

▼ Week 5: Testing Health Information Systems

Overview

In week 5, you will learn about key testing principles. We will also briefly touch upon automated testing as an alternative to manual testing

Learning outcomes

Completing week 5 should help you work toward achieving the following learning outcomes:

- L03: **Understanding the purpose of system testing and being aware of the different types of testing**
- L08: **Critically appraising the proposed design of a system and assessing whether it is fit for purpose**

How this week relates to your assignment

- Your assignment should include a **system implementation plan**. It needs to clarify **how you are going to assess whether the system is fit for purpose and ready to be implemented**.
- This week will help you to understand the **types of functional and non-functional tests** you could consider to include in your **overall test plan** and what they may entail. In addition, you could include a more detailed test plan for 1-2 scenarios that relate to key system features. During the face-to-face days, you will gain some hands-on experience with developing and executing a test plan.

Materials and activities

[5.1 Testing: Key principles, modes and methods](#)

[5.2 Interview with Amanda Reilly](#)

[5.3 Quiz](#)

▼ 5.1 Testing: key principles, modes and methods

The first bug in computer testing was discovered in 1947 by a woman called Grace Hopper. She was testing the Harvard Mark II computer and 'debugged' it by removing a moth that was stuck between the computer's electromagnetic relays. Nowadays, 'debugging' is no longer about finding insects in the physical machine, but about **tracing and correcting errors in software**

applications. Also, **testing of information systems has become about much more than just identifying and correcting bugs.**

Types of testing

There are many types of testing; some estimates say there are more than a hundred (and counting). Instead of being familiar with them all, the key thing you should be aware of is that they can be broadly grouped into three main categories:

- 1. Functional testing** – includes tests that verify the system against the functional requirements as specified by the user. Functional tests describe what the system does, both in isolation and in interaction with other systems and the context it is deployed in. Examples of functional tests are unit tests, integration tests and user acceptance tests; these will be explained in more detail in the video tutorials (see video below).
- 2. Non-functional testing** – non-functional tests focus on the performance, reliability, scalability and other non-functional aspects of the system. They describe how well the system works and are usually performed after the functional tests have been completed. Examples of non-functional tests are: performance tests (does the system perform well under the anticipated workload); load tests (how many people can simultaneously log into the system); penetration (or pen) tests (are there any security weaknesses in the system that make it vulnerable for unauthorised access, such as in cyber attacks); usability testing (is the system easy to use, e.g. is navigating the system intuitive).
- 3. Maintenance testing** – maintenance tests happen after a system has been deployed, and are triggered by modification or migration of the system. A regression test is a common example, where the aim is to confirm that system functionalities are unaffected by changes in, for example, the system itself (e.g. the programming code) or in other systems it interacts with (e.g. an upgrade of the operating system it runs on). You can do a regression test by re-executing (a selection of) existing test cases.

Manual versus automated testing

From what you have learned about testing so far, you have probably already guessed that it takes a lot of time and people to do it properly. If done manually, a test requires a human to type in test data, execute the test case, verify the results and then generate a test report. But often, tests could be automated; especially simple, repetitive tasks and non-functional and maintenance tests (e.g. performance tests, regression tests). However, not all tests lend themselves to automation, such as user acceptance tests or usability tests.

For automated testing, you need to develop the scripts that execute the tests. This requires an initial time investment from people with the right skills. However, once developed, the scripts can run tests automatically and unsupervised. Please keep in mind that there is some effort involved to make sure the scripts stay aligned with changes in the system.

Some benefits of automated testing are:

- Frees up skilled human testers and developers
- Can be run outside work hours without supervision

- Less prone to human error
- Generally faster to run an entire set of test cases (i.e. a test suite), often with better coverage
- No detailed knowledge required of the test suite to run the automated test
- Generates detailed test reports and documents early, quickly and often
- Automated regression tests provide a safety net.

