2. Testing Throughout The Life Cycle

Hans Schaefer
hans.schaefer@ieee.org
http://www.softwaretesting.no

Contents

Terminology

Verification

To check if a product fulfills what was defined on a higher abstraction level (i.e. code against design).

Is the product built right?

Validation

To check if a product is usable in practice, i.e. to check against stakeholder requirements and needs.

Is it the right product? (value for user)

Test oracle

A mechanism for determining if a test result is right or not (for example a program, computation, or visual check by the tester).

Terminology

Robustness testing

Testing to determine the robustness of the software product.

Robustness: The degree to which a component or system can function correctly in the presence of invalid inputs or stressful environmental conditions. [IEEE 610] See also error tolerance, fault-tolerance.

Negative testing: Tests aimed at showing that a component or system does not work. Negative testing is related to the testers' attitude rather than a specific test approach or test design technique, e.g. testing with invalid input values or exceptions. [After Beizer].

Terms
-> Development models
Test levels
Test types and what they test
Testing in maintenance

Development Models

Software Development Models

Two main models:

One time- (sequential) Development (f.ex. the Waterfall model)
- **with low risk**
- **stable requirements**
- **if the general solution is known**

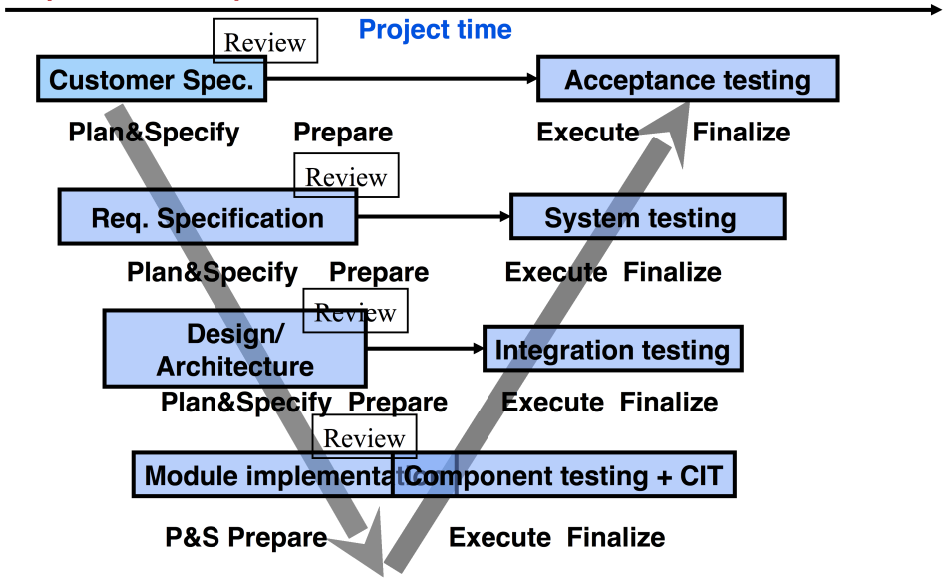
Iterative, incremental, evolutionary models
- **if something is not sure and one needs feedback**

Principles for Good Testing

- Test does not come at the end, but is integrated in the model!
- Only test execution is at the end, everything else as early as possible!
- The earlier one develops test cases, the earlier one can find bugs or possible bugs!

The General V-Model (in principle) -

sequential development model



Why the V-Model

A sequential model, improves the Waterfall model

Usually four test levels

Every test level shall find faults from the corresponding construction level.

A later level is a safety net for earlier ones.

Early test preparation!

Find faults when you prepare the test, not during execution!

Test preparation gives you a better understanding of the task.

V = Victory!

Problems With the V-Model

Early test material is devaluated by later changes.

It is tempting to postpone early tests until later (too much work for test environment).

Sequential execution of the test levels

For a discussion of the boundaries of the up-front test method, see

http://www.ddj.com/architect/199600262

How to Apply the V-Model

- Run a V for every increment / release.
- Run a V on every prototype.
- Make test checklists (*use cases*, draft test specifications) as early as possible.
- Make test data (concrete test cases) during coding.
- Keep the test material, update and automate it.
  - Update it for every release as you learn more about defects.
- Regression test for every release!
  - (Run a smoke test for every Build).

