Certified Tester

Advanced Level Syllabus

Version 2007

International Software Testing Qualifications Board

Revision History

Version	Date	Remarks
ISEB v1.1	04SEP01	ISEB Practitioner Syllabus
ISTQB 1.2E	SEP03	ISTQB Advanced Level Syllabus from EOQ-SG
V2007	12OCT07	Certified Tester Advanced Level syllabus version 2007

Table of Contents

Acknowledgements

This document was produced by a core team from the International Software Testing Qualifications Board Advanced Level Working Party: Bernard Homès (chair), Graham Bath, Rex Black, Sigrid Eldh, Jayapradeep Jiothis, Paul Jorgensen, Vipul Kocher, Judy McKay, Thomas Mueller, Klaus Olsen, Randy Rice, Jürgen Richter, Eric Riou Du Cosquer, Mike Smith, Geoff Thompson, Erik Van Veenendaal.

The core team thanks the review team and all national boards for their suggestions and input.

At the time the Advanced Level Syllabus was completed the Advanced Level Working Party had the following membership (alphabetical order):

Graham Bath, Rex Black, Robert Bender, Chris Carter, Maria Clara Choucair, Sigrid Eldh, Dorothy Graham, Bernard Homès (chair), Jayapradeep Jiothis, Vipul Kocher, Anastasios Kyriakopoulos, Judy McKay, Thomas Mueller, Klaus Olsen, Avinoam Porat, Meile Posthuma, Erkki Pöyhönen, Jürgen Richter, Eric Riou Du Cosquer, Jan Sabak, Hans Schaefer, Maud Schlich, Rajesh Sivaraman, Mike Smith, Michael Stahl, Geoff Thompson, Erik Van Veenendaal.

The following persons participated in the reviewing, commenting and balloting of this syllabus:

Bernard Homès (chair)

Reto Armuzzi
Sue Atkins
Graham Bath
Paul Beekman
Armin Beer
Rex Black
Francisca Blunschi
Armin Born
Con Bracke
Chris Carter
Maria Clara Choucair
Robert Dankanin
Piet de Roo
Sigrid Eldh
Tim Edmonds
Erwin Engelsma
Graham Freeburn
Dorothy Graham
Brian Hambling
Jeff B Higgott
Bernard Homès
Rob Hendriks
Dr Suhaimi Ibrahim
Phillip Isles
Pr. Paul C. Jorgensen
Vipul Kocher
Anastasios Kyriakopoulos
Junfei Ma
Fergus McClachlan
Judy McKay
Don Mills
Gary Mogyorodi
Richard Morgan
Silvio Moser
Ernst Müller
Reto Müller
Thomas Müller
Peter Mullins
Beat Nagel
Richard Neeve
Klaus Olsen
Dale Perry
Helmut Pichler
Jörg Pietzsch
Avionam Porat
Iris Pinkster
Horst Pohlmann
Meile Posthuma
Eric Riou Du Cosquer
Stefan Ruff
Hans Schaefer
Maud Schlich
Rajesh Sivaraman
Mike Smith
Katja Stalder
Neil Thompson
Benjamin Timmermans
Chris van Bael
Jurian van de Laar
Marnix van den Ent
Mark van der Zwan
Stephanie van Dijck
Jan van Moll
Erik Van Veenendaal
Roland Weber
Phillip Whettlock
Derek Young
Mike Young
Wenqiang Zheng.

This document was formally released by the General Assembly of ISTQB® on 12 October 2007.

0. Introduction to this syllabus

0.1 The International Software Testing Qualifications Board

The International Software Testing Qualifications Board (hereinafter called ISTQB®) is made up of Member Boards representing countries or regions around the world. At the time of release, the ISTQB® consisted of 33 Member Boards. More details on the structure and membership of the ISTQB may be found at www.istqb.org.

Purpose of this document

This syllabus forms the basis for the International Software Testing Qualification at the Advanced Level. The ISTQB® provides this syllabus as follows:

1. To Member Boards, to translate into their local language and to accredit training providers. National boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To Exam Boards, to derive examination questions in their local language adapted to the learning objectives for each module.
3. To training providers, to produce courseware and determine appropriate teaching methods.
4. To certification candidates, to prepare for the exam (as part of a training course or independently).
5. To the international software and systems engineering community, to advance the profession of software and systems testing, and as basis for books and articles.

The ISTQB® may allow other entities to use this syllabus for other purposes, provided they seek and obtain prior written permission.

The Certified Tester Advanced Level in Software Testing

The Advanced Level qualification is aimed at people who have achieved an advanced point in their careers in software testing. This includes people in roles such as testers, test analysts, test engineers, test consultants, test managers, user acceptance testers and software developers. This Advanced Level qualification is also appropriate for anyone who wants a deeper understanding of software testing, such as project managers, quality managers, software development managers, business analysts, IT directors and management consultants. To receive Advanced Level certification, candidates must hold the Foundation Certificate and satisfy the Exam Board which examines them that they have sufficient practical experience to be considered Advanced Level qualified. Refer to the relevant Exam Board to understand their specific practical experience criteria.

Level of knowledge

Learning objectives for each chapter are divided such that they can be clearly identified for each individual module. Further details and examples of learning objectives are given in section 0.3.

This syllabus' content, terms and the major elements (purposes) of all standards listed shall at least be remembered (K1), even if not explicitly mentioned in the learning objectives.

Examination

All Advanced Certificate examinations must be based on this syllabus and on the Foundation Level Syllabus. Answers to examination questions may require the use of material based on more than one section of this and the Foundation Level Syllabus. All sections of this and the Foundation Level Syllabus are examinable.

The format of the examination is defined by the Advanced Exam Guidelines of the ISTQB®. Individual Member Boards may adopt other examination schemes if desired.

Exams may be taken as part of an accredited training course or taken independently (e.g., at an examination center). Exams may be taken on paper or electronically, but all exams must be proctored / observed (supervised by a person mandated by a National or Examination Board).

Accreditation

An ISTQB® Member Board may accredit training providers whose course material follows this syllabus. Training providers should obtain accreditation guidelines from the board or body that performs the accreditation. An accredited course is recognized as conforming to this syllabus, and is allowed to have an ISTQB® examination as part of the course.

Further guidance for training providers is given in Appendix C – Notice to Training Providers

Level of detail

The level of detail in this syllabus allows internationally consistent teaching and examination. In order to achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the Advanced Level
- Learning objectives for each knowledge area, describing the cognitive learning outcome and mindset to be achieved
- A list of information to teach, including a description, and references to additional sources if required
- A list of terms that students must be able to recall and have understood
- A description of the key concepts to teach, including sources such as accepted literature or standards

The syllabus content is not a description of the entire knowledge area of software testing; it reflects the level of detail to be covered in Advanced Level training courses.

How this syllabus is organized

There are 10 major chapters, each with an introductory section that provides an insight on how they apply to the different testing professionals (modules).

For training purposes, sections 0.3 to 0.6 are provided with the specific learning objectives for each module, per chapter. These sections also provide the minimum time expected for training these topics.

It is strongly suggested to simultaneously read the syllabus and study the learning objectives of that specific chapter. This will allow the reader to fully understand what is required and what are the essentials of each chapter for each of the three modules.

Terms & Definitions

Many terms used in the software literature are used interchangeably. The definitions in this Advanced Level Syllabus are available in the Standard glossary of terms used in software testing, published by the ISTQB®.

Approach

There are a number of ways to approach testing, such as those based on the specifications, code structure, data, risks, processes, standards and similar lists of taxonomies. Different processes and tools provide support to the testing processes; methods are available to improve existing processes.

This Advanced Level Syllabus is organized around the approaches proposed in ISO 9126, with a separation of functional, non-functional and supporting approaches. Supporting processes and some improvement methods are mentioned. Selection of this organization and processes is done on an arbitrary basis considered to provide a sound basis for the Advanced Level testers and test managers.

0.2 Expectations

The Advanced Level certification described in this syllabus will be examined with three major task descriptions in mind, each representing basic responsibilities and expectations within an organization. In any organization, responsibilities and associated tasks may be split between different individuals or covered by a single individual. The working responsibilities are outlined below.

0.2.1 Advanced Level Test Manager.

Advanced Level Test Management professionals should be able to:

- Define the overall testing goals and strategy for the systems being tested
- Plan, schedule and track the tasks
- Describe and organize the necessary activities
- Select, acquire and assign the adequate resources to the tasks
- Select, organize and lead testing teams
- Organize the communication between the members of the testing teams, and between the testing teams and all the other stakeholders
- Justify the decisions and provide adequate reporting information where applicable

0.2.2 Advanced Level Test Analyst.

Advanced Level Test Analysts should be able to:

- Structure the tasks defined in the test strategy in terms of business domain requirements
- Analyze the system in sufficient detail to meet the user quality expectations
- Evaluate the system requirements to determine domain validity
- Prepare and execute the adequate activities, and report on their progress
- Provide the necessary evidence to support evaluations
- Implement the necessary tools and techniques to achieve the defined goals

0.2.3 Advanced Level Technical Test Analyst.

Advanced Level Technical Test Analysts should be able to:

- Structure the tasks defined in the test strategy in terms of technical requirements
- Analyze the internal structure of the system in sufficient detail to meet the expected quality level
- Evaluate the system in terms of technical quality attributes such as performance, security, etc.
- Prepare and execute the adequate activities, and report on their progress
- Conduct technical testing activities
- Provide the necessary evidence to support evaluations
- Implement the necessary tools and techniques to achieve the defined goals

0.3 Learning Objectives / Level of Knowledge

The following learning objectives are defined as applying to this syllabus. Each topic in the syllabus will be examined according to the learning objective for it.

Level 1: Remember (K1)
The candidate will recognize, remember and recall a term or concept.
Keywords: Remember, recall, recognize, know.
Example
Can recognize the definition of "failure" as:

- "non-delivery of service to an end user or any other stakeholder" or
- "actual deviation of the component or system from its expected delivery, service or result".

Level 2: Understand (K2)
The candidate can select the reasons or explanations for statements related to the topic, and can summarize, differentiate, classify and give examples for facts (e.g. compare terms), the testing concepts, test procedures (explaining the sequence of tasks).
Keywords: Summarize, classify, compare, map, contrast, exemplify, interpret, translate, represent, infer, conclude, categorize.
Examples
Explain the reason why tests should be designed as early as possible:

- To find defects when they are cheaper to remove.
- To find the most important defects first.

Explain the similarities and differences between integration and system testing:

- Similarities: testing more than one component, and can test non-functional aspects.
- Differences: integration testing concentrates on interfaces and interactions, and system testing concentrates on whole-system aspects, such as end to end processing.

Level 3: Apply (K3)
The candidate can select the correct application of a concept or technique and apply it to a given context. K3 is normally applicable to procedural knowledge. There is no creative act involved like evaluating a software application, or creating a model for a given software. When we have a given model and cover in the syllabus the procedural steps to create test cases from a model, then it is K3.
Keywords: Implement, execute, use, follow a procedure, apply a procedure.
Example

- Can identify boundary values for valid and invalid partitions.
- Use the generic procedure for test case creation to select the test cases from a given state transition diagram (and a set of test cases) in order to cover all transitions.

Level 4: Analyze (K4)
The candidate can separate information related to a procedure or technique into its constituent parts for better understanding, and can distinguish between facts and inferences. Typical application is to analyze a document, software, project situation and propose appropriate actions to solve a problem or task.
Keywords: Analyse, differentiate, select, structure,, focus, attribute, deconstruct, evaluate, judge, monitor, coordinate, create, synthesize, generate, hypothese, plan, design, construct, produce.
Example

- Analyze product risks and propose preventive and corrective mitigation activities.
- Describe which portions of an incident report are factual and which are inferred from results.

Reference (For the cognitive levels of learning objectives)
Bloom, B. S. (1956). *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain,* David McKay, Co. Inc.
Anderson, L. W. and Krathwohl, D. R. (eds) (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon.

0.4 Learning Objectives for Test Managers

This section provides a list of detailed learning objectives for the Test Manager module.
In general all parts of this syllabus are examinable at a K1 level. That is the candidate will recognize, remember and recall a term or concept.
For this reason the table below only contains learning objectives on K2, K3 and K4 levels.

Introduction to Test Manager Syllabus – [60 minutes]
(Including revision of ISTQB® Foundation Level syllabus)

Chapter 1: Basic Aspects of Software Testing – [150 minutes]
1.2 Testing in the Software Lifecycle
- (K2) Describe how testing is a part of any software development and maintenance activity
- (K4) Analyze software life-cycle models and outline the most appropriate tasks/test activities to be executed.(distinguish between test and development activities)

1.3 Specific Systems
- (K2) Explain by giving examples the specifics of testing systems of systems
- (K2) Explain why the three main outcomes of testing safety critical systems are required to demonstrate compliance to regulations

1.4 Metrics & Measurement
- (K2) Describe and compare the standard testing related metrics
- (K3) Monitor testing activities by measuring the test object(s) and the test process

Chapter 2: Testing Processes – [120 minutes]
2.3 Test planning & control
- (K2) Describe, giving examples, how test strategies affect test planning
- (K2) Compare test work products and explain by giving examples relations between development and testing work products
- (K2) Classify test control activities related to determine if test mission, strategies, and objectives have been achieved

2.5 Test implementation & execution
- (K2) Explain the pre-conditions for test execution
- (K2) Explain by giving examples the advantages and disadvantages or early test implementation considering different testing techniques
- (K2) Explain the reasons why users and/or customers might be included in test execution.
- (K2) Describe how the level of test logging might vary depending on test level

2.6 Evaluating Exit Criteria and Reporting
- (K2) Summarize the information necessary to collect during the test process to support accurate reporting and evaluation against exit criteria

2.7 Test Closure Activities
- (K2) Summarize the four groups of test closure activities
- (K3) Generalize lessons learned in test closure phase in order to discover areas to improve or repeat

Chapter 3: Test Management – [1120 minutes]
3.2 Test Management Documentation
- (K4) Outline test management documents such as Test Plan, Test Design Specification, and Test Procedure in compliance with IEEE 829
- (K2) Describe at least 4 important elements of a test strategy / approach and which documents according to IEEE 829 contain elements of test strategy
- (K2) Illustrate how and why deviations from the test strategy are managed in the other test management documents

3.3 Test Plan Documentation
- (K2) Summarize the IEEE 829 structure of a master test plan

- (K2) Paraphrase and interpret the topics suggested by the standard IEEE 829 structure of a test plan with respect to tailoring to an organization, risk of a product, and risk, size and formality of a project

3.4 Test Estimation

- (K3) Estimate the testing effort for a small sample system using a metrics based and an experience-based approach considering the factors that influence cost effort and duration
- (K2) Understand and give examples to the factors listed in the syllabus which may lead to inaccuracies in estimates

3.5 Scheduling Test Planning

- (K2) Explain the benefits of early and iterative test planning. Support your explanation by examples

3.6 Test Progress Monitoring & Control

- (K2) Compare the different procedures for controlling test progress
- (K2) Give at least 5 conceptually different examples about how test progress findings influence the course of the test process
- (K4) Use findings related to the test progress observed during monitoring and control activities and measures in order to outline an action plan to improve the current test process. Suggest improvements
- (K4) Analyze test results and determine the test progress, documented into a monitoring report and a final test summary report covering all 4 reporting dimensions

3.7 Business Value of Testing

- (K2) Give examples (measures) for each of the 4 categories determining the “cost of quality”.
- (K3) For a given context list the quantitative and/or qualitative values that apply

3.8 Distributed, Outsourced & Insourced Testing

- (K2) List risks, commonalities and differences between the 3 staffing strategies (distributed, outsourced & in-sourced testing)

3.9 Risks-Based Testing

3.9.1 Introduction to Risk Based Testing

- (K2) Explain the different ways, how risk-based testing responds to risks
- (K4) Identify risk within a project and product, and determine the adequate test strategy and test plan based on these risks

3.9.2 Risk Management

- (K3) Execute a risk analysis for product from a testers perspective, following the specific approach FMEA
- (K4) Summarize the results from the various perspectives on risk typically held by key project stakeholders, and use their collective judgment in order to outline test activities to mitigate risks

3.9.3 Risk Management in the Lifecycle

- (K2) Describe characteristics of risk management that require it to be an iterative process
- (K3) Translate a given risk based test strategy to test activities and monitor its effects during the testing
- (K4) Analyze and report test results and determine / propose residual risks to enable project managers to make intelligent release decisions

3.10 Failure Mode and Effects Analysis

- (K2) Describe the concept of FMEA, explain its application in projects and benefits on projects by example

3.11 Test Management issues

- (K2) Compare the test management issues for the Exploratory Testing, Systems of Systems, and testing safety-critical systems related to strategy, benefits and disadvantages, adequacy and their impact on planning, coverage, and monitoring and control

Chapter 4: Test Techniques – [0 minutes]

No Learning objectives (at any K-level) apply for the test manager.