▼ Software Testing Tutorials for Beginners

Now watch the six tutorials in the video below (18:20). You can stop the video after the sixth tutorial, and we apologise for the robotic voice-over in tutorials 4 and 5.

The video will talk about why testing is important; the seven principles of testing; the role of testing within the system development life cycle; and about different types of functional testing.

<https://www.youtube.com/watch?v=goaZTAzsLMk&t=13s>

What is software testing?

- It is a process used to identify the correctness, completeness and quality of developed computer software.
- It includes a set of activities conducted with the intent of finding errors in software so that it could be corrected before the product is released to the end users.
- Software bugs can be expensive and/or dangerous

Seven software testing principles

1. **Exhaustive testing is not possible:** if you were to test all the possible combinations of situations, project execution time and costs will rise exponentially. We need the **optimal amount of testing** based on the **Risk assessment** of the application.
2. **Defect clustering:** a **small number of modules** contain **most of the defects detected**. By experience, you can identify such risky modules. However, this approach leads to the "Pesticide paradox".
3. **Pesticide paradox:** if the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs. To overcome this, the **test cases need to be regularly reviewed and revised**, adding new and different test cases to help find more defects.
4. **Testing shows the presence of defects:** you can never claim a software product is completely bug free. Testing reduces the probability of undiscovered defects remaining in the software, but even if no defects are found, it is not a proof of software correctness or that no bugs remain in the system.
5. **Absence of errors is a fallacy:** the software could be 99% bug-free but still not meet the needs/requirements of the client.
6. **Early testing:** testing should start as early as possible in the Software Development Life Cycle. In this way, any defects in the requirements/design phase are captured as well.

7. **Testing is context dependent:** different types of software will need different types of testing.

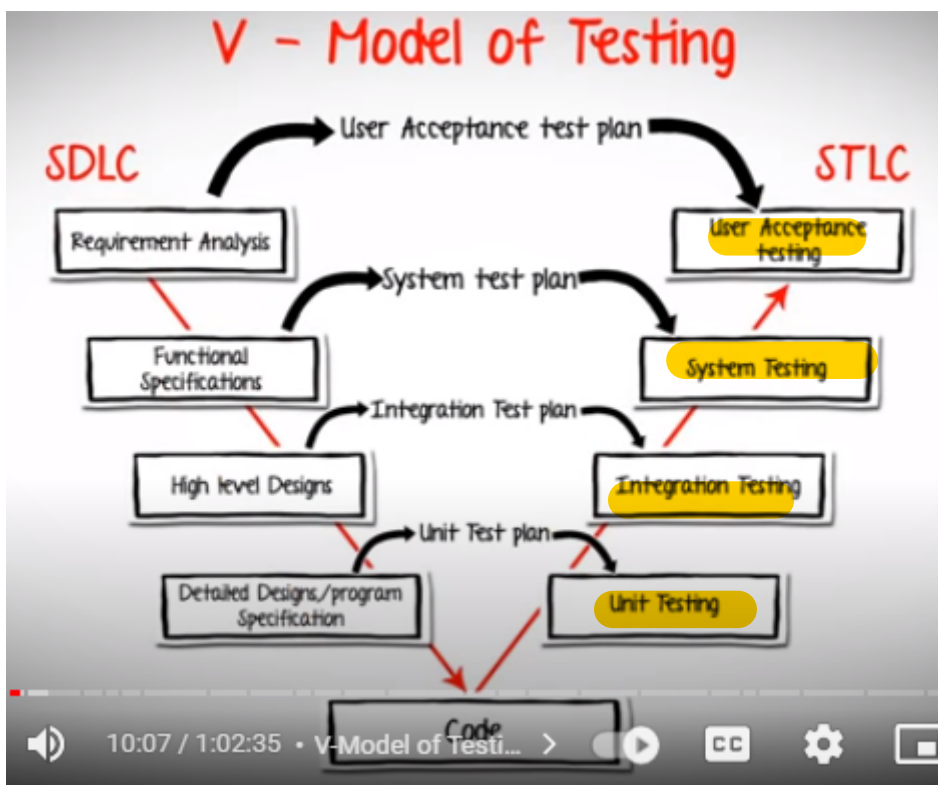
Software testing: Software Development Life Cycle (SDLC) vs Software Testing Life Cycle (STLC)

