Chapter 2 - Testing throughout the software life cycle

Lan Anh Tran

March 18, 2025

Contents

1 Introduction

Software development follows various models that define how projects progress from conception to completion. These models guide the organization of development activities, including testing, which is a crucial part of ensuring software quality. The choice of a development model influences how and when testing is performed. This chapter explores different software development models, including sequential, iterative, and incremental approaches, and their impact on testing processes.

2 Software development models

Testing is important in the software development life cycle. The life cycle model will determine how to orginize the testing. Testing is highly related to software development activities.

Definition 2.1. *Verification: Is the deliverable built according to the specifications?*

Definition 2.2. *Validation: Is the deliverable fit for purpose, e.g. does it provide a solution to the problem?*

2.1 Sequential development models

Definition 2.3. *Sequential development model: A type of development life cycle model in which a complete system is developed in a linear way of several discrete and successive phases with no overlap between them.*

The Waterfall Model

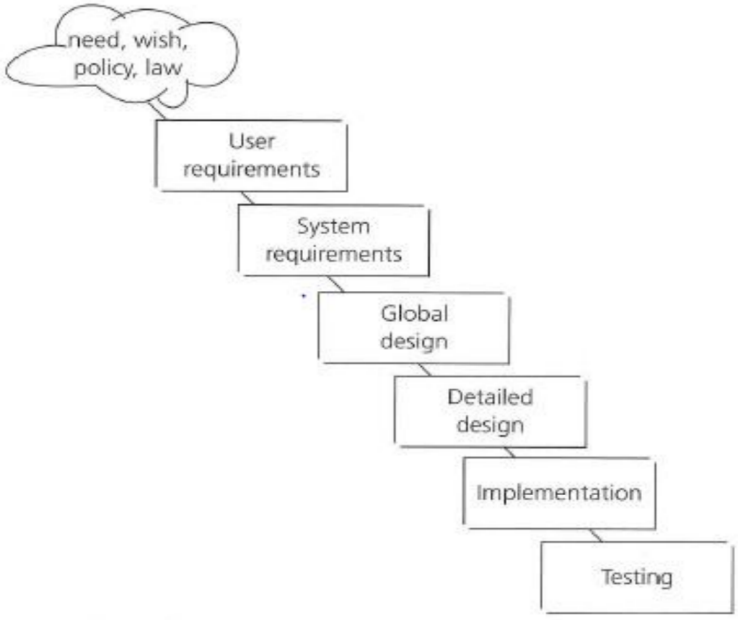

Figure 1: Waterfall model

The waterfall model (in Figure 1) was one of the earliest models to be designed. It has a natural timeline where tasks are executed in a sequential fashion. We start at the

top of the waterfall with a feasibility study and flow down trough the various project tasks, finishing with implementation into the live environment.

The V-model

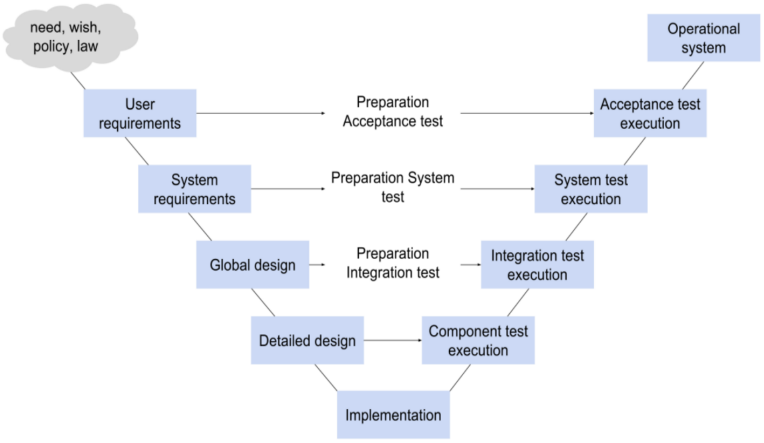

Figure 2: The V-model

The V-model wass developed to address some of the problems experienced using the traditional waterfall approach. Defects were being found too late in the life cycle, as testing was not involved until the end of the project. The V-model provides guidance on how testing begins as early as possible in the life cycle. Testing can also be integrated into each phase of the life cycle. Within the V-model, validation testing takes place especially during the early stages, rieviewing the user requirements an late in the life cycle, during user acceptance testing.

