

Testing throughout the SDLC

SE401: Software Quality Assurance and Testing

Outline

- Software development models
- Test Levels
- Test Types
- Maintenance testing

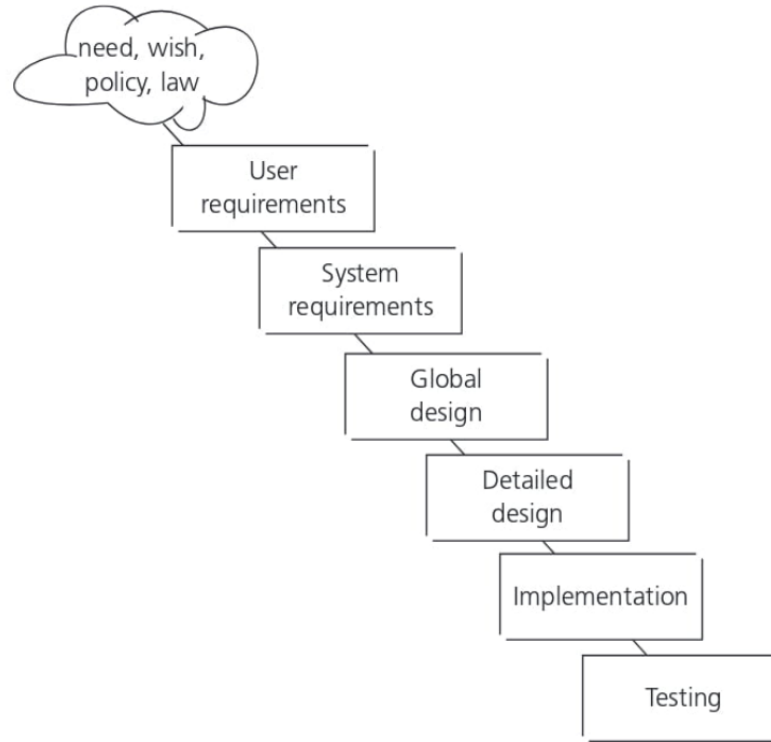
Software development models

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

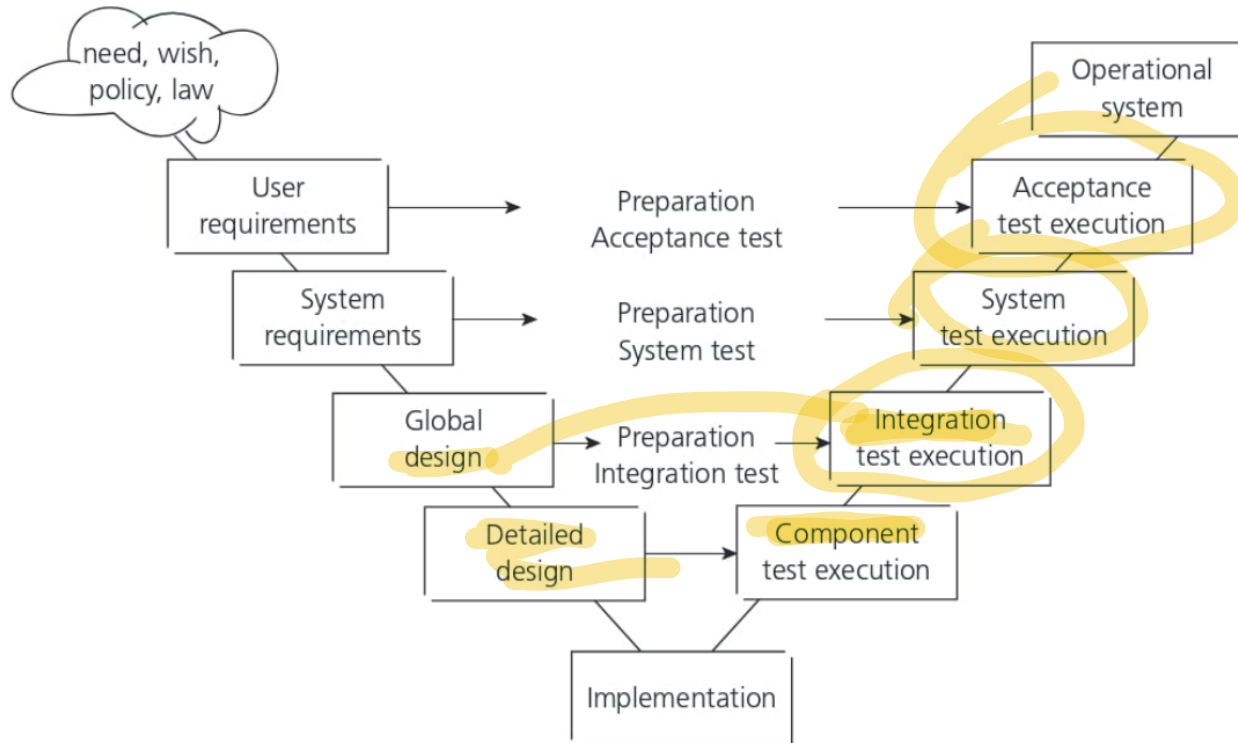
Testing is focused on...