Chapter 5: Test of Software Characteristics – [0 minutes]

No Learning objectives (at any K-level) apply for the test manager.

Chapter 6: Reviews – [120 minutes]
6.2 The Principles of Reviews

- (K2) Explain the benefits of reviews compared to dynamic testing and other static testing techniques

6.4 Introducing Reviews

- (K2) Compare review types with each other and show their relative strengths, weaknesses and fields of use.
- (K3) Lead a review team through a formal review following the steps identified
- (K4) Outline a review plan as part of a quality/test plan for a project considering review techniques considering defects to be found, available skills of staff, and aligned with appropriate dynamic testing approaches

6.5 Success Factors for Reviews

- (K2) Explain the risks behind not considering the technical, organizational factors and people issues for performing reviews.

Chapter 7: Incident Management – [80 minutes]

- (K3) Process a defect following the incident management life cycle procedure as proposed by IEEE Standard 1044-1993
- (K3) Evaluate defect reports against IEEE Standard 1044-1993 and the applied defect taxonomy in order to improve their quality.
- (K4) Analyze the defect reports created over time and update the defect taxonomy

Chapter 8: Standards & Test Improvement Process – [120 minutes]

- (K2) Summarize sources of software standards and explain its usefulness for software testing

8.4 Improving the Test Processes

- (K3) Write and test improvement plan using the generic steps involving the right persons
- (K2) Summarize the testing improvement process as defined by TMM, TPI, CTP, STEP, and the process areas verification and validation in CMMI
- (K2) Explain the evaluation criteria of the test improvement models TMM, TPI, CTP, STEP, and the process areas verification and validation in CMMI

Chapter 9: Test Tool & Automation – [90 minutes]
9.2 Test Tool Concepts

- (K2) Compare the elements and aspects within each of the test tool concepts “Benefits & Risks”, “Test Tool Strategies”, “Tool Integration”, “Automation Languages”, “Test Oracles”, “Tool Deployment”, “Open Source Tools”, “Tool Development”, and “Tool Classification”
- (K2) Describe why and when it is important to create a test tool strategy or road-map for your test tool
- (K2) Understand the different phases in test tool implementation

9.3 Test Tool Categories

- (K2) Summarize the test tool categories by objectives, intended use, strengths, risks and examples
- (K2) Summarize specific requirements to test tools and open source test tools used for testing Safety Critical systems
- (K2) Describe important aspects and consequences of different Test Tools and their implementation, usage and effects on the test process.
- (K2) Describe when and why implementing your own tool is an option and its benefits, risks and consequences.

Chapter 10: People Skills – Team Composition – [240 minutes]
10.2 Individual Skills

- (K3) Use a given questionnaire in order to determine strengths and weaknesses of team members related to use of software systems, domain and business knowledge, areas of systems development, software testing and interpersonal skills

10.3 Test Team Dynamics

- (K3) Perform a gap analysis in order to determine the required technical and soft skills for open positions in an organization.

10.4 Fitting Testing Within an Organization

- (K2) Characterize the various organizational options and compare them with in-/out source and in-/off-shoring.

10.5 Motivation

- (K2) Provide example of motivating and demotivating factors for testers

10.6 Communication

- (K2) Describe by example professional, objective and effective communication in a project from the tester perspective. You may consider risks and opportunities.

0.5 Learning Objectives for Test Analysts

This section provides a list of detailed learning objectives for the Test Analyst module.
In general all parts of this syllabus is on a K1 level. That is the candidate will recognize, remember and recall a term or concept. For this reason the table below only contains learning objectives on K2, K3 and K4 levels.

Introduction to Test Analyst Syllabus – [60 minutes]
(Including revision of ISTQB® Foundation Level syllabus)

Chapter 1: Basic Aspects of Software Testing – [30 minutes]

Chapter 2: Testing Processes – [180 minutes]
2.4 Test analysis & design

- (K2) Explain the causes of functional testing taking place in specific stages of an application's life cycle
- (K2) Exemplify the criteria that influence the structure and level of test condition development
- (K2) Describe how test analysis and design are static testing techniques that can be used to discover defects
- (K2) Explain by giving examples the concept of test oracles and how a test oracle can be used in test specifications

2.5 Test implementation & execution

- (K2) Describe the pre-conditions for test execution, including: testware; test environment; configuration management; and defect management

2.6 Evaluating Exit Criteria and Reporting

- (K3) Determine from a given set of measures if a test completion criterion has been fulfilled.

Chapter 3 : Test Management – [120 minutes]
3.9.2 Risks Based Testing

- (K3) Prioritize test case selection, test coverage and test data based on risk and document this appropriately in a test schedule and test procedure
- (K2) Outline the activities of a risk based approach for planning and executing domain testing

Chapter 4 : Test Techniques – [1080 minutes]
4.2 Specification based

- (K2) List examples of typical defects to be identified by each specific specification-based techniques, provide corresponding coverage criteria
- (K3) Write test cases from given software models using the following test design techniques (The tests shall achieve a given model coverage)
    - Equivalence partitioning
    - Boundary value analysis
    - Decision tables
    - State Transition Testing
    - Classification Tree Method
    - Pairwise testing
    - Use cases
- (K4) Analyze a system, or its requirement specification, in order to determine which specification-based techniques to apply for specific objectives, and outline a test specification based on IEEE 829, focusing on functional and domain test cases and test procedures

4.4 Defect and Experience Based

- (K2) Describe the principle and reasons for defect-based techniques and differentiate its use from specification- and structure-based techniques
- (K2) Explain by examples defect taxonomies and their use

- (K2) Understand the principle of and reasons for using experienced-based techniques and when to use them
- (K3) Specify, execute and report tests using exploratory testing
- (K2) Classify defects to be identified by the different types of software fault attacks according to the defects they target
- (K4) Analyze a system in order to determine which specification-based defect-based or experienced-based techniques to apply for specific goals.

Chapter 5: Test of Software Characteristics – [210 minutes]
5.2 Quality Attributes for Domain Testing
- (K4) Explain by examples what testing techniques listed in chapter 4 are appropriate to test of accuracy, suitability, interoperability, functional security, and accessibility characteristics.
- (K3) Outline, design, specify and execute usability tests using appropriate techniques and covering given test objectives and defects to be targeted.

5.3 Quality Attributes for Technical Testing
- (K2) Explain the reasons for including efficiency, reliability and technical security tests in a testing strategy and provide examples of defects expected to be found
- (K2) Characterize non-functional test types for technical testing by typical defects to be targeted (attacked), its typical application within the application life-cycle, and test techniques suited to used for test design.

Chapter 6 : Reviews – [180 minutes]
- (K3) Use a review checklist to verify code and architecture from a testers perspective
- (K3) Use a review checklist to verify requirements and use cases from a testers perspective
- (K2) Compare review types with each other and show their relative strengths, weaknesses and fields of use.

Chapter 7 : Incident Management – [120 minutes]
- (K4) Analyze, classify and describe functional and non-functional defects in understandable defect reports

Chapter 8 : Standards & Test Improvement Process – [0 minutes]
No Learning objectives (at any K-level) apply for the test analyst.

Chapter 9: Test Tools & Automation – [90 minutes]
9.2 Test Tool Concepts
- (K2) Compare the elements and aspects within each of the test tool concepts “Benefits & Risks”, “Test Tool Strategies”, “Tool Integration”, “Automation Languages”, “Test Oracles”, “Tool Deployment”, “Open Source Tools”, “Tool Development”, and “Tool Classification”

9.3 Test Tool Categories
- (K2) Summarize the test tool categories by objectives, intended use, strengths, risks and provide examples
- (K2) Map the tools of the tool categories to different levels and types of testing

Chapter 10: People Skills – Team Composition – [30 minutes]
10.6 Communication
- (K2) Describe by example professional, objective and effective communication in a project from the tester perspective. You may consider risks and opportunities..

0.6 Learning Objectives for Technical Test Analysts

This section provides a list of detailed learning objectives for the Technical Testing Analyst module
In general all parts of this syllabus is on a K1 level. That is the candidate will recognize, remember and recall a term or concept. For this reason the table below only contains learning objectives on K2, K3 and K4 levels.

Introduction to Technical Test Analyst Syllabus –[60 minutes]
(Including revision of ISTQB® Foundation Level syllabus)

Chapter 1: Basic Aspects of Software Testing – [30 minutes]

Chapter 2: Testing Processes – [180 minutes]
2.4 Test analysis & design

- (K2) Explain the stages in an application's lifecycle where non-functional tests and architecture-based tests may be applied . Explain the causes of non-functional testing taking place only in specific stages of an application's lifecycle
- (K2) Exemplify the criteria that influence the structure and level of test condition development
- (K2) Describe how test analysis and design are static testing techniques that can be used to discover defects
- (K2) Explain by giving examples the concept of test oracles and how a test oracle can be used in test specifications

2.5 Test implementation & execution

- (K2) Describe the pre-conditions for test execution, including: testware; test environment; configuration management; and defect management

2.6 Evaluating Exit Criteria and Reporting

- (K3) Determine from a given set of measures if a test completion criterion has been fulfilled .

Chapter 3: Test Management – [120 minutes]
3.9.2 Risk Management

- (K2) Outline the activities of a risks based approach for planning and executing technical testing

Chapter 4: Test Techniques – [930 minutes]
4.2 Specification based

- (K2) List examples of typical defects to be identified by each specific specification-based techniques
- (K3) Write test cases from given software model in real-life using the following test design techniques (the tests shall achieve a given model coverage)
    - Equivalence partitioning
    - Boundary value analysis
    - Decision tables
    - State transition testing
- (K4) Analyze a system, or its requirement specification in order to determine which specification-based techniques to apply for specific objectives, and outline a test specification based on IEEE 829, focusing on component and non-functional test cases and test procedures

4.3 Structure based

- (K2) List examples of typical defects to be identified by each specific specification-based techniques
- (K3) Write test cases in real-life using the following test design techniques (The tests shall achieve a given model coverage)
    - Statement testing
    - Decision testing

    - Condition determination testing
    - Multiple condition testing
- (K4) Analyze a system in order to determine which structure-based technique to apply for specific test objectives
- (K2) Understand each structure-based technique and its corresponding coverage criteria and when to use it
- (K4) Be able to compare and analyze which structure-based technique to use in different situations

4.4 Defect and Experienced Based

- (K2) Describe the principle and reasons for defect-based techniques and differentiate its use from specification- and structure-based techniques
- (K2) Explain by examples defect taxonomies and their use
- (K2) Understand the principle of and reasons for experienced-based techniques and when to use them
- (K3) Specify, execute and report tests using exploratory testing
- (K3) Specify tests using the different types of software fault attacks according to the defects they target
- (K4) Analyze a system in order to determine which specification-based, defect-based or experienced-based techniques to apply for specific goals.

4.5 Static Analysis

- (K3) Use the algorithms “Control flow analysis”, “Data flow analysis” to verify if code has not any control or data flow anomaly
- (K4) Interpret the control and data flow results delivered from a tool in order assess if code has any control or data flow anomaly
- (K2) Explain the use of call graphs for the evaluation of the quality of architecture. This shall include the defects to be identified, the use for test design and test planning, limitations of results

4.6 Dynamic Analysis

- (K2) Explain how dynamic analysis for code can be executed and summarize he defects that can be identified using that technique, and its limitations

Chapter 5: Test of Software Characteristics – [240 minutes]

5.2 Quality Attributes for Domain Testing

- (K2) Characterize non-functional test types for domain testing by typical defects to be targeted (attacked), its typical application within the application life-cycle, and test techniques suited to used for test design.
- (K4) Specify test cases for particular types of non-functional test types and covering given test objectives and defects to be targeted.

5.3 Quality Attributes for Technical Testing

- (K2) Characterize non-functional test types for technical testing by typical defects to be targeted (attacked), its typical application within the application life-cycle, and test techniques suited to used for test design.
- (K2) Understand and explain the stages in an application’s lifecycle where security, reliability and efficiency tests may be applied (including their corresponding ISO9126 sub-attributes)
- (K2) Distinguish between the types of faults found by security, reliability and efficiency tests, (including their corresponding ISO9126 sub-attributes)
- (K2) Characterize testing approaches for security, reliability and efficiency quality attributes and their corresponding ISO9126 sub-attributes.
- (K3) Specify test cases for security, reliability and efficiency quality attributes and their corresponding ISO9126 sub-attributes.
- (K2) Understand and explain the reasons for including maintainability, portability and accessibility tests in a testing strategy
- (K3) Specify test cases for maintainability and portability types of non-functional test

Chapter 6: Reviews – [180 minutes]

- (K4) Outline a review checklist in order to find typical defects to be found with code and architecture review
- (K2) Compare review types with each other and show their relative strengths, weaknesses and fields of use.

Chapter 7: Incident Management – [120 minutes]

- (K4) Analyze, classify and describe functional and non-functional defects in understandable defect reports

Chapter 8: Standards & Test Improvement Process – [0 minutes]

No Learning objectives (at any K-level) apply for the technical test analyst.

Chapter 9: Test Tools & Automation – [210 minutes]

9.2 Test Tool Concepts

- (K2) Compare the elements and aspects within each of the test tool concepts "Benefits & Risks", "Test Tool Strategies", "Tool Integration", "Automation Languages", "Test Oracles", "Tool Deployment", "Open Source Tools", "Tool Development", and "Tool Classification"

9.3 Test Tools Categories

- (K2) Summarize the test tool categories by objectives, intended use, strengths, risks and examples
- (K2) Map the tools of the tool categories to different levels and types of testing

9.3.7 Keyword-Driven Test Automation

- (K3) Create keyword / action word tables using the key-word selection algorithm to be used by a test-execution tool
- (K3) Record tests with Capture-Replay tools in order to make regression testing possible with high quality, many testcases covered, in a short time-frame

9.3.8 Performance Testing Tools

- (K3) Design a performance test using performance test tools including planning and measurements on system characteristics

Chapter 10: People Skills – Team Composition – [30 minutes]

10.6 Communication

- (K2) Describe by example professional, objective and effective communication in a project from the tester perspective. You may consider risks and opportunities.

1. Basic Aspects of Software Testing

Terms:

Ethics, measurement, metric, safety critical systems, system of systems, software lifecycle.

1.1 Introduction

This chapter introduces some central testing themes that have general relevance for all testing professionals, whether Test Managers, Test Analysts or Technical Test Analysts. Training providers will explain these general themes in the context of the module being taught, and give relevant examples. For example, in the “Technical Test Analyst” module, the general theme of “metrics and measures” (section 1.4) will use examples of specific technical metrics, such as performance measures.

In section 1.2 the testing process is considered as part of the entire software development lifecycle. This theme builds on the basic concepts introduced in the Foundations Syllabus and pays particular attention to the alignment of the testing process with software development lifecycle models and with other IT-processes.

Systems may take a variety of forms which can influence significantly how testing is approached. In section 1.3 two specific types of system are introduced which all testers must be aware of, systems of systems (sometimes called “multi-systems”) and safety critical systems.

Advanced testers face a number of challenges when introducing the different testing aspects described in this syllabus into the context of their own organizations, teams and tasks.

1.2 Testing in the Software Lifecycle

Testing is an integral part of the various software development models such as:

- Sequential (waterfall model, V-model and W-model)
- Iterative (Rapid Application Development RAD, and Spiral model)
- Incremental (evolutionary and Agile methods)

The long-term lifecycle approach to testing should be considered and defined as part of the testing strategy. This includes organization, definition of processes and selection of tools or methods.

Testing processes are not carried out in isolation but interconnected and related to others such as:

- Requirements engineering & management
- Project management
- Configuration- and change management
- Software development
- Software maintenance
- Technical support
- Production of technical documentation

Early test planning and the later test execution are related in the sequential software development models. Testing tasks can overlap and/or be concurrent.

Change and configuration management are important supporting tasks to software testing. Without proper change management the impact of changes on the system can not be evaluated. Without configuration management concurrent evolutions may be lost or mis-managed.

Depending on the project context, additional test levels to those defined in the Foundation Level Syllabus can also be defined, such as:

- Hardware-software integration testing
- System integration testing
- Feature interaction testing
- Customer Product integration testing

Each test level has the following characteristics:

- Test goals
- Test scope
- Traceability to test basis
- Entry and Exit criteria
- Test deliverables including reporting
- Test techniques
- Measurements and metrics
- Test tools
- Compliance with organization or other standards

Depending on context, goals and scope of each test level can be considered in isolation or at project level (e.g. to avoid unnecessary duplication across different levels of similar tests).