"Trade-offs between Productivity and Quality in Selecting Software Development Practices", Alan MacCormack, Chris F. Kemerer, Michael Cusumano, Bill Crandall, IEEE Software Magazine, 5/2003

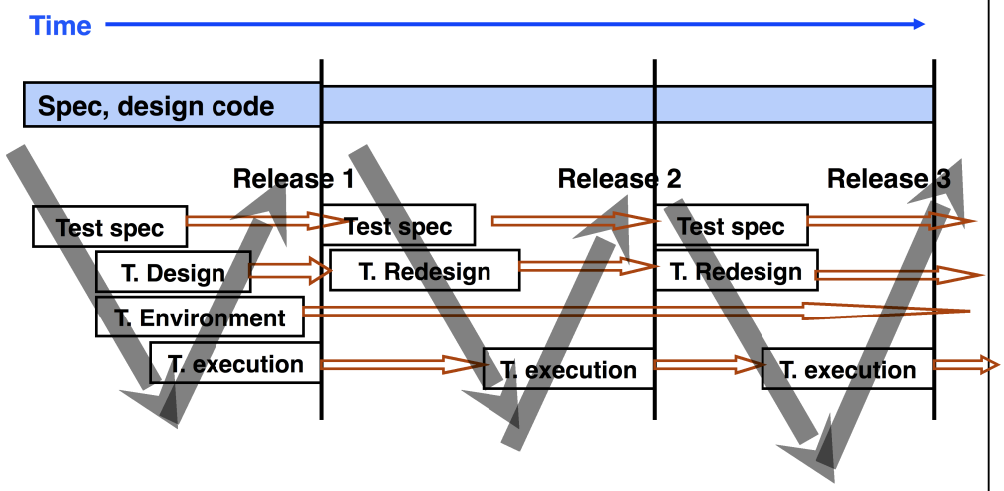
Testing for Iterative-incremental Development Methods

- Iterative Development method: Several small developments, in a row, each follows the life cycle
- Examples: Prototyping, RAD (Rapid Application Development), RUP (”Rational Undefined Process” :-)), SCRUM, "agile" development models
- The increment can be tested at different levels as part of development.
- In addition test of the integrated system.

Se mer på www.agile101.net

Very important: Regression testing!

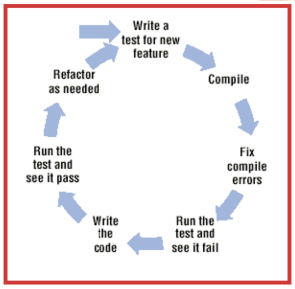
The Real Way of Working with Test



Principles for Good Testing in All Life Cycle Models

- Every development activity has its corresponding test activity.
- Every test level has a specific objective.
- Test preparation shall start at the same time as the corresponding development activity -> feedback, thinking, improvement / adaptation.
- Testers can review drafts for development documents, at the same time as test design.
- The number of test levels depends on system complexity. Especially integration test is often partitioned into several levels. Test levels can also be combined.

Test-Driven Development (TDD) (not for exam)



Development of test cases before the code
Automation of the whole test
Running the test (failing)
before the code is ready
Implementation of code to pass the test -> next iteration

This is consequently applying the V-model principle of early test preparation!

A consequence of the thought about early test development and regression test (incremental development)

Ref: IEEE Software Magazine 3/2007 (whole issue)

The End Game (not for exam)

1. Stop testing and plan the end game.
2. Rank the known failures by severity.
3. Estimate how many (severe) failures can be fixed in the time remaining.
4. Drop unfixable features from the release plan.
5. If features cannot be dropped, change the release date.
6. Remove the bugs as decided in step 2 and retest / regression test.

Source: Weinberg, Perfect Software and other illusions about testing

Terms
Development models
-> Test levels
Test types amd what they test
Testing in maintenance

Test Levels

Tasks for the Test Levels

Component (Unit-, Module-) test: Coding problems, problems in detailed design, algorithms.

Integration test: Problems in interfaces, working together, design, architecture.

System test: Problems in requirements, problems with system attributes.

Acceptance test: Deviations from the customer interpretation of requirements and needs. The system does not work on the customer platform and in the customer environment.