- **Verification** *am i Building the SW correctly.*
 - *Is the deliverable built according to the specifications?*
- **Validation** *am i Building The right SW?*
 - *Is the deliverable fit for purpose, e.g. does it provide a solution to the problem?*

Waterfall model



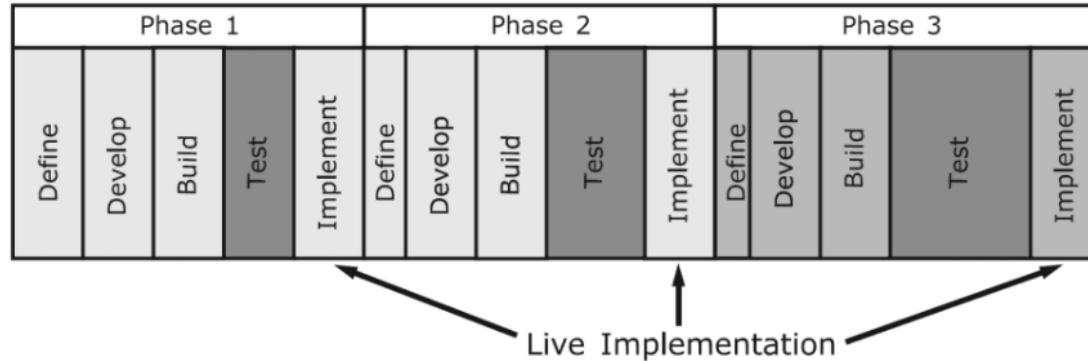
The V-model



The V model

- Testing needs to begin as early as possible in the life cycle.
- Testing can be integrated into each phase of the life cycle.
- Within the V-model, validation testing takes place especially during
 - the early stages ,i.e. reviewing the user requirements
 - and late in the life cycle, i.e. during user acceptance testing

Iterative-incremental development model



- **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.

Testing within a life cycle model

- In any life cycle model, there are several characteristics of good testing:

Every development activity has a corresponding testing activity

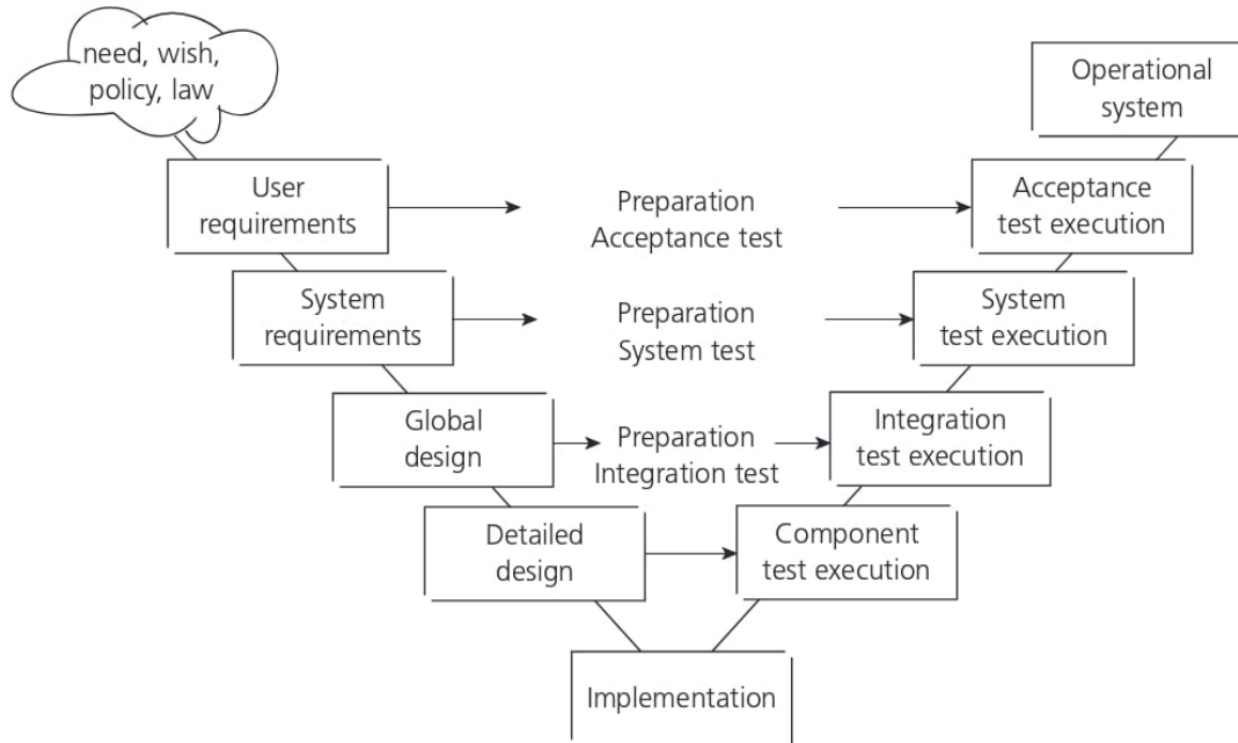
Each test level has test objectives specific to that level.

