ISTQB
Practitioner Certificate in Software Testing

Author: Bart Franco
Mario Peeters

DOCUMENT CHANGE HISTORY

Date	Version	Reason
2006-10-21	0.1	Creation
2006-10-24	0.2	Added introduction, Change history
2006-10-25	0.3	Added abbreviations list
2006-11-03	0.4	Updated after first review
2006-11-05	0.5	Reshuffled chapters, added pictures/tables
2006-11-06	0.6	Peer Review
2006-11-07	1.0	Updated with peer review comments
2006-12-05	1.1	Updated error on p65

1 Introduction

1.1 Purpose

This document is based on the syllabus of the ISTQB Practitioner course. As such it can be used as a preparation for the exam for achieving the ISTQB Practitioner Certificate in Software Testing.
The document can also be used within your population as a guide to set up and maintain a structured test process within their job assignments and to come to a common mindset about testing.

Remarks and suggestions can always be given to the authors of the document and will be (after approval) be incorporated in the document:

Bart Franco: bart.franco@dotc.be
Mario Peeters: mario.peeters@tass.be

1.2 Abbreviations

AT	Alpha Test
BVA	Boundary Value Analysis
C/E graphing	Cause Effect graphing
CTM	Classification Tree Method
ECT	Elementary Comparison Test
EP	Equivalence Partitioning
ET	Exploratory Testing
FMEA	Failure Mode Effect Analysis
FPA	Function Point Analysis
IEEE	Institute of Electrical and Electronics Engineers
IFPUG	International Function Point Users Group
ISTQB	International Software Testing Qualification Board
KISS	Keep It Simple and Short
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
PBR	Perspective Based Reading
RAD	Rapid Application Development
ROI	Return On Investment
RPN	Risk Priority Number
SA	Static Analysis
SUMI	Software Usability Measurement Inventory
TEEM	Test Effort Estimation Model
TMap	Test management approach
TMI	Test Process Improvement
WBD	Work Break Down

2 Testing in the life cycle

2.1 Lifecycle models

The choice for a development lifecycle depends on the project aims and goals:

- Business drivers
- Project drivers
- Goals

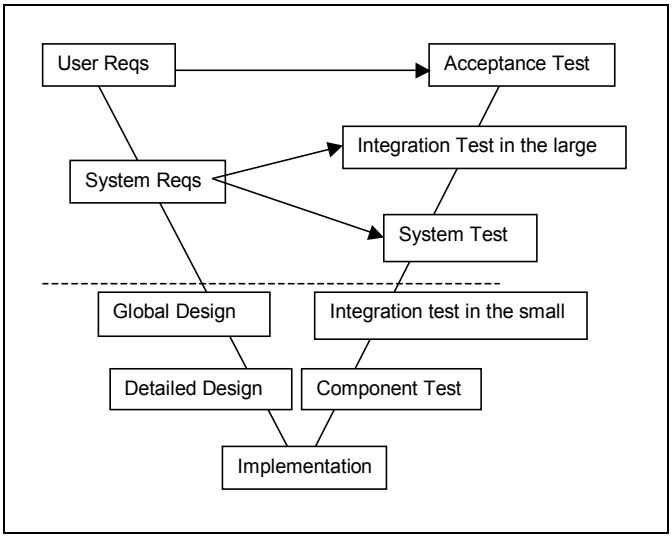
There are several lifecycle models and each affects the testing that needs to be carried out.

Lifecycle models can be divided into two categories: sequential and iterative models.

2.1.1 Sequential lifecycle models are:

- **Waterfall**. In this model the development phases flow into one another, starting with a feasibility study in the top and implementation at the bottom. Test related activities tend to be executed at the end of the project lifecycle, on the critical path. Also it is difficult to get feedback from 'lower activities' to activities above.

- **V model**. To address the problems of the waterfall, the V model bends the testing activities around the point of code delivery. Testing activities are carried out as early as possible, related to the design and build activities:



2.1.2 Iterative lifecycle models are:

- **Pre-planned incremental delivery (phased)**. Instead of one lifecycle, several smaller lifecycles are executed. With each increment new functionality is added. This supports early market presence. With each increment more testing is involved, because the new functionality must be tested, regression testing of the existing functionality and integration tests of the new and existing parts has to be executed. Notice: the full functionality is known beforehand.

- **Spiral**. In situations where the full set of system requirements is not known, building prototypes and simulations can reduce the risks. Each phase starts (and continues) with an analysis of alternatives and risks to achieve the goals.

- **RAD**. Parallel development method where simultaneously functions are developed and integrated within time boxes leading to prototypes. Later product specification needs to be created and the project must be placed under formal control before production implementation. Configuration management is very important. Testing needs to incorporate the changes in the project and deliverables.

- **Extreme programming.**
  Some characteristics:
  - Business stories for describing functionalities
  - On site of the customer for continuous feedback and acceptance testing
  - Pair programming and shared code ownership
  - Design the tests first
  - Continuous integration of code
  - Implementation of the simplest solution for today's problems.

- **Evolutionary.** More focus on delivery than development. Each evolutionary cycle delivers a live working system during a (short or long) period of time. With each cycle the functionality is extended relying on active customer feedback. Based on existing functionality the customer decides if the development proceeds, what new functionality needs to be development, or to halt the project? Decisions are purely based on business and return on investment.

2.2 Planning using the V-model

The V model:

- Encourages early involvement of the test team
- Adds value at requirements definition/specification stages
- Encourages early detection of faults
- Encourages reviews and static testing before code delivery
- Minimizes testing impact on the development critical paths

Several test execution phases are part of the V model:

- Component test
- Integration test in the small
- System testing
- Integration testing in the large
- Acceptance testing

It's good if an organization plans ahead for each phase what the contents is.
Test Level Characteristics per test phase that should be determined are:

1. Objective
2. Scope
3. Owner – who is accountable, responsible
4. Entry criteria
5. Exit criteria
6. Test deliverables
7. Typical techniques
8. Tools
9. Standards
10. Metrics

Entry/Exit criteria because:

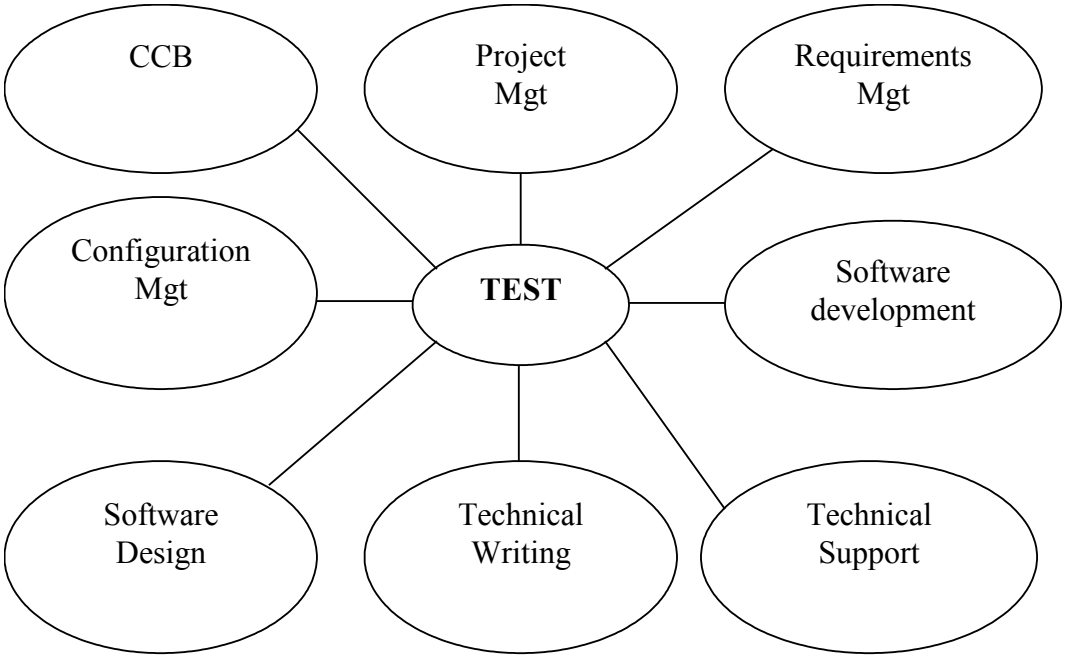
- They provide a measurable state of product readiness to progress to the next stage
- Exit criteria provide focus on the delivery requirements from the delivering team
- Entrance criteria provide focus on acceptance requirements from the receiving team
- The quality gate is shown as a milestone on the project plan and is on the critical path
- It prevents the false economics of progressing the product before it is ready in response to external pressures
- It confirms that all planned tasks have been completed prior to product promotion to the next stage

2.3 Relationships – pre-requisites for thorough testing

Important in general is:

- Know the development life cycle and plan your testing to fit in, for example: phased releases have cumulative test cases and more regression
- Ensure that project, requirements and configuration management are in place and operational
- Establish early contact with all parties from the other project areas, for example configuration management, development, project management, users, customer, …
- Foster two-way open communications and team-working

The test process does not exist in isolation in a project, but rather works together with other processes, teams:



2.3.1 Structured project management

This means:

- Documented project plan
- Defined life cycle
- Identified milestones and deliverables
- Estimation and scheduling according to procedure
- Project risk management
- Tracking of effort, scheduling and risk
- Stored measurement data

The project plan is usually a high level plan made up from detailed plans like development, test and integration plan. Plans must be maintained by the owner and include:

- Monitoring of progress
- Milestones to focus of critical tasks
- Display of dependencies
- Individual resource allocation

Project management is ultimately responsible for all deliverables and needs to manage and communicate changes.

2.3.2 Structured requirements management

Without managed requirements, test designs are almost impossible to make!
This means:

- Requirements are documented
- Changes to requirements are managed
- Requirements are reviewed
- Requirements are the basis for plans and activities

2.3.3 Structured configuration management

Without configuration management you have no idea what has been changed and needs to be tested. Structured configuration management has correct processes and controls in place concerning:

- Release management
- Build management
- Configuration management
- Fault and change management (Change Control Board)
- Documentation control

Symptoms of bad Configuration Management:

- Faults that have been fixed earlier re-appear
- Expected files are not included in the release
- Unexpected changes are included
- Features and functions disappear between releases
- Variations in code functionality on different environments
- Wrong code versions delivered
- Different versions of baseline documentation in use
- Code delivered does not match release note
- Can't find the latest version of the build
- No idea which user has which version
- Wrong functionality tested
- Wrong functionality shipped

Etc…

2.3.4 Software Development

Life cycle has a big impact on testing. Phased releases have cumulative test cases and more regression. A small change can mean a lot of testing, re-testing and regression testing. Development (effort) and testing (effort) are related but the relationship is not always linear (directly proportional). Also agree release time, content and procedures. Interact regarding faults and change management.

2.3.5 Technical Support

Testers can ask Technical Support questions regarding live faults, quick fixes and (potential) workarounds. Testers can improve their tests by finding out from Technical Support about faults found in the real world. Testers can help Technical Support by carrying out priority regression testing.

2.3.6 Technical Writing

Technical authors can gain information on the system operation and performance by working with the test team. The test team can validate any products, e.g. user guides. They can help Technical Authors understand how the product will be used.

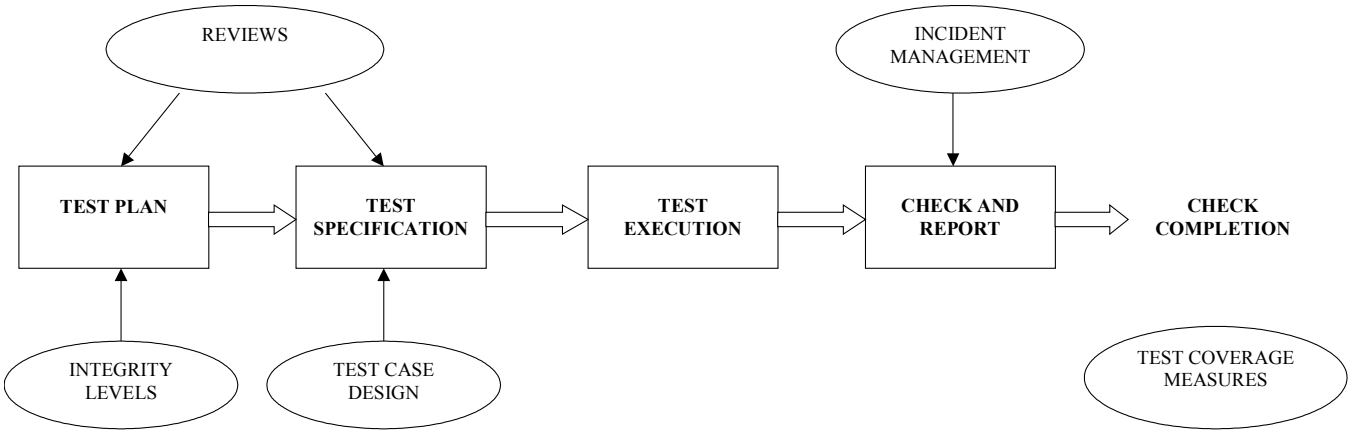
2.3.7 Software Design

During test case design, numerous queries will arise from a technical design nature, which will require confirmation. Including a designer in the review process for testing is useful as it can often highlight areas that have been missed or are considered to have insufficient coverage from a technical design viewpoint.