SDLC Waterfall Method

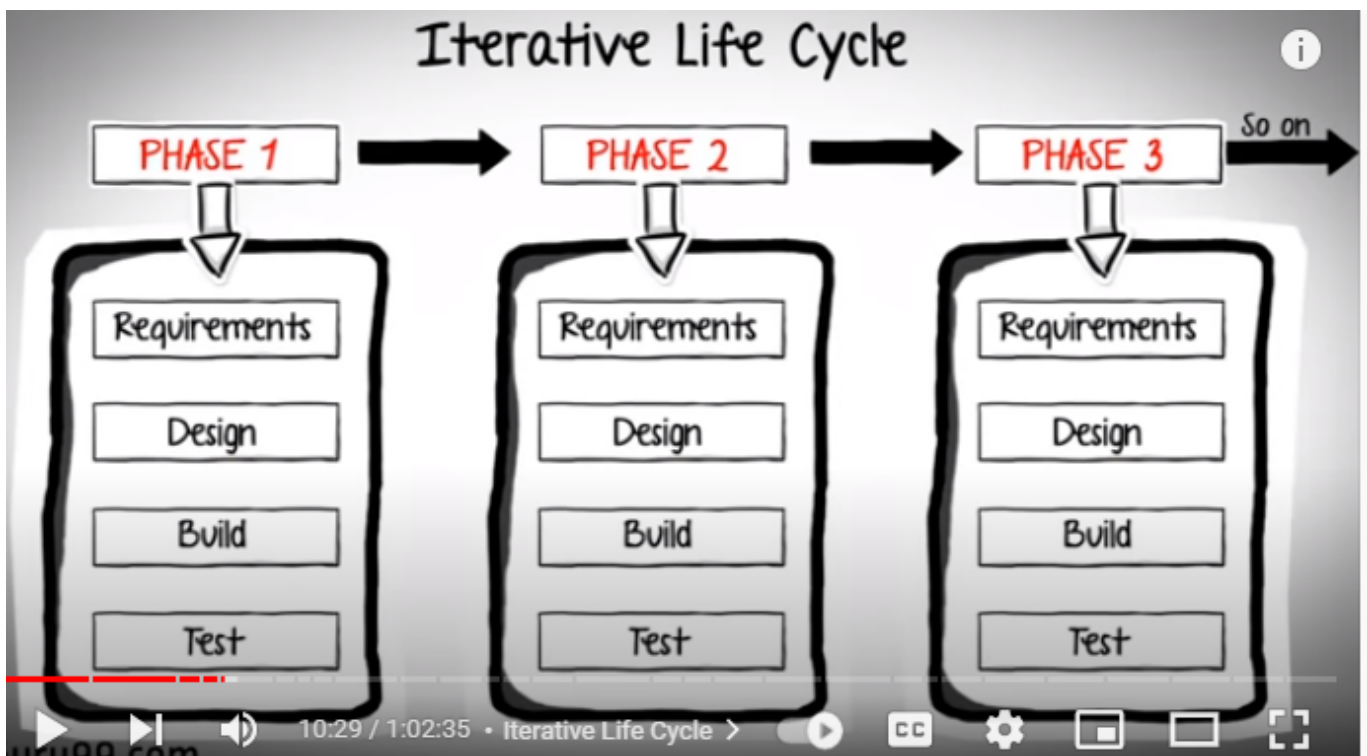
1. **Requirements stage:** gather as much information as possible about the details and specifications of the desired software from the client.
2. **Design stage:** plan the programming languages (e.g. JAVA, .php, .net) and database (e.g. Oracle, MySQL, etc) that would be suitable for the project.
3. **Build stage:** actually code the software.
4. **Test stage:** Test the software to verify that it is built as per the specifications given by the client.
5. **Maintainance**

In this approach to software development, testing is carried out only at stage 4. However, if there are mistakes in the previous phases (e.g. Requirements, Design/Architecture), errors will be present in the end product...However, the earlier in the life cycle an error is detected, the cheaper it is to fix it.