Detailed Definitions of Test Levels

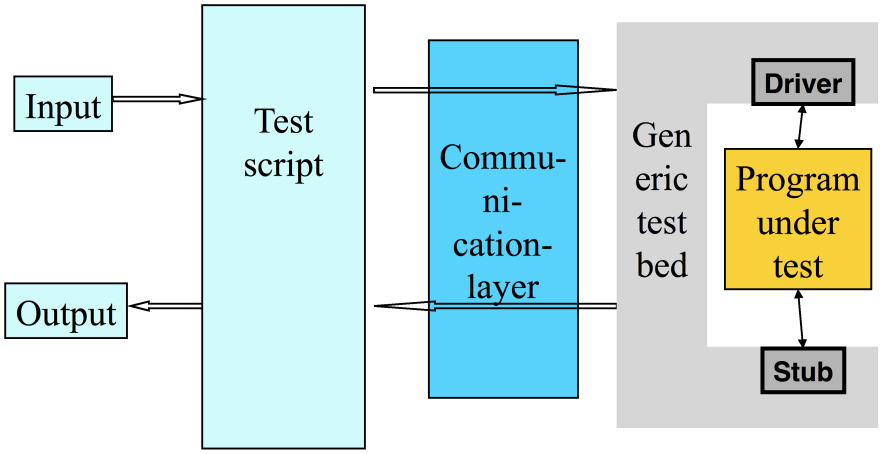
Component (Unit-, Module-) test: Test of components / modules isolated
In order to find detail design and coding bugs.
Component (Unit-, Module-) integration test: Test of component interaction.
Components on the same platform or in the same process. To find interface bugs.
Normally no "official" test.
(higher level) Integration test: Test of interaction between subsystems, modules on different platforms or in different processes. Normally called "integration test".
System test: Test against the requirements
Functional test: Test of every function against its requirements. To find high level-design faults and misunderstood requirements.
Interaction test: Test of functions with each other, to detect conflicts between them.
Nonfunctional test: Test of special things and system attributes.
System integration test: Test of interaction between the system and other external systems. To find interface problems.
Acceptance test: Customer test against contract requirements and expectations. Is the system usable in practice? Testing including system, users, processes.

Component test / Module test / Unit test

Typical test objects: components, programs, data conversion / migration programs, database modules
Test basis: Component requirements, detailed design, code

Isolated components
On development platform
Access to development tools and debugger
Access to the code
Developer involved
Bugs are repaired at once and in an informal way
Often not seen as a formal test
Stubs, drivers, simulators
Test of functionality and structure
Maybe test of non-functional attributes

Design of Test Environment



Test Driver

Tasks:

- Call the module under test
- Prepare / update the environment
- Log and list output
- Analysis functions

Test driver can be generated automatically by tools or replaced by debuggers.

Test driver should be kept for regression test.

Stub

Tasks: Replaces a component not yet part of the test.

Complexity increasing the more you put into a stub.

- **Exit immediately**
- **Log that you have been here**
- **Give a constant output**
- **Read output from a file or terminal**
- **Debugging facilities**
- **Simulation of real component**

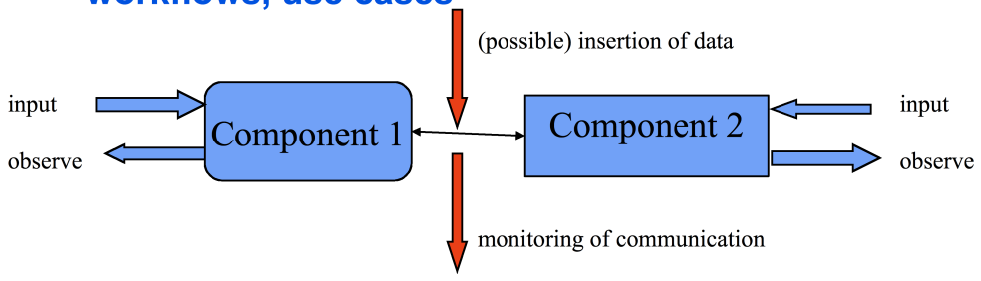
Stubs can be generated automatically by a tool or replaced by a debugger.
Stubs should be kept for regression test.

Also: ”Dummies”, ”Mockups” or ”mock objects”

Integration Test

Typical test objects: sub-systems, interfaces, database implementation, infrastructure, system configuration and control data

Test basis: Software and system design, architecture, workflows, use cases



“... The biggest problem we have in software development is that code isn’t always a good neighbor.” (Tom and Mary Poppendieck)

Integration Test in Practice

Often several levels

Component- or Module integration: Test of communication between modules, often on development platform, informally done by developer

”Real” integration: Test of communication between larger components, sub systems
Needs test environment, test lab

System integration: Communication between new system and other systems. Several platforms, customers can be involved, several suppliers, ...

Integration Strategies

Step wise:
- Isolates bugs to the last integrated parts
- Reduces the risk in later test
- Requires a lot of simulations, test environment, expensive
- Sequential

Larger parts at the same time (”big bang”):
- Problems to isolate bugs
- Less cost for simulation and environment
- Sliding boundary to system test
- Useful for new versions of existing product
- Larger risk in later test

Integration Strategies

Depend on system architecture.