3 Test process

The generic test process: PSERC

Planning
Specification
Execution
Recording and Checking
Completion checking



3.1 Documentation Overview

Test Process	Test Deliverables
Test Planning	- Test plan
Test Specification	- Test design - Test cases - Test procedures
Test Recording & Checking	- Test log - Incident reports
Test Completion Control	- Test summary report

3.2 Test Planning

Purpose:

- Focuses the mind on what is required
- Includes high-level and detailed test plans
- Should include time for the planning activity itself
- To get a clear scope, objectives, prioritization and risk focused testing.
- Define an approach per test level
- Includes risk analysis
- Describe the test techniques to be used
- Includes milestones, budget, schedule

Basis for test planning should be the test management documentation. This includes:

- Test policy
- Test strategy
- Project test plan
- Phase test plan

Each document varies in strategic aim; use level of detail and content

3.3 Test Specification

A specification is a detailed description of something to be done or made …

3.3.1 Analyses

- Identify (and review) test coverage items, e.g. test situations, conditions, attributes, state transitions, etc.
- You will need:
  - Functional specification
  - Knowledge of the software specification
  - Alternatively when formal docs are missing: access to business- end technical experts (test oracles)
- To be reviewed for testability
- !!! If you cannot identify the expected outcome, then you cannot test!!!
- Test Analysis is an approach refinement to the Test Plan
- Contains:
  - Test unit vs. test techniques table
  - Identification of the test coverage items

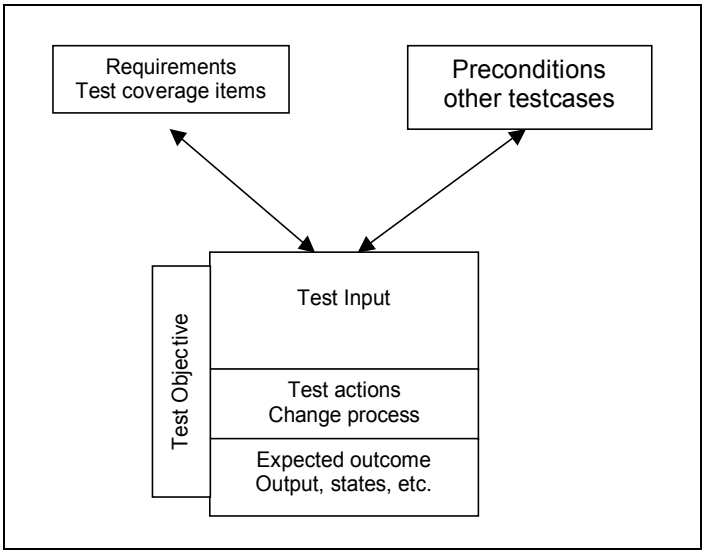
Note: Test Specification will also be needed for non-functional testing (different techniques/skills)

3.3.2 Design

- Create test cases that exercise the test coverage items
- Test design specification, test case specification (logical test cases)

3.3.3 Build

- Describe the test procedure (Physical test cases)
- Organize test data, test environments for the test cases
- Define test execution scenario



Test cases must be:

- **Measurable**
- **Objective**
- **Repeatable**
- **Risk prioritized**

Detailed test procedures ensure that:

- “No” expertise is required for execution, so other resources can assist
- The tests are repeatable (also helpful for development!)
- They can be used as test logs if used to record the test results

The test execution schedule shows:

- What test procedures are scheduled for execution during the test phase
- The planned execution week for each test procedure
- Dependencies between the test procedures
- Who will carry out the test
- High level timings
- Test progress milestones
- Decision points

The test execution schedule is highly dynamic and will require regular updating during test execution.

To complete the test case specification:

- Test details – aim, coverage
- Basis of the test case – baseline docs, risks, use
- Information required – initial state, data actions, expected outcomes, expected results, finish state

Risk analysis and Test planning may imply that you have to set priorities in analyzing and designing test cases!

3.4 Test Execution

As first step of the test execution, the following activities are executed:

- Verify test procedures and scripts
- Verify test environment
- All identified test roles are captured by persons
- People that have to execute the tests are available
- All process are operational (defect reporting, Configuration Management, etc.)
- Pre-test (most important functionality)
- Compliance check against entry criteria

For multiple reasons it can be advantageous to plan a handover period where a development representative sits in the early test stages after handover:

- Visual confirmation of any installation problems
- Development witness major problems first-hand
- Development gains confidence in testing
- Tester becomes familiar with the product under guidance
- Fosters development/test team building and co-operation

During test execution the following activities are executed:

- Test execution while following the detailed test procedure
- Execution time is logged
- Suspension/resumption criteria are applied
- Further test specification in cases like:
    - Component is buggy
    - Insufficient test coverage
    - An assumption that was made has changed
    - New/changed risks
    - Faults in the test design or procedure

3.5 Test Recording and Checking

Important: accuracy!

Activities:

- Compare actual with expected results
- Log and analyze any discrepancies (incident / defect management)
- Establish where the discrepancy lies
- Plan re-testing to confirm removal of faults
- Keeping track of the test using Test record

Discrepancies or defects can be caused by errors:

- In the test preparation
- In the product under test
- In the baseline documentation

Contents of test record:

- Identification of the item under test (including version)
- Test environment details
- Test procedure id's and version
- Test execution details (date, time, user)
- Test results (# passes and failures)
- Coverage measurements
- Anomalous events
- Incident report id's

Test record can be used to:

- Produce metrics
- Identify common problems
- Defects in the same area
- Number of defects found in a phase

3.6 Test Completion Checking

Test completion criteria are Criteria for determining when planned testing is complete, defined in terms of a test measurement technique.

Alternative names: exit criteria / release criteria (BS7925/1)

The criteria are documented in the Test Plan and should be specified for each test level/phase.

It's very good to use completion criteria as input to schedule and monitor the progress.

Test completion answers questions like:

- When do we stop testing?
- When have you finished?
- Have you done enough testing to meet your obligations to the project?

Use of test completion criteria:

- Help to focus testing on the test objectives
- Allows end position to be identified
- Allow progress towards the end position to be measured and monitored
- Prevent 'wrong' decisions being made under (time) pressure
- Tell when you have finished the testing as agreed during the planning phase

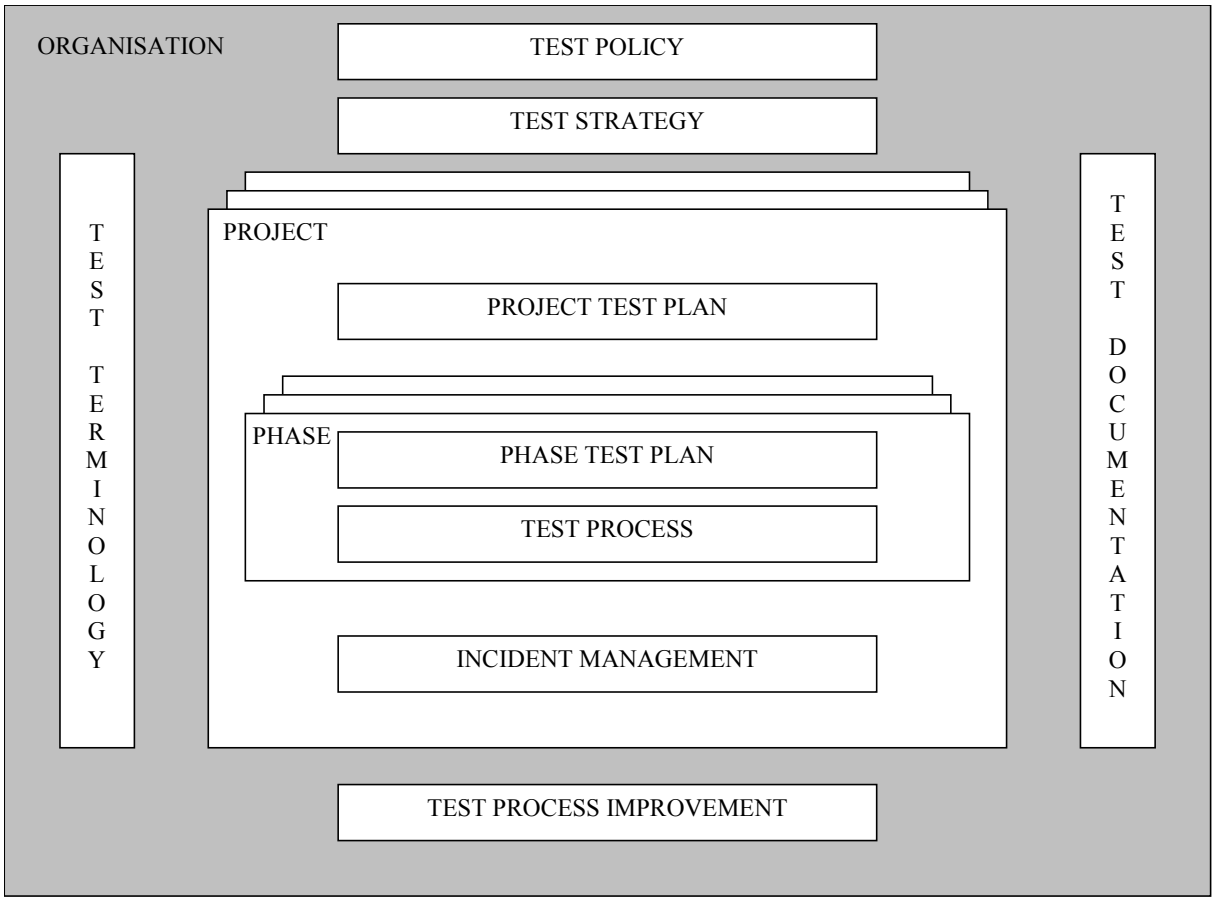
Examples:

- SC 90% for highly critical items
- SC 70% for low critical items
- Overall requirement coverage 80%
- All high priority tests running at 100%
- All outstanding, priority 1 and 2, defect reports have been fixed, retested and closed
- All other outstanding defect reports agreed by the business
- A test evaluation report has been produced
- Mean Time Between Failures
- SUMI user satisfaction score

When Completion criteria are not met:

- Create additional tests
- Test further
- Review the criteria, but be cautious with the relaxation. And if relaxation: justify, document and sign off the risks

4 Test Management Documentation



4.1 Test Policy

A document describing the philosophy of the organization towards software testing. Test policy statements reflect the nature of the business, the risks associated with the products and market and the business attitude regarding the required quality of products and deliverables.
The test policy provides the direction for the lower level test documentation, helping to keep the testing focused on the test objectives stated.

Test policy contents:

1. **Rule set**
   Consists of the policy statements that are the key drivers for the test process. They give the controls to the process.

   Example: the company will not implement untested software into live operation

2. **Definition of testing**
   Consists of the statement detailing what the organization understands by the term testing in terms of what is required and what needs to be achieved.

   Example: Testing will confirm that the software functions as detailed in the product design documentation

3. **The organizational approach to testing**
   Description how the test function fits into the overall IT structure, the requirements towards communication and links the other teams involved in the development activities.

   Example: the company structure will allow for three distinct, independent and independently managed teams: design, development and testing

4. The testing process
The setup of the test process, usually in relation to the overall development and/or project approach.

Example: the testing approach carried out is based on the V-model

5. Evaluation of testing
The approach to evaluating the success and value of the testing activities.

Example: live defects will be evaluated to gauge test effectiveness

6. Quality level
The companies required quality levels.

Example: no more than X high-severity defects per 1000 lines of delivered code to be found in the first six months of operation

7. Test process improvement
The approach to test process improvement.

Example: The TMM plan will be applied to all test activities within the overall company-wide CMM improvement program

4.2 Test Strategy

The test Strategy dictates the test framework required to meet the test policy. It is a high level document (i.e. one or more projects) defining the test approach in terms of the test phases to be performed and the testing within those phases. The strategy defined must complement the other strategic working practices and development procedures in order to be successful.

The strategy document is not always one document, but can be comprised of multiple documents.

Characteristics of the test strategy document:

Sections
1. The risks addressed by the testing of software
2. The testing to be executed to address the identified risks

Input
1. Test policy
2. Risk registers
3. TPI information
4. Business sector, organization and structure

Details of the test phases:
The phases of testing required including the structure, content and controls for each of the phases. For each phase the following information is described:

1. **Entry and exit criteria**. Required for each phase in order to manage the process as the required products move between the test phases. The criteria ensure that all the test objectives and deliverables have been achieved for each phase.

2. **Approach to testing.** The defined approach to testing along with a rational (reasons) for its selection: top-down, bottom-up, priority driven.

3. **Test case design techniques**. The techniques that are to be considered for each phase. Component tests for example include more white box techniques than other tests.

4. **Test completion criteria.** The criteria linked to risk that must ensure that the proof exists to support the decisions made regarding the promotion of the product between phases.

5. **Degree of test independence.** Dictation of the independence of testing required to achieved the desired levels of software quality and the justification for the decision.

6. **Standards.** The standards that apply to each test phase and which standard requirements will be complied with.

7. **Test environments**. The test environments to be used.

8. **Test automation**. The approach to test tools and their use.

9. **The degree of reuse of software**. Dictation to what level the various software items will be reused between levels of testing, including stubs, drivers, test harnesses, automated scripts and test data.

10. **Retesting and regression testing**. Examples are scheduling of releases, timing and contents, risk assessment.

11. **The test process to be used**. V model or other models, including the details of the test process that re applied and procedures and templates to be used.

12. **Measures and metrics**. Details of the measurements to be made, the data that needs to be captured at each specific point, the analysis of data and the use of the results.

13. **Incident management.** The strategy for managing incidents, the process, responsibilities and tools.

100% testing of software is impossible, which is why a test strategy is developed, determined by the risk analysis. The aim is to focus the most testing effort on the areas where defects are most likely to be found, or where the defects are most likely to have the most impact, or both.

4.3 Project Test Plan

The Project Test Plan describes how the overall test strategy will be applied to a specific project.

Characteristics:
- Focus on a particular project: confirm compliance, or explain non-compliance with the Test Strategy
- Linked to the Project Plan
- Identification of project testing milestones, likely costs and schedules (high-level)
- Coordination between test levels

Typical details:
1. Objective and scope
2. Risks and mitigation
3. Phases and deliverables, including
    a. Relations between phases
    b. Entry/exit criteria per phase (when to enter next phase)
4. Organization
5. Infrastructure
6. High-level schedule

Contents:
- Time scales for the required testing
- Assess the cost for the project
- Assess the test resource requirements
- Levels of testing required
- Number of test cycles
-
- Contributions, roles and responsibilities for everyone involved.
- Identification of project deliverables to be tested

Goal:
- Satisfy management and users that adequate testing will be performed

4.4 Phase Test Plan

A phase test plan is a document providing detailed information on testing within a specified phase of the project.
(E.g. system test, integration)

Contents:
- Modules, functions, processes to be tested
- Details of business risks associated with those functions
- Test coverage requirements
- Test techniques to be applied
- Methods and tools
- Test specification and execution requirements
- Which environments will be used
- Details the test schedule
- Details of any variations from the project test plan or the test strategy.
- ….

Test Phase Plan is the lowest level of Test Management Documentation – on this level; tracking is possible as it contains a detailed structure.

5 Test Plan Documentation

Prescribes scope, approach, resources and schedule of the intended testing activities.

For:
- Management/Marketing: information on cost, constraints, timing, level of quality reachable in timeframe
- Customers/Users: information on level of quality that can be expected, define user involvement and user resources
- Test team: information on goals, responsibilities, approach
- Development team: information on (measurable) expected quality level

A test plan will vary, e.g. project test plan, phase test plan, development test plan, release test plan, performance test plan, …

Test assignment as input for the test project:
- Test objectives
- Expectations
- Sponsor
- Responsibilities test team
- Scope of test
- Constraints (project planning, policy, strategy, standards, time, resources…)

Notice: the assignment must be written by the person giving the assignment, so he/she is directly involved. If he/she does not do this, an interview is possible, but only with open questions!

Important things to remember:
- The chapters are linked, build the new chapter on the previous
- KISS (keep it simple and short)
- Mind your readers
- The test plan is a tool, not a product
- Use measurable criteria
- Keep exploratory and ad-hoc testing possible

5.1 Test Plan Outline (IEEE-829)

1. **Test Plan identifier** – unique number, version, identification when update is needed (for example at x % requirements slip), change history

2. **Introduction** (= management summary) – in short what will and will not be tested, references.