Testing activities must be aligned to the chosen software development lifecycle model, whose nature may be sequential (e.g. Waterfall, V-model, W-model), iterative (e.g. Rapid Application Development RAD, and Spiral model) or incremental (e.g. Evolutionary and Agile methods).

For example, in the V-model, the ISTQB® fundamental test process applied to the system test level could align as follows:

- System test planning occurs concurrently with project planning, and test control continues until system test execution and closure are complete.
- System test analysis and design occurs concurrent with requirements specification, system and architectural (high-level) design specification, and component (low-level) design specification.
- System test environment (e.g. test beds, test rig) implementation might start during system design, though the bulk of it would typically occur concurrently with coding and component test, with work on system test implementation activities stretching often until just days before the start of system test execution.
- System test execution begins when the system test entry criteria are all met (or waived), which typically means that at least component testing and often also component integration testing are complete. System test execution continues until system test exit criteria are met.
- Evaluation of system test exit criteria and reporting of system test results would occur throughout system test execution, generally with greater frequency and urgency as project deadlines approach.
- System test closure activities occur after system test exit criteria are met and system test execution is declared complete, though they can sometimes be delayed until after acceptance testing is over and all project activities are finished.

For each test level, and for any selected combination of software lifecycle and test process, the test manager must perform this alignment during the test planning and/or project planning. For particularly complex projects, such as systems of systems projects (common in the military and large corporations), the test processes must be not only aligned, but also modified according to the project's context (e.g. when it is easier to detect a defect at higher level than at a lower level).

1.3 Specific Systems

1.3.1 Systems of Systems

A system of systems is a set of collaborating components (including hardware, individual software applications and communications), interconnected to achieve a common purpose, without a unique management structure. Characteristics and risks associated with systems of systems include:

- Progressive merging of the independent collaborating systems to avoid creating the entire system from scratch. This may be achieved, for example, by integrating COTS systems with only limited additional development.
- Technical and organizational complexity (e.g. among the different stakeholders) represent risks for effective management. Different development lifecycle approaches may be adopted for contributing systems which may lead to communication problems among the different teams involved (development, testing, manufacturing, assembly line, users, etc). Overall management of the systems of systems must be able to cope with the inherent technical complexity of combining the different contributing systems, and be able to handle various organizational issues such as outsourcing and offshoring.
- Confidentiality and protection of specific know-how, interfaces among different organizations (e.g. governmental and private sector) or regulatory decisions (e.g. prohibition of monopolistic behavior) may mean that a complex system must be considered as a system of systems.
- Systems of systems are intrinsically less reliable than individual systems, as any limitation from one (sub)system is automatically applicable to the whole systems of systems.
- The high level of technical and functional interoperability required from the individual components in a system of systems makes integration testing critically important and requires well-specified and agreed interfaces.

1.3.1.1 Management & Testing of Systems of Systems

Higher level of complexity for project management and component configuration management are common issues associated with systems of systems. A strong implication of Quality Assurance and defined processes is usually associated with complex systems and systems of systems. Formal development lifecycle, milestones and reviews are often associated with systems of systems.

1.3.1.2 Lifecycle Characteristics for Systems of Systems

Each testing level for a system of systems has the following additional characteristics to those described in section 1.2 Testing in the Software Lifecycle:

- Multiple levels of integration and version management
- Long duration of project
- Formal transfer of information among project members
- Non-concurrent evolution of the components, and requirement for regression tests at system of systems level
- Maintenance testing due to replacement of individual components resulting from obsolescence or upgrade

Within systems of systems, a testing level must be considered at that level of detail and at higher levels of integration. For example “system testing level” for one element can be considered as “component testing level” for a higher level component.

Usually each individual system (within a system of systems) will go through each level of testing, and then be integrated into a system of systems with the associated extra testing required.

For management issues specific to systems of systems refer to section 3.11.2.

1.3.2 Safety Critical Systems

"Safety critical systems" are those which, if their operation is lost or degraded (e.g. as a result of incorrect or inadvertent operation), can result in catastrophic or critical consequences. The supplier of the safety critical system may be liable for damage or compensation, and testing activities are thus used to reduce that liability. The testing activities provide evidence that the system was adequately tested to avoid catastrophic or critical consequences.

Examples of safety critical systems include aircraft flight control systems, automatic trading systems, nuclear power plant core regulation systems, medical systems, etc.

The following aspects should be implemented in safety critical systems:

- Traceability to regulatory requirements and means of compliance
- Rigorous approach to development and testing
- Safety analysis
- Redundant architecture and their qualification
- Focus on quality
- High level of documentation (depth and breadth of documentation)
- Higher degree of auditability.

Section 3.11.3 considers the test management issues related to safety critical systems.

1.3.2.1 Compliance to Regulations

Safety critical systems are frequently subject to governmental, international or sector specific regulations or standards (see also 8.2 Standards Considerations). Those may apply to the development process and organizational structure, or to the product being developed.

To demonstrate compliance of the organizational structure and of the development process, audits and organizational charts may suffice.

To demonstrate compliance to the specific regulations of the developed system (product), it is necessary to show that each of the requirements in these regulations has been covered adequately. In these cases, full traceability from requirement to evidence is necessary to demonstrate compliance. This impacts management, development lifecycle, testing activities and qualification/certification (by a recognized authority) throughout the development process.

1.3.2.2 Safety Critical Systems & Complexity

Many complex systems and systems of systems have safety critical components. Sometimes the safety aspect is not evident at the level of the system (or sub-system) but only at the higher level, where complex systems are implemented (for example mission avionics for aircraft, air traffic control systems).

Example: a router is not a critical system by itself, but may become so when critical information requires it, such as in telemedical services.

Risk management, which reduces the likelihood and/or impact of a risk, is essential to safety critical development and testing context (refer to chapter 3). In addition Failure Mode and Effect Analysis (FMEA) (see section 3.10) and Software Common Cause Failure Analysis are commonly used in such context.

1.4 Metrics & Measurement

A variety of metrics (numbers) and measures (trends, graphs, etc) should be applied throughout the software development life cycle (e.g. planning, coverage, workload, etc). In each case a baseline must be defined, and then progress tracked with relation to this baseline.

Possible aspects that can be covered include:

1. Planned schedule, coverage, and their evolution over time
2. Requirements, their evolution and their impact in terms of schedule, resources and tasks
3. Workload and resource usage, and their evolution over time
4. Milestones and scoping, and their evolution over time
5. Costs, actual and planned to completion of the tasks
6. Risks and mitigation actions, and their evolution over time
7. Defects found, defect fixed, duration of correction

Usage of metrics enables testers to report data in a consistent way to their management, and enables coherent tracking of progress over time.

Three areas are to be taken into account:

- Definition of metrics: a limited set of useful metrics should be defined. Once these metrics have been defined, their interpretation must be agreed upon by all stakeholders, in order to avoid future discussions when metric values evolve. Metrics can be defined according to objectives for a process or task, for components or systems, for individuals or teams. There is often a tendency to define too many metrics, instead of the most pertinent ones.
- Tracking of metrics: reporting and merging metrics should be as automated as possible to reduce the time spent in producing the raw metrics values. Variations of data over time for a specific metric may reflect other information than the interpretation agreed upon in the metric definition phase.
- Reporting of metrics: the objective is to provide an immediate understanding of the information, for management purpose. Presentations may show a “snapshot” of the metrics at a certain time or show the evolution of the metric(s) over time so that trends can be evaluated.

1.5 Ethics

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is necessary, among other reasons to ensure that the information is not put to inappropriate use. Recognizing the ACM and IEEE code of ethics for engineers, the ISTQB® states the following code of ethics:

PUBLIC- Certified software testers shall act consistently with the public interest.

CLIENT AND EMPLOYER - Certified software testers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.

PRODUCT - Certified software testers shall ensure that the deliverables they provide (on the products and systems they test) meet the highest professional standards possible.

JUDGMENT- Certified software testers shall maintain integrity and independence in their professional judgment.

MANAGEMENT - Certified software test managers and leaders shall subscribe to and promote an ethical approach to the management of software testing.

PROFESSION - Certified software testers shall advance the integrity and reputation of the profession consistent with the public interest.

COLLEAGUES - Certified software testers shall be fair to and supportive of their colleagues, and promote cooperation with software developers.

SELF - Certified software testers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

2. Testing Processes

Terms:

BS 7925/2, exit criteria, IEEE 829, test case, test closure, test condition, test control, test design, test execution, test implementation, test planning, test procedure, test script, test summary report, test log.

2.1 Introduction

In the ISTQB® Foundation Level Syllabus, the following fundamental test process was described as including the following activities:

- Planning and control
- Analysis and design
- Implementation and execution
- Evaluating exit criteria and reporting
- Test closure activities

These activities can be implemented sequentially or some can be in parallel e.g. analysis and design could be could be implemented in parallel with Implementation and execution, whereas the other activities could be implemented sequentially.

Since test management is fundamentally related to the test process, test managers must be able to apply all of this section's content to managing a specific project. For Test Analysts and Technical Test Analysts, however, the knowledge acquired at Foundation level is largely sufficient, with the exception of the test development tasks listed above. The knowledge required for these tasks is covered generally in this section and then applied in detail in chapter 4 Test Techniques and chapter 5 Testing of Software Characteristics.

2.2 Test Process Models

Process models are approximations and abstractions. Test process models do not capture the entire set of complexities, nuances, and activities that make up any real-world project or endeavor. Models should be seen as an aid to understanding and organizing, not as immutable, revealed truth.

While this syllabus uses the process described in the ISTQB® Foundations Level Syllabus (see above) as an example, there are additional important test process models, examples of three of them are listed below. They are all test process models and test process improvement models (Practical Software Testing includes the Test Maturity Model), and are defined in terms of the levels of maturity they support. All three test process models, together with TPI®, are discussed further in section 8.3 Test Improvement Process.

- Practical Software Testing – Test Maturity Model [Burnstein03]
- Critical Testing Processes [Black03]
- Systematic Test and Evaluation Process (STEP)

2.3 Test Planning & Control

This chapter focuses on the processes of planning and controlling testing.

Test planning for the most part occurs at the initiation of the test effort and involves the identification and implementation of all of the activities and resources required to meet the mission and objectives identified in the test strategy.

Risk based testing (see chapter 3 Test Management) is used to inform the test planning process regarding the mitigating activities required to reduce the product risks identified e.g. if it is identified that serious defects are usually found in the design specification, the test planning process could result in additional static testing (reviews) of the design specification before it is converted to code. Risk based testing will also inform the test planning process regarding the relative priorities of the test activities.

Complex relationships may exist among test basis, test conditions, test cases and test procedures such that many to many relationships may exist among these work products. These need to be understood to enable test planning and control to be effectively implemented.

Test control is an ongoing activity. It involves comparing actual progress against the plan and reporting the status, including deviations from the plan. Test control guides the testing to fulfill the mission, strategies, and objectives, including revisiting the test planning activities as needed.

Test control must respond to information generated by the testing as well as to changing conditions in which a project or endeavor exists. For example, if dynamic testing reveals defect clusters in areas that were deemed unlikely to contain many defects, or if the test execution period is shortened due to a delay in starting testing the risk analysis and the plan must be revised. This could result in the re-prioritization of tests and re-allocation of the remaining test execution effort.

The content of test planning documents is dealt with in chapter 3 Test Management.

Metrics to monitor test planning and control may include:

- Risk and test coverage
- Defect discovery and information
- Planned versus actual hours to develop testware and execute test cases

2.4 Test Analysis & Design

During test planning, a set of test objectives will be identified. The process of test analysis and design uses these objectives to:

- Identify the test conditions
- Create test cases that exercise the identified test conditions

Prioritization criteria identified during risk analysis and test planning should be applied throughout the process, from analysis and design to implementation and execution.

2.4.1 Identification of Test Conditions

Test conditions are identified by analysis of the test basis and objectives to determine what to test, using test techniques identified within the Test Strategy and/or the Test Plan.

The decision to determine the level and structuring of the test conditions can be based upon the functional and non-functional features of the test items using the following:

1. Granularity of the test basis: e.g. high level requirements may initially generate high level test conditions e.g. Prove screen X works, from which could be derived a low level test

condition e.g. Prove that screen X rejects an account number that is one digit short of the correct length
2. Product risks addressed: e.g. for a high risk feature detailed low level test conditions may be a defined objective
3. Requirements for management reporting and information traceability
4. Whether the decision has been taken to work with test conditions only and not develop test cases e.g. using test conditions to focus unscripted testing

2.4.2 Creation of Test Cases

Test cases are designed by the stepwise elaboration and refinement of the identified test conditions using test techniques (see chapter 4) identified in the test strategy. They should be repeatable, verifiable and traceable back to requirements.

Test case design includes the identification of:

- the preconditions such as either project or localized test environment requirements and the plans for their delivery
- the test data requirements
- the expected results and post conditions

A particular challenge is often the definition of the expected result of a test; i.e., the identification of one or more test oracles that can be used for the test. In identifying the expected result, testers are concerned not only with outputs on the screen, but also with data and environmental post-conditions.

If the test basis is clearly defined, this may theoretically be simple. However, test bases are often vague, contradictory, lacking coverage of key areas, or plain missing. In such cases, a tester must have, or have access to, subject matter expertise. Also, even where the test basis is well specified, complex interactions of complex stimuli and responses can make the definition of expected results difficult, therefore a test oracle is essential. Test execution without any way to determine correctness of results has a very low added value or benefit, generating spurious incident reports and false confidence in the system.

The activities described above may be applied to all test levels, though the test basis will vary. For example, user acceptance tests may be based primarily on the requirements specification, use cases and defined business processes, while component tests may be based primarily on low-level design specification.

During the development of test conditions and test cases, some amount of documentation is typically performed resulting in test work products. A standard for such documentation is found in IEEE 829. This standard discusses the main document types applicable to test analysis and design, Test Design Specification and Test Case Specification, as well as test implementation. In practice the extent to which test work products are documented varies considerably. This can be impacted by, for example:

- project risks (what must/must not be documented)
- the “value added” which the documentation brings to the project
- standards to be followed
- lifecycle model used (e.g. an agile approach tries to minimize documentation by ensuring close and frequent team communication)
- the requirement for traceability from test basis, through test analysis and design

Depending on the scope of the testing, test analysis and design may address the quality characteristics for the test object. The ISO 9126 standard provides a useful reference. When testing hardware/software systems, additional characteristics may apply.

The process of test analysis and design may be enhanced by intertwining it with reviews and static analysis. For example, carrying out test analysis and test design based on the requirements specification is an excellent way to prepare for a requirements review meeting. Similarly, test work products such as tests, risk analyses, and test plans should be subjected to reviews and static analyses.

During test design the required detailed test infrastructure requirements may be defined, although in practice these may not be finalized until test implementation. It must be remembered that test infrastructure includes more than test objects and testware (example: rooms, equipment, personnel, software, tools, peripherals, communications equipment, user authorizations, and all other items required to run the tests).

Metrics to monitor test analysis and design may include:

- Percentage of requirements covered by test conditions
- Percentage of test conditions covered by test cases
- Number of defects found during test analysis and design

2.5 Test Implementation & Execution

2.5.1 Test Implementation

Test implementation includes organizing the test cases into test procedures (test scripts), finalizing test data and test environments , and forming a test execution schedule to enable test case execution to begin. This also includes checking against explicit and implicit entry criteria for the test level in question.

Test procedures should be prioritized to ensure the objectives identified within the strategy are achieved in the most efficient way, e.g. running the most important test procedures first could be an approach.

The level of detail and associated complexity for work done during test implementation may be influenced by the detail of the test work products (test cases and test conditions). In some cases regulatory rules apply, and tests should provide evidence of compliance to applicable standards such as the United States Federal Aviation Administration's DO-178B/ED 12B.

As identified in 2.4 above, test data is needed for testing, and in some cases these sets of data can be quite large. During implementation, testers create input and environment data to load into databases and other such repositories. Testers also create scripts and other data generators that will create data that is sent to the system as incoming load during test execution.

During test implementation, testers should finalize and confirm the order in which manual and automated tests are to be run. When automation is undertaken, test implementation includes the creation of test harnesses and test scripts. Testers should carefully check for constraints that might require tests to be run in particular orders. Dependencies on the test environment or test data must be known and checked.

Test implementation is also concerned with the test environment(s). During this stage it should be fully set up and verified prior to test execution. A fit for purpose test environment is essential: The test environment should be capable of enabling the exposure of the defects present under test conditions, operate normally when failures are not occurring, and adequately replicate if required e.g. the production or end-user environment for higher levels of testing.