To address this problem, the **V-Model of Testing** for software development was created: **for every phase in the development life cycle, there is a corresponding testing phase.**



Apart from the V-Model, there are also **Iterative Life Cycle** development models (e.g. **Rapid Application Development** and **Agile Development**): the development is carried out in phases, with each phase adding some functionality to the software. Each phase has its own set of development and testing activities:



Testing in Life-Cycle Models:

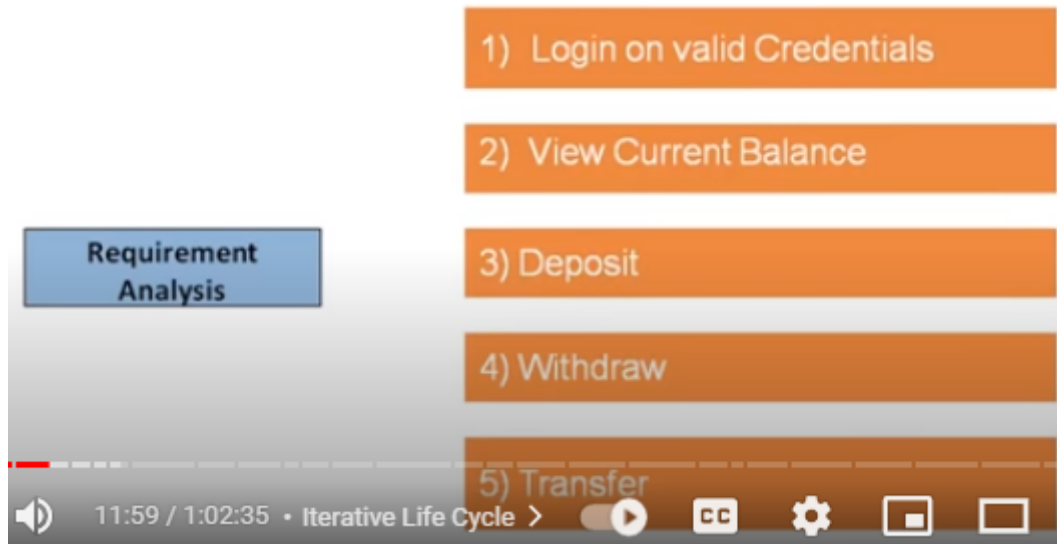
- There are **numerous development life-cycle models**
- The development model selected for a project depends on the **aims and goals of that project.**
- **Testing** is *not* a stand-alone activity and it has to **adapt with the development model** chosen for the project.

- In any model, **testing** should be performed at **all levels** (from requirements phase up to maintenance).