The analysis & design of tests for a given test level should begin during the corresponding development activity

Testers should be involved in reviewing documents as soon as drafts are available in the development life cycle

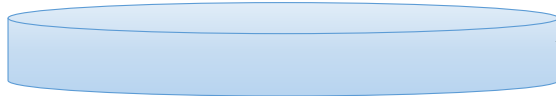
- Test levels can be combined or reorganized depending on the nature of the project or the system architecture

Testing within a life cycle model

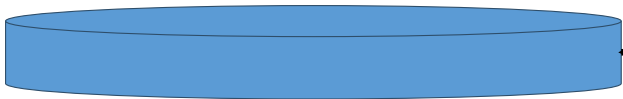


Test Levels

- **Acceptance:** is the responsibility of the customer – in general. The goal is to gain confidence in the system



System: The behavior of the whole product as defined by the project scope



Integration: Interfaces between components ; interactions with other systems (OS, HW, ...)



- **Unit:** Any module, program, object separately testable

↳ component

Test Levels

- For each test level, please note:
 - The generic objectives
 - The test basis (docs/products used to derive test cases)
 - The test objects (what is being tested)
 - Typical defects and failures to be found
 - Specific approaches and responsibilities

Component testing - objectives

- Verifying the functioning of software items (modules, methods, objects, classes etc.) that are separately testable
- Component testing includes testing of functionality and specific non functional characteristics, such as:
 - resource behavior (e.g. memory leaks)
 - robustness testing
 - structural testing (e.g. branch coverage).

Component testing - test basis

- All materials that are applicable for the component under test:
 - Specification of the component
 - Software design
 - The data model *class diagrams.*
 - As well as the code itself

Component testing

- Writing code to test the project code
- Stubs, drivers and simulators may be used.
- **Stubs:**
 - Code that replaces a called component in order to simulate its purpose (i.e. "hard-coded" data to replace data from a database).
- **Drivers:**
 - Code that replaces an other software component in order to call the component under test.

Component testing - example

```
public void getLocationListTest()
{
    //Arrange
    VisitorRepositoryStub stub = new VisitorRepositoryStub();
    VisitorBLL bll = new VisitorBLL(stub);
    List<LocationSummary> expectedResult = stub.getLocationList("nb-NO");
    //Act
    List<LocationSummary> result = bll.getLocationList("nb-NO");
    //Assert
    Assert.Equal(expectedResult.Count(), result.Count());
    Assert.Equal(expectedResult[0].id, result[0].id);
}
```

Component testing - 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.
- TDD test-driven development
 - prepare and automate test cases before coding
 - used in XP (extreme programming)

Integration testing - objectives

- Tests interfaces between components
- Test interactions with different parts of a system, such as:
 - the operating system
 - file system
 - hardware
 - interfaces between systems

Integration testing - test basis

- Software and system design
- The system architecture
- Workflows/use cases

Integration testing - test objects

- The item under test includes
 - Builds including some or all component of the system
 - The database elements
 - System infrastructure
 - Interface between components or objects
 - System configuration
 - Configuration data

Integration testing - types

- **Component integration**



- Tests the interactions between software components is done after component testing

- **System integration**

- Tests the interactions between different systems is done after system testing.

Integration testing - approaches and responsibilities

- 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”. *Past exam.*
- Both functional and structural approaches may be used.
- Ideally, testers should understand the architecture and influence integration planning.
- Can be done by the developers or by a separate team.

System testing - objectives

- Testing the behavior of the whole system as defined by the scope of the project.