During test implementation, testers must ensure that those responsible for the creation and maintenance of the test environment are known and available and that all the testware and test support tools and associated processes are ready for use. This includes configuration management, incident management, and test logging and management. In addition, testers must verify the procedures that gather data for exit criteria evaluation and test results reporting.

It is wise to use a balanced approach to test implementation. For example, risk-based analytical test strategies are often blended with dynamic test strategies. In this case, some percentage of the test execution effort is allocated to testing which does not follow predetermined scripts.

Testing without scripts should not be ad hoc or aimless as this can be unpredictable in duration unless time boxed (see SBTM). Over the years, testers have developed a variety of experience-based techniques, such as attacks (see section 4.4 and [Whittaker03]), error guessing [Myers79], and exploratory testing. Test analysis, test design, and test implementation still occur, but they occur primarily during test execution. When following such dynamic test strategies, the results of each test influence the analysis, design, and implementation of the subsequent tests. While these strategies are lightweight and often effective at finding bugs, they require expert testers, can be unpredictable in duration, often do not provide good coverage information, and may be difficult to repeat without specific tool assistance for regression testing.

2.5.2 Test Execution

Test execution begins once the test object is delivered and entry criteria to test execution are satisfied. Tests should be executed according to the test procedures, though some amount of latitude may be given to the tester to ensure coverage of additional interesting test scenarios and behaviors that are observed during testing (any failure detected during such deviation must describe the variations from the written test procedure that are necessary to reproduce the failure). Automated tests will follow their defined instructions without deviation.

At the heart of the test execution activity is the comparison of actual results with expected results. Testers must bring the utmost attention and focus to these tasks, otherwise all the work of designing and implementing the test can be wasted when failures are missed (false positives) or correct behavior misclassified as incorrect (false negatives). If the expected and actual results do not match, an incident has occurred. Incidents must be carefully scrutinized to establish the cause (which might or might not be a defect in the test object) and to gather data to assist with the resolution of the incident. Incident management is discussed in detail in chapter 7.

When a defect is identified, the test specification should be carefully evaluated to ensure that it is correct. A test specification can be incorrect for a number of reasons, including problems in test data, defects in the test document, or a mistake in the way it was executed. If it is incorrect, it should be corrected and re-run. Since changes in the test basis and the test object can render a test specification incorrect even after it has been run successfully many times, testers should remain aware of the possibility that the observed results may therefore be due to an incorrect test.

During test execution, test results must be logged appropriately. Tests which were run but for which results were not logged may have to be repeated to identify the correct result, leading to inefficiency and delays. (Note that adequate logging can address the coverage and repeatability concerns associated with dynamic test strategies.) Since the test object, testware, and test environments may all be evolving, logging should identify the specific versions tested.

Test Logging provides a chronological record of relevant details about the execution of tests.

Results logging applies both to individual tests and to events. Each test should be uniquely identified and its status logged as test execution proceeds. Any events that affect the test execution should be logged. Sufficient information should be logged to measure test coverage and document reasons for delays and interruptions in testing. In addition, information must be logged to support test control, test progress reporting, measurement of exit criteria, and test process improvement.

Logging varies depending on the level of testing and the strategy. For example, if automated component testing is occurring, the automated tests gather most of the logging information. If manual acceptance testing is occurring, the test manager may compile or collate the log. In some cases, as with test implementation, logging is influenced by regulation or auditing requirements.

The IEEE 829 standard includes a description of information that should be captured in a test log.

- Test log identifier
- Description
- Activity and event entries

BS-7925-2 also contains a description of information to be logged.

In some cases, users or customers may participate in test execution. This can serve as a way to build their confidence in the system, though that presumes that the tests are mostly unsuccessful in finding defects. Such an assumption is often invalid in early test levels, but might be valid during acceptance test.

Metrics to monitor test implementation and execution may include:

- Percentage of test environments configured
- Percentage of test data records loaded
- Percentage of test conditions and cases executed
- Percentage of test cases automated

2.6 Evaluating Exit Criteria and Reporting

Documentation and reporting for test progress monitoring and control are discussed in detail in section 3.6. From the point of view of the test process, test progress monitoring entails ensuring the collection of proper information to support the reporting requirements. This includes measuring progress towards completion.

Metrics to monitor test progress and completion will include a mapping to the agreed exit criteria (agreed during test planning), which may include one or all of the following:

- Number of test conditions, test cases or test specifications planned and those executed broken down by whether they passed or failed
- Total defects raised, broken down by severity and priority for those fixed and outstanding
- Number of changes (change requests) raised, accepted (built) and tested
- Planned expenditure versus actual expenditure
- Planned elapsed time versus actual elapsed time
- Risks identified broken down by those mitigated by the test activity and any left outstanding
- Percentage of test time lost due to blocking events
- Retested items
- Total test time planned against effective test time carried out

For test reporting IEEE 829 specifies a Test Summary Report, consisting of the following sections:

- Test summary report identifier
- Summary
- Variances
- Comprehensive assessment
- Summary of results
- Evaluation
- Summary of activities
- Approvals

Test reporting can occur after each of the test levels is completed as well as at the end of all of the test effort.

2.7 Test Closure Activities

Once test execution is determined to be complete, the key outputs should be captured and either passed to the relevant person or archived. Collectively, these are test closure activities. Test closure activities fall into four main groups:

1. Ensuring that all test work is indeed concluded. For example, all planned tests should be either run or deliberately skipped, and all known defects should either be fixed and confirmation tested, deferred for a future release, or accepted as permanent restrictions.
2. Delivering valuable work products to those who need them. For example, known defects deferred or accepted should be communicated to those who will use and support the use of the system, and tests and test environments given to those responsible for maintenance testing. Another work product may be a Regression test pack (either automated or manual).
3. Performing or participating in retrospective meetings (Lessons Learned) where important lessons (both from within the test project and across the whole software development lifecycle) can be documented and plans established to ensure that they are either not repeated in future or where issues cannot be resolved they are accommodated for within project plans. For example,
   a. Due to late discovery of unanticipated defect clusters, the team might have discovered that a broader cross-section of user representatives should participate in quality risk analysis sessions on future projects.
   b. Actual estimates may have been significantly misjudged and therefore future estimation activities will need to account for this together with the underlying reasons, e.g. was testing inefficient or was the estimate actually lower than it should have been.
   c. Trends and cause and effect analysis of defects, by relating them back to why and when they occurred and looking for any trends: e.g. whether late change requests affected the quality of the analysis and development, or looking for bad practices: e.g. missing a test level, which would have found defects earlier and in a more cost effective manner, for perceived saving of time. Also, relating defect trends to areas such as new technologies, staffing changes, or the lack of skills
   d. Identification of potential process improvement opportunities.
4. Archiving results, logs, reports, and other documents and work products in the configuration management system, linked to the system itself. For example, the test plan and project plan should both be stored in a planning archive, with a clear linkage to the system and version they were used on.

These tasks are important, often missed, and should be explicitly included as part of the test plan.

It is common for one or more of these tasks to be omitted, usually due to premature dissolution of the team, resource or schedule pressures on subsequent projects, or team burnout. On projects carried out under contract, such as custom development, the contract should specify the tasks required.

Metrics to monitor test closure activities may include:

- Percentage of test cases run during test execution (coverage)
- Percentage of test cases checked into re-usable test case repository
- Ratio of test cases automated: to be automated
- Percentage of test cases identified as regression tests
- Percentage of outstanding defect reports closed off (e.g. deferred, no further action, change request, etc.)
- Percentage of work products identified and archived.

3. Test Management

Terms:

FMEA, level test plan, master test plan, product risk, project risk, risk-based testing, risk analysis, risk identification, risk level, risk management, risk mitigation, risk type, test control, session-based test management, test estimation, test level, test management, test monitoring, test plan, test policy, test point analysis, test scheduling, test strategy, wide band delphi.

3.1 Introduction

This entire chapter covers areas of knowledge required specially for test managers.

3.2 Test Management Documentation

Documentation is often produced as part of test management. While the specific names and scope of the test management documents tend to vary, the following are common types of test management documents found in organizations and on projects:

- Test policy, which describes the organization's philosophy toward testing (and possibly quality assurance).
- Test strategy (or test handbook), which describes the organization's methods of testing, including product and project risk management, the division of testing into steps, levels, or phases, and the high-level activities associated with testing.
- Master test plan (or project test plan, or test approach), which describes the application of the test strategy for a particular project, including the particular levels to be carried out and the relationship among those levels.
- Level test plan (or phase test plan), which describes the particular activities to be carried out within each test level, including expanding on the master test plan for the specific level being documented.

In some organizations and on some projects, these types of documents may be combined into a single document, the contents of these types of documents may be found in other documents, and some of the contents of these types of documents may be manifested as intuitive, unwritten, or traditional methodologies for testing. Larger and more formal organizations and projects tend to have all of these types of documents as written work products, while smaller and less formal organizations and projects tend to have fewer such written work products. This syllabus describes each of these types of documents separately, though in practice the organizational and project context determines the correct utilization of each type.

3.2.1 Test Policy

The test policy describes the organization's philosophy toward testing (and possibly quality assurance). It is set down, either in writing or by management direction, laying out the overall objectives about testing that the organization wants to achieve. This policy may be developed by the Information Technology, Research and Development, or Product Development department, but should reflect the organizational values and goals as they relate to testing.

In some cases, the test policy will be complementary to or a component of a broader quality policy. This quality policy describes the overall values and goals of management related to quality.

Where a written test policy exists, it may be a short, high-level document that:

- Provides a definition of testing, such as building confidence that the system works as intended and detecting defects.
- Lays out a fundamental test process, for example, test planning and control, test analysis and design, test implementation and execution, evaluating of test exit criteria and test reporting, and, test closure activities.
- Describes how to evaluate the effectiveness and efficiency of testing, for example, the percentage of defects to be detected (Defect Detection Percentage or DDP) and the relative cost of defects detected in testing as opposed to after release.
- Defines desired quality targets, such as reliability (e.g. measured in term of failure rate) or usability.
- Specifies activities for test process improvement, for example, application of the Test Maturity Model or Test Process Improvement model, or implementation of recommendations from project retrospectives.

The test policy may address test activities for new development as well as for maintenance. It may also reference a standard for testing terminology to be used throughout the organization.

3.2.2 Test Strategy

The test strategy describes the organization’s methods of testing, including product and project risk management, the division of testing into levels, or phases, and the high-level activities associated with testing. The test strategy, and the process and activities described in it, should be consistent with the test policy. It should provide the generic test requirements for the organization or for one or more projects.

As described in the Foundation Syllabus, test strategies (also called test approaches) may be classified based on when test design begins:

- Preventative strategies design tests early to prevent defects
- Reactive strategies where test design comes after the software or system has been produced.

Typical strategies (or approaches) include:

- Analytical strategies, such as risk-based testing
- Model-based strategies, such as operational profiling
- Methodical strategies, such as quality-characteristic based
- Process- or standard-compliant strategies, such as IEEE 829-based
- Dynamic and heuristic strategies, such as using bug-based attacks
- Consultative strategies, such as user-directed testing
- Regression testing strategies, such as extensive automation.

Different strategies may be combined. The specific strategy selected should be appropriate to the organization’s needs and means, and organizations may tailor strategies to fit particular operations and projects.

In many instances, a test strategy explains the project and product risks and describes how the test process manages these risks. In such instances, the connection between risks and testing should be explicitly explained, as are options for reducing and managing these risks.

The test strategy may describe the test levels to be carried out. In such cases, it should give high-level guidance on the entry criteria and exit criteria of each level and the relationships among the levels (e.g., division of test coverage objectives).

The test strategy may also describe the following:

- Integration procedures
- Test specification techniques
- Independence of testing (which may vary depending on level)
- Mandatory and optional standards

- Test environments
- Test automation
- Reusability of software work products and test work products
- Re-testing and regression testing
- Test control, and reporting
- Test measurements and metrics
- Incident management
- Configuration management approach of testware

Both short and long term test strategies should be defined. This can be done in one or more documents. Different test strategies are suitable for different organizations and projects. For example, where security- or safety-critical applications are involved, a more intensive strategy may be more appropriate than in other cases.

3.2.3 Master Test Plan

The master test plan describes the application of the test strategy for a particular project, including the particular levels to be carried out and the relationship among those levels. The master test plan should be consistent with the test policy and strategy, and, in specific areas where it is not, should explain those deviations and exceptions. The master test plan should complement the project plan or operations guide in that it should describe the testing effort that is part of the larger project or operation.

While the specific content and structure of the master test plan varies depending on the organization, its documentation standards, and the formality of the project, typical topics for a master test plan include:

- Items to be tested and not to be tested
- Quality attributes to be tested and not to be tested
- Testing schedule and budget (which should be aligned with the project or operational budget)
- Test execution cycles and their relationship to the software release plan
- Business justification for and value of testing
- Relationships and deliverables among testing and other people or departments
- Definition of what test items are in scope and out of scope for each level described
- Specific entry criteria, continuation (suspension/resumption) criteria, and exit criteria for each level and the relationships among the levels
- Test project risk.

On smaller projects or operations, where only one level of testing is formalized, the master test plan and the test plan for that formalized level will often be combined into one document. For example, if system test is the only formalized level, with informal component and integration testing performed by developers and informal acceptance testing performed by customers as part of a beta test process, then the system test plan may include the elements mentioned in this section.

In addition, testing typically depends on other activities in the project. Should these activities not be sufficiently documented, particularly in terms of their influence and relationship with testing, topics related to those activities may be covered in the master test plan (or in the appropriate level test plan). For example, if the configuration management process is not documented, the test plan should specify how test objects are to be delivered to the test team.

3.2.4 Level Test Plan

The level test plan describes the particular activities to be carried out within each test level, where necessary expanding on the master test plan for the specific level being documented. It provides schedule, task, and milestone details not necessarily covered in the master test plan. In addition, to the extent that different standards and templates apply to specification of tests at different levels, these details would be covered in the level test plan.

On less-formal projects or operations, a single level test plan is often the only test management document which is written. In such situations, some of the informational elements mentioned in sections 3.2.1, 3.2.2 and 3.2.3 could be covered in this document.

3.3 Test Plan Documentation Templates

As mentioned in section 3.2, the specific content and structure of the master test plan varies depending on the organization, its documentation standards, and the formality of the project. Many organizations develop or adapt templates to ensure commonality and readability across projects and operations, and templates are available for test plan documentation.

The IEEE 829 "Standard for Software Testing Documentation" contains test documentation templates and guidance for applying them, including for preparing test plans. It also addresses the related topic of test item transmittal (i.e., release of test items to testing).

3.4 Test Estimation

Estimation, as a management activity, is the creation of an approximate target for costs and completion dates associated with the activities involved in a particular operation or project. The best estimates:

- Represent the collective wisdom of experienced practitioners and have the support of the participants involved
- Provide specific, detailed catalogs of the costs, resources, tasks, and people involved
- Present, for each activity estimated, the most likely cost, effort and duration

Estimation of software and system engineering has long been known to be fraught with difficulties, both technical and political, though project management best practices for estimation are well established. Test estimation is the application of these best practices to the testing activities associated with a project or operation.

Test estimation should include all activities involved in the test process, i.e. test planning and control, test analysis and design, test implementation and execution, test evaluation and reporting, and test closure activities. The estimated cost, effort, and, especially, duration of test execution is often of the most interest to management, as test execution is typically on the project critical path. However, test execution estimates tend to be difficult to generate and unreliable when the overall quality of the software is low or unknown. A common practice is also to estimate the number of test cases required. Assumptions made during estimation should always be documented as part of the estimation.

Test estimation should consider all factors that can influence the cost, effort, and duration of the testing activities. These factors include (but are not limited to) the following:

- Required level of quality of the system
- Size of the system to be tested
- Historical data from previous test projects (this may also include benchmark data)
- Process factors, including: test strategy development or maintenance lifecycle and process maturity; and the accuracy of the project estimate.
- Material factors, including: test automation and tools, test environment, test data, development environment(s); project documentation (e.g., requirements, designs, etc.), and reusable test work products.
- People factors, including: managers and technical leaders; executive and senior management commitment and expectations; skills, experience, and attitudes in the project team; stability of

the project team; project team relationships; test and debugging environment support; availability of skilled contractors and consultants; and domain knowledge.

Other factors that may influence the test estimate include complexity of the process, technology, organization, number of stakeholders in the testing, many sub teams, especially geographically separated; significant ramp up, training, and orientation needs; assimilation or development of new tools, techniques, custom hardware, number of testware; requirements for a high degree of detailed test specification, especially to an unfamiliar standard of documentation; complex timing of component arrival, especially for integration testing and test development; and, fragile test data (e.g., data that is time sensitive).