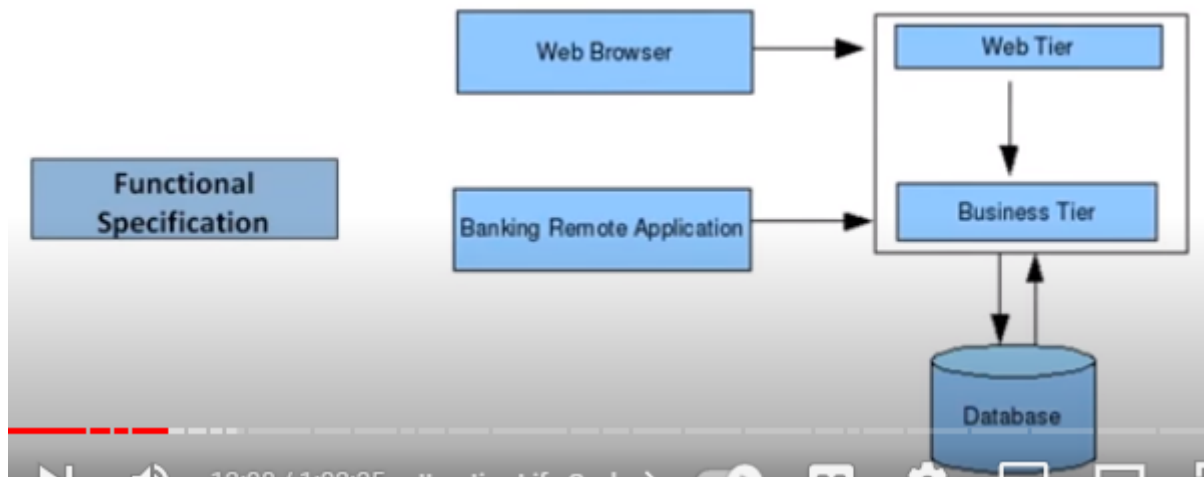
▼ What is Unit Testing?

In the SDLC branch of the V-Model, we have 5 stages (example of banking app):

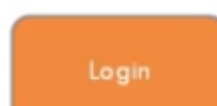
1. Requirement analysis: the client decides the functionalities they want into a system:

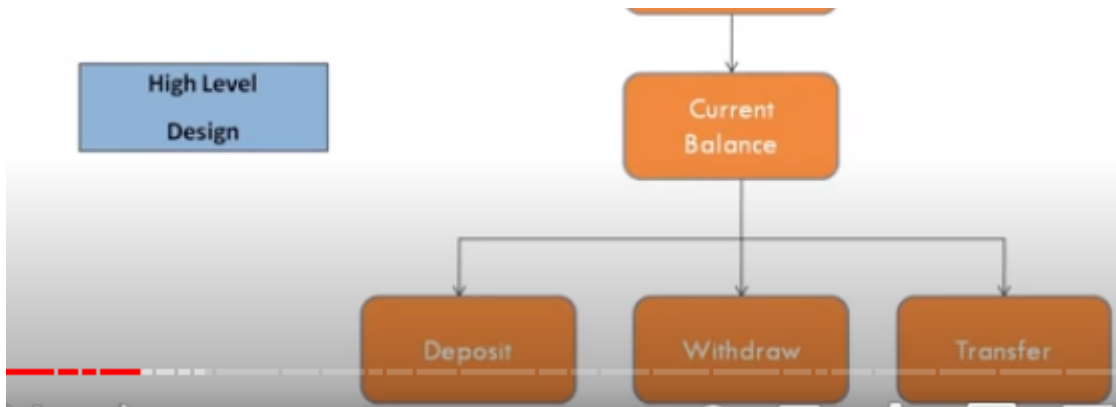


2. Functional specification: architecture, database and operating environment design are finalised.

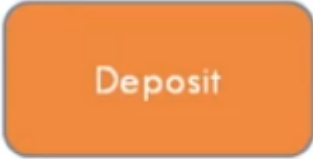


3. High-level design: the application is broken down in **modules** and **programmes**.





4. **Detail design:** pseudocode for **functions** for **each module** is documented.