3. **Test items** (derived from risk analysis and test strategy), including:
   - Version
   - Risk level
   - References to the documentation
   - Reference incidents reports
   - Items excluded from testing

4. & 5. **Features to be tested** (derived from risk analysis and test strategy), including:
   - Detail the test items
   - All (combinations of) software features to be tested or not (with reason)
   - References to design documentation
   - Non-functional attributes to be tested

6. **Approach** (derived from risk analysis and test strategy), including:
   - Major activities, techniques and tools used (add here a number of paragraphs for items of each risk level)
   - Level of independence
   - Metrics to evaluate coverage and progression
   - Different approach for each risk level
   - Significant constraints regarding the approach

7. **Item pass/fail criteria** (or: Completion criteria), including:
   - Specify criteria to be used
   - Example: outstanding defects per priority
   - Based on standards (ISO9126 part 2 & 3)
   - Provides unambiguous definition of the expectations
   - Do not count failures only, keep the relation with the risks

8. **Suspension and resumption** (for avoiding wastage), including:
   - Specify criteria to suspend all or portion of tests (at intake and during testing)
   - Tasks to be repeated when resuming test

9. **Deliverables** (detailed list), including:
   - Identify all documents -> to be used for schedule
   - Identify milestones and standards to be used

10. **Testing tasks** for preparing the resource requirements and verifying if all deliverables can be produced. The list of tasks is derived from Approach and Deliverables and includes:
    - Tasks to prepare and perform tests
    - Task dependencies
    - Special skills required
    - Tasks are grouped by test roles and functions
    - Test management
    - Reviews
    - Test environment control

11. **Environmental needs** (derived from points 9 and 10) for specifying the necessary and desired properties of the test environments, including:
    - Hardware, communication, system software
    - Level of security for the test facilities
    - Tools
    - Any other like office requirements

12. **Responsibilities** for managing, designing, preparing, executing, witnessing, checking and resolving, including:
    - Groups responsible for proving test items and environmental needs

13. **Staffing and training needs**, including:
    - Staffing needs by skill level (more than helping hands, a team with a mixture of skills with domain knowledge)
    - Training options
    - Extracted from Testing tasks and Responsibilities
    - Extracted from Environmental needs

14. **Schedule**, also used for communication to other team plans. The schedule is derived from Testing tasks and Staff and training. The schedule includes:
    - Milestones of the project and transmittal events
    - Additional test milestones
    - Estimated time for each test task (don't use man-month)
    - Schedule for each task, test milestone
    - Periods of use of each testing resource

15. **Project risks and contingencies**, for identifying the risks with each component, task, schedule and to be prepared. The risks include:
    - High-risk assumptions (**notice: assumptions and conditions as in TMap are not used in IEEE 829, they are risks if they have not become available…**)
    - Likelihood, impact and trigger
    - Contingency plan for each risk

16. **Approvals** for getting everybody to 'buy in' and avoid discussions in the future when defects are found or testing is finished. Make sure to:
    - Specify names and titles of all that must approve the plan
    - Contact each stakeholder and explain the purpose of the plan
    - Explain the contents using thorough walkthroughs or individual meetings

6 Testing and product risks

6.1 Definitions

Risk:
Is a factor that could result in a future negative consequence.

Risk Identification
What is the risk, where is it?

Risk Analysis
What is the likelihood and the impact?

Risk Mitigation
Try to reduce the likelihood …

Risk based testing:
What could go wrong if the testing fails or is not executed? Risk is about prioritization of the tests we can run within a series of limitations of time, cost and quality goals. We must be aware that risk based testing is applicable to both new products and maintenance.

Risk is measured by a combination of likelihood and impact

6.2 Risk identification techniques

1. **Expert interviews**. Interviews with experts when key stakeholders are not available. Disadvantage: lack of cross-fertilisation of ideas

2. **Independent assessment**. Use of external experts (can still be within the same company) that are experienced with the identification of risk, but have no vested interest in the viability, delivery or commerciality of the product.

3. **Risk templates**. Documents for recording the risks during identification.

4. **Lessons learnt**. Issues recorded for other aspects of the system or project under test. The lessons learned should focus on establishing areas where certain types of risks occurred that were not previously identified or considered.

5. **Risk workshops**. Organized events managed by a facilitator that can be one-time of in a series. Ideal is to use it as a brainstorm session while using checklists. By using several perspectives a complete picture of the risks is gained. (FMEA technique)

6. **Brainstorming.** Multi-disciplinary teams thinking about risks, where no one or no aspect of the project is in the majority.

7. **Prompt lists and checklists**. Prompt lists stimulating people to think about the areas where risks should be considered and also give an indication of the type of problems the system could be subjected to. Checklists are more detailed lists of risks that can be addressed.

8. **Fault prediction**. Establishing the likelihood of an error in the components of the system and perhaps the type of error that are expected. Can be done for example using fault trees or predictive techniques and models.

9. **Self-assessment**. Assessment of risks after using one the previous techniques or using your own knowledge.

6.3 Risk analysis

Identification of likelihood and impact; looks into complex and potentially error prone areas

If risks cannot be calculated quantitively (impact x likelihood), risks can be defined qualitatively (based on perceived risk).

If testing is cut short (or not done) then the risks that have not been addressed will remain.

Risks first have to be identified through analysis of the situation. It is easy to identify risks, but the difficulty is to identify the risks that matter. Then risks must be monitored and kept updated.

The users are ultimately responsible for the risk. They will have been involved in the identification stage, the prioritization, the analysis and mitigation exercise. The testers only guide the users; they do not 'take over'.

For release to production the testers role is to ensure that the information presented to the users is objective, accurate and focused on the right aspects of the system.

Differentiated Test Approach

- *Target Settings:* Different risks can be tested with different techniques, coverage, …
- *Priority Settings:* Give priority to high risk areas
- *Describe the risk:* Remaining risks in the delivered system (open risks)

6.4 Risk mitigation

Mitigation of risk is linked to the categorization and classification of risks.

Risk mitigation alternatives:

- Do nothing (cost/benefit balancing)
- Share
- Preventive action
- Detective action
- …and put contingency in place, should it happen (corrective action).

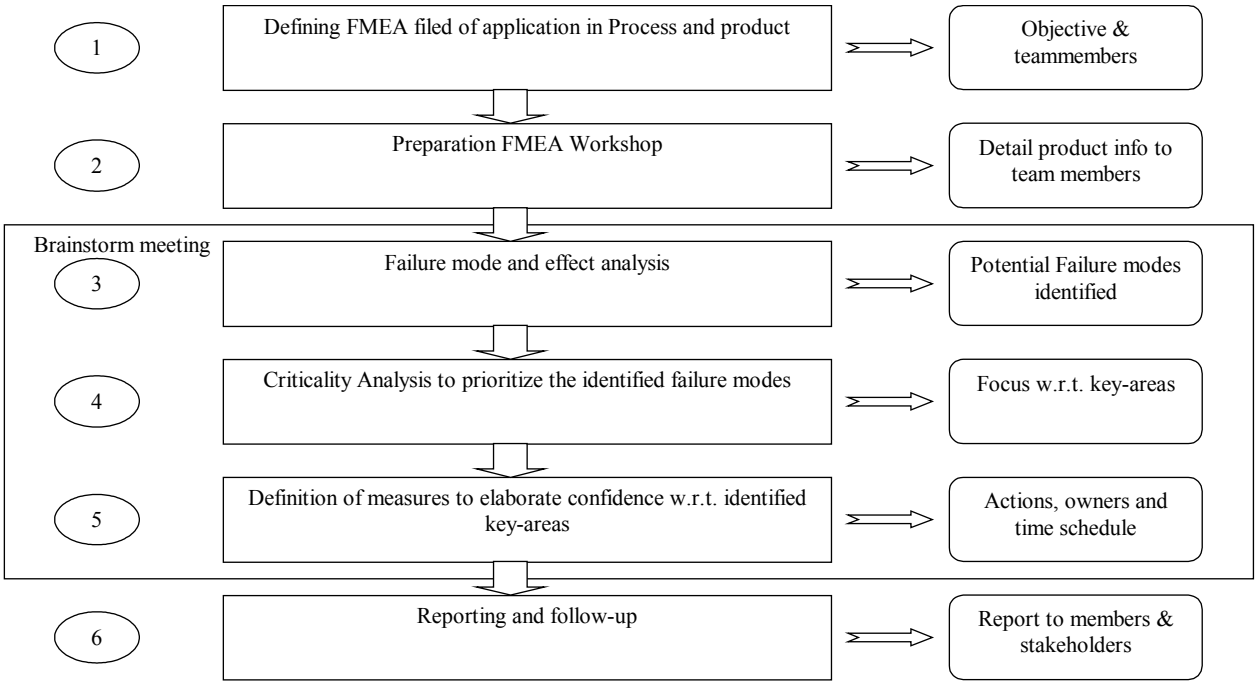
Try to reduce the likelihood of the risk.

Use the differentiated test approach:

- Test design techniques (BS-7925/2)
- Static testing (more risk means more review)
- Completion criteria
- Most experienced person
- Priority settings
- Re-test
- Regression test

Different techniques focus on different defects!

6.5 FMEA process



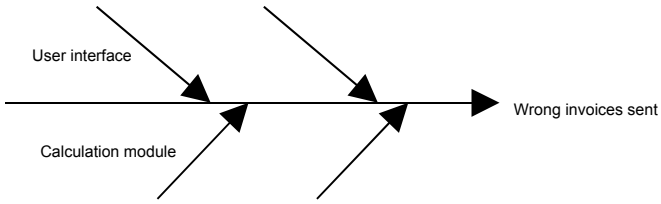
6.5.1 Risk identification

Identifying all factors (especially internal and external business factors) that can go wrong, including all causes for those factors. A possible result is a Ishikawa or Fishbone diagram

6.5.2 Critical Analysis

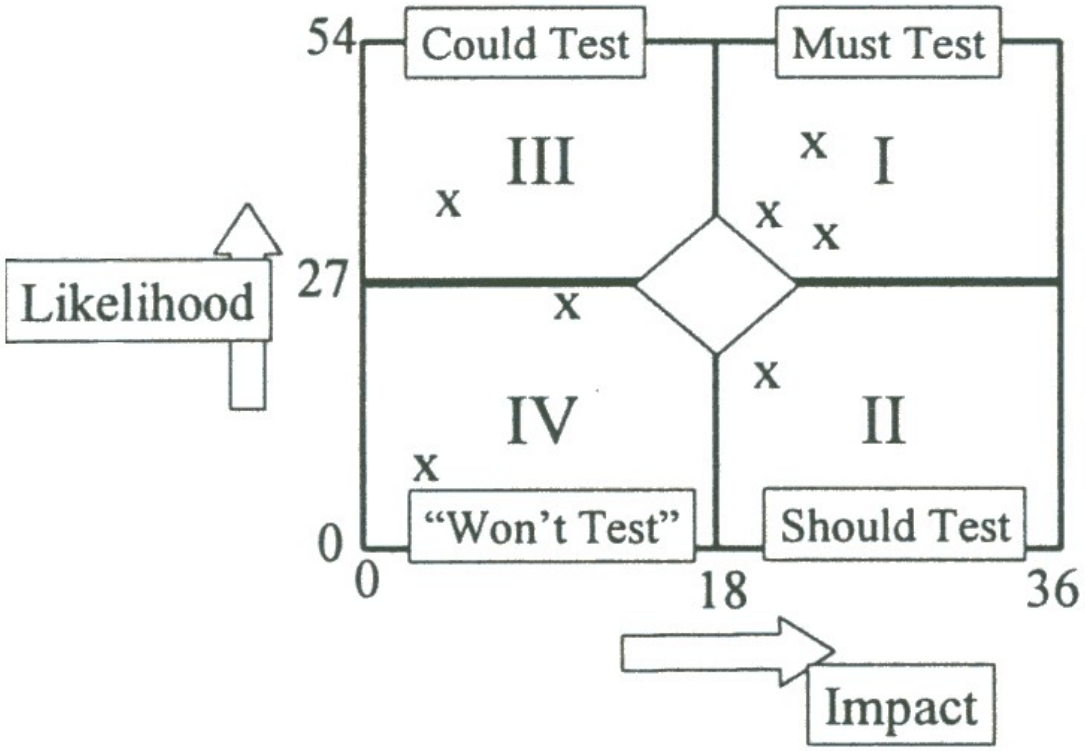
After identification of risks using FMEA, the criticality of risks can be determined. The term used is RPN (Risk Priority Number). The RPN is the result of Impact x likelihood. Each impact and likelihood of a risk receives a number between 1 – 10. An example of a scale is 2, 4, 6, 8, 10. Each number in the scale receives a meaning. Then for each risk the impact x likelihood is calculated. For example 6x8. The total results receive a meaning also using a scale.

Notice that risk is usually not calculated, but perceived by the stakeholders.



6.5.3 Stakeholders involvement

- Identify the stakeholders (project manager, architect, senior tester, user, …)
- Ask them to fill in the risk table
- Perceived likelihood and impact (their views will differ)
- Establish consensus, through a meeting
- Make the functional risk matrix



Risk Matrix

6.6 Likelihood vs. Impact

For identifying risks and their priority, common factors can be considered.

Likelihood on defects is determined by:
- Complexity
- New development/level of re-uses
- Interrelationships (# interfaces)
- Size
- Technology
- Inexperience (of development team)

Impact:
- User importance / selling item
- Financial or other damage
- Usage intensity
- External visibility

For each function the likelihood and impact is determined using the factors and a cross-table is made. For graphical representation a risk matrix can be made.

Using the results for the test strategy an example would be:

70% statement coverage pair inspection	formal test spec, BVA, 90% statement coverage
free format	70% statement coverage

Example low level test

EP	Process cycle test C/E graphing
Error guessing	Use cases EP

Example high level test

6.7 Non-functional aspects

Non-functional risks should also be considered. The ISO 9126 lists the most common used level of attributes: 6 attributes. Every attribute has a number of sub-attributes, which leads to a total of 21 sub-attributes:

- Portability
  - Adaptability
  - Installability
  - Co-existence
  - Replaceability
- Efficiency
  - Time Behavior
  - Resource Utilization
- Reliability
  - Maturity
  - Fault tolerance
  - Recoverability
- Functionality
  - Suitability
  - Accuracy
  - Interoperability
  - Security
- Usability
  - Understandability
  - Learnability
  - Operability

    - Attractiveness
- Maintainability
    - Analyzability
    - Changeability
    - Stability
    - Testability

For identifying these attributes a Tool structured questionnaire can be used. With the questionnaire, users answer questions in their business language and the answers are used for deducting the relevant attributes.

The result is a quality profile, with for each quality attribute:

- Selection of sub-attributes
- Risk level
- Context description

Risk and metrics – tracking the lifecycle of the risks provides metrics in a number of areas:
Number of tests
Initial classification to final classification
Cost to fix
Status at implementation
Live failure comparison

6.8 Risk and test strategy

The stakeholders (like end-users, system managers, live support, etc.) are best placed to assess the likely business impact of a risk.