Estimation can be done either bottom-up or top-down. The following techniques can be used in test estimation:

- Intuition and guesses
- Past experience
- Work-breakdown-structures (WBS)
- Team estimation sessions (e.g., Wide Band Delphi)
- Three point estimate
- Test Point Analysis (TPA) [Pol02]
- Company standards and norms
- Percentages of the overall project effort or staffing levels (e.g., tester-developer ratios)
- Organizational history and metrics, including metrics-derived models that estimate the number of defects, the number of test cycles, the number of tests cases, each tests' average effort, and the number of regression cycles involved
- Industry averages and predictive models such as test points, function points, lines of code, estimated developer effort, or other project parameters

In most cases, the estimate, once prepared, must be delivered to management, along with a justification (see section 3.7). Frequently, some negotiation ensues, often resulting in a reworking of the estimate. Ideally, the final test estimate represents the best-possible balance of organizational and project goals in the areas of quality, schedule, budget, and features.

3.5 Scheduling Test Planning

In general, planning for any set of activities in advance allows for the discovery and management of risks to those activities, the careful and timely coordination with others involved, and a high-quality plan. The same is true of test planning. However, in the case of test planning, additional benefits accrue from advanced planning based on the test estimate, including:

- Detection and management of project risks and problems outside the scope of testing itself
- Detection and management of product (quality) risks and problems prior to test execution
- Recognition of problems in the project plan or other project work products
- Opportunities to increase allocated test staff, budget, effort and/or duration to achieve higher quality
- Identification of critical components (and thus the opportunity to accelerate delivery of those components earlier)

Test scheduling should be done in close co-operation with development, since testing heavily depends on the development (delivery) schedule.

Since, by the time all the information required to complete a test plan arrives, the ability to capitalize on these potential benefits might have been lost, test plans should be developed and issued in draft form as early as possible. As further information arrives, the test plan author (typically a test manager), can add that information to the plan. This iterative approach to test plan creation, release, and review also allows the test plan(s) to serve as a vehicle to promote consensus, communication, and discussion about testing.

3.6 Test Progress Monitoring & Control

There are five primary dimensions upon which test progress is monitored:

- Product risks
- Defects
- Tests
- Coverage
- Confidence

Product risks, incidents, tests, and coverage can be and often are measured and reported in specific ways during the project or operation. Preferably these measurements are related to the defined exit criteria as stated in the test plan. Confidence, though measurable through surveys, is usually reported subjectively.

Metrics related to product risks include:

- Number of remaining risks (incl. type and level of risks)
- Number of risks mitigated (incl. type and level of risks)

Metrics related to defects include:

- Cumulative number reported (identified) versus cumulative number resolved (disposed)
- Mean time between failure or failure arrival rate
- Breakdown of the number of defects associated with: particular test items or components; root causes, sources; test releases; phase introduced, detected or removed; and, in some cases, owner
- Trends in the lag time from defect reporting to resolution

Metrics related to tests include:

- Total number planned, specified (developed), run, passed, failed, blocked, and skipped
- Regression and confirmation test status
- Hours of testing planned per day versus actual hours achieved

Metrics related to test coverage include:

- Requirements and design element coverage
- Risk coverage
- Environment/configuration coverage

These measurements may be reported verbally in narrative form, numerically in tables, or pictorially in graphs and may be used for a number of purposes, including:

- Analysis, to discover what is happening with the product, project, or process via the test results
- Reporting, to communicate test findings to interested project participants and stakeholders
- Control, to change the course of the testing or the project as a whole and to monitor the results of that course correction

Proper ways to gather, analyze, and report these test measures depends on the specific information needs, goals, and abilities of the people who will use those measurements.

When using test results to influence or measure control efforts on the project, the following options should be considered:

- Revising the quality risk analysis, test priorities, and/or test plans
- Adding resources or otherwise increasing the test effort
- Delaying the release date
- Relaxing or strengthening the test exit criteria

Implementing such options typically requires consensus among project or operation stakeholders and consent by project or operation managers.

The way a test report is set up depends largely on the target audience, e.g. project management or business management. For a project manager, having detailed information on defects is likely be of interest; to the business manager, the status of the product risks could be the key reporting issue.

The IEEE 829 “Standard for Software Testing Documentation” provides a template for reporting test summary information.

Based on divergence from the test plan as stated in the test progress report, test control should be performed. Test control is aimed at minimizing the divergence from the test plan. Possible control measures include:

- Revisit the priority of test cases
- Obtain additional resources
- Extending the release date
- Change the scope (functionality) of the project
- Revisit the test completion criteria (only with approval of stakeholders).

3.7 Business Value of Testing

While most organizations consider testing valuable in some sense, few managers, including test managers, can quantify, describe, or articulate that value. In addition, many test managers, test leads, and testers focus on the tactical details of testing (aspects specific to the task or level, while ignoring the larger strategic (higher level) issues related to testing that other project participants, especially managers, care about.

Testing delivers value to the organization, project, and/or operation in both quantitative and qualitative ways:

- Quantitative values include finding defects that are prevented or fixed prior to release, finding defects that are known prior to release, reducing risk by running tests, and delivering information on project, process, and product status.
- Qualitative values include improved reputation for quality, smoother and more-predictable releases, increased confidence, building confidence, protection from legal liability, and reducing risk of loss of whole missions or even lives.

Test managers and test leads should understand which of these values apply for their organization, project, and/or operation, and be able to communicate about testing in terms of these values.

A well-established method for measuring the quantitative value and efficiency of testing is called cost of quality (or, sometimes, cost of poor quality). Cost of quality involves classifying project or operational costs into four categories:

- Costs of prevention
- Costs of detection
- Costs of internal failure
- Costs of external failure

A portion of the testing budget is a cost of detection, while the remainder is a cost of internal failure. The total costs of detection and internal failure are typically well below the costs of external failure, which makes testing an excellent value. By determining the costs in these four categories, test managers and test leads can create a convincing business case for testing.

3.8 Distributed, Outsourced & Insourced Testing

In many cases, not all of the test effort is carried out by a single test team, composed of fellow employees of the rest of the project team, at a single and same location as the rest of the project team. If the test effort occurs at multiple locations, that test effort may be called distributed. If the test effort is carried out at one or more locations by people who are not fellow employees of the rest of the project team and who are not co-located with the project team, that test effort may be called outsourced. If the test effort is carried out by people who are co-located with the project team but who are not fellow employees, that test effort may be called insourced.

Common across all such test efforts is the need for clear channels of communication and well-defined expectations for missions, tasks, and deliverables. The project team must rely less on informal communication channels like hallway conversations and colleagues spending social time together. Location, time-zone, cultural and language differences make these issues even more critical.

Also common across all such test efforts is the need for alignment of methodologies. If two test groups use different methodologies or the test group uses a different methodology than development or project management, that will result in significant problems, especially during test execution.

For distributed testing, the division of the test work across the multiple locations must be explicit and intelligently decided. Without such guidance, the most competent group may not do the test work they are highly qualified for. Furthermore, the test work as a whole will suffer from gaps (which increase residual quality risk on delivery) and overlap (which reduce efficiency).

Finally, for all such test efforts, it is critical that the entire project team develop and maintain trust that each of the test team(s) will carry out their roles properly in spite of organizational, cultural, language, and geographical boundaries. Lack of trust leads to inefficiencies and delays associated with verifying activities, apportioning blame for problems, and playing organizational politics.

3.9 Risk-Based Testing

3.9.1 Introduction to Risk-Based Testing

Risk is the possibility of an undesired outcome. Risks exist whenever some problem may occur which would decrease customer, user, participant, or stakeholder perceptions of product quality or project success.

Where the primary effect of the potential problem is on product quality, potential problems are called product risks (or quality risks). Examples include a possible reliability defect (bug) that could cause a system to crash during normal operation. Where the primary effect of the potential problem is on project success, that potential problem is called a project risk (or a planning risk). Examples include a possible staffing shortage that could delay completion of a project.

Not all risks are of equal concern. The level of risk is influenced by different factors:

- the likelihood of the problem occurring
- the impact of the problem should it occur

In risk-based testing, testing is conducted in a manner that responds to risk in three ways:

- Allocation of test effort, selection of techniques, sequencing of test activities, and repair of defects (bugs) must be made in a way that is appropriate to the level of risk associated with each significant, identified product (quality) risk.
- Planning and management of the testing work in a way that provides mitigation and contingency responses for each significant, identified project (planning) risk.
- Reporting test results and project status in terms of residual risks; e.g. based on tests which have not yet been run or have been skipped, or defects that have not yet been fixed or retested.

These three types of responses to risk should occur throughout the lifecycle, not simply at the beginning and end of the project. Specifically, during the project, testers should seek to do the following:

- Reduce risk by finding the most important defects (for product risks) and by putting into action appropriate mitigation and contingency activities spelled out in the test strategy and test plan.

- Make an evaluation of risk, increasing or decreasing the likelihood or impact of risks previously identified and analyzed based on additional information gathered as the project unfolds.

In both cases, actions taken influence how testing responds to risk.

Risk-based testing can be seen as analogous to insurance in many ways. One buys insurance where one is worried about some potential risk, and one ignores risks that one is not worried about. Quantitative analysis similar to risk evaluation that actuaries and other insurance professionals do may be applicable, but typically risk-based testing relies on qualitative analyses.

To be able to properly carry out risk-based testing, testers should be able to identify, analyze and mitigate typical product and project risks related to safety, business and economic concerns, system and data security, and technical and political factors.

3.9.2 Risk Management

Risk management can be thought of as consisting of three primary activities:

1. Risk identification
2. Risk analysis
3. Risk mitigation (also referred to as risk control)

These activities are in some sense sequential, but the need for continuous risk management mentioned in the previous and following sections means that, during most of the project, all three types of risk management activity should be used iteratively.

To be most effective, risk management includes all stakeholders on the project, though sometimes project realities result in some stakeholders acting as surrogates for other stakeholders. For example, in mass-market software development, a small sample of potential customers may be asked to help identify potential defects that would impact their use of the software most heavily; in this case the sample of potential customers serves as a surrogate for the entire eventual customer base.

Because of their particular expertise, test analysts should be actively involved in the risk identification and analysis process.

3.9.2.1 Risk Identification

For both product and project risks, testers can identify risks through one or more of the following techniques:

- Expert interviews
- Independent assessments
- Use of risk templates
- Lessons learned (e.g. project evaluation sessions)
- Risk workshops (e.g. FMEA)
- Brainstorming
- Checklists
- Calling on past experience

By calling on the broadest possible sample of stakeholders, the risk identification process is most likely to detect the largest possible number of significant risks.

In some approaches to risk identification, the process stops at the identification of the risk itself.

Certain techniques, such as Failure Mode and Effect Analysis (FMEA), require for each potential failure mode to proceed to the identification of the effects of the failure mode on the rest of the system (including upper level systems in case of Systems of Systems), and on the potential users of the system.

Other techniques, such as Hazard Analysis, require an anticipation of the source of the risk.

For a description of the use of Failure Mode Effect, Failure Mode and Effect Analysis and Criticality Analysis, see section 3.10 and [Stamatis95], [Black02], [Craig02], [Gerrard02].

3.9.2.2 Risk Analysis

While risk identification is about identifying as many pertinent risks as possible, risk analysis is the study of these identified risks. Specifically, categorizing each risk and determining the likelihood and impact associated with each risk.

Categorization of risk means placing each risk into an appropriate type. Typical quality risk types are discussed in the ISO 9126 standard of quality characteristics to classify risks. Some organizations have their own set of quality characteristics. Note that, when using checklists as a foundation of risk identification, selection of the type of risk often occurs during the identification.

Determining the level of risk typically involves assessing, for each risk item, the likelihood of occurrence and the impact upon occurrence. The likelihood of occurrence is often interpreted as the likelihood that the potential problem can exist in the system under test. In other words, it arises from technical risk.

Factors influencing technical risk include:

- Complexity of technology and teams
- Personnel and training issues among the business analysts, designers, and programmers
- Conflict within the team
- Contractual problems with suppliers
- Geographical distribution of the development organization
- Legacy versus new approaches
- Tools and technology
- Bad managerial or technical leadership
- Time, resource and management pressure
- Lack of earlier quality assurance
- High change rates
- High earlier defect rates
- Interfacing and integration issues.

The impact upon occurrence is often interpreted as the severity of the effect on the users, customers, or other stakeholders. In other words, it arises from business risk. Factors influencing business risk include:

- Frequency of use of the affected feature
- Damage to image
- Loss of business
- Potential financial, ecological or social losses or liability
- Civil or criminal legal sanctions
- Loss of license
- Lack of reasonable workarounds
- Visibility of failure leading to negative publicity

Testers can approach establishing the level of risk either quantitatively or qualitatively. If likelihood and impact can be ascertained quantitatively, one can multiply the two values together to calculate the cost of exposure. This is the expected loss associated with a particular risk.

Typically, though, the level of risk can only be ascertained qualitatively. That is, one can speak of likelihood being very high, high, medium, low, or very low, but one cannot say for certainly whether the likelihood is 90%, 75%, 50%, 25%, or 10%. This qualitative approach should not be seen as inferior to quantitative approaches; indeed, when quantitative approaches are used inappropriately, they mislead the stakeholders about the extent to which one actually understands and can manage risk. Informal approaches such as those described in [vanVeenendaal02], [Craig02] and [Black07b] are often qualitative and less rigorous.

Unless risk analysis is based upon extensive and statistically valid risk data (as is the case in the insurance industry), regardless of whether risk analysis is qualitative or quantitative, the risk analysis will be based on the perceived likelihood and impact. This means that personal perceptions and opinions about these factors will influence the determined level of risk. Project managers, programmers, users, business analysts, architects and testers typically have different perceptions and thus possibly different opinions on the level of risk for each risk item. The risk analysis process should include some way of reaching consensus or, in the worst case, establishing through dictate an agreed upon level of risk. Otherwise, risk levels cannot be used as a guide for risk mitigation activities.

3.9.2.3 Risk Mitigation

Once a risk has been identified and analyzed, there are four main possible ways to handle that risk:

1. Mitigate the risk through preventive measures to reduce likelihood and/or impact.
2. Make contingency plans to reduce impact if the risk becomes an actuality.
3. Transfer the risk to some other party to handle.
4. Ignore and accept the risk.

The options to select one of these depend on the benefits and opportunities created by each option, as well as the cost and, potentially, additional risks associated with each option.

Mitigation strategies

In most risk-based testing strategies, risk identification, risk analysis, and the establishment of the risk mitigation activities are the foundation of the master test plan and the other test plans. The level of risk associated with each risk item determines the extent of the testing effort (i.e., mitigation action) taken to deal with each risk. Some safety related standards (e.g., FAA DO-178B/ED 12B, IEC 61508), prescribe the test techniques and degree of coverage based on the level of risk.

Project risk mitigation

If project risks are identified, they may need to be communicated to and acted upon by the project manager. Such risks are not always within the power of the testing organization to reduce. However, some project risks can and should be mitigated successfully by the test manager, such as:

- Test environment and tools readiness
- Test staff availability and qualification
- Low quality of inputs to testing
- Too high change traffic for inputs to testing
- Lack of standards, rules and techniques for the testing effort.

Approaches to risk mitigation include early preparation of testware, pre-testing test equipment, pre-testing earlier versions of the product, tougher entry criteria to testing, requirements for testability, participation in reviews of earlier project results, participation in problem and change management, and monitoring of the project progress and quality.

Product risk mitigation

When one is talking about a product (quality) risk, then testing is a form of mitigation for such risks. To the extent that one finds defects, testers reduce risk by providing the awareness of defects and opportunities to deal with them before release. To the extent testers do not find defects, testing reduces risk by ensuring that, under certain conditions (i.e., the conditions tested), the system operates correctly.

Product (quality) risks can be mitigated by non-test activities. For example, if it is identified that the requirements are not well written, an efficient solution would be to implement thorough reviews as a mitigating action, as opposed to writing and prioritizing tests that will be run once the badly written software becomes a design and actual code.

The level of risk is also used to prioritize tests. In some cases, all of the highest-risk tests are run before any lower risk tests, and tests are run in strict risk order (often called “depth-first”); in other cases, a sampling approach is used to select a sample of tests across all the identified risks using risk to weight the selection while at the same time ensuring coverage of every risk at least once (often called "breadth-first").