Detail Design

Function depositMoney(int amount) {

// Only Declarations of the functions

//No Actual Code

//This helps in maintaining uniformity across the project and avoid errors

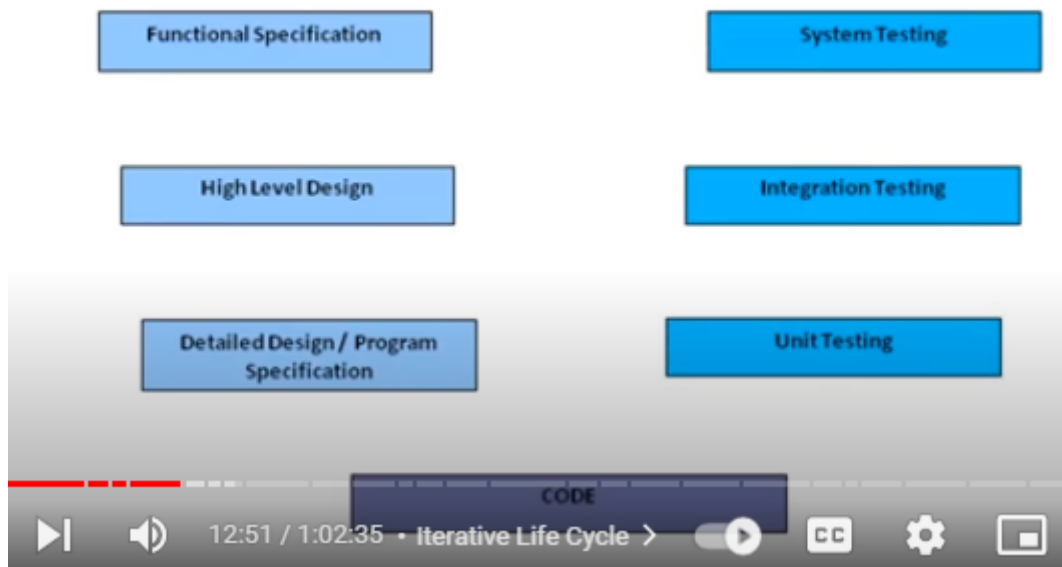
}

5. Coding phase

During all these software development phases, the tester is doing **corresponding testing activities**, according to the test phase that is currently getting implemented:

Requirement Analysis

User Acceptance Testing



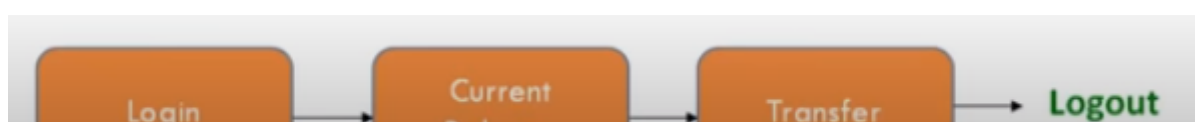
Unit Testing

- It is also called **component testing**
- It is performed on a standalone module to check whether it has been developed correctly
- Unit testing should be done by developers, although in reality they do not have time/capacity to do that and prefer someone else to do it for them (so they ask an independent tester to do that)



System testing

- It is concerned with the **behaviour of a system as a whole**
- Unlike Integration Testing, which focuses on data transfer amongst modules, System Testing checks **complete end-to-end scenarios**, as a way a customer would use the system.
- Apart from **functional, non-functional requirements** are also checked (e.g. **performance** and **reliability**)



▼ Acceptance Testing

- It is usually done at client location by the client after the other phases of testing have fixed other defects.
- Its focus is not to find defects, but to **check whether the system meets the requirements**.
- Acceptance testing can be done in 2 ways:
 - **Alpha Testing**: a small set of employees of the client would use the system as the end-user would
 - **Beta Testing**: a small set of customers would use the system directly

Integration Testing

- Individual modules are **combined** and **tested as a group**.
- **Data transfer between modules** is tested thoroughly.
- Integration testing is carried out by testers.
- Different types:
 - **Big-Bang Integration Testing**:
 - Wait for all modules to be developed before testing
 - Time-consuming
 - Difficult to trace the root cause of defects
 - **Incremental Integration Testing**:
 - Models are tested as and when they become available. Higher-level modules can be tested first (**top-down approach**: creation of **stubs** for not-yet ready lower-level modules) or lower-level ones (**bottom-up approach**: creation of **drivers** for not-yet ready higher-level modules)

Smoke/Sanity Testing

- To **check critical functionalities** of a system before it is accepted for major testing
- Sanity Testing is quick and not exhaustive
- Its goal is not to find defects but to check system health.