The project team members are best placed to assess the likelihood of an error occurring.

The test strategy describes how the test stages and techniques are used for addressing specific types of risks. Testers have the responsibility to ensure that the test coverage for any identified risk is in line with the priority of that risk.
Risks determine:

- The thoroughness of testing
- The type of testing
- The testing priority: give higher priority to testing of higher-risk areas

!!! Risk analysis is useless if you eventually test everything in the same manner!!!

6.9 From test strategy to techniques

Risk level and –type determine the technique:

- Often expert-based decision (many selection factors)
- Decide, founded on type of system and expected results
- Combine different views!
- Assign them to a test level
- And define a technique table

Selection factors for test techniques:

- Risk level and type
- Test objective
- Available documentation
- Knowledge of testers
- Time and budget
- Supporting tools
- Development life cycle
- Product life cycle
- Previous experience on (type of) defects
- Measurements on test effectiveness
- International standards
- Test point analysis

- Customers'/contractual requirements
- Auditability / traceability

Notice: These kinds of selection factors are ideally recorded in a test strategy document.

!!! A shortage of testing at an early stage (component) cannot be corrected at a later stage (system) – faults will remain!!!

Subsume ordering: where one technique can be shown to include criteria in another, it can be said to subsume the other.

- Error guessing → Use Cases → EP → ECT or C/E Graphing
- Error guessing → Use Cases → PCT (1) → PCT (2)
- Increasing levels of switch coverage
- Free format → x % SC → y % SC → z % SC
- BVA and Syntax are very powerful at low level

Limitations to subsuming:

- Cost effectiveness not addressed
- Ordering only applies to a single criterion
- If one technique covers another it does not mean that the technique is suitable for all types of defects

Results from research on test effectiveness: better results when the attention for testing is not focused on one test level, but to several: component, inspection, ST, AT.

7 Testing and Project Risks

In general:

- The same 3 steps and approach as with product risks
- Initially carried out at test planning phase
- Tracked throughout development and test process
- Risk identification often supported by a risk identification checklist (i.e. TMAP)
- Risks can be internal or external to the test process (dependencies)
- Use a checklist
- Identify using a risk workshop or interview experts

The risk table is part of the test plan and periodic test progress and quality report.

For each risk, the risk table identifies:

- Chance (high, medium, low)
- Impact type (product quality, lead time, effort)
- Impact size/effect (high, medium, low)
- Trigger (how will we know it's happening)
- Contingency actions

For risk mitigation, discuss the risks thoroughly with business and project management.

Track identified mitigation actions.

Update the test plan and schedule when Murphy strikes.

8 Test Estimation and Scheduling

8.1 Benefits and difficulties

Test estimation benefits:

- Makes the consequences of the test strategy visible and tangible in terms of effort, cost and lead time
- Risk-based discussion with management!
- Well-founded resource claims
- Evaluate influencing factors and measure productivity – improve test process

Why estimation is difficult:

- Requirements incomplete / ambiguous
- Many influencing factors
- Test input quality unknown
- Changing technology and processes

In general: uncertainty due to a lack of information, that's why all assumptions must be documented and checked as information becomes available.

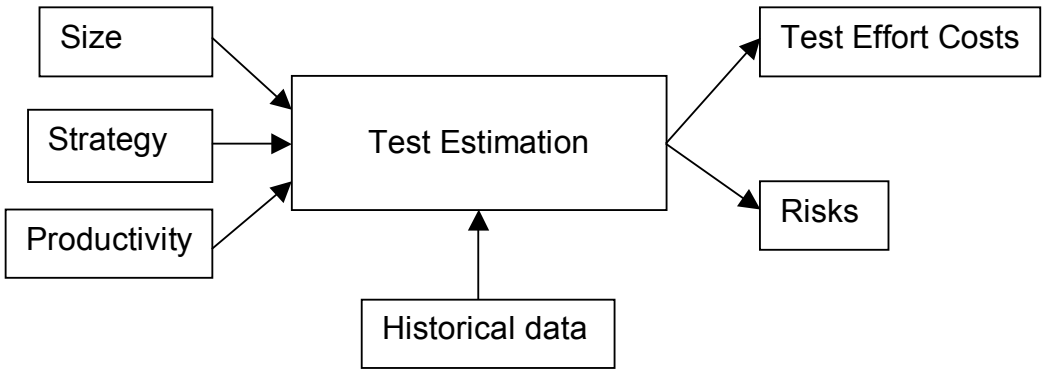
Estimates for test execution:

- Include estimation for rework!
- Use average number of re-tests from previous projects.
- What about estimation for tools and test environment?

Risks management around estimates:

- Include a contingency budget? 20%?
- Don't give in easily when the schedule is under pressure:
  - Relax entry criteria, but understand and communicate the risks involved
  - Add staff (but do they have the right skills?)
  - Reduce test execution time, accept lesser quality (relax exit criteria) → risk based, or overall lower coverage
  - Reduce the number of features in the product

8.2 Estimation cornerstones



8.3 Methods for estimation

8.3.1 Top Down methods

From projects to phases. Then breakdown between phases etc. Usually however, the arguments for the estimation and breakdown are not strong.

8.3.1.1 Related to software development

Testing tasks related to software development.
Example: testing as 30% or 40% of development effort.

8.3.1.2 Metrics

Several formula based estimation approaches exist:

- number of test cycles or time per test phase based on historical data.
- TEEM - A method for mature organisations where measurement rules are used for estimation. These rules are determined by influencing factors: size, strategy, environment and productivity. Rules are adapted for each project. The more projects are run, the more reliable the rules become.

8.3.1.3 Function Point Analysis

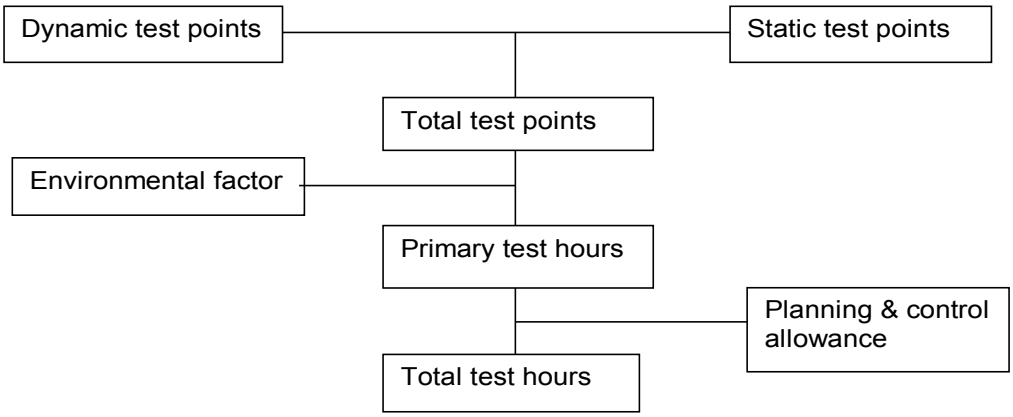
Size measurement of the system based on a user level. The complexity per function is determined.
Examples: IFPUG, FPA.

8.3.1.4 Test Point Analysis

Technique using function points as a base and then estimating the time for all testing activities.

Needed:

- Size of test object: # function points, corrective factors (complexity, interfacing, uniformity)
- Importance of subsystems/functions (user importance, usage intensity)
- Quality attributes: usability, efficiency, …etc., risk level
- Productivity: Productivity figures per organization, skills test team, historical data. Productivity never higher than 70% due to meetings part-time work, etc.
- Environmental factors: test basis, development test, tools, environment, …



TPA can still be used when no function points exist
Estimation without function points:

X requirements critical * A * 1,25 = …
Y requirements normal * A * 1,00 = …
Z requirements low * A * 0.75 = …

…with A = average time to test a requirement

To estimate testing time, apply environmental and productivity factor, plus planning and control allowance.

8.3.2 Bottom Up methods

Bottom up method where the project is divided into it's smallest tasks, each task is estimated in detail and then the total estimation for test project is calculated.

The test strategy and (standard) approach (e.g. IEEE 829/TMap) is used as a base for identifying the tasks for producing deliverables and other tasks. Usually a checklist is used for listing the tasks in the test life cycle.

8.3.2.1 Wide Band Delphi

Bottom-up technique where people with knowledge from the project use Work Breakdown (WBS) data from previous projects to estimate the needed time for tasks. The estimation is made during a meeting chaired by a Moderator. Each participant prepares the meeting by making his own estimation including assumptions.

Important with this method:

- Each task must be comprehensive, max 40 hours.
- Agreement on % maximum deviation. If all participants stay within the deviation, the average of all estimations is used. If estimates exist outside the deviation, the low and high explain their reasoning.
- The moderator gathers the data, explain the WBS and discuss requirement quality, complexity, etc.
- Each participant estimates (each task max X hours) and documents assumptions (supported by an "influencing factors" checklist)
- Moderator calculates averages and deviations and collects assumptions, then re-distributes
- Discuss outcome (based on criterion % max), If necessary, a new estimation round is carried out

8.3.2.2 Intuition / Guesswork

This is the least accurate estimation method.
Should not be used.

8.3.2.3 Previous experience / Historical data

Use metrics from previous projects to estimate the test effort.
This gives a fast, but not so accurate estimation.

8.3.2.4 Work breakdown (WBD)

From test strategy and (standard) test approach determine all deliverables to be developed
A WBD contains the tasks needed to produce the deliverables (test plan, design, script, reports) and other tasks (reviews, test environment, management and control).

9 Test Completion Phase

Activities:

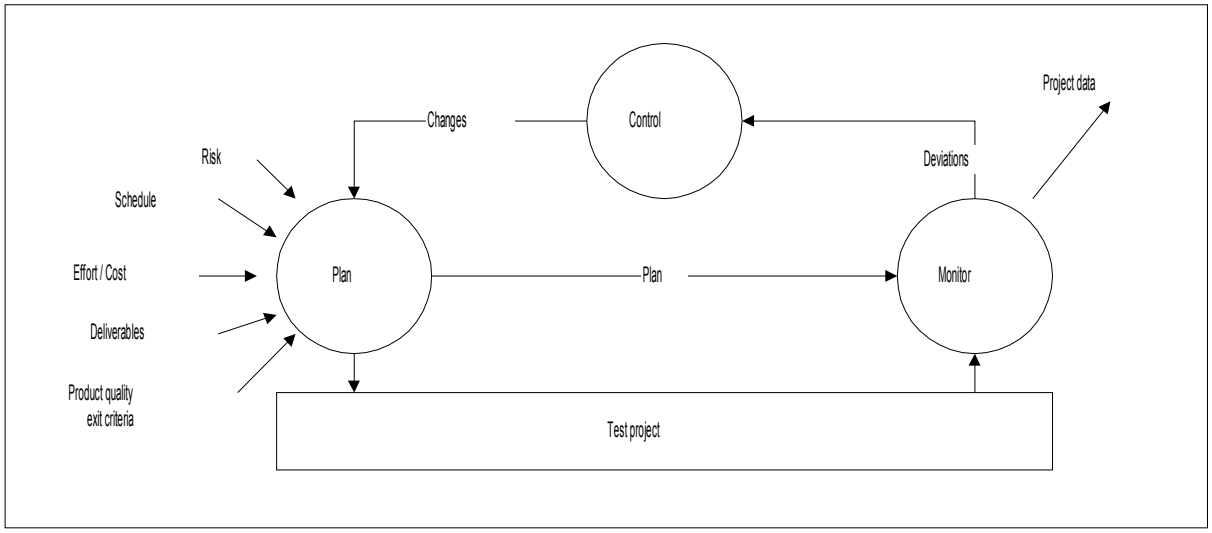
- Evaluation of test object / test process
  - Gather and complete measurement data (effort per activity, lead-time, defects, …)
  - Calculate performance indicators and key metrics
  - Interview stakeholders
  - Evaluate test process against test plan (strategy, tools, standards, infrastructure, training, procedures, reporting, …)
- Write evaluation report
- Preserve testware
- Discharge test team

Content of evaluation report:

1. Introduction
2. Test summary (objectives, test executed, major findings and results, deviations from test plan and test process
3. Product quality (incidents, non-functional metrics, quality test basis)
4. Actual vs. planning
5. Evaluation test process
6. Risk management
7. Test deliverables

10 Test Monitoring and Control

10.1 Basics of monitoring and control



Set up a monitoring and control system at the start:

- Tracking
- Reporting
- Meetings

10.2 Tracking using test reports

Test report contents:

- Report id
- Test area / level / phase
- Period covered by the report
- Management summary
- Status of progress / quality / risks (actual vs. planned)
- Other outstanding activities
- Contingency actions proposed for the next period (think along, don't just sit with your arms crossed)

When the focus shifts during the project, e.g. from effort to quality focused, the report changes too!

Preparation phase report:

- Effort spent per test task
- Effort spent per tester
- Progress on test designs (# completed vs. total # → %)
- Progress on other deliverables
- Early quality indicators (static test or component test information)
- Test process risk

Execution phase report:

- Product quality information (i.e. defects per test hour)
    - Defects, trends, completion criteria, non-functional
    - Detailed information on "showstoppers", outstanding risks
- Progress test execution
    - # (High priority) test cases executed vs. total # → %
    - # Requirements / risk items covered
- Effort per task / per tester
- Progress other deliverables
- Test process risks

10.3 Controlling the test process

Deviations? Discuss, agree, define and track corrective actions

Update plan, re-establish requirements in case of:

- Changing requirements
- Significant deviation from plan
- Pre-defined criteria

In case of slippage it is important to discuss the slippage with the stakeholders. Then review your options:

- Defer release data if not fixed
- Allocate additional resources
- Execute highest priority test cases in remaining time available
- Review test completion criteria
- Deliver less functionality

With all tracking, reporting and monitoring, the slippage should not be a surprise!

11 People Skills

Testing skills:

1. Test knowledge
2. Domain knowledge
3. IT knowledge
4. Social skills

A successful tester needs a balance in these skills.

For each test level different skills are required.
- Component test: developer
- System test: Professional tester
- Acceptance: Business expertise

Soft skills and attitude:
- Communication
- Giving/receiving constructive criticism / positive feedback
- Influencing people
- Negotiation
- Selling / promoting testing
- Stress resistant
- Be accurate
- Have a sense of humor

Essential communication skills:
1. Speak from practice
2. Be with your audience
3. Check your terminology
4. Show respect
5. Provoke interesting questions
6. Explain things that matter
7. Be quick
8. Show how everyone is involved
9. Be prepared

Belbin team roles:

Roles	**Positive / negative sides**
Coordinator	organizer/manipulative
Resource	investigator bridge builder/over run
Implementer	disciplined/inflexible
Specialist	expert/closed
Plant	innovative/sensitive
Shaper	driver/temperamental
Monitor	evaluator methodical/negative
Completer-finisher	focused/obsessive
Team-worker	popular/indecisive

When hiring a new tester think of the saying: recruit for attitude, train for excellence.