Other risk mitigation actions that can be used by testers include

- Choosing an appropriate test design technique
- Reviews and inspection
- Reviews of test design
- Level of independence
- Most experienced person
- The way re-testing is performed
- Regression testing

In [Gerrard02] the concept of test effectiveness is introduced to indicate (as a percentage) how effective testing is expected to be as a risk reduction measure. One would tend not to apply testing to reduce risk where there is a low level of test effectiveness.

Whether risk-based testing proceeds depth-first or breadth-first, it is possible that the time allocated for testing might be consumed without all tests being run. Risk-based testing allows testers to report to management in terms of the remaining level of risk at this point, and allows management to decide whether to extend testing or to transfer the remaining risk onto the users, customers, help desk/technical support, and/or operational staff.

Adjusting testing for further test cycles

If there is time left to test further, any next test cycles should be adapted to a new risk analysis. The main factors here are any totally new or very much changed product risks; unstable or defect-prone areas discovered during the testing; risks from fixed defects; the attempt to concentrate testing on typical faults found during testing; and, potentially under-tested areas (low test coverage). Any new or additional test cycle should be planned using an analysis of such risks as an input. It is also a highly recommended practice have an updated risk sheet at each project milestone.

3.9.3 Risk Management in the Lifecycle

Ideally, risk management occurs throughout the entire lifecycle. If an organization has a test policy document and/or test strategy, then these should describe the general process by which product and project risks are managed in testing, and show how that risk management is integrated into and affects all stages of testing.

Risk identification and risk analysis activities can begin during the initial phase of the project, regardless of the software development lifecycle model followed. Risk mitigation activities can be planned and put into place as part of the overall test planning process. In the master test plan and/or level test plans, both project and product risks can be addressed. The type and level of risk will influence the test levels in which the risks will be addressed, the extent of effort to use in mitigating the risk, the test and other techniques to apply to reduce the risk, and the criteria by which to judge adequacy of risk mitigation.

After planning is complete, risk management, including identification, analysis, and reduction, should be ongoing in the project. This includes identifying new risks, re-assessing the level of existing risks, and evaluating the effectiveness of risk mitigation activities. To take one example, if a risk identification and analysis session occurred based on the requirements specification during the requirements phase, once the design specification is finalized, a re-evaluation of the risks should occur. To take another example, if during testing a component is found to contain considerably more than the expected number of defects, one can conclude that the likelihood of defects in this area was higher than anticipated, and thus adjust the likelihood and overall level of risk upward. This could result in an increase in the amount of testing to be performed against this component.

Once the initial risk identification and analysis activities are complete, and as the reduction activities are carried out, one can measure the degree to which risks have been reduced. This can be done by tracing test cases and discovered defects back to their associated risks. As tests are run and defects found, testers can examine the remaining, residual level of risk. This supports the use of risk-based testing in determining the right moment to release. For an example of reporting test results based on risk coverage, see [Black03].

Test reporting should address risks covered and still open, as well as benefits achieved and not yet achieved.

3.10 Failure Mode and Effects Analysis

The failure mode and effects analysis (FMEA) and a variant including criticality analysis (FMECA) are iterative activities, intended to analyze the effects and criticality of failure modes within a system. The application of these analyses to software is sometimes termed SFMEA and SFMECA where the S stands for Software. In the following sections, only FMEA is used but the information is applicable to the other three methods as well.

Testers must be able to contribute to the creation of the FMEA document. This includes understanding the purpose and application of these documents as well as being able to apply their knowledge to help determine risk factors.

3.10.1 Areas of Application

FMEA should be applied:

- where the criticality of the software or system under consideration must be analyzed, to reduce the risk of failure (e.g. safety critical systems like aircraft flight control systems)
- where mandatory or regulatory requirements apply (see also section 1.3.2 Safety Critical Systems)
- to remove defects at early stages
- to define special test considerations, operational constraints, design decisions for safety critical systems

3.10.2 Implementation Steps

FMEA should be scheduled as soon as preliminary information is available at a high level, and extended to lower levels as more details become available. The FMEA can be applied at any level of the system or software decomposition depending on the information available and the requirements of a program.

For each critical function, module or component, iteratively:

- Select a function and determine its possible failure modes, i.e. how the function can fail
- Define the possible causes for those failures, their effects and design mechanisms for reduction or mitigation of the failures

3.10.3 Benefits & Costs

FMEA provides the following advantages:

- Expected system failures caused by software failures or usage errors can be revealed
- Systematic use can contribute to overall system level analysis
- Results can be used for design decisions and/or justification
- Results may be used to focus testing to specific (critical) areas of the software

The following considerations must be made when applying the FMEA:

- Sequences of failures are seldom considered
- Creation of FMEA tables can be time consuming

- Independent functions may be difficult to define
- Error propagation may not easily be identified

3.11 Test Management Issues

3.11.1 *Test Management Issues for Exploratory Testing*

Session-based test management (SBTM) is a concept for managing exploratory testing. A session is the basic unit of testing work, uninterrupted, and focused on a specific test object with a specific test objective (the test charter). At the end of a single session, a report, typically called a session sheet is produced on the activities performed. SBTM operates within a documented process structure and produces records that complement verification documentation.

A test session can be separated into three stages:

- Session Setup: Setting up the test environment and improving the understanding of the product.
- Test Design and Execution: Scanning the test object and looking for problems
- Defect Investigation and Reporting: Begins when the tester finds something that looks to be a failure.

The SBTM session sheet consists of the following:

- Session charter
- Tester name(s)
- Date and time started
- Task breakdown (sessions)
- Data files
- Test notes
- Issues
- Defect

At the end of each session the test manager holds a debriefing meeting with the team. During debriefing the manager reviews the session sheets, improves the charters, gets feedback from the testers and estimates and plans further sessions.

The agenda for debriefing session is abbreviated PROOF for the following:

- Past : What happened during the session?
- Results : What was achieved during the session?
- Outlook : What still needs to be done?
- Obstacles : What got in the way of good testing?
- Feelings : How does the tester feel about all this?

3.11.2 *Test Management Issues for Systems of Systems*

The following issues are associated with the test management of systems of systems:

- Test management is more complex because the testing of the individual systems making up the systems of systems may be conducted at different locations, by different organizations and using different lifecycle models. For these reasons the master test plan for the systems of systems typically implements a formal lifecycle model with emphasis on management issues such as milestones and quality gates. There is often a formally defined Quality Assurance process which may be defined in a separate quality plan.
- Supporting processes such as configuration management, change management and release management must be formally defined and interfaces to test management agreed. These processes are essential to ensure that software deliveries are controlled, changes are introduced in a managed way and the software baselines being tested are defined.

- The construction and management of representative testing environments, including test data, may be a major technical and organizational challenge.
- The integration testing strategy may require that simulators be constructed. While this may be relatively simple and low-cost for integration testing at earlier test levels, the construction of simulators for entire systems may be complex and expensive at the higher levels of integration testing found with systems of systems. The planning, estimating and development of simulators is frequently managed as a separate project.
- The dependencies among the different parts when testing systems of systems generate additional constraints on the system and acceptance tests. It also requires additional focus on system integration testing and the accompanying test basis documents, e.g. interface specifications.

3.11.3 *Test Management Issues for Safety Critical Systems*

The following issues are associated with the test management of safety-critical systems:

- Industry-specific (domain) standards normally apply (e.g. transport industry, medical industry, and military). These may apply to the development process and organizational structure, or to the product being developed. Refer to chapter 6 for further details.
- Due to the liability issues associated with safety critical systems, formal aspects such as requirement traceability, test coverage levels to be achieved, acceptance criteria to be achieved and required test documentation may apply in order to demonstrate compliance.
- To show compliance of the organizational structure and of the development process, audits and organizational charts may suffice.
- A predefined development lifecycle is followed, depending on the applicable standard. Such lifecycles are typically sequential in nature.
- If a system has been categorized as “critical” by an organization, the following non-functional attributes must be addressed in the test strategy and/or test plan:
  - Reliability
  - Availability
  - Maintainability
  - Safety and security

  Because of these attributes, such systems are sometimes called RAMS systems

3.11.4 *Other Test Management Issues*

Failure to plan for non-functional tests can put the success of an application at considerable risk. Many types of non-functional tests are, however, associated with high costs, which must be balanced against the risks.

There are many different types of non-functional tests, not all of which may be appropriate to a given application.

The following factors can influence the planning and execution of non-functional tests:

- Stakeholder requirements
- Required tooling
- Required hardware
- Organizational factors
- Communications
- Data security

3.11.4.1 Stakeholder Requirements

Non-functional requirements are often poorly specified or even non-existent. At the planning stage, testers must be able to obtain expectation levels from affected stakeholders and evaluate the risks that these represent.

It is advisable to obtain multiple viewpoints when capturing requirements. Requirements must be elicited from stakeholders such as customers, users, operations staff and maintenance staff; otherwise some requirements are likely to be missed.

The following essentials need to be considered to improve the testability of non-functional requirements:

- Requirements are read more often than they are written. Investing effort in specifying testable requirements is almost always cost-effective. Use simple language, consistently and concisely (i.e. use language defined in the project data dictionary). In particular, care is to be taken in the use of words such as “shall” (i.e. mandatory), “should” (i.e. desirable) and “must” (best avoided or used as a synonym for ”shall”).
- Readers of requirements come from diverse backgrounds.
- Requirements must be written clearly and concisely to avoid multiple interpretations. A standard format for each requirement should be used.
- Specify requirements quantitatively where possible. Decide on the appropriate metric to express an attribute (e.g. performance measured in milliseconds) and specify a bandwidth within which results may be evaluated as accepted or rejected. For certain non-functional attributes (e.g. usability) this may not be easy.

3.11.4.2 Required Tooling

Commercial tools or simulators are particularly relevant for performance, efficiency and some security tests. Test planning should include an estimate of the costs and timescales involved for tooling. Where specialist tools are to be used, planning should take account of learning curves for new tools or the cost of hiring external tool specialists.

The development of a complex simulator may represent a development project in its own right and should be planned as such. In particular, the planning for simulators to be used in safety-critical applications should take into account the acceptance testing and possible certification of the simulator by an independent body.

3.11.4.3 Hardware Required

Many non-functional tests require a production-like test environment in order to provide realistic measures. Depending on the size and complexity of the system under test, this can have a significant impact on the planning and funding of the tests. The cost of executing non-functional tests may be so high that only a limited amount of time is available for test execution.

For example, verifying the scalability requirements of a much-visited internet site may require the simulation of hundreds of thousands of virtual users. This may have a significant influence on hardware and tooling costs. Since these costs are typically minimized by renting (e.g. “top-up”) licenses for performance tools, the available time for such tests is limited.

Performing usability tests may require the setting up of dedicated labs or conducting widespread questionnaires. These tests are typically performed only once in a development lifecycle.

Many other types of non-functional tests (e.g. security tests, performance tests) require a production-like environment for execution. Since the cost of such environments may be high, using the production environment itself may be the only practical possibility. The timing of such test executions must be planned carefully and it is quite likely that such tests can only be executed at specific times (e.g. night-time).

Computers and communication bandwidth should be planned for when efficiency-related tests (e.g. performance, load) are to be performed. Needs depend primarily on the number of virtual users to be simulated and the amount of network traffic they are likely to generate. Failure to account for this may result in unrepresentative performance measurements being taken.

3.11.4.4 Organizational Considerations

Non-functional tests may involve measuring the behavior of several components in a complete system (e.g. servers, databases, networks). If these components are distributed across a number of different sites and organizations, the effort required to plan and co-ordinate the tests may be significant. For example, certain software components may only be available for system testing at particular times of day or year, or organizations may only offer support for testing for a limited number of days. Failing to confirm that system components and staff from other organizations are available “on call” for testing purposes may result in severe disruption to the scheduled tests.

3.11.4.5 Communications Considerations

The ability to specify and run particular types of non-functional tests (in particular efficiency tests) may depend on an ability to modify specific communications protocols for test purposes. Care should be taken at the planning stage to ensure that this is possible (e.g. that tools provide the required compatibility).

3.11.4.6 Data Security Considerations

Specific security measures implemented for a system should be taken into account at the test planning stage to ensure that all testing activities are possible. For example, the use of data encryption may make the creation of test data and the verification of results difficult.

Data protection policies and laws may preclude the generation of virtual users on the basis of production data. Making test data anonymous may be a non-trivial task which must be planned for as part of the test implementation.

4. Test Techniques

Terms:

BS 7925-2, boundary value analysis (BVA), branch testing, cause-effect graphing, classification tree method, condition testing, condition determination testing, control flow analysis, D-D path, data flow analysis, decision table testing, decision testing, defect-based technique, defect taxonomy, dynamic analysis, error guessing, equivalence partitioning, exploratory testing, experienced-based technique, LCSAJ, memory leak, multiple condition testing, pair wise testing, path testing, requirements-based testing, software attacks, specification-based technique, static analysis, statement testing, state transition testing, structure-based technique, test charter, use case testing, wild pointer.

4.1 Introduction

Test design techniques considered in this chapter include the following categories:

- Specification-based (or behavior-based or black-box)
- Structure-based (or white-box)
- Defect-based
- Experienced-based

These techniques are complementary and may be used as appropriate for any given test activity, regardless of which level of testing is being performed. While any of these techniques could be employed by Test Analysts and Technical Test Analysts, for the purposes of this syllabus, the following division is made based on most common usage:

• Specifications-based →	Test analysts and Technical test analysts
• Structure-based →	Technical test analysts
• Experienced-based →	Test analysts and Technical test analysts
• Defect-based →	Test analysts and Technical test analysts

In addition to these areas, this chapter also includes a discussion of other techniques, such as attacks, static analysis and dynamic analysis. These techniques are normally performed by technical testing analysts.
Note that specification-based techniques can be either functional or non-functional. Non-functional techniques are discussed in the next chapter.

4.2 Specification-based

Specification based techniques are a way to derive and select test conditions or test cases based on an analysis of the test basis documentation for a component or system without reference to its internal structure.

Common features of specification-based techniques:

- Models, either formal or informal, are used for the specification of the problem to be solved, the software or its components.
- From these models test cases can be derived systematically.

Some techniques also provide coverage criteria, which can be used for measuring the task of test design. Completely fulfilling the coverage criteria does not mean that the entire set of tests is complete, but rather that the model no longer suggests any useful tests based on that technique.

Specification-based tests are often requirements-based tests. Since the requirements specification should specify how the system is to behave, particularly in the area of functionality, deriving tests from the requirements is often part of testing the behavior of the system. The techniques discussed earlier in this subsection can be applied directly to the requirements specification, creating the model for system behavior from the text or graphics of the requirements specification itself, and then the tests follow as described earlier.

In the Advanced Level Syllabus, the following specifications-based test design techniques are considered:

Name	Technique	Coverage Criteria
Equivalence partitioning	See ISTQB® Foundation Syllabus section 4.3.1 for a description.	Number of covered partitions / total number of partitions.
Boundary value analysis (BVA)	See ISTQB® Foundation Syllabus section 4.3.2 for a description. Note that BVA can be applied using either 2 or 3 values. The decision which one to apply is most likely to be risk-based.	Number of distinct boundary value covered / total number of boundary values
Decision table testing and Cause-effect graphing	See ISTQB® Foundation Syllabus section 4.3.3 for a description of Decision table testing. With decision table testing every combination of conditions, relationships and constraints is tested. In addition to decision tables, a graphical technique using logical notation called cause-effect graphs can also be used. Note next to testing every combination of conditions also collapsed tables are possible. Using full decision tables or collapsed decision table is most likely to be risk-based. [Copeland03]	Number of combination of conditions covered / maximum number of combination of conditions.
State transition testing	See ISTQB® Foundation Syllabus section 4.3.4 for a description.	For single transitions, the coverage metrics is the percentage of all valid transitions exercised during test. This is known as 0-switch coverage. For n transitions the coverage measure is the percentage of all valid sequences of n transitions exercised during test. This is known as (n-1) switch coverage.
Classification tree method, orthogonal arrays and all pairs tables	Factors or variables of interest and the options or values those can take on are identified, and singletons, pairs, triples, or even higher-order combinations of those options or values are identified. [Copeland03] Classification tree method uses a graphical notation to show the test conditions (classes) and combinations addressed by the test cases [Grochtmann94]	Depends on the technique applied, e.g. pair wise is distinct from classification trees.
Use case testing	See ISTQB® Foundation Syllabus section 4.3.5 for a description.	No formal criterion applies.

Sometimes techniques are combined to create tests. For example, the conditions identified in a decision table might be subjected to equivalence partitioning to discover multiple ways in which a condition might be satisfied. Tests would then cover not only every combination of conditions, but also, for those conditions which were partitioned, additional tests would be generated to cover the equivalence partitions.