Regression/Maintenance Testing

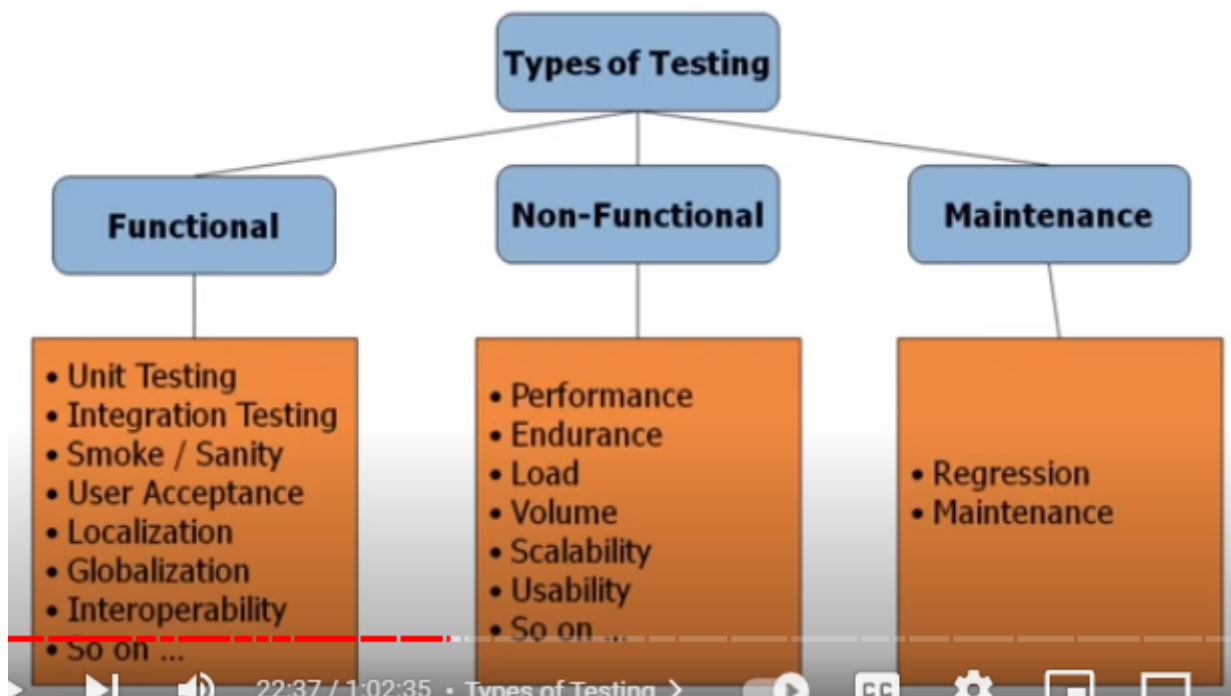
- Once a system has been deployed, testing any of its enhancements, changes or corrections forms part of Maintenance Testing
- It is carried out to **check that modification in software has not caused unintended adverse side effects**.

Non-Functional Testing

- Apart from Functional Testing, non-functional requirements like **performance, usability** and **load factor** are also important.

- **Performance Testing** is carried out to **check and fine-tune system response time**, with the goal of reducing it at an acceptable level.
- **Load Testing: system performance** can be compromised at different processing/requests loads, i.e. number of people accessing the system.

Types of Testing



[Back to summary.](#)

5.2 Interview with Amanda Reilly

We have interviewed Amanda Reilly, who is the end-to-end test manager for IT Testing Services at the University of Manchester. In the interview, Amanda explains the types of tests her team use, how they develop and execute test plans, and how they communicate and work with stakeholders and manage their expectations. Watching the interview should give you additional insight into how system testing works in practice.