Depend on cost for test environments (drivers, stubs).

No general strategy is good for every problem.

Integration Strategies

Control flow based
- top-down
- bottom-up
- Server before client

Layer based
- First the lower layers, then the upper ones using their services

Function based
- All components implementing a function

Business based
- All component implementing a process

Data model based
- All components using a specific part of the data model

Tester Tasks with Integration

Early feedback

- Which interfaces make testing difficult?
- Which components must be delivered first (to ease integration)?
- Standardization of interfaces.
- Require ”test hooks”, back doors to make testing easier.

System Test

Typical test objects: System, user and operation manuals, system configuration and configuration data

Test basis: System and Software requirement specification, functional specification, risk analysis reports

Platform and environment as near as possible to customer reality

based also on

- Models
- Use cases
- Business process-flow
- Interaction with platform and resources
- Generally accepted requirements to systems
- Data quality characteristics

System Test - What to Include?

- Function test
- Function interaction, flow
- Non-functional attributes (-> pg. 43)
  - Efficiency (stress, load, volume test)
  - Usability
  - Safety
  - Robustness
  - Portability
  - Maintainability

System test - who?

- Normally independent test team
- Large task to make test environment
- Must not be project-blind
- Customer viewpoint

Problem: Often incomplete / undocumented requirements!

System test - Main Techniques

- Black box test (based on specifications)
- White box not much used, rather ”gray box” - focus on user visible design details
- Much focus on test of attributes!

A kind of white box test:
Test of all menu structures

Acceptance Test

- Typical test objects: Business processes on fully integrated system, operational and maintenance processes, user procedures, forms, reports, configuration and control data.
- Test basis: User requirements, system requirements, use cases, business processes, risk analysis reports

- Customer’s test
- Formal test against acceptance criteria
- Does the system work for the business processes and in relation to requirements and contract?
- Is the system ready for use?
- Customers and their users shall be part of this!

- Example failure in business processes: Heathrow Terminal 5, March 2008

Acceptance Test

The goal is NOT to find failures, but to build trust! (by measuring quality)

-> It must be assured that the faults are found before. -> Requirements to and follow up of supplier test.

Acceptance Test (AT), Special Case

Standard systems (COTS- commercial-off-the-shelf) are tested when being installed.

AT can be followed by system integration test

Usage attributes should be tested earlier (prototypes)

AT of changes can (and should) be prepared before system test

Acceptance Test, Special Types

User Acceptance Test (The users check if they can use the system in practice)
Operational Testing (all tasks in operation, like backup, restore, user management, maintenance, security check, recovery,...)
Verification against contract, regulations, laws, safety regulations, etc. ("Compliance testing", contract acceptance testing, certification)
For standard systems: Alpha test, Beta test, Field test, Pilot test
For special systems: "Factory Acceptance Test", "Site Acceptance Test"

Cited from syllabus: Developers of market, or COTS, software often want to get feedback from potential or existing customers in their market before the software product is put up for sale commercially. Alpha testing is performed at the developing organization's site. Beta testing, or field testing, is performed by people at their own locations. Both are performed by potential customers, not the developers of the product.
Organizations may use other terms as well, such as factory acceptance testing and site acceptance testing for systems that are tested before and after being moved to a customer's site.