2.2 Iterative and incremental development models

Not all life cycles are sequential. There are also iterative and incremental life cycles where, instead of one large development timeline from beginning to endre, we cycle trough a number of smaller self-contained life cycle phases for the same project.
Exaples of iterative and incremental development models:

- Rational Unified Process (RUP)
- Scrum
- Kanban
- Spiral (and prototyping)

Iterative-incremental development is the process of establishing requirements, designing, building and testing a system carried out as a series of shorter development cycles. An increment, added to others developed previously, forms a growing partial system, which should also be tested. Regression testing is increasingly important to all iteration phases after the first one. The testing is often less formal, and common issues that might come is: more regression testing, defects outside the scope of the iteration or increment may be forgotten, less thorough testing.

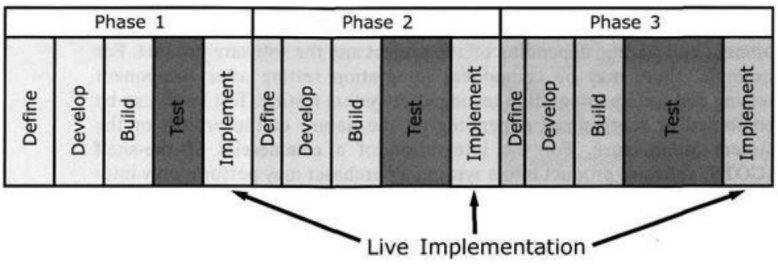

Figure 3: Iterative development model

2.3 Testing within a life cycle model

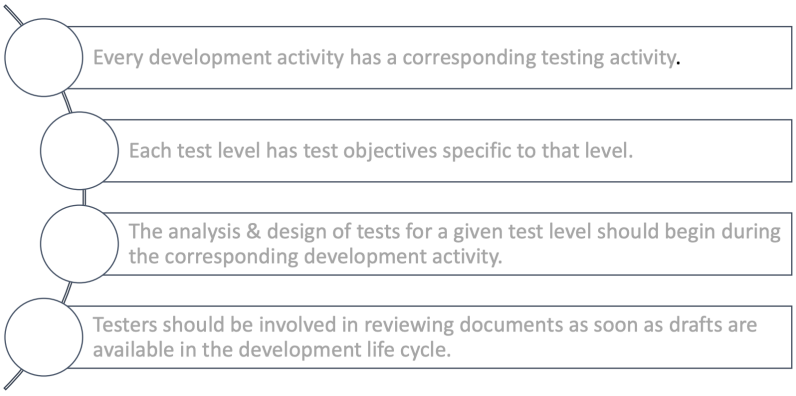

Figure 4: Good characteristics of good testing

Test levels can be combined or reorganized depending on the nature of the project or the system architecture.

3 Test Levels

Testing consist of four levels. From bottom and up, we have Unit/Component testing, Integration testing, System testing and Acceptance testing. For each test levels, importants details to note is: the generic objectives, the test basis (docs/products used to derive test cases), test objects (what is being tested), typical defects and failure to be found, sepcific approaches.

3.1 Component testing

Definition 3.1. *Component testing: The testing of individual hardware or software components.*

Test objectives
The objectives of components testing include:

- Reducing risk
- Verifying whether or not functional and non-functional behaviours of the component are as they should be.

- 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
- Verifying the functioning of software items (modules, methods, objects, classes, etc.) that are separately tastable.

Test basis

All meterials that are applicable for the component under test:

- Specification of the component
- Software design
- The data model
- The code itself

Test objects

What are we testing at this level? We could say the smallest thing that can be sensibly tested on its own. Typical test objects for component testing include:

- Components themselves, units or modules
- Code and data structures
- Classes
- Database models

Typical defects and failure

Examples of defects and failures that can typically be revealed by component testing include:

- Incorrects functionality
- Data flow problems
- Incorrect code or logic

Specific approaches and responsibilities

Component testing usually involves the programmer who wrote the code. Defects are fixed as soon as they are found, without formal recording of incidents. Test-driven development prepare and automate test cases before coding. And is used in extreme programming.