Information in an end-of-test report

- What has been tested and in which environment (i.e. coverage)
- How things have been tested
- Overviews of the number, type and source (e.g. code, missed requirements) of identified defects
- Detailed testing schedule and testing observations
- Agreed 'known issues' that will taken across to the next phase of testing (User Acceptance Testing)

How to get most of the Testing Team

- Engage testers early on (at project inception if possible)
- Have a good project plan that includes detailed test activities and milestones
- Support managing the test activities in line with the project plan and assist the Test Team with addressing defects if needed

[Back to summary.](#)

▼ 5.3 Quiz

1. What is the general purpose of testing a system?

- a. Identify bugs in the software
- b. Check whether the system meets the users' requirements
- c. Assess the system's response times

d. Can be any of the above

That's correct. Different types of testing have different purposes depending on the system and the aims of the project, so it can be any of the above.

2. Which of the following testing principles refers to the situation where a user is unhappy with a system, even if tests have shown it to be bug-free?

- b. Absence of errors fallacy

A system can be bug-free, but still be unusable for the customer. This happens if the system is tested thoroughly, but against the wrong requirements. This situation is referred to as the 'absence-of-errors fallacy', so b was indeed the correct answer.

3. What does the testing principle 'the pesticide paradox' refer to?

- c. Changing tests more often during a project will increase the usefulness of testing
- Correct. The pesticide paradox is the testing principle that refers to the fact that using the same tests over and over again will limit their ability to identify problems. To overcome this, tests need to be reviewed, altered and added to constantly.*

4. Which of the following statements does NOT apply to the V-model of testing?

- a. It specifies a series of linear test stages that happen one at a time

b. It is an iterative system development model

Indeed. The V-model of testing can be seen as an extension of the waterfall model, but instead of one test phase towards the end of the development process, there is a corresponding testing phase for each phase. However, each phase happens one after the other, so unlike agile, it cannot be considered an iterative model.

- c. It includes integration testing
- d. It is an extension of the waterfall model

5. Compared to testing in a waterfall approach, testing in an agile approach:

- a. Happens at different timepoints in the project

That's right. In a waterfall approach, testing will happen as the last step of the development process, whereas in an agile approach, you will test much earlier in the project (which is one of the 7 testing principles) and continue testing throughout. This allows you to 'fail fast and often' (and adapt fast and often), instead of saving failure till the end, where changes to the system are likely to be more costly.

6. What are characteristics of an integration test?

- a. It tests individual modules in combination
- b. It is usually carried out by testers
- c. It tests the data transfer between individual modules

d. All of the above

Correct. These are all characteristics of an integration test. An integration test is usually executed by testers and assesses integration and data transfer between system modules, which are combined and tested as a group.

7. What is the purpose of a user acceptance test?

- c. To assess whether the system meets the needs and requirements of the user

The developers' interpretation of the requirements may be different from what the user actually needs. The user acceptance test is a way to check this. It is the final functional testing phase and is executed—not surprisingly—by users. It is different from a usability test, which focuses more on assessing if users find the system easy to use and navigate.

8. Testing how many concurrent users the system can support is best described as a:

- a. Non-functional test

Correct! Looking at how the system behaves if many users access it simultaneously is also known as 'load testing'. It helps to determine the system's performance under real-life load conditions, and is an example of a non-functional test.

9. Which of the following kinds of test is least likely to lend itself for automation?

- b. Usability test

In general, functional tests, especially those that are executed by users, are less or not suitable for automated testing. Although a usability test is considered a non-functional test, it does involve users and is all about the users' perception of the system's ease-of-use. It will be hard –if not impossible—to automate this. In contrast, other non-functional tests (e.g. performance tests, load tests) as well as regression tests are obvious candidates for automation.

[Back to summary.](#)