12 Reviews

12.1 Definition

Review = *"An evaluation of software elements or project status to ascertain discrepancies from planned results and to recommend improvement"*

Objective: not just to find defects, but also to find defects earlier in the life cycle and to remove the causes from the process

ROI = (C + (0.25 * M)) * 3 – X

Good outcome is 1:2 or 1:3

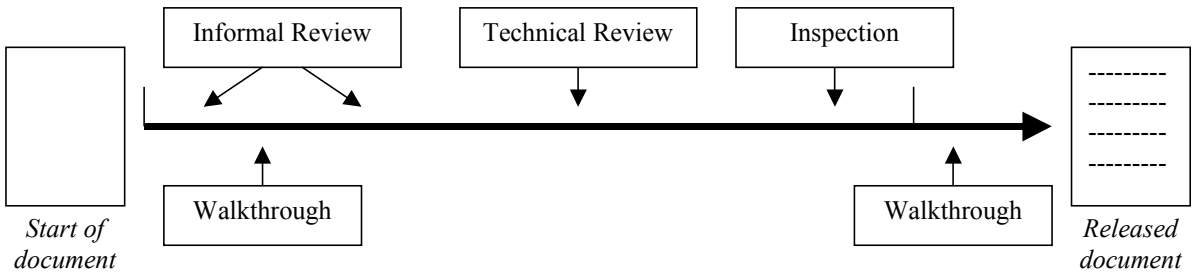
X = time spent in reviews (cost)
C = number of critical errors
M = number of majors (1 in 4 leads to failure, hence the 0.25)
3 = time to solve 1 fault found in testing (conservative!)

Traditional preparation problem	**Solution**
Non-distinct goals & responsibilities	Roles
Non-specific (all inspectors required to find defects of any type)	Rules, checklists
Non-systematic (no set of procedures to follow)	Perspective-based reading (PBR)

Key success factors:

- Focus on role, but record all defects
- Focus not only on the product
- Preparation procedures
- Checking, not reading
- Optimum checking rate
- PBR
- Severity concept – critical, major, minor

Reviews in time:



When will reviews not work?

- Quality not important enough
- Management not interested
- Untrained engineers and moderators
- Information on document quality is used by management to evaluate individual performance
- Inspections implemented in a too formal or theoretical manner
- Review/inspection remarks are given/taken personally

12.2 Implementing reviews

Actions to do before starting reviews:

- Provide people some knowledge
- Take a critical document to review
- Train the engineers and moderator
- Make a master review plan
- Improve by using metrics and feedback

When will reviews not work?

- Management support
- Amateurs at work
- Using metric for individual performance evaluation
- Too formal / theoretical implementation
- Multiple moderator at work

12.3 Types of product reviews

Walkthrough – author guides the group through a document and his thought processes, so all understand the same thing and reach consensus on changes to make (often with people outside the software project on higher levels) → *validation*

Technical review (= peer review) – group discusses a document and tries to reach agreement about the content (approach), how things will be done.

Inspection – formal individual and group checking using sources and standards, according to detailed and specific rules (checklists), support the author, find defects → *verification*

Informal review – a type of document review that doesn't follow a formal process and entry/exit criteria. Ask a colleague to provide comments. Rarely involves a meeting.

Management review – a project review in which the project status is evaluated against the plan (resources, cost, progress, quality) and appropriate actions are defined.

Audit – independent evaluation of project and product using defined criteria (e.g. internal audit by QA-group)

12.3.1 Informal Review

- Formality : Informal
- Primary Goal : Finding faults
- Secondary Goal : None
- Entry Criteria : None, document is presented in its current state, does not have to be finished.
- Meeting : None
- Process Leader : Author
- Optimum team size : 2
- Preparation : None
- Materials : Document to review
- Participants : Anyone
- Logging : Informal notes
- Exit Criteria : None
- Follow Up : Informal
- Strengths : Quick and cheap
- Weaknesses : Find less faults than other methods
- Applicable for : Unfinished documents or prior to other formal review with more people.

12.3.2 Walkthrough

- Formality : rather formal
- Primary Goal : Finding faults
- Secondary Goal : Gather information, explain and evaluate content, common understanding, establish consensus, education of reviewers.
- Entry Criteria : Product and source documents distributed. Should be read but not formally reviewed. No formal criteria.
- Meeting : Either to check direction of document or to explain contents to others.
- Process Leader : Moderator (often Test Manager)
- Optimum team size : 5 - 8
- Preparation : Not extensive. Defects are found during the meeting. Roles are not needed.
- Materials : Document to review
- Participants : Peers, technical leaders and participants from outside the software discipline.
- Logging : Separate scribe.
- Exit Criteria : No formal follow up as a later review is likely.
- Follow Up : Moderator checks whether action items are closed.
- Strengths : Education of participants, sharing of knowledge, useful for non-tech staff.
- Weaknesses : Finds less faults than other technical reviews and inspections.
- Applicable for : Unfinished documents in an early stage, to check if direction is desirable. Nearly finished documents must get a deployment agreement.

12.3.3 Technical Review

- Formality : less formal
- Primary Goal : Finding faults
- Secondary Goal : Assess concepts and discuss alternatives, establish consistency, get feedback, inform participants.
- Entry Criteria : No formal criteria (document maybe 50-80 % complete).
  Kick-off is very important; author can explain what (not) to review.
- Meeting : Logging and discussion not separated. Check only criticals and majors, keep track of people issues.
- Process Leader : Technical leader will act as moderator.
- Optimum team size : 4 - 6
- Preparation : Roles based on specific question of the author. Not too many pages.
- Materials : Document to review.
- Participants : Experts, peers and sometimes project leader.
- Logging : Author.
- Exit Criteria : Author updates the document. Technical leader verifies.
- Follow Up : Expert-based decision regarding the follow-up.
- Strengths : Good for sharing knowledge with technical peers, making strategic technical discussions.
- Weaknesses : No reference to external documents.
- Applicable for : Unfinished documents, to check technical content.

12.3.4 Inspection

Steps of the Inspection:

- Planning
- Kick off
- Review
- Logging meeting
- Rework and follow-up

12.3.4.1 Planning

Consists of:

- Entry evaluation (textual errors, status of the document)
- Define inspection strategy
- Gathering documents
- Invitation and planning

For the inspection strategy it is import to:

- Know the objective of the inspection like effectiveness, efficiency, common understanding, etc.
- Decide on the team size, roles and assignments
- Decide what part of the document must be inspected (**chunking)**

Perspective based reading

The role that each participant will play determines for a large part the success of an inspection. Each participant uses a certain perspective to review a document (PBR).

In general there are 4 types of roles and each role uses **rules** for the inspection. These rules tell the participant what the focus is of the review. Terms used here are:

- Abstraction model = What = do I get the information I need
- Usage model = How =does this info help me
- Reading technique as a result of the Abstraction and usage model = inspection procedure for accomplishing a particular task.

The roles participant can have are:

- Type 1: Higher level documents
- Type 2: Compliance to procedures and standards
- Type 3: Related documents
- Type 4: Content of the document under inspection from the view of the documents users.

These roles are then customized using PBR for each project / document that is reviewed and a checklist is made.

12.3.4.2 Kick off

With the kick off the materials are distributed among the participants and the moderator explains the inspection procedure. Think of the roles, as the forms to use.

Especially a standard review form must be explained so that everyone knows what and how to inspect, who the participants are and record the results including time.

Also important to explain: the checking rate, i.e. the pages per hour. This is the most critical factor for good results.

Also important are the severities that the participants must understand and use. The severities are links to the rules. For example: the Type 4 reviewer may be more concerned with the downstream damage then a Type 2 reviewer.

12.3.4.3 Review

Each participant reviews the document. Important is that they:

- Review using their perspective, rules and checklists
- Focus on the important defect (e.g. not textual errors)
- Follow the checking rate
- Use the review form / defect logging sheet

The moderator collects the defect logging sheets, merges them and sends it to all participants. Double defects are kept double, because a participant may mean something else in the defect.

Possibly handy is the use of a central tool. It doesn't replace the logging meeting, but the scribe is not needed and the moderator can participate as reviewer. Example: ReviewPro.

12.3.4.4 Logging meeting

At the start of the logging, the moderator collects the execution data from the reviewers like time used, defects found, logging rate, defect density. Based on this the moderator can decide to cancel the meeting if the review was not successfully executed.

Then the logging starts, either using a central list of all defects of all reviewers, the lists that each participant brings with him or the defects are logged during the meeting.

After logging it is decided what the follow up actions are:

- Accepted as is
- Update and start new review process
- Update and only check by moderator afterwards (only when document complies to the exit criteria).

Important also are the completion criteria. The criteria are product related (quality of the document) or process related (quality of review activities). Examples are: Criticals/majors per page and the checking rate.

Also process improvement may be relevant, think of lessons learned, mistakes made.

After the logging, **discussion** can start on relevant issues.

Then a **causal analysis** is executed to find out what caused the defects. Only a few important defects are discussed. Notice that a causal analysis is only useful if there is a function in the organization that uses the results!

12.3.4.5 Rework and follow-up

The author updates the document. If other documents need to changes, Change Requests are created. The defect logging form is used as a checklist, but that does not limit the author to make other changes.

When finished the moderator checks the changes and calls for a meeting if necessary.

13 Incident management

Steps:

1. **Recognition** – when an anomaly is found
2. **Investigation** – each anomaly is investigated to identify all known related issues and propose solutions
3. **Action** – a plan of action is formulated on the basis of the investigation
4. **Disposition** – once all actions required are complete the anomaly shall be closed

At each step there are administrative activities.

1. Recognition:
   - Recording – who, what, where
   - Classifying – project activity, suspected cause, repeatability, …
   - Identifying impact – severity, priority, customer value, impact on schedule/cost/risk etc.

2. Investigation:
   - Recording – effort, investigator name, start & end time
   - Classifying – actual cause, source, type
   - Identifying impact – review + update impact assessment from previous step

3. Action:
   - Recording – items to be fixed, require action including corrective actions to prevent reoccurrence
   - Classifying – resolution (immediate, eventual, deferred, no fix) and corrective (departmental, corporate, …)
   - Identifying impact – review + update impact assessment from previous step

4. Disposition
   - Recording – action implemented, date closed, verification method (test case)
   - Classification – closed, deferred, merged, referred to other project
   - Identifying impact – review + update impact assessment from previous step

Severity vs. Priority:

- Severity: Technical impact on the system under test
- Priority: The business impact or impact on project

Why use a standard:

- Allows standard classification of anomalies by everyone
- Allows identification of life cycle activities where most problems are introduced and to evaluate the used methods
- A reference for defining defect management system.

Above list taken from: IEEE 1044 – Standard classification for software anomalies

14 Test Techniques

Test case design technique is a method used to derive or select test cases.

They are used to ensure that the test cases are:

- measurable
- controllable
- repeatable

Risk level, risk type and test completion criteria state the type of coverage and the extent of coverage. They also dictate the test techniques to be used when testing. Test cases are then designed to satisfy the test completion criteria

Test techniques provide an understanding of the complexities imposed by most systems. The use of techniques forces testers into thinking about what they test and why they are testing.

Static vs. Dynamic:
dynamic test case *design* is carried out at the same time as the static *review* activities. Early in the V-model lifecycle. The dynamic test cases are executed once the system has been built and delivered into the test environment.
Static testing techniques are primarily techniques aimed at preventing faults being propagated to the next phase or into the test environment. Dynamic techniques (black or white) are intended to find faults in the translation of the specifications into the actual system.

Systematic vs. Non-systematic:
Systematic techniques are based around a set of rules that make the activity mechanistic, repeatable and measurable.
Non-systematic techniques are based on the experience of the tester (e.g. Exploratory testing). The biggest draw back with these techniques is the lack of documentary evidence and therefore repeatability of the test scenarios.

Functional vs. Non-functional:
Functional test techniques relate to establishing what the system does. Do the functions work?
Non-functional techniques focus on establishing how the system does what it does, i.e. is it secure? How quickly does it respond? Is it reliable? ..Non-functional techniques often use tools to execute the tasks with supporting tools analysing the dynamic system activity, e.g. memory use, concurrent threads, …

Advantages/Disadvantages of test techniques:

Advantages	Disadvantages
Objectivity	Requires training to some degree
Formal coverage measures	Time to implement – culture change
Early defect finding	Everyone must be “bought in”
Traceability	Not seen as useful for all applications
Coverage independent of the tester	Takes more time than less formal design
Way to differentiate test depth based on risks using different test techniques	Does not cover all situations (error guessing still useful)
High level of re-use (re-usable testware)	Little use of domain and product knowledge of tester
Repeatability and reproducibility	
Audit trails	
Higher defect finding capability	

The following paragraphs list a number of test techniques and give a short explanation about them. At the end of the document a table is given in which the test techniques are classified (systematic vs non-systematic, functional vs. non-functional, …)

14.1 Static Analysis

Static analysis involves no dynamic execution of the software under test and can detect possible defects in an early stage, before running the program.

Static analysis = automated technique to “walk through” the source code and detect constructs non-complying with pre-defined rules.

Rules come from convention in the industry, department or project.

Static analysis is also used to force developers to not use risky or buggy parts of the programming language by setting rules that must not be used.

The added value of static analysis:

- Unique defects are detected that cannot or hardly be detected using dynamic tests.
    - Unreachable code
    - Variable use (undeclared, unused)
    - Uncalled functions
    - Boundary value violations
- Source code is transferable to other / future developers
- Less defects in later tests

In time, Static Analysis (SA) is done after coding and before compilation, unit tests etc.

CODING -> STATIC ANALYSIS -> COMPILATION -> DYNAMIC ANALYSIS

Interesting statistics:

- 60% of faults could have been detected by static analysis
- 40% of faults detected by static analysis become a failure in the field.

Quality attributes that can be the focus of static analysis:

- Reliability
- Maintainability
- Testability
- Re-usability
- Portability
- Efficiency

When implementing static analysis consider the following issues:

- Organizational impact:
    - Limiting coding freedom
    - Other people than developers look at the code
    - Changes for developers and developers team
- Software life cycle impact
    - Increased attention for quality during the implementation stage (bottom of V model)
    - Possibly only delivery of validated code

Code qualify measures:

- Lines of code
- Comment frequency
- Nesting
- Number of function calls
- Cyclomatic complexity

.

14.2 Equivalence partitioning

Steps:

- Identify relevant input terms
- Identify equivalence classes using the rules (module C06, sheet 9)
- Identify test cases (a test case can only have 1 invalid Equivalence Class)
- Output partitioning (extension to decision coverage): are all results covered by the test cases.