System testing - test basis

- System requirements specification , both functional and non functional
- Business processes
- Risk analysis
- Use cases
- Other high level descriptions of the system behavior, interactions with OS/system resources
- Requirements may exist as text and/or models.
- Testers also need to deal with incomplete or undocumented requirements.

System testing - test objects

- The entire integrated system
- User manuals
- Operation manuals
- System configuration information
- Configuration data

System testing - approaches and responsibilities

OS For example

- Test environment 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

Acceptance testing - 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
 - Non-functional characteristics of the system



Acceptance testing - test basis

- User requirements specification
- Use cases
- System requirements specification
- Business processes
- Risk analysis



Acceptance testing - types

- **User acceptance testing** –validate the fitness for use of the system by users.
- **Operational testing, usually done by the system administrators:**
 - testing of backup/restore
 - disaster recovery
 - user management
 - maintenance tasks
 - periodic checks of security vulnerabilities



Acceptance testing - types

- **Contract and regulation acceptance testing** performed against a contract's acceptance criteria (i.e. governmental, legal or safety regulations)
- **Alpha and beta testing**
 - Alpha testing is performed at the developing organization's site.
 - Beta testing (field testing), is performed by people at their own locations.
- Both are performed by potential customers, not the developers of the product.

Acceptance testing - responsibilities

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

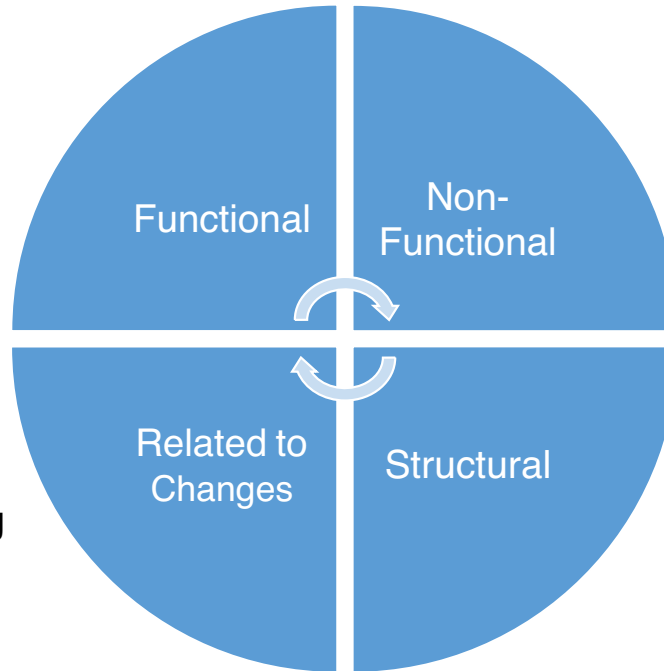
Test Types

“What” the system does:

- Suitability
- Interoperability
- Security
- Accuracy
- Compliance

“How” the system works:

- Performance, Load, Stress
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability



- Confirmation Testing
- Regression Testing

- Code Coverage

Functional testing (Black box testing)

- **Objectives**

- 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:
 - requirements specification
 - business processes
 - use cases
 - functional specifications
 - may be undocumented

Non functional testing

- **Objectives**

- Measuring 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
- You can find more about them in 'Software Engineering –Software Product Quality' (ISO 9126).

Software Product Quality



Structural testing (white box testing)

- **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.

Structural testing (white box testing)

- **Test basis**

- Structural testing is based on the structure of the code as well as the architecture of the system (e.g. a calling hierarchy, a business model or a menu structure)

- **Approach**

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

- **Tools**

- Coverage measurement tools assess the percentage of executable elements(e.g. statements or decision outcomes) that have been exercised.

Testing related to changes, confirmation and regression testing

- **Confirmation testing**

- After a defect is detected and fixed, the software should be retested to confirm that the original defect has been successfully removed.

- **Regression testing**

- The repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the change(s).

Testing related to changes

- **Objective**

- To verify that modifications in the software or the 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.

Testing related to changes

- **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.

Maintenance testing

- **Objectives**

- Maintenance testing is done on an existing operational system , and is triggered by modifications migration, or retirement of the software or system.

Maintenance testing - types

- **Modifications**

- Planned enhancement changes(e.g. release-based)
- Corrective and emergency changes (patches)
- Changes of environment (operating system or database upgrades)

- **Migration** (e.g. from one platform to another)

- operational tests of the new environment
- tests on the changed software.

- **Retirement of a system**

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

Maintenance testing

- **Scope**

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

Maintenance testing

- **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.