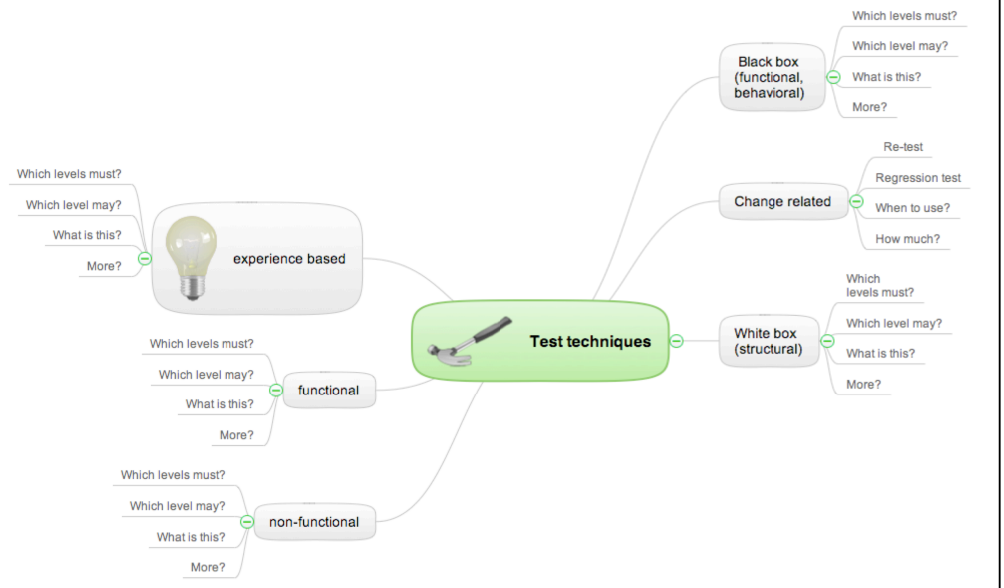
Acceptance Test - Main Techniques

As in system test
Functional test (single functions) -> less!
-> Business process test, "soap opera test" (several functions) -> more!
Attribute test (load, response times, usability, ...)

Questions?

Test Types and What They Test

Test type: A group of activities that shall verify a certain area or has a certain goal.



Functional Testing (vs. Non-functional)

Test of what the system does.
The functionality.

Special types:

- **Security testing (security functionality)**
- **Interoperability testing (working together with other systems or components)**

Non-functional Testing (vs. Functional)

Test of product attributes / characteristics
Definition see ISO/IEC 9126-1

Reliability
Efficiency (performance, stress test)
Usability
Portability
Maintainability

Can be both black box and white box!

Such tests can be done at all test levels. Most important during system- and acceptance test.
Requirements should be quantified!

ISO/IEC 9126-1 Quality Model

		Internal and external quality				Quality in use
Functionality	Reliability	Usability	Efficiency	Maintainability	Portability	
Suitability	Maturity	Understandability	Time behavior	Changeability	Adaptability	Effectiveness
Accuracy	Fault tolerance	Learnability	Resource behavior	Stability	Installability	Productivity
Interoperability	Recoverability	Operability	Compliance	Testability	Co-existence	Safety
Security	Compliance	Attractiveness		Compliance	Replaceability	Satisfaction
Compliance		Compliance			Compliance	

Functional Testing - Black Box Testing (vs. White Box)

Test of what the system does.
Against requirements, user manual, use cases, user stories, ...
Alternative name: Behavioral testing

Allows early preparation of test.

Can be used on all Test levels!

Black Box Testing
Measuring Coverage

Which parts of the specification are tested?
Every requirement?
Every menu?
Every window/screen?
Every external interface?
Every command?
Etc.

This is measured with cross references specification<->test.

Structural Testing - White Box

(vs. Black Box)

Test in order to cover the architecture, structure and all details in the code of the system

White box / glass box / open box testing

To measure the comprehensiveness of testing

Much in component testing, less in higher test levels.

White Box Testing

Measuring Test Coverage

To which degree is the structure / code executed under testing.

For example coverage of statements, branches, conditions, interfaces (-> Chapter 4).

Test coverage can be measured automatically by a tool.

Test coverage can be increased by developing extra test cases.

Testing In Maintenance / Related To Changes

Terms
Development models
Test levels
Test types and what they test
-> Testing in maintenance

Why?

- **Risk with changes due to**
  - Modifications, migration, or retirement of the software or system
  - New or changed control data
  - Environment changes
  - Changes of standards (that must be followed)
  - Updates due to newly discovered vurnerabilities
  - Coupling with new (external) systems
  - Etc.

CTFL Syllabus: The planning of releases in advance is crucial for a successful maintenance testing. A distinction has to be made between planned releases and hot fixes.

Test Related To Changes

Re-testing / Confirmation testing

Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions.

Regression testing

Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.

Regression Test

May be needed on every test level
Should be automated (otherwise too difficult to do)
Functional, non-functional, structural tests

Not necessarily ALL existing test cases! -> impact and risk analysis.

Where can side-effects occur?
Not easy, often impossible due to complex structure

Automatic tests tend to find less and less defects (pesticide paradox).

How Much Test In Maintenance?

Depends on risk
- Risk in the change itself
- Size of change
- Size of system
- Criticality

Very difficult if no specifications or no test material is present.

Testing can be done at all test levels.
No general rules!

NOT IN THE PRODUCTION ENVIRONMENT!

The Total Test

- Consists of many test cases
- Test cases should be grouped and ordered into test suites
- Consists of all possible test types

Questions?