3.2 Integration testing

Definition 3.2. *Integration testing: Testing preformed to expose defects in the interfaces and in the interactions between integrated components or systems.*

Test objectives
The objectives of integration testing include:

- Reducing risk
- Verifying wheter 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 interface themselves or in the components or systems being tested together
- Preventing defects from escaping to later testing

Test basis
Examples of work products that can be used as a test basis for integration testing include:

- Software and system design
- Sequence diagrams
- Interfaces and communication protocol specifications
- Use cases
- Architechture at component or system level
- Workflows
- External interface definitions

Test objects
The emphasis here is in testing things with others which have already been tested individually. We are interested in how things work togheter and how thet interact. Typical test objects for integration testing include:

- Subsystems
- Databases
- Infrastructure
- Interfaces APIs
- Microservices

Typical defects and failures
Examples of defects and failures that can typically be revealed by *component intgration* testing include:

- Incorrect data, missing data or incorrect data encoding
- Incorrect sequencing or timing of interface calls
- Interface mismatch
- Failures in communication between components
- Unhandled or imprroperly handled communication failures between components.
- Incorrect assumptions about the meaning, units or boundaries of the data being passed between components.

Exaples of defects and failures that can typically be revealed by *system intgration* testing include:

- Inconsistent message structures between systems
- Incorrect data, missing data or incorrect data encoding
- Interface mismatch
- Failures in communication between systems
- Unhandled or improperly handled communication failures between systems

Approaches and responsibilites
Start integration with those components that are expected to cause most problems. To reduce the risk of late defect discovery, integration should normally be incremental rather than "big bang". Both functional and structural approaches may be used. Ideally, testers should understand the architecture and influence integration planning. Integration testing can be done by the developers or by a seperate team.

Integration testing types
Component integration testing test the interactions between software components and is done after component testing. It is a good candidate for automation. In iterative and incremental development, both component tests and integration test are usually part of continuos integration, which may involve automated build, test and release to end users or to a next level. At least, this is the theory. In practice, component integration testing may not be done at all, or misunderstood and as a consequence not done well.
System integration testing test the interactions between different systems, packages and microservices, and may be done after system testing. System integration testing may also test interfaces to and provided by external organization (such as web services).

3.3 System testing

Definition 3.3. *System testing: Testing an integrated system to verify that it meets specified requirements. (Note that the ISTQB definition implies that system testing is only about verification of specified requirements. In practice, system testing is often also about validation that the system is suitable for its intended users, as well as verifying againts any type of requirement.)*

Test objectives
The objectives of system testing include:

- Reducing risk
- Verifying whether or not functional and non-functional behaviours of the system are as they should be
- 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
- Testing the behavior of the whole system as defines by the scope of the project.

Test basis
What should the system as a whole be able to do? Examples of work products that can be used as a test basis for system testing include:

- Software and system requirement specifications (functional and non-functional)
- Risk analysis reports
- Use cases
- Epics and user stories
- Models of system behavior
- State diagrams
- System and user manuals
- Testers also need to deal woth incomplete or undocumented requirements

Test objects
What are we actually testing at this level? The emphasis here is in testing the whole system, from end to end, encompassing everything that the system need to do. Typical test objects for integration testing include:

- Applications
- Hardware/software
- Operating systems
- System under test
- System configuration and configuration data

Typical defects and failure
Examples of defects and failures that can typically be revealed by system testing include:

- Incorrects calculations
- Incorrects or unexpected system functional or non-functional behavior

- Incorrects control and/or data flows within the system
- Failure to properly an completely carry out end-to-end functional tasks
- Failure of the system to work properly in the production environments
- Failure of the system to work as described in system and user manuals

Specific approaches and responsibilites
Test encironment should correspond to the production environment as much as possible. First, the most suited black-box technique. Then, white-box technique to assess the thoroughness of testing. An independent test team may be responsible for the testing. The level of independence is based on the applicable risk level.