4.3 Structure-based

A structure-based test design technique, also known as white-box or code-based test techniques, is one in which the code, the data, the architecture or system flow is used as the basis for test design, with tests derived systematically from the structure. The technique determines the elements of the structure which are considered. The technique also provides coverage criteria, which say when test derivation can conclude. These criteria do not mean that the entire set of tests is complete, but rather that the structure under consideration no longer suggests any useful tests based on that technique. Each criterion has to be measured and associated with an objective defined by each project or company.

Common features of structure-based techniques:

- Information about how the software is constructed is used to derive the test cases, for example, code and design.
- The extent of coverage of the software can be measured for existing test cases, and further test cases can be derived systematically to increase coverage.

In the Advanced Level Syllabus, the following structure-based test design techniques are considered:

Name	**Technique**	**Coverage Criteria**
Statement testing	The executable (non-comment, non-whitespace) statements are identified.	number of statements executed / total number of statements
Decision testing	The decision statements, such as if/else, switch/case, for, and while statements, are identified.	Number of decision outcomes executed / total number of decision outcomes.
Branch testing	The branches, such as if/else, switch/case, for, and while statements, are identified. Note that decision and branch testing are the same at 100% coverage, but can be different at lower coverage levels	Number of branches executed / total number of branches.
Condition testing	The true/false and case labels are identified.	number of boolean operand values executed / total number of boolean operand values
Multiple condition testing	All possible combinations of true/false conditions are identified.	number of boolean operand value combinations executed / total number of boolean operand value combinations
Condition determination testing	The possible combinations of true/false conditions that can affect decisions (branches) are identified.	number of boolean operand values shown to independently affect the decision / total number of boolean operands
LCSAJ (loop testing)	The possible conditions that control loop iteration are identified. Linear Code Sequence and Jump (LCSAJ) is used to test a specific section of the	number of executed LCSAJs / total number of LCSAJs.

Name	Technique	Coverage Criteria
	code (a linear sequence of executable code) that starts at the beginning of a particular control flow and ends in a jump to another control flow or a jump to the end of the program. These code fragments are also known as DD-Paths (for decision-to-decision path). This technique is used to define specific test cases and the associated test data required to exercise the selected control flow. Designing these tests requires access to a model of the source code that defines the control flow jumps. LCSAJ can be used as a basis for code coverage measurement.	
Path testing		
	The unique sequences of statements (paths) are identified.	number of executed paths / total number of paths.

One common use for structural coverage criteria is to measure the completeness or incompleteness of a set of tests designed using specification -based, and/or experienced-based techniques. Testing tools called code coverage tools are used to capture the structural coverage achieved by these tests. If important structural coverage gaps are identified (or the coverage mandated by applicable standards is not reached), then tests are added to address those gaps using structural and other techniques.

Analysis of coverage

Dynamic testing using structure-based or other techniques may be conducted to determine if a specific code coverage is attained or not. This is used to provide evidence in the case of critical systems where a specific coverage has to be obtained (see also section 1.3.2 Safety Critical Systems). The results can indicate that some sections of the code are not exercised, and thus lead to the definition of additional test cases to exercise these sections.

4.4 Defect- and Experience-based

4.4.1 *Defect-based techniques*

A defect-based test design technique is one in which the type of defect sought is used as the basis for test design, with tests derived systematically from what is known about the defect.

The technique also provides coverage criteria which are used to determine when test derivation can conclude. As a practical matter, the coverage criteria for defect-based techniques tend to be less systematic than for behavior-based and structure-based techniques, in that the general rules for coverage are given and the specific decision about what constitutes the limit of useful coverage is discretionary during test design or test execution. The coverage criteria do not mean that the entire set of tests is complete, but rather that defects being considered no longer suggest any useful tests based on that technique.

In the Advanced Level Syllabus, the following defect-based test design technique is discussed:

Name	**Technique**	**Coverage Criteria**
Taxonomies (categories & lists of potential defects)		
	The tester who uses the taxonomy samples from the list, selecting a potential problem for analysis. Taxonomies can list root cause, defect and failure. Defect taxonomies list most common defects in the software under test. The list is used to design test cases.	Appropriate data defects and types of defects.

4.4.2 *Experienced-based techniques*

There are other test design techniques which consider defect history but do not necessarily have systematic coverage criteria. These are categorized as experienced-based test techniques

Experience-based tests utilize testers' skills and intuition, along with their experience with similar applications or technologies. These tests are effective at finding defects but not as appropriate as other techniques to achieve specific test coverage levels or producing reusable test procedures.

When using dynamic and heuristic approaches, testers tend to use experience-based tests, and testing is more reactive to events than pre-planned approaches. In addition execution and evaluation are concurrent tasks. Some structured approaches to experience-based tests are not entirely dynamic; i.e. the tests are not created entirely at the same time as they execute test.

Note that although some ideas on coverage are presented in the following table, experienced-based techniques do not have formal coverage criteria.

In this syllabus, the following experience-based test design techniques are discussed:

Name	Technique	Coverage Criteria
Error guessing	The tester uses experience to guess the potential errors that might have been made and determines the methods to uncover the resulting defects. Error guessing is also useful during risk analysis to identify potential failure modes. [Myers97]	When a taxonomy is used, appropriate data faults and types of defects. Without a taxonomy, coverage is limited by the experience of the tester and the time available.
Checklist-based	The experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified. These checklists are built based on a set of standards, experience, and other considerations. A user interface standards checklist employed as the basis for testing an application is an example of checklist-based test.	Each item on the checklist has been covered.
Exploratory	The tester simultaneously learns about the product and its defects, plans the testing work to be done, designs and executes the tests, and reports the results. Good exploratory tests are planned, interactive, and creative. The tester dynamically adjusts test goals during execution and prepares only lightweight documentation. [Veenendaal02]	Charters may be created that specify tasks, objectives, and deliverables, and an exploratory testing session is planned that identifies what to achieve, where to focus, what is in and out of scope, and what resources should be committed. In addition, a set of heuristics about defects and quality may be considered.
Attacks	The tester makes a directed and focused evaluation of system quality by attempting to force specific failures to occur. The principle of software attacks, as described in [Whittaker03], is based on the interactions of the software under test with its operating environment. This includes the user interface, operating system with the kernel, the APIs and the file systems. These interactions are based on precise exchanges of data. Any misalignment in one (or more) of the interactions can be the cause of a failure.	The different interfaces of the application being tested. The principal interfaces are the user interface, operating system, kernel, APIs, and file system.

Defect- and experienced-based techniques apply knowledge of defects and other experiences to increase defect detection, They range from “quick tests” in which the tester has no formally pre-planned activities to perform, through pre-planned sessions to scripted sessions. They are almost always useful but have particular value in the following circumstances:

- No specifications are available
- There is poor documentation of the system under test
- Insufficient time is allowed to design and create test procedures
- Testers are experienced in the domain and/or the technology
- Seek diversity from scripted testing
- Analyze operational failures

Defect- and experience-based techniques are also useful when used in conjunction with behavior-based and structure-based techniques, as they fill the gaps in test coverage that result from systematic weaknesses in these techniques.

4.5 Static Analysis

Static analysis is concerned with testing without actually executing the software under test and may relate to code or the system architecture.

4.5.1 Static Analysis of Code

Static analysis is any type of testing or examination that can be done without executing the code. There are a number of static analysis techniques and these are discussed in this section.

4.5.1.1 Control Flow Analysis

Control flow analysis provides information on the logical decision points in the software system and the complexity of their structure. Control flow analysis is described in the ISTQB® Foundation Level Syllabus and in [Beizer95].

4.5.1.2 Data Flow Analysis

Data flow analysis is a structured test technique that tests the paths between where a variable is set to where it is subsequently used. These paths are termed definition-use (du-pairs) or set-use pairs. In this method, test sets are generated to achieve 100% coverage (where possible) for each of these pairs.

This technique, although termed data flow analysis, also considers the flow of the control of the software under test as it pursues the set and use of each variable and may have to traverse the control flow of the software. See also [Beizer95]

4.5.1.3 Compliance to Coding Standards

During static analysis, compliance to coding standards can also be evaluated. Coding standards cover both the architectural aspects and the use (or prohibition of use) of some programming structures.

Compliance to coding standards enables the software to be more maintainable and testable. Specific language requirements can also be verified using static testing.

4.5.1.4 Generate code metrics

Code metrics can be generated during static analysis that will contribute to a higher level of maintainability or reliability of the code. Examples of such metrics are:

- Cyclomatic complexity
- Size
- Comment frequency
- Number of nested levels
- Number of function calls.

4.5.2 Static Analysis of Architecture

4.5.2.1 Static Analysis of a Web Site

The architecture of a web site can also be evaluated using static analysis tools. Here the objective is to check if the tree-like structure of the site is well-balanced or if there is an imbalance that will lead to:

- More difficult testing tasks
- Increased workload for maintenance
- Difficulty of navigation for the user

Some specific testing tools that include a web spider engine can also provide, via static analysis, information on the size of the pages and on the time necessary to download it, and on whether the page is present or not (i.e. http error 404). This provides useful information for the developer, the webmaster and the tester.

4.5.2.2 Call Graphs

Call graphs can be used for a number of purposes:

- Designing tests that call the specific module
- Establishing the number of locations within the software from where the module is called
- Providing a suggestion for the order of integration (pair-wise and neighboring integration [Jorgensen02])
- Evaluate the structure of the total code and its architecture.

Information on calling modules can also be obtained during dynamic analysis. The information obtained refers to the number of times a module is being called during execution. By merging the information obtained from call graphs during static analysis with the information obtained during dynamic analysis, the tester can focus on modules that are called often and/or repeatedly. If such modules are also complex (see 1.3.2.2 Safety Critical Systems & Complexity) they are prime candidates for detailed and extensive testing.

4.6 Dynamic analysis

The principle of dynamic analysis is to analyze an application while it is running. This frequently requires some kind of instrumentation, inserted in the code either manually or automatically.

4.6.1 Overview

Defects that are not immediately reproducible can have significant consequences on testing effort and on the ability to release or productively use software. Such defects may be caused by memory leaks, incorrect use of pointers and other corruptions (e.g. of the system stack) [Kaner02]. Due to the nature of these defects, which may include the gradual worsening of system performance or even system crashes, testing strategies must consider the risks associated with such defects and, where appropriate, perform dynamic analysis to reduce them (typically by using tools).

Dynamic analysis is performed while the software is being executed. It may be applied to:

- Prevent failures from occurring by detecting wild pointers and loss of system memory
- Analyze system failures which cannot easily be reproduced
- Evaluate network behavior
- Improve system performance by providing information on run-time system behavior

Dynamic analysis may be performed at any test level, but is particularly useful in component and integration testing. Technical and system skills are required to specify the testing objectives of dynamic analysis and, in particular, to analyze results.

4.6.2 Detecting Memory Leaks

A memory leak occurs when the memory (RAM) available to a program is allocated by that program but subsequently not released due to programming errors. The program thereby loses the ability to access this piece of memory and could eventually run out of usable memory.

Memory leaks cause problems which develop over time and may not always be immediately obvious. This may be the case if, for example, the software has been recently installed or the system restarted, which often occurs during testing. For these reasons, the negative affects of memory leaks may often first be noticed when the program is in production.

The symptoms of a memory leak are a steadily worsening of system response time which may ultimately result in system failure. While such failures may be resolved by re-starting (re-booting) the system, this may not always be practical or even possible.

Tools identify areas in the code where memory leaks occur so that they can be corrected. Simple memory monitors can also be used to obtain a general impression of whether available memory is reducing over time, although a follow-up analysis would still be required if this were the case.

There are other kinds of leaks that should be considered. Examples include file handles, semaphores and connection pools for resources.

4.6.3 Detecting Wild Pointers

"Wild" pointers within a program are pointers which are in some way unusable. For example, they may have "lost" the object or function to which they should be pointing or they do not point to the area of memory intended (e.g. beyond the allocated boundaries of an array). When a program uses wild pointers, a variety of consequences may occur:

1. The program may perform as expected. This may be the case where the wild pointer accesses memory which is currently not used by the program and is notionally "free".
2. The program may crash. In this case the wild pointer may have caused a part of the memory to be used which is critical to the running of the program (e.g. the operating system).
3. The program does not function correctly because objects required by the program cannot be accessed. Under these conditions the program may continue to function, although an error message may be issued.
4. Data may be corrupted by the pointer and incorrect values subsequently used.

Note that any changes made to the program's memory usage (e.g. a new build following a software change) may trigger any of the four consequences listed above. This is particularly critical where initially the program performs as expected despite the use of wild pointers, and then crashes unexpectedly (perhaps even in production) following a software change. It is important to note that such failures are symptoms of an underlying error (i.e. the wild pointer) but not the error itself. (Refer to [Kaner02], "Lesson 74").

Tools may identify wild pointers as they are used by the program, irrespective of their consequences on the program's execution.

4.6.4 Analysis of Performance

Dynamic analysis may be conducted to analyze program performance. Tools help identify performance bottlenecks and generate a wide range of performance metrics which can be used by the developer to "tune" the system. This is also referred to as "performance profiling".

5. Testing of Software Characteristics

Terms

Accessibility testing, accuracy testing, efficiency testing, heuristic evaluation, interoperability testing, maintainability testing, operational acceptance test (OAT), operational profile, portability testing, recoverability testing, reliability growth model, reliability testing, security testing, suitability testing, SUMI, usability testing

5.1 Introduction

While the previous chapter described specific techniques available to the tester, this chapter considers the application of those techniques in evaluating the principal attributes used to describe the quality of software applications or systems.

In this syllabus the quality attributes which may be evaluated by a Test Analyst and Technical Test Analyst are considered in separate sections. The description of quality attributes provided in ISO 9126 is used as a guide to describing the attributes.

An understanding of the various quality attributes is a basic learning objective of all three modules. Depending on the specific quality attribute covered, a deeper understanding is developed in either the test analyst or the technical test analyst module, so that typical risks can be recognized, appropriate testing strategies developed and test cases specified.

5.2 Quality attributes for domain testing

Functional testing is focused on "what" the product does. The test basis for functional testing is generally a requirements or specification document, specific domain expertise or implied need. Functional tests vary according to the test level or phase in which they are conducted. For example, a functional test conducted during integration testing will test the functionality of interfacing modules which implement a single defined function. At the system test level, functional tests include testing the functionality of the application as a whole. For systems of systems, functional testing will focus primarily on end to end testing across the integrated systems.

A wide variety of test techniques is employed during functional test (see section 4). Functional testing may be performed by a dedicated tester, a domain expert, or a developer (usually at the component level).

The following quality attributes are considered:

- Accuracy
- Suitability
- Interoperability
- Functional security
- Usability
- Accessibility

5.2.1 Accuracy Testing

Functional accuracy involves testing the application's adherence to the specified or implied requirements and may also include computational accuracy. Accuracy testing employs many of the test techniques explained in chapter 4.

5.2.2 Suitability Testing

Suitability testing involves evaluating and validating the appropriateness of a set of functions for its intended specified tasks. This testing can be based on use cases or procedures.

5.2.3 Interoperability Testing

Interoperability testing tests whether a given application can function correctly in all intended target environments (hardware, software, middleware, operating system, etc.). Specifying tests for interoperability requires that combinations of the intended target environments are identified, configured and available to the test team. These environments are then tested using a selection of functional test cases which exercises the various components present in the environment.

Interoperability relates to how different software systems interact with each other. Software with good interoperability characteristics can be integrated easily with a number of other systems without requiring major changes. The number of changes and the effort required to perform those changes may be used as a measure of interoperability.

Testing for software interoperability may, for example, focus on the following design features:

- The software's use of industry-wide communications standards, such as XML.
- Ability of the software to automatically detect the communications needs of the systems it interacts with and switch accordingly.

Interoperability testing may be particularly significant for

- organizations developing Commercial Off the Shelf (COTS) software and tools
- developing systems of systems

This form of testing is primarily performed in system integration testing.

5.2.4 Functional Security Testing

Functional security testing (penetration testing) focuses on the ability of the software to prevent unauthorized access, whether accidental or deliberate, to functions and data. User rights, access and privileges are included in this testing. This information should be available in the specifications for the system. Security testing also includes a number of aspects which are more relevant for Technical Test Analysts and are discussed in section 5.3 below.

5.2.5 Usability Testing

It is important to understand why users might have difficulty using the proposed software system. To do this it is first necessary to appreciate that the term “user” may apply to a wide range of different types of persons, ranging from IT experts to children or people with disabilities.