Advantages:

- Reduce the number of test cases
- Focus both on valid and invalid test cases
- Applicable for all test levels
- When it is impractical or not needed to cover every possible input / output

Disadvantages:

- Dependencies between input attributes NOT taken into account
- Not a mathematical formula to calculate the number of test cases

14.3 Boundary value analysis

Boundary value analysis uses a model of the component that partitions the input and output of that component, with numeric inputs that can be divided in equivalence classes.
The values at and around the boundaries of an equivalence class are referred to as boundary values.
These are the values at which often a large number of defects can be found.
When determining the test cases, values closely to these boundaries are chosen so that each boundary is tested with a minimum of two test cases, or three for full boundary value analysis.

Use the following example :

During the calculation of the benefit the following are relevant : Every person receives a minimum benefit of 3500. All persons whose civil status is Married or that live in the postal area 5000 until 5600 receive an additional benefit of 1500.

In addition it is determined whether a person has worked and that his/her age is higher than 40. If this is the case the benefit is again raised with 1000. Alternatively (else) for persons that are married and have exactly 4 children the benefit is raised by 500.

Equivalence Table

Attributes	**Rule**	**Valids**	**Invalids**
Civil status	*Boolean*	*Married*	*Not married*
Postal area	*Range*	*[5000-5600]*	*<5000, >5600*
Worked	*Boolean*	*Yes*	*No*
Age	*Boundary*	*>40*	*<=40*
Children	*Number*	*4*	*<4,>4*

Boundary Values Table

Attributes	**Valids**	**Invalids**
Civil status	*Married*	*Single* *Divorced* *…*
Postal area	*5000* *5001* *5599* *5600*	*0* *4999* *5601* *9999*
Worked	*Yes*	*No*
Age	*41*	*0* *39* *40*
Children	*4*	*0* *3* *5* *9*

14.4 Syntax test

The objective of a syntax test is to find defects in primary input validations, user-interfaces, screen navigation, error messages and the layout of screens.
Applicable standards, specifications of screens, error messages and reports in the functional specifications and the data model can be used as test basis documentation.
During syntax testing all (screen) items are related to the attributes of the data model. The data model is used to determine the type of (screen) items and the restrictions that apply for the item itself.
Also, an investigation is carried out of the available (navigation) options, related to the screens and screen items.

This test technique is relatively straightforward. For most systems it is recommended to apply sampling on syntax testing since testing every screen and interface in detail is usually not the most effective way to spend test effort.

14.5 Cause/Effect Graphing

Formal, time-consuming
High level of coverage
Formal function testing technique
Equals Branch Condition Combination Coverage (without using extensions only!)

Steps

- Study specification for causes and effects
- Establish decision table (#cases = 2^n with n = #conditions)
- Determine test cases

Determine causes:

- Split compound conditions into single conditions
- Positive conditions (don't use NOT)
- Rank conditions by priority – highest priority first

- Decision table: start with all '1' and go to all '0'

Extension: simplification = Delete 'non-relevant' test cases. This is a risk-management decision!
Based on "Output partitioning".

If for two test columns only one condition differs and the results are the same, these columns can be joined. Compare two columns that only differ in one place (condition). Always delete the column on the right. See examples in: module c08, sheet 9, 10, 11.

Extension: test measure
For each group of "n" conditions all possible combinations
When conditions are relatively independent
In combination with compound conditions
Example:

Test case →	1	2	3	4
C1	1	0	1	0
C2	1	1	0	0
C3	1	0	0	1
A1	…	…	…	…
A2	…	…	…	…

This is test measure 2: for each group of 2 conditions all possible combinations are tested.

Reducing test situations: AND/OR conditions

A OR B		A AND B	
1	0	0	1
0	1	1	0
0	0	1	1

Reduction is 25% → Modified Condition Decision Coverage

14.6 Classification Tree Method

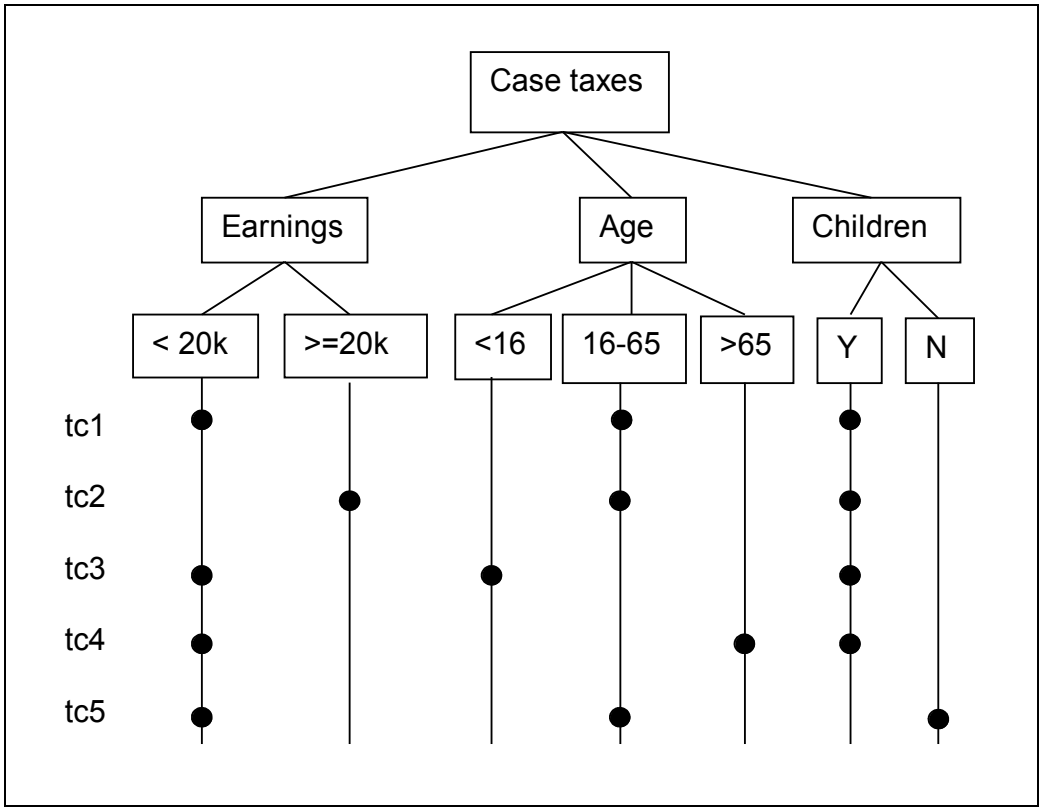
Good defect detection rate
Systematic and structured technique

Steps:

- Select test objects from SUT
- Design classification tree
  - Attributes of a test object are identified and classified
  - For each classification, input values are identified (EP rules)
- Combine classes to form test cases, combining input values from each classification

!!! Nothing is said about outputs!!!

Example:



Invalid test cases are tested individually. Valid and indifferent test cases are combined.

14.7 Elementary Comparison Test

Steps

- Study documentation, translate text into pseudo code
- Identify the conditions (look for IF)
- Determine test situations – conditions true or false

A	**OR**	**B**	**True is either one is true**
1		0	True as a result of A
0		1	True as a result of B
0		0	False

So in this case the # of test cases can be limited by leaving out the: 1 1 condition
Why? A and B have both been true, the chances that the system does not function well on the true-true situation is small.

A	**AND**	**B**	**True if both are true**
0		1	False due to A
1		0	False due to B
1		1	True

So in this case the # of test cases can be limited by leaving out the: 0 0 condition
Why? A and B have both been false, the chances that the system does not function well on the false-false situation is small.

Keep an eye on conditions that can only be true when another is false:

Example: A AND (B OR C) → B OR C = D

A	**AND**	**D**
1		1
1		0
0		1

B	**OR**	**C**
0		0
0		1
1		0

A	**AND**	**(B**	**OR**	**C**)
1		0		1
1		1		0
1		0		0
0		1		1
0		0		0

Steps in detail:

Case table:

Test case →	1	2	3	…
Parameter 1	value	value	value	…
Parameter 2	value	value	…	…
…		…	…	

Coverage:

Test case →	1	2	3	4	…
Conditions					
C1.1	x		x		
C1.2		x	x		
C2.1	x			x	
C2.2		x			
C2.3			x	x	
Etc.					

Where

A	AND	B
1		0 → C2.1
0		1 → C2.2
1		1 → C2.3

Example from course exercise:

1. Create the pseudo code

C1 IF CS - "Cohabit" OR (5000 < PA AND PA < 5600)
THEN 1500 additional benefit
ELSE no action

C2 IF Worked = Yes AND Age > 40
THEN 1.000 additional benefit

C3 ELSE IF CS = "Cohabit" AND # of children = 4
THEN 500 additional benefit ELSE no action

2. Create the test situations

C1	CS = "Cohabit"	OR	(5000 <= PA	AND	PA < 5600)
1.	O		O		I
2.	O		I		O
3.	I		O		I
4.	O		I		I

C2	Worked = Yes	AND	Age > 40
1.	I		I
2.	I		O
3.	O		I

C3	CS = "Cohabit"	AND	# of children = 4
1.	I		I
2.	I		O
3.	O		I

C3: notice dependency with C2 ! This condition only applied when C2 is false !

3. Create test cases

Logical test cases:

	1	*2*	3	4	*5*
CS = Cohabit	0	0	1	0	1
5000 < PA	0	1	0	1	-(0)
PA < 5600	1	0	1	1	-(1)
Worked = Yes	1	1	0	-(1)	0
Age > 40	1	0	1	-(1)	1
# of children = 4	-(0)	1	1	-(1)	0

Implemented test cases:

	1	*2*	*3*	*4*	5
Civil Status	Married	Single	Cohabit	Single	Cohabit
5000 < PA < 5600	4999	5600	4000	5000	1000
Worked	Yes	Yes	No	Yes	No
Age > 40	41	40	99	60	45
# of children = 4	5	4	4	4	1
+ 1500			X	X	X
+ 1000	X			X	
+ 500			X		
Expected Result	4500	3500	5500	6000	5000
Actual result					
PR number					

While creating test cases, track what test situations you already have covered. Example:

	1	2	3	4	5
Cl.l	X				
Cl. 2		X			
Cl. 3			X		X
C1.4				X	
C2.1	X			X	
C2.2		X			
C2.3			X		X
C3.1			X		
C3.2					X
C3.3		X			

14.8 State Transition Testing

N-switch coverage = a form of state testing in which test cases are designed to execute all valid sequences of N+1 transitions.

Test cases for 1-switch look as follows:

- Start state (name/id)
- Event (above arrow)
- Action (below arrow)
- Next state (name/id)
- Event
- Action
- Final state

Sequence the test cases so that the precondition is reached via the previous test case → nice for execution

State Table: states on the left, all possible actions/triggers across the top – outcome of each trigger on that specific state in the crossing cell.

14.9 Maintainability

Definition: a set of attributes that bear on the effort needed to make specified modifications

Sub-attributes:
- Analyzability: the capability to be diagnosed for deficiencies or new functionality
- Changeability: the extent in which changes can be implemented
- Stability: frequency of defects after new release
- Testability: the effort needed for testing modifications

Characteristics:
- Should be based on maintainability requirements
- Should be addressed in the test strategy, test plan, completion criteria, quality report, etc.
- Typically addressed at lower test levels
- Often supported by static analysis tools

Static techniques for testing maintainability:
- Assessment used checklist with weighted factors
  Determine important characteristics, define checklist with interview and expertise
  Think of process measures: 3, 4, 5 GL development, standards
  Think of product measures: documentation, consistency, traceability, and complexity of program, algorithmic, structure

  Example: Appendix A

- Metrics

  Product oriented:
  - Lines Of Code
  - Comment Frequency
  - Nesting
  - Number of Function Calls
  - Cyclomatic Complexity (Number of Decisions + 1)

  Process oriented (i.e. implicit measurements):
  - Analyzability: defect analysis time, Documentation availability
  - Changeability: Average time to change, Change parameter success ratio
  - Stability: Frequency of defects after new release
  - Testability: Regression test effort, test effort per unit.

- Design and code inspection (PBR role: conformance to standards)

- Static analysis (style guide, structure analysis)

Dynamic tests are possible, but less usable. Think of: explicit test cases using real life case based on maintenance history, execute typical maintenance activities and measure the actual effort spent and other indicators.

Maintenance testing: testing during maintenance releases:

1. Planned (perceptive, preventive, adaptive, corrective)
   Usually high-level test plan (this is how we work) for several maintenance items
   Include change control and release procedures including impact analysis
   Standard regression test set may be required.

2. Not planned – ad hoc
   Use risk analysis upfront and determine standard plan and test cases.

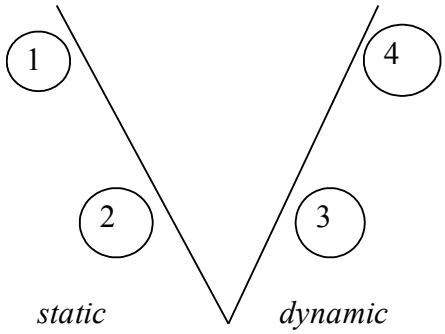
14.10 Usability

14.10.1 Heuristic evaluation

= Testing/evaluating the usability of a software product against a number of usability principles.

Focus of Usability:

- Narrow focus: only the usability of a program, screen (colors, buttons, etc.)
- Broad focus: the usability of the system for the user in his environment and for his business process.



1. Establish and validate usability requirements, define test plan and strategy
2. Inspect specifications and design from a usability perspective
3. Verify implementation (screen testing)
4. Validate implementation (user testing)

HE steps:

- Planning – select most critical usability parts (strategy), 2-5 experts, testers, developers, roles
- Kick-off – explain users and scenarios
- Individual checking – supported by checklist
- Gather findings and debriefing – discuss possible solution, categorize per heuristic, critical/major/minor

Additional notice:

- Not by users (they are biased), but by experts
- Expertise determines success
- Start execution early: during design process (prototyping)
- First the narrow focus, the broad focus.

Heuristics:

1. Visibility of system status
2. Match between system and real world
3. User control and freedom
4. Consistency and standards
5. Design useful error messages
6. Recognition rather than recall
7. Flexibility and efficiency in use
8. Aesthetic and minimalist design
9. Error prevention
10. Help and documentation

Benefits of usability tests:

- Low costs
- Quick tests

14.10.2 Usability (lab) testing

Identifies usability problems that are very likely to occur in real use. End users perform tasks with a product, while being observed. Users are interviewed after the test (SUMI?)