3.4 Acceptance testing

Definition 3.4. *Acceptance testing: Formal testing with respect to user needs, requirements, and business process conducted to determine wheter or not a system satisfies the acceptance criteria and to enable the user, costumers or other authorized entity to determine whether or not to accept the system.*

Test objectives
The questions to be answered:

- Can the system be released?
- What are the outstanding risks?
- Has development met its obligations?

The goal is to establish confidence in the system and the non-functional characteristics of the system

Test basis

- User requirements specification
- User stories
- Use cases
- System requirements specification
- Business process
- Risk analysis

Responsibilities

- Costumers or users of a system
- Stakeholders may be involved

Test types
We have contract and regulation acceptance testing that preformes againts a contracts acceptance criteria. We also have alpha and beta testing, where alpha testing is performed at the developing organizations site. And beta testing (field testing), that is performed by people at their own locations. Both are preformed by potential customers, not the developers of the product.

4 Test Types

We have four test types: functional, non-functional, structural and related to changes.

4.1 Testing of function

This type focuses on ”what” the system does.

Test obectives
Test what a system should do and consider the external behavior of the software.

Test levels
May be performed at all test levels.

Test basis
The expected behavior description can be found in work products such as: requirement specification, business processes, use cases, functional specifications, may be undocumented

4.2 Testing of non-functional software characteristics

This type focuses on ”how” the system works.

Test objectives
Mesuring characteristics of software that can be quantified on a varying scale: e. g. response times for performance testing.

Test levels
May be performed at all test levels.

Software Product Quality

Figure 5: A model over ISO 9126 - what does a software prduct quality contain?

4.3 Testing of software structure

Focus on code coverage.

Test objectives

Measuring the thoroughness of testing through assessment of the coverage of a set of structural elements or coverage items.

Test levels

May be performed at all test levels, but especially in component testing and component integration testing.

Test basis

Structural testing is based on the structure of the code as well as the architecture of the system.

Test approach

Structural techniques are best used after specification-based techniques, in order to help measure the thoroghness of testing.

Test tools

Coverage measurement tools assess the percentage of executable elements.

4.4 Testing related to changes

We have confirmation testing (definition 4.1) and regression testing (definition 4.2) in this test type.

Definition 4.1. *Confirmation testing: After a defect is detected and fixed, the software should be retested to confirm that the orgiginal defect has been successfully removed.*

Definition 4.2. *Regression testing: The repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the changes.*

Test objectives

To verify that modifications in the software or environment have not caused unintended side effects and that the system still meets its requirements.

Test levels

May be performed at all test levels, applies to functional, non-functional and structural testing

Test approach

The extent of regression testing is based on the risk of finding defects in software that was working previously. Regression test suites are run many times and generally evolve slowly, so regression testing is a strong candidate for automation. If the regression test suite is very large it may be more appropriate to select a subset for execution.

5 Maintenance testing

Test objectives
Maintenance testing is done on a existing operational system, and is triggered by modifications, migratio, or retirement of the software or system.

Test types
Modification

- Planned enhancement changes
- Corrective and emergency changes (patched)
- Changes of environment (operating system or database upgrades)

Migration

- Operational tests of the new environment
- Tests on the changed software

Retirement of a system

- The testing of data migration or archiving if long dataretention periods are required.

Scope
The scope of maintenance testing is related to: the risk of the change, the size of the existing system and the size of the change.

Test levels
Depending on the changes, maintenance testing may be done at any or all test levels and for any or all test types.

Approach
Determining how the existing system may be affected by changes is called impact analysis and is used to help decide how much regression testing to do.

Note
Maintenance testing can be difficult if specifications are out of date or missing.

6 Conclusion

The selection of a software development model plays a significant role in determining the structure and timing of testing activities. Sequential models like Waterfall provide a linear approach, whereas iterative models, such as Scrum and Spiral, allow for continuous refinement through repeated cycles. Regardless of the model used, testing remains essential at every stage to ensure the software meets functional and non-functional requirements. By understanding these models and their integration with testing, teams can improve software quality and reduce the risk of defects reaching production.