coverage
* You write automated tests and a tool tells you which part of the code was run and what wasn’t so that you can see what still needs testing
* FMI: guru99.com/code-coverage.html

Coverage Gap:

* What you don’t have tests for yet, the percentage of uncovered code

Non-functional Testing

* Testing the non-functional aspects of the code eg. Is it slow? Easy to use? Hackable?
* Answers:
* Text

  Description automatically generated
* Types of non-functional testing include:
  + Usability
  + Performance
  + Security

Usability

* Such as: Alpha and beta testing – where real users give feedback on what could be improved

Performance testing

* Usually a specialist area
* Broad, includes: stress testing, load testing, endurance testing, scalability

Security Testing

* Penetration testing/ethical hacking/white-hat hacking

All these are potentially separate career paths so you don’t have to cover the whole thing

Black Box vs. Whitebox Testing

Black box

* You don’t look at the internals of the software or the modules
* Relies on the requirements for how the box should behave
* You send input, check output
* Might need to test in different environments
* For example: Might need to time how long to get the results back – performance test
* You can test integration and system testing in this way, adding elements and see how it affects the user experience

White box

* You know what is inside the box and use that knowledge
* Usually carried out by devs
* Also uses requirements, but also the tech spec
* Does the same thing, but on a more granular level

Change Related Testing

* Testing in response to change
* Say a bug happens and the devs fix it, the testers then do two types of tests:
  + Confirmation Testing
    - Was the bug really fixed?
    - All failed tests should be re-run, this focuses *only* on the area which was broken
  + Regression Testing
    - Did the fix or a change break anything else?
    - A fix in one place can break something somewhere else – this is known as a **regression** and errors of this type are called **regression bugs**
    - Text

      Description automatically generated with medium confidence
    - Tests the rest of the system to ensure it still works as intended with no unintended changes
    - This can be difficult with hundreds of thousands of tests, especially if it is a small bug – this is where test automation comes in, it prevents testing becoming a bottleneck although it is a serious investment
    - In the absence of automated testing you prioritise the more critical parts of the system

Change Types:

* There are many types of change which would cause you to re-run tests, for example:
  + Graphical user interface, text, application

    Description automatically generated
  + Any change to the app ‘ecosystem’

Further Notes:

A test can be a combination of multiple types of test, for example, it can be *both* a confirmation test and a functional test

You describe your test according to the main focus, so if you are mostly focused on behaviour you say ‘black-box’ and if you are making sure it is not still broken ‘confirmation testing’

Even though it is the same action, it tells people what your focus is

**Important:** You can do any test type at any test level

Table

Description automatically generated

Blakc box testing tend to be carried out at the higher levels, white box testing is mostly at the unit level, integration is in between, so often called ‘Grey-Box Testing’

Summary:

* Test Levels
  + Unit/module tesing
* Integration testing
* System testing
  + Includes End to End or Front to Back tests
* Acceptance testing
* Functional vs. Non-Functional testing
* White box vs. Black Box testing
* Regression and confirmation testing – can happen at every level

Maintenance testing

* Covers the maintenance phase of the lifecycle
* Most projects need maintenance
* Systems MUST be maintained because change is inevitable

Overview

* What kind of maintenance does software require
* What is the role of impact testing?

Two types of change: Planned and Unplanned

Unplanned maintenance

* More difficult to deal with i.e. – bugs in production, can be functional or non-functional
* Requires a ‘hotfix’ – means that time and resources must be reallocated fro the issue, also leaves potential security vulnerabilities

Planned Changes

* Software enhancements
* Operational and environmental upgrades – lack of supported software -ie. If you start using SQL2021 database you know you will need to schedule to move all the data in 10 years when it stops being supported, lots of testing
* Retirement – it may be deemed not fit for purpose after 20 years, it’s ‘shelf life’, but the functionality is still needed so you might need to re-write with newer tech

Impact Analysis

* Text

  Description automatically generated
* You should ask questions such as:
* Graphical user interface, text

  Description automatically generated
* In order to identify the scope of the change and allocate time and resourses
* You may not be able to run your old tests, you may need to modify them or carry out new ones
* Then confirmation and regression changes are needed
* You should analyse the situation and prioritise accordingly:
  + What has changed? What is effected or likely to be effected?
* Challenges of Impact Analysis:
  + Outdated specifications – treat with continuous discussion with experienced people
  + Lack of knowledge – may result from those who really know the system leaving
  + Outdated test cases with no traceability betw tests and requirements – the tests may not have been changed with everything else
  + Weak or non-existent tool support

Summary for Course 2:

* SDLC
* Multiple models exist – Sequential and Iterative
* Most applications are layered – interconnected network of models
* Levels: unit, integration, system, acceptance
* Types: Functional, Non-functional, black box, white box, etc.
* Testing is a continuous process (maintenance testing)