Everything on Testing

Different testing processes.

Unit/Component test	Whether a unit does or does not satisfy requirements and/or unit's implementation structure, does not match design
Integration testing	Whether combinations of components are incorrect or inconsistent. Formal and organized
System test	Concerns issues and behaviour that can only be exposed by testing the entire system.
Validation/acceptance test	Whether the system meets the client's requirements, related to used cases/scenarios

Verification vs Validation

Verification

Checking if software does what the specifications says it should.

Validation

Making sure the specification and system we build is really what the client wants. It requires carefully organised test plans designed around use cases and run through developer and client together.

The impossibility of perfect testing

The ideal exhaustive testing is to try all possible inputs, but that's not feasible. Test data therefore must be selected/designed.

The testing process

Component testing

Testing of individual components. Tests are derived from component descriptions/specifications.

Integration testing

Testing of groups of components, integrated to create a system or subsystem. Tests are based on system specification

Testing formality

Testing should be treated as a formal procedure. The documentation is vital as part of a formal process.

Test data and test cases

Test data: Input devised to test the system.
Predicted outputs: The result that should be produced from the test data inputs if the system operates according to its specification.

Test cases: Record of the test chosen to tests the system with rational test data and predicted outputs

Test case design
Test cases must be chosen/designed systematically. There are two approaches:

Black box testing	Knowing the expected functions
White box testing	Knowing the internal operation of the code

Black box testing
Don't know how the software specifics work.

Look at expected values, test cases based on system specification

Input data and output results often fall into different classes. Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member.

Test cases should be chosen from each partition, also test input values at boundaries between partitions and any special values

White box testing
Also known as structural testing or glass-box testing

Test cases are derived according to the programs structure. The objective is to exercise all program statements. Test data is selected to force execution of statements.

Predicted outputs are determined for the test data

Path Testing : Check all possible moves/statements through the code and execute them at least once. A flow diagram is required for this

Unit Testing
Test individual system components (classes, methods). Can be black-box or whitebox testing

It must execute a component under test in a test harness. The test harness comprises stubs and/or drivers, components that are depended on, are replaced by stubs

A stub is an empty or dummy implementation of a component that respects its public interface.

A driver organises the tests, this can be a junit test method. The driver instantiates the component under test and calls its methods. It supplies test data, gathers and reports the results

Integration Testing

Test complete systems or subsystems composed of integrated components.

It’s blackbox testing with tests derived from the specification.

The main problem with integration testing is that it is hard to localise errors. A solution to this is incremental integration testing, which reduces the problem by starting from individual components and building it up to groups

Components are executed under test harness (stubs, drivers)

Top-Down testing
Start with high level components and form groups from the top down. Individual lower level components are replaced by stubs

Bottom-up testing
Aggregate/group lower level individual components into levels until the complete system is created. Higher level drivers must be written to carry out the tests

Most integration involves a combination of these two strategies, however bottom-up testing is more useful

<u>Other aspects of testing</u>
Regression testing
Constantly repeating previous tests to make sure no bugs have appeared, when a new component is added, this is a good way to avoid knock-off effects. Re-testing the system following maintenance, to ensure continued correctness.

Comprehensive test documentation is essential.

Alpha Testing
Comes after all internal testing. An early release to selected real users (non-experts necessarily) expected to report bugs and observation to developers

Beta Testing
Following alpha testing its a wide set of representative users, before final release to all users

Stress Testing
Exercises the system beyond its maximum design load. Stressing the system test failure behaviour. System should not fail catastrophically. Also checks for unacceptable loss of service or data.

Usability, Security, Performance Testing