Steps

- Set test goals and formulate test criteria
- Based on usability requirements
- Time needed to accomplish task, # wrong choices, # times documentation used, …
- Understandability menu-options, opinion on attractiveness
- Establish test scenario
- Based on context (task) analysis, use cases, etc.
- Pitfalls: too directive, mismatch with user language, tasks influence each other, …
- Test the scenarios!
- Select representative test participants
- Based on context (user) analysis
- Profile: novice/experienced, age, role, …
- Instructions: manual, reference card, phone to helpdesk
- Create test environment – observation or usability lab
- Conduct test
- Report

Cost/Benefits

- Representative for real use
- But: time consuming
- But: usability expertise needed to interpret results
- But: 3-10 users
- Expensive but high benefits with a lab

Life cycle phase: working prototype, working product

14.10.3 SUMI

SUMI = software usability measurement inventory

- Broad focus: Measure user satisfaction
- Well-founded questionnaire
- User scores are standardized using a reference database
- Objective information on users' subjective attitude to six usability aspects

SUMI:

- Intended for use by users
- Running version of software required
- Supported by an analysis tool
- Specialized versions available for multimedia and internet applications

SUMI metrics:

- Affect (user's feeling about interacting with the product)
- Efficiency (user's perception of efficiency of task performance)
- Helpfulness (communicative skills of the product – e.g. help messages)
- Control (user's feelings about product response)
- Learnability (how quickly familiar with the product + quality of documentation)

Cost/Benefits:

- Easy to use; fast, well-founded results
- But: limited detailed analysis possible
- Objective indicator
- Low cost
- But: late in lifecycle

Minimum of 10 users doing the 'same' tasks (hard!)

14.11 Performance

= The capability of the software product to provide appropriate performance, relative to the resources used, under specified conditions

Sub attributes:
- Time behavior – appropriate response/processing times and throughput rates
- Resource behavior – appropriate amounts and types of resources

Important:
- Start on time!
- Can be used for: Verification test against requirements using real-life situations
- Can be used for: Validating the infrastructure
- Can be used for understanding scalability
- There is a difference between performance, volume and stress.
- Test objective must be clearly specified (requirements validation, infrastructure validation…)
- Think of test plan strategy, completion criteria, etc.
- Operation profiles often used “again”
- Checking and analysis require expertise (often white box approach).
- Different tools play an important role

Performance test process:
1. Start up and pre-requisites
2. Tool evaluation and selection
3. Test items and test cases
4. Test checking and recording
5. Checking for test completion

1. **Start and pre-requisites**

Prepare the test. Think of:
- Clear test objectives.
- Requirements specification including items like specified maximum data volumes, response times, etc that you later can use as completion criteria.
- Do you have mass data and what about privacy?
- Performance test tools and production-like environment

3. **Test items and test cases**
From operational modes and profiles. Specified conditions are analyzed per test item (database settings, concurrent users)

This leads to a load model:
- Requirements to be covered/questions to be answered
- Test item(s)
- Test configuration/environment
- Load to be generated
- Measurements
- Test data built around operational profile using black box techniques.

Description of load:
Poor description - # simultaneous users
Good description - # transactions per time unit

Possible measures:

- Response time
- Mean response time
- Worst case response time ratio
- Throughput time
- Turn around time
- Memory utilization
- Transmission resource utilization
- Mean occurrence of transmission error

Test execution

- Beware: do automatic running of the performance test (tool) and the logging and monitoring tool affect the performance?
- Entry criteria: maturity level of core functionality (stability, correctness)

Additional Static techniques: Performance assessment (like maintainability approach) and architectural and code reviews (algorithmic complexity, SQL statements).

Additional dynamic tests: exploratory, stopwatch (implicit measurement), benchmarking, random test (BS7925-2) / real life test (TMap).

14.12 Reliability

= The capability of the software product to maintain a specified level of performance when used under specified conditions

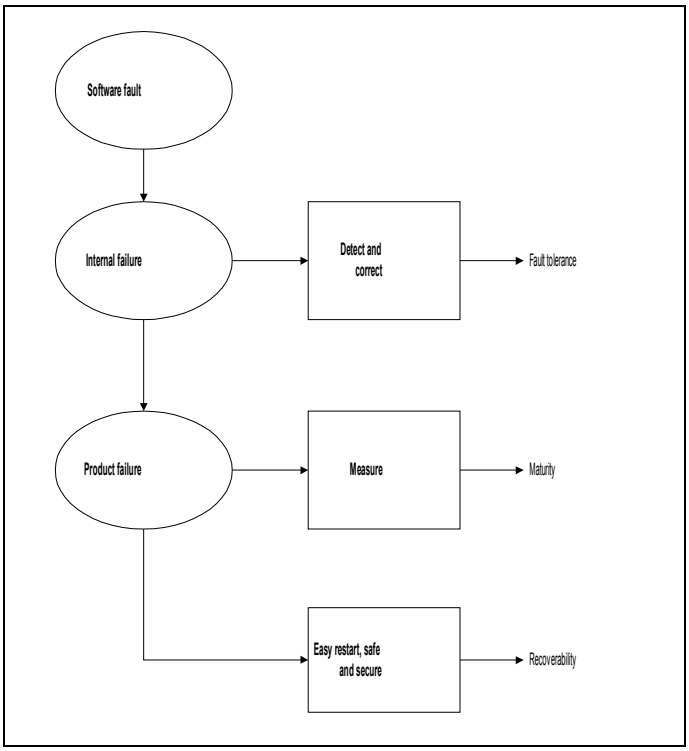
Sub-attributes:

- Maturity – frequency of failure in the software product
- Fault tolerance – maintain performance in case of software faults or interface infringements
- Recoverability – re-establish performance and recover data affected in case of a failure

Characteristics:

- Addressed at high test levels – when the system changes, reliability should be retested
- Highly design development related
- Should be based on reliability requirements and addressed in test strategy, test plan, completion criteria, etc
- Includes a definition of reliability failure

Reliability interpretation:



Static techniques:

- Reliability assessment – checklist same for maintainability
- Metrics
    - Maturity: MTBF, #breakdowns, test coverage, estimated latent fault density
    - Fault tolerance: failure avoidance, incorrect input operation avoidance
    - Recoverability: MTTR, mean restart time
- Design and code inspection (reliability coding standards, like nested levels, Cyclomatic complexity)

Dynamic techniques:

- Syntax checking – aimed at fault tolerance
- Random testing using operational profiles (TMap: real life test) using operational profiles
- Volume, load, stress testing (looking for boundaries)

Using Operational Profiles (operation = major task performed by a system):

- Identify and define operational modes: intensive/normal use, peak/prime/off hours, max/average/low #users
- Per operational mode define test cases: identify user group, task frequency per time unit, relative frequency
- Build test cases to match the profile: detailed content list less critical, each task at least once, use data preparation tools
- Execute test cases randomly
- Use results to drive decision

14.13 Exploratory Testing

Exploratory testing = Simultaneous exploration, design and execution.

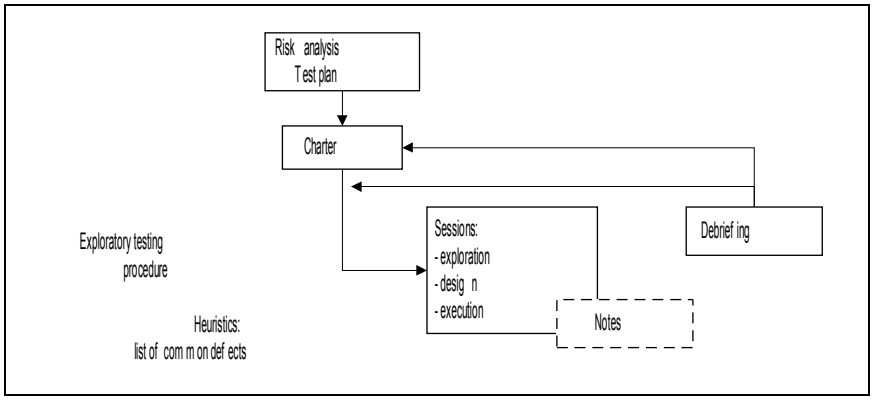
It is:

- An interactive test process
- Using information gained while testing to design new and better tests
- Formal, so different from error-guessing and ad-hoc testing
- Testers have skills to listen, read, think and report rigorously and effectively.

Applicable when:

- Little or no specification is available
- Investigation/isolation of a particular defect
- Investigation into a particular risk – to evaluate the need for scripted tests
- No time to specify and script tests
- To diversify testing

When less applicable: module c14, sheet 7.



For preparing and executing exploratory tests, test charters are used with items like:

- What (scope)
- What not (out of scope)
- Why (questions to be answered
- How (brainstorm)
- Expected problems
- Reference

For describing the results of the test, session sheets are used:

- Test coverage outline
- Name tester
- Test execution log
- Defects found
- Quality indicator (# major defects per hour)
- New risks encountered
- Issues, questions, anomalies

Debriefing: at the end of the session for discussing priority of defects, risks mitigated, etc.

14.14 Use Cases

Objective:
- Validation
- Testing most important tasks (regression testing)
- Flow through the system

Type of defects:
- Functionality, usability
- Integration defects

Qualification:
- Informal, dialogue-type system

Steps:
1. Identification of users (actors) of the system
2. Identification of use cases for each actor
3. Identification of paths for each use case
4. Decide on use cases to elaborate (risk based testing)

Use case template:
- Name and summary
- Frequency
- Actors
- Preconditions
- Description (basic course and deviations) → loud & clear
- Exceptions (failure conditions)
- Post conditions (expected results)
- Additional attention points (performance, messages)
- To be inspected with ‘product management’ and development.

Cost/Benefits:
- Main tasks/functions tested (disadvantage: limited (code) coverage)
- Preferably split test specification and execution (ask users to run the test cases)
- Preferably early involvement
- User focused: real-world defects
- Different view → different defects

14.15 Process Cycle Test

Objective:
Test the suitability of the IS with respect to the organizational procedures, e.g. task descriptions, user manual, and identify possible usability problems

Type of defects:
- Inconsistency within procedures or task descriptions
- Tasks that cannot (fully) be carried out with the system
- Incomplete system, security, incomplete forms, …

Steps
- Determine decision points (number all ins & outs / before & after)
- Determine path combinations (different coverage measures)
- Specify test cases (actions and results)
- Specify initial database
- Specify test procedure
- Deliverables: test design and test procedure

Test Measure 1: touch all branches at least once
Test Measure 2: all combinations before & after each decision point

Cost/Benefits
- Thorough insight in suitability
- Structured; independent and testware
- Variation of coverage (measures)
- But: documentation required
- Extend scope by adding open questions
- But: knowledge: expertise on Process Cycle Test
- Costs moderate
- Users involved

Test level: acceptance test

15 Test Phases

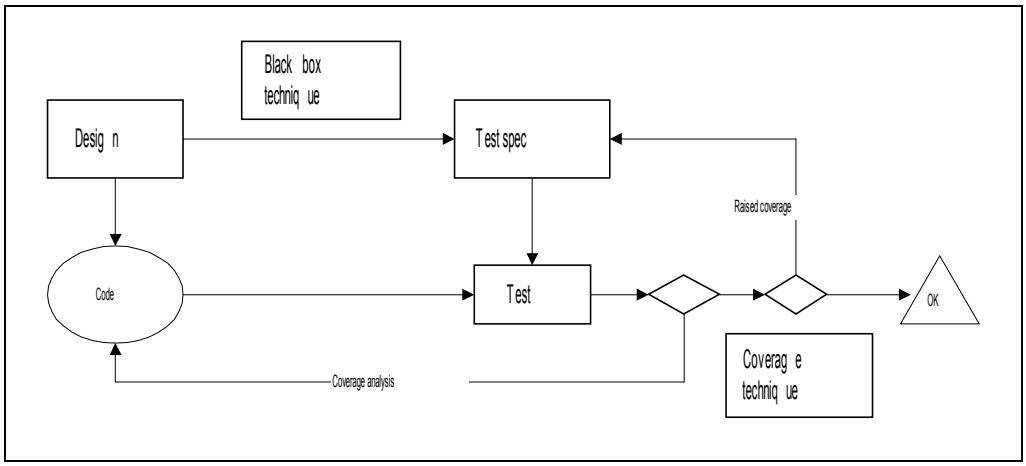
15.1 Component Testing

BS 7925/2

Component testing = the testing of individual components before integrating into the system. The focus is on the internal program structure.

Usually stubs and drivers are needed.

Basic scheme of component testing and coverage:



Process description:

- Determine test objective (exit criteria) in terms of coverage metric. Think of # conditions/decisions, usage of tools
- Select a corresponding black box technique based on exit criteria.

15.1.1 Component Test Life Cycle

- Plan
- Specify
- Execute
- Check results
- Check coverage

15.1.2 Structural Testing

- Static Analysis
- Dynamic Analysis

Static and Component

- Static analysis
- Maintainability
- Usability – heuristic evaluation
- Coverage techniques
- Component testing approach
- Dynamic analysis

15.2 Integration Testing

Integration testing in the small = testing performed to expose faults in the interfaces and in the interaction between integrated components.

Integration strategies:
- Bottom-up using drivers (easy)
- Top-down using stubs (hard)
- Big bang
- Functional incremental – risk priority

Typical test techniques: EP, BVA, syntax, state, fault tolerance, performance, and resource behavior.

Integration testing in the large:
- Interacting groups of systems
- Systems must be tested and passed for integration
- Interoperability, fault tolerance testing
- May take large scale planning and co-ordination
- Separate phase test plan
- Done by system test team?
- Typical test techniques:
- EP, C/E graph, ECT, Classification trees, State transition, syntax at UI-level
- Exploratory testing
- Non-functional attributes: Performance, security, reliability, portability, usability (e.g. Heuristic evaluation)

Requirements testing:
- Based on requirements document and FS
- Test cases are traceable to the requirements
- Requirements coverage: which requirements have tested and passed, tested and failed, not tested
- Danger in generating a ‘tick-list’ of requirements to be tested
- Requirements are tested in isolation
- System not seen ‘in the whole’

15.3 System Testing

Integration and System

- Integration testing techniques
- System testing overview
- Cause/effect graphing
- Classification tree method
- Elementary comparison test
- State transition testing
- Performance testing
- Reliability testing

15.4 Acceptance Testing

Acceptance testing (beta)
= Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component (BS7925-1)

Acceptance testing = the process of comparing the end product to the current needs of its end users (Kit 1995)

- Responsibility user/customer
- Does the system comply to the requirements?
- Has the right system been built?
- Contractual against acceptance criteria

Typical test techniques
- EP
- Sometimes C/E graphing or ECT (formal verification possible but preferred at system test level)
- Use cases / process cycle test
- Non-systematic techniques (ET)

Typical non-functional attributes
- Usability
- Sometimes performance (real life environment)

Possibilities for integrated test (ST + AT):
- Reduces time to market
- Early defects
- User involvement, one environmental model
- But: conflict of interest possible
- But: are they really the same? Think of quality attributes, technique, views, environment
- But: impact on quality

16 Test Tools

16.1 Static Analysis Tools

Static analysis tools are tools to analyze the source code:

- # Comments
- Complexity
- Use of variables
- Coding standards

Advantages:

<TBD>

Disadvantages:

<TBD>

16.2 Test Running Tools

Usefulness of test tools:

- Can speed up the process
- Can enable (better) regression tests
- Can prevent manual repetition errors
- “Special tests” like performance/load are supported
- Once implemented, they save time and effort
- Improved test planning
- Allows test designer to concentrate on new problems – less boring activities

Mainly: record & playback, test programming, data driven testing

Record and playback Advantages / disadvantages, see module C13, sheet 7.

Starting Data driven testing (DDT) is a project on its own! (Tool script language)

Test frame:

- Implementing action words
- Test engineer defines and documents action words
- Navigator creates the action word functions on various levels
- Real navigation scripts re-used for various applications

Are you ready for test automation?

- Test automation is software development
- Test automation is a long-term investment
- Assess your resources: people & skills
- There is no one-size-fits-all approach
- Test cases determine the quality of the test
- You need to gauge your maturity

16.2.1 Data Driven Testing

Advantages:

- More maintainable
- Test data accessible for non-technical person
- Test preparation can start earlier
- Higher robustness (more unattended test sessions)
- Useful for regression testing

Disadvantages:

- Test flow ‘hidden’ in test scripts
- Result checking by tool
- Online systems
- Less suitable for user scenario testing
- Skills needed in scripting language
- Test control and traceability

16.3 Test Management Tools

Types:

- Requirements management
- Configuration management
- Incident management
- Test management

Test management tools can be stand-alone or integrated

Test management tool:

- Test planning
- Scheduling and running tests
- Progress reporting
- Testware management
- Links to attached files
- Requirements testability
- Integrated incident management
- Driver for automated testing

16.4 Performance Test Tool

The goal of a performance test tool is to simulate load and see how the systems reacts, performs.

Advantages:

- Real-life test situations

Disadvantages:

- Specialists tool (you need resources, expertise)

16.5 Comparison Tool

The goal of a comparison test tool is to compare output from different test runs.

Advantages:

- Saves time
- Use during regression test

Disadvantages:

- The generated output must be structured in one way …

16.6 Oracle Tool

When developing a new platform, this tool can predict how this new platform will perform, in relation to the current platform.

Advantages:

- Makes a prediction

Disadvantages:

- Analyzing the results takes a lot of time.

17 Appendix A: Maintainability Checklist

Analyzability			
Nr	*Checklist item*	Weighting	Score
	Is the design documentation up-to-date?		
	Is the design documentation available?		
	Is the design documentation retrievable?		
	Are the problem reports supported by log files?		
	Is the level of comments in the code sufficient?		
	Is the cyclomatic complexity of the code within limits?		
	Is the size of the components not too large?		
	Is the number of interfaces (e. g. function calls) not too large?		
	Are traceability matrices available?		
	Has the design documentation been inspected		
	Has a Standard development environment been used?		
	Has generated code been applied?		
	Is historical defect data available regarding analysis and solutions?		
	Have coding standards and a style guide been applied?		

Changeability			
Nr.	*Checklist item*	*Weighting*	Score
1	Is the size of the components not too large?		
2	Is the number of nested levels in the code not too large?		
3	Is the cyclomatic complexity of the code within limits?		
4	Is the level of comments in the code sufficient?		
5	Is design documentation available, consistent and up-to-date?		
6	Is the program structure clear and straightforward?		
7	Does the code conform to a style guide (readability)?		
8	Do the project members have enough domain knowledge?		
9	Has a Standard development environment been used?		
10	Is the number of external interfaces not too large?		

Stability			
Nr.	*Checklist item*	*Weighting*	Score
1	Is the size of the components not too large?		
2	Is the number of nested levels in the code not too large?		
3	Is the level of comments in the code sufficient?		
4	Is the number of anomalies solved in the past large?		
5	Has the component been re-used a number of times?		
6	Are CM tools provided to track changes?		
7	Are the test results stored for automatic comparison?		
8	Is the number of interfaces (function calls, coupling) not too large?		
9	Has the database been normalized to the third degree?		
10	Are Standard routines applied for "returning code"?		
11	Is a full regression test set available?		
12	Is the amount of dynamically allocated memory minimized?		
13	Is sufficient attention given to error handling in the code?		

Testability			
Nr.	*Checklist item*	*Weighting*	Score
1	Is a regression test set available?		
2	Is configuration management applied for both the test object and the testware?		
3	Is the documentation (test basis) of an adequate level?		
4	Is an adequate test environment available?		
5	Is an incident management tool available with defect history?		
6	Are logging facilities provided?		
7	Are record and playback tools available (with automated test scripts)?		
8	Is the coverage of the test script known?		
9	Do the testers have enough domain knowledge?		
10	Is the cyclomatic complexity within limits?		
11	Is a Standard test interface available?		

Appendix B: Review Process Form

REVIEW PROCESS FORM ❑ Walkthrough ❑ Technical Review ❑ Inspection			Doc-id(s): Start date:		Version:
Project:	**Meeting**	**Date:**		**Time:**	**Location:**
Document Name:					

	Pages/lines of code						
	❑ Req	❑ Spec	❑ Arch	❑ Design	❑ Code	❑ Test Spec	❑ Other……
Goals / Specific questions:							
Related documents / standards:							
Applied rules and checklists:							

	Name	Role	Prep time

Planning time:				New items logged:	
Kick-off meeting time:					
Total Preparation time:					
Logging time:					
Discussion time:				**C**ritical items logged:	
Rework time:				**M**ajor items logged:	
Follow-up time:				**m**inor items logged:	
Total time:				Total items logged:	

	❑ Only Moderator ❑ All Participants ❑ New Review	
		Moderator signature:

Remarks on review procedure:

Improve Quality Services BV	**Review Process Form**	**SEQ-Number:**

REVIEW LOGGING FORM	Doc-id(s):		Project:	
❑Walkthrough ❑Technical Review ❑Inspection	Meeting	Date:	Time:	Location:

Document name:

	Location Pag	Line	Who	Rule	Defect / Issue description				N	Q	Rework / action performed / outcome discussion
1	12	14		DT2	Selection is not clear		X				
2	12	15		DT2	Selection is not clear		X				
3	12	17		DT2	Selection is not clear		X				
4	12	34		DT1	Search by list is not as described in FRS		X				
5	12	39		DT1	No drop down list		X				
6	13	31		DT2	No credit card drop down list		X				

Improve Quality Services BV	Review Logging Form	Page totals						Page … of …

Appendix C: Coverage

Statement Coverage

Statement testing requires that each statement (node) be covered at least once as a minimum.

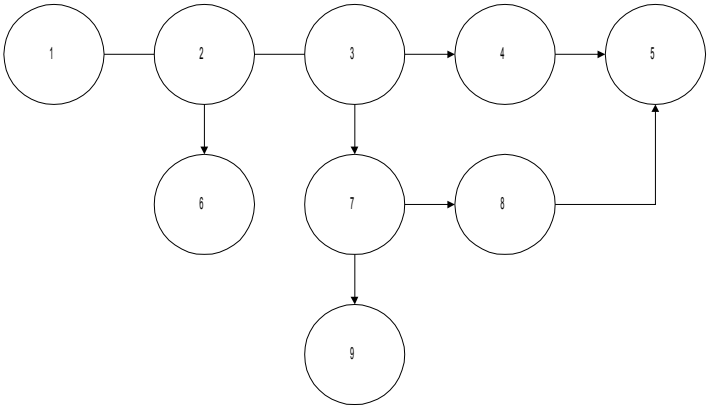
Decision Coverage

Decision coverage requires that each decision outcome (branch) is covered at least once as a minimum.

Branch Coverage

Branch and Decision coverage are different at lower levels when there is more than one entry/exit point → for decision coverage; only branches linked to a decision are counted.

Example:



When a test case follows the path: 1 – 2 – 6, decision 2 is the decision point and the coverage is:

Decision coverage:	1 out of 6:	17%
Branch coverage:	2 out of 9:	22%

Branch Condition Coverage

Branch Condition Coverage = each single condition true/false

Branch Condition Combination Coverage

Branch Condition Combination Coverage = all possible combinations within one decision are tested
Modified Condition Decision Coverage = every condition determines twice the result of the compound condition

Path testing = testing designed to execute all or selected paths through a program → *many* cases!!! (E.g. loops)

Dynamic Analysis: bounds checkers, memory testers, leak detectors
Detect memory problems
Detect overwriting/over reading array boundaries
Detect memory allocated but not freed
Detect reading and using uninitialised memory

An example:

V = A and (B or C)

- Statement coverage

Every statement is executed once.

Test case	A	B	C	Result
1	1	1	1	1

- Branch / Decision coverage

Every decision outcome is tested once.

Test case	A	B	C	Result
1	0	1	1	0
2	1	1	0	1

- Branch Condition coverage

Each single condition is once true and once false.

Test case	A	B	C	Result
1	0	1	1	0
2	1	0	0	0

- Branch Condition Combination coverage

All possible combinations are executed.

Test case	A	B	C	Result
1	0	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	1	1	0
5	1	0	0	0
6	1	0	1	1
7	1	1	0	1
8	1	1	1	1

Appendix D: List of Standards

- BS : British Standards
- BS7925-1 : Software Testing Vocabulary
- BS7925-2 : Software Component Testing
- CMM-SW : Capability Maturity Model for Software
- CMMI-SE/SW : Capability Maturity Model Integration
- IEEE : Institute of Electrical and Electronics Engineering
- IEEE 610 : Standard Computer Dictionary
- IEEE 610.12 : Software Engineering Terminology
- IEEE 730 : Standard for Software Quality Assurance Plans
- IEEE 829 : Standard for Software Test Documentation
- IEEE 1008 : Standard for Software Unit Testing
- IEEE 1012 : Standard for Software Verification and Validation
- IEEE 1028 : Standard for Software Reviews
- IEEE 1044 : Standard for Classification for Software Anomalies
- IEEE 1044.1 : Guide to Classification for Software Anomalies
- TMM : Testing Maturity Model

18 Appendix E: Test Techniques Classification

Characteristics	Static	Dynamic	Black Box	White Box	Formal	Informal	Formal coverage	Informal coverage
Techniques								
Equivalence Partitioning		X	X		X		X	
Ad Hoc testing		X	X	X		X		X
Branch (Decision) Coverage		X		X	X		X	
State Transition Testing		X	X		X		X	
Walkthrough	X		X	X	X			X
Cause / Effect Graphing		X	X		X		X	
Complexity Analysis	X				X			X
Random Testing		X	X		X			X
Statement Coverage		X		X	X		X	
Data Flow Testing	X	X		X	X		X	
Syntax Testing		X	X		X			X
Inspection	X		X	X	X			X
Error Guessing		X	X	X				X
Boundary Value Analysis		X	X		X		X	
Informal Review	X		X	X		X		X
Classification Tree Method								
Elementary Comparison Test								
Exploratory Testing								
Use Cases								
Process Cycle Test								
Maintainability								
Usability Lab Testing								
Heuristic Evaluation								
SUMI								
Performance								
Reliability								

Appendix F: V-model table

	Component Test	**Integration in the small**	**System test**	**Integration in the large**	**Acceptance Test**
Objective	To show that each component functions as intended (according to the component specification) and to provide visibility into the quality of the component	To confirm functionality of modules when components are combined together. Verification against the global design, with focus on interaction and interfacing between modules.	To show that the system meets the functional specification and all the non-functional requirements.	To confirm system and network functionality when the system is integrated into an existing network of systems.	To show the delivered system meets the business needs (validation), formal acceptance of the product
Scope	Test Input Fields, GUIS, code, calculations, etc	Test all interfaces, data exchange, and variable passing between components.	Functionality, non-functional attributes such as performance, security, installation, error handling, recovery etc	System interfaces, files, data, operational profiles, usually end to end testing based on business processes.	Requirements based testing. The whole system, test cases based on requirements document, covers functionality, usability, help, user guides etc.
Responsibility	Development	Integration team	Test team (within development)		User / Customer or representative
Who does it	Developer typically, development owned	Integration tester, integrator / development responsibility	Independent Test team e technical experts / development responsibility	Independent test team	Users, business representatives, live support, or test (user/customer responsibility)
Entry Criteria	Component is complete and ready to test (compiles, has been reviewed, sw design has been approved, complies to static criteria).	Components passed component test phase (complies to exit criteria), integration test spec reviewed and approved, global design reviewed and approved, passed confidence test.	Integration test phase passed, development sign off, test team acceptance (intake/ confidence), release notes available	System test sign off for each system being integrated	Sign off from System Test/ Integration test phase (system test report available), user requirements reviewed and approved, test plan reviewed and approved.
Exit Criteria	Component passes all tests and is signed off, level of coverage is reached, test report written, cyclomatic complexity within criteria	All integration test cases executed, all critical tests passed, number of defects within set limits, test results documented, test report written	All system test cases run and complete, no high priority defects outstanding, Mean Time Between Failure (MTBF), # defects per test hour under threshold, requirements coverage	All integration test cases complete, no category A or B bugs outstanding, system meets reliability requirements	All acceptance test cases completed, no category A or B business priority defects outstanding, list with known defects, business sign off for live implementation, MTBF, acceptance test report approved
Test Deliverables	Component test report/results, test log, test code & test data, component sign off of developer/ stubs and drivers.	Integration test report/result, problem reports, sign off, integration test specification	System test plan, test report, test results, test specifications and – procedures, test evaluation report (recommendations for product and project)	Integration test report, test results, configuration guide, integrated test model	Acceptance test report, results, test logs, problem reports, change requests, test specifications and test procedures.

Typical Test Techniques	White box coverage techniques, syntax testing, Boundary Value Analysis, Ad hoc	White box test techniques, equivalence partitioning, state transition testing, syntax, cause effect graphing.	Black Box (e.g. Equivalence Partitioning, state transition, cause/effect graphing), specialist non-functional test techniques (e.g. error-guessing)	Black Box	Equivalence partitioning, exploratory testing, use case testing, error guessing, process cycle test
Metrics	Number of defects found/fixed, priority, statistical analysis	Number of faults found, priority, statistical analysis	Tests passed/failed/run, number of faults and priority, environment log, test logs, progress reports time & effort planned v spent, requirements coverage, test effectiveness	Test passed/failed/run, number of faults and priority test coverage, test effectiveness	Number of faults, business priority, tests passed/failed/run
Test Tools	Static analysis tools, debuggers, harnesses, dynamic analysis, coverage, comparator	Test harnesses (stubs, drivers), simulators, record/playback for regression testing, defect management, test management	Performance monitoring, data generators, capture/replay, test management tools	Harnesses, stubs, drivers, data generators, simulators, capture/replay, comparators	Capture replay, comparators, performance, defect management, test management
Applicable Testing Standard	BS7925-2 (component testing standard, TMap, MISRA coding standards	BS7925-2 (component testing standard, TMap, IEEE 829 for documentation	BS7925-2 (component testing standard, TMap, IEEE 829 for documentation, ISO 9126 for non-functional exit criteria	See useful standards for software testing	BS7925-2 (component testing standard, TMap, IEEE 829 for documentation, ISO 9126 for non-functional exit criteria
Typical non functional test types	Resource usage (e.g. memory, semaphores, …), time behavior, portability, maintainability	Resource usage (e.g. memory), performance	Reliability, performance, usability, portability		Usability, performance, security