Some national institutions (e.g. the British Royal National Institute for the Blind), recommend that web pages are accessible for disabled, blind, partially sighted, mobility impaired, deaf and cognitively-disabled users. Checking that applications and web sites are usable for the above users, would also improve the usability for everyone else.

Usability testing measures the suitability of the software for its users, and is directed at measuring the following factors with which specified users can achieve specified goals in particular environments or contexts of use:

- Effectiveness: the capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use
- Efficiency: the capability of the product to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use
- Satisfaction: the capability of the software product to satisfy users in a specified context of use

Attributes that may be measured are:

- Understandability: attributes of the software that affect the effort required by the user to recognize the logical concept and its applicability
- Learnability: attributes of software that affect the effort required by the user to learn the application
- Operability: attributes of the software that affect the effort required by the user to conduct tasks effectively and efficiently
- Attractiveness: the capability of the software to be liked by the user

Usability evaluation has two purposes:

- To remove usability defects (sometimes referred to as formative evaluation)
- To test against usability requirements (sometimes referred to as summative evaluation)

Tester skills should include expertise or knowledge in the following areas:

- Sociology
- Psychology
- Conformance to national standards (for example, American Disabilities Act)
- Ergonomics

Performing validation of the actual implementation should be done under conditions as close as possible to those under which the system will be used. This may involve setting up a usability lab with video cameras, mock up offices, review panels, users, etc. so that development staff can observe the effect of the actual system on real people.

Many usability tests may be executed as part of other tests, for example during functional system test. To achieve a consistent approach to the detection and reporting of usability faults in all stages of the lifecycle, usability guidelines may be helpful.

5.2.5.1 Usability Test Specification

Principal techniques for usability testing are:

- Inspection, evaluation or review
- Performing verification and validation of the actual implementation
- Performing surveys and questionnaires

Inspection evaluation or review

Inspection or review of the specification and designs from a usability perspective that increase the user’s level of involvement can be cost effective in finding problems early.

Heuristic Evaluation (systematic inspection of a user interface design for usability) can be used to find the usability problems in the design so that they can be attended to as part of an iterative design process. This involves having a small set of evaluators examine the interface and judge its compliance with recognized usability principles (the "heuristics").

Validation of the actual implementation

For performing validation of the actual implementation, tests specified for functional system test may be developed as usability test scenarios. These test scenarios measure specific usability attributes, such as speed of learning or operability, rather than functional outcomes.

Test scenarios for usability may be developed to specifically test syntax and semantics.

- Syntax: the structure or grammar of the interface (e.g. what can be entered to an input field)
- Semantics: meaning and purpose (e.g. reasonable and meaningful system messages and output provided to the user)

Techniques used to develop these test scenarios may include:

- Black box methods as described, for example, in BS-7925-2
- Use Cases, either in plain text or defined with UML (Unified Modeling Language)

Test scenarios for usability testing include user instructions, allowance of time for pre and post test interviews for giving instructions and receiving feedback and an agreed protocol for running the sessions. This protocol includes a description of how the test will be carried out, timings, note taking and session logging, and the interview and survey methods to be used.

Surveys and questionnaires

Survey and questionnaire techniques may be applied to gather observations of user behavior with the system in a usability test lab. Standardized and publicly available surveys such as SUMI (Software Usability Measurement Inventory) and WAMMI (Website Analysis and MeasureMent Inventory) permit benchmarking against a database of previous usability measurements. In addition, since SUMI provides concrete measurements of usability, this provides a good opportunity to use them as completion / acceptance criteria.

5.2.6 Accessibility Testing

It is important to consider the accessibility of software to those with particular requirements or restrictions in its use. This includes those with disabilities. It should consider the relevant standards, such as the Web Content Accessibility Guidelines, and legislation, such as Disability Discrimination Acts (UK, Australia) and Section 508 (US).

5.3 Quality attributes for technical testing

Quality attributes for Technical Test Analysts focus on "how" the product works, rather than the functional aspects of "what" it does. These tests can take place at any test level, but have particular relevance for:

Component test (especially real time and embedded systems)

- Performance benchmarking
- Resource usage

System Test and Operational Acceptance Test (OAT)

- Includes any of the quality attributes and sub-attributes mentioned below, according to risks and available resources
- Technically-oriented tests at this level are aimed at testing a specific system, i.e. combinations of hardware and software, including servers, clients, databases, networks and other resources

Frequently, the tests continue to be executed after the software has entered production, often by a separate team or organization. Measurements of quality attributes gathered in pre-production tests may form the basis for Service Level Agreements (SLA) between the supplier and the operator of the software system.

The following quality attributes are considered:

- Technical security
- Reliability
- Efficiency
- Maintainability

- Portability

5.3.1 *Technical Security Testing*

Security testing differs from other forms of domain or technical testing in two significant areas:

1. Standard techniques for selecting test input data may miss important security issues
2. The symptoms of security faults are very different from those found with other types of testing

Many security vulnerabilities exist where the software not only functions as designed, but also performs extra actions which are not intended. These side-effects represent one of the biggest threats to software security. For example, a media player which correctly plays audio but does so by writing files out to unencrypted temporary storage exhibits a side-effect which may be exploited by software pirates.

The principal concern regarding security is to prevent information from unintended use by unauthorized persons. Security testing attempts to compromise a system's security policy by assessing a system's vulnerability to threats, such as

- Unauthorized copying of applications or data (e.g. piracy)
- Unauthorized access control (e.g. ability to perform tasks for which the user does not have rights)
- Buffer overflow (buffer overrun), which may be caused by entering extremely long strings into a user interface input field. Once a buffer overflow has been caused, an opportunity for running malicious code instructions may exist.
- Denial of service, which prevents users from interacting with an application (e.g. by overloading a web-server with "nuisance" requests)
- Eavesdropping on data transfers via networks in order to obtain sensitive information (e.g. on credit card transactions)
- Breaking the encryption codes used to protect sensitive data
- Logic bombs (sometimes called Easter Eggs in the USA), which may be maliciously inserted into code and which activate only under certain conditions (e.g. on a specific date). When logic bombs activate, they may perform malicious acts like the deletion of files or formatting of disks.

Particular security concerns may be grouped as follows:

- User interface related
  - Unauthorized access
  - Malicious inputs
- File system related
  - Access to sensitive data stored in files or repositories
- Operating System related
  - Storage of sensitive information such as passwords in non-encrypted form in memory. Crashing a system through malicious inputs may expose this information.
- External software related
  - Interactions which may occur among external components that the system utilizes. These may be at the network level (e.g. incorrect packets or messages passed) or at the software component level (e.g. failure of a software component on which the software relies).

It should be noted that improvements which may be made to the security of a system may affect its performance. After making security improvements it is advisable to consider the repetition of performance tests

5.3.1.1 Security Test Specification

The following approach may be used to develop security tests.

- Perform information retrieval
  A profile or network map of the system is constructed using widely available tools. This information may include names of employees, physical addresses, details regarding the internal networks, IP-numbers, identity of software or hardware used, and operating system version.
- Perform a vulnerability scan
  The system is scanned using widely available tools. Such tools are not used directly to compromise the systems, but to identify vulnerabilities that are, or that may result in, a breach of security policy. Specific vulnerabilities can also be identified using checklists such as those provided at www.testingstandards.co.uk
- Develop “attack plans” (i.e. a plan of testing actions intended to compromise a particular system’s security policy) using the gathered information. Several inputs via various interfaces (e.g. user interface, file system) need to be specified in attack plans to detect the most severe security faults.
- The various “attacks” described in [Whittaker04] are a valuable source of techniques developed specifically for security testing. For more information on “attacks” refer to section 4.4.

5.3.2 Reliability Testing

An objective of reliability testing is to monitor a statistical measure of software maturity over time and compare this to a desired reliability goal. The measures may take the form of a Mean Time Between Failures (MTBF), Mean Time to Repair (MTTR) or any other form of failure intensity measurement (e.g. number of failures per week of a particular severity). The development of the monitored values over time can be expressed as a Reliability Growth Model.

Reliability testing may take the form of a repeated set of predetermined tests, random tests selected from a pool or test cases generated by a statistical model. As a result, these tests may take a significant time (in the order of days).

Analysis of the software maturity measures over time may be used as exit criteria (e.g. for production release). Specific measures such as MTBF and MTTR may be established as Service Level Agreements and monitored in production.

Software Reliability Engineering and Testing (SRET) is a standard approach for reliability testing.

5.3.2.1 Tests for Robustness

While functional testing may evaluate the software’s tolerance to faults in terms of handling unexpected input values (so-called negative tests), technically oriented tests evaluate a system’s tolerance to faults which occur externally to the application under test. Such faults are typically reported by the operating system (e.g. disk full, process or service not available, file not found, memory not available). Tests of fault tolerance at the system level may be supported by specific tools.

5.3.2.2 Recoverability Testing

Further forms of reliability testing evaluate the software system’s ability to recover from hardware or software failures in a predetermined manner which subsequently allows normal operations to be resumed. Recoverability tests include Failover and Backup & Restore tests.

Failover tests are performed where the consequences of a software failure are so high that specific hardware and/or software measures have been implemented to ensure system operation even in the event of failure. Failover tests may be applicable, for example, where the risk of financial losses is extreme or where critical safety issues exist. Where failures may result from catastrophic events this form of recoverability testing may also be called “disaster recovery” testing.

Typical hardware measures might include load balancing across several processors, clustering servers, processors or disks so that one can immediately take over from another should it fail (e.g.

RAID: Redundant Array of Inexpensive Disks). A typical software measure might be the implementation of more than one independent instance of a software system (for example, an aircraft's flight control system) in so-called redundant dissimilar systems. Redundant systems are typically a combination of software and hardware measures and may be called duplex, triplex or quadruplex systems, depending on the number of independent instances (2, 3 or 4 respectively). The dissimilar aspect for the software is achieved when the same software requirements are provided to two (or more) independent and not connected development teams, with the objective of having the same services provided with different software. This protects the redundant dissimilar systems in that a similar defective input is less likely to have the same result. These measures taken to improve the recoverability of a system may directly influence its reliability as well and may also be considered when performing reliability testing.

Failover testing is designed to explicitly test such systems by simulating failure modes or performing them in a controlled environment. Following failure the failover mechanism is tested to ensure that data is not lost or corrupted and that any agreed service levels are maintained (e.g. function availability, response times). For more information on failover testing, see www.testingstandards.co.uk.

Backup and Restore tests focus on the procedural measures set up to minimize the effects of a failure. Such tests evaluate the procedures (usually documented in a manual) for taking different forms of backup and for restoring that data should a loss or corruption of data take place. Test cases are designed to ensure that critical paths through the procedure are covered. Technical reviews may be performed to "dry-run" these scenarios and validate the manuals against the actual installation procedure. Operational Acceptance Tests (OAT) exercise the scenarios in a production or production-like environment to validate their actual use.

Measures for Backup and Restore tests may include the following:

- Time taken to perform different types of backup (e.g. full, incremental)
- Time taken to restore data
- Levels of guaranteed data backup (e.g. recovery of all data no more than 24 hours old, recovery of specific transaction data no more than one hour old)

5.3.2.3 Reliability Test Specification

Reliability tests are mostly based on patterns of use (sometimes referred to as "Operational Profiles") and can be performed formally or according to risk. Test data may be generated using random or pseudo-random methods.

The choice of reliability growth curve should be justified and tools can be used to analyze a set of failure data to determine the reliability growth curve that most closely fits the currently available data.

Reliability tests may specifically look for memory leaks. The specification of such tests requires that particular memory-intensive actions be executed repeatedly to ensure that reserved memory is correctly released.

5.3.3 Efficiency Testing

The efficiency quality attribute is evaluated by conducting tests focused on time and resource behavior. Efficiency testing relating to time behavior is covered below under the aspects of performance, load, stress and scalability testing.

5.3.3.1 Performance Testing

Performance testing in general may be categorized into different test types according to the non-functional requirements in focus. Test types include performance, load, stress and scalability tests.

Specific performance testing focuses on the ability of a component or system to respond to user or system inputs within a specified time and under specified conditions (see also load and stress below). Performance measurements vary according to the objectives of the test. For individual software components performance may be measured according to CPU cycles, while for client-based systems performance may be measured according to the time taken to respond to a particular user request. For systems whose architectures consist of several components (e.g. clients, servers, databases) performance measurements are taken between individual components so that performance "bottlenecks" can be identified.

5.3.3.2 Load Testing

Load testing focuses on the ability of a system to handle increasing levels of anticipated realistic loads resulting from the transaction requests generated by numbers of parallel users. Average response times of users under different scenarios of typical use (operational profiles) can be measured and analyzed. See also [Splaine01]

There are two sub-types of load testing, multi-user (with realistic numbers of users) and volume testing (with large numbers of users). Load testing looks at both response times and network throughput.

5.3.3.3 Stress Testing

Stress testing focuses on the ability of a system to handle peak loads at or beyond maximum capacity. System performance should degrade slowly and predictably without failure as stress levels are increased. In particular, the functional integrity of the system should be tested while the system is under stress in order to find possible faults in functional processing or data inconsistencies.

One possible objective of stress testing is to discover the limits at which the system actually fails so that the "weakest link in the chain" can be determined. Stress testing allows additional components to be added to the system in a timely manner (e.g. memory, CPU capability, database storage).

In spike testing, combinations of conditions which may result in a sudden extreme load being placed on the system are simulated. "Bounce tests" apply several such spikes to the system with periods of low usage between the spikes. These tests will determine how well the system handles changes of loads and whether it is able to claim and release resources as needed. See also [Splaine01].

5.3.3.4 Scalability Testing

Scalability testing focuses on the ability of a system to meet future efficiency requirements, which may be beyond those currently required. The objective of the tests is to judge the system's ability to grow (e.g. with more users, larger amounts of data stored) without exceeding agreed limits or failing. Once these limits are known, threshold values can be set and monitored in production to provide a warning of impending problems.

5.3.3.5 Test of Resource Utilization

Efficiency tests relating to resource utilization evaluate the usage of system resources (e.g. memory space, disk capacity and network bandwidth). These are compared under both normal loads and stress situations, such as high levels of transaction and data volumes.

For example for real-time embedded systems memory usage (sometimes referred to as a "memory footprint") plays a significant role in performance testing.

5.3.3.6 Efficiency Test Specification

The specification of tests for efficiency test types such as performance, load and stress are based on the definition of operational profiles. These represent distinct forms of user behavior when interacting with an application. There may be several operational profiles for a given application.

The numbers of users per operational profile may be obtained by using monitoring tools (where the actual or comparable application is already available) or by predicting usage. Such predictions may be based on algorithms or provided by the business organization, and are especially important for specifying the operational profile for scalability testing.

Operational profiles are the basis for test cases and are typically generated using test tools. In this case the term “virtual user” is typically used to represent a simulated user within the operational profile.

5.3.4 Maintainability Testing

Maintainability tests in general relate to the ease with which software can be analyzed, changed and tested. Appropriate techniques for maintainability testing include static analysis and checklists.

5.3.4.1 Dynamic Maintainability Testing

Dynamic maintainability testing focuses on the documented procedures developed for maintaining a particular application (e.g. for performing software upgrades). Selections of maintenance scenarios are used as test cases to ensure the required service levels are attainable with the documented procedures.

This form of testing is particularly relevant where the underlying infrastructure is complex, and support procedures may involve multiple departments/organizations. This form of testing may take place as part of Operational Acceptance Testing (OAT). [www.testingstandards.co.uk]

5.3.4.2 Analyzability (corrective maintenance)

This form of maintainability testing focuses on measuring the time taken to diagnose and fix problems identified within a system. A simple measure can be the mean time taken to diagnose and fix an identified fault.

5.3.4.3 Changeability, Stability and Testability (adaptive maintenance)

The maintainability of a system can also be measured in terms of the effort required to make changes to that system (e.g. code changes). Since the effort required is dependent on a number of factors such as software design methodology (e.g. object orientation), coding standards etc., this form of maintainability testing may also be performed by analysis or review. Testability relates specifically to the effort required to test the changes made. Stability relates specifically to the system’s response to change. Systems with low stability exhibit large numbers of “knock-on” problems (also known as “ripple effect”) whenever a change is made. [ISO9126] [www.testingstandards.co.uk]

5.3.5 Portability Testing

Portability tests in general relate to the ease with which software can be transferred into its intended environment, either initially or from an existing environment. Portability tests include tests for installability, co-existence/compatibility, adaptability and replaceability.

5.3.5.1 Installability Testing

Installability testing is conducted on the software used to install other software on its target environment. This may include, for example, the software developed to install an operating system onto a processor, or an installation “wizard” used to install a product onto a client PC. Typical installability testing objectives include:

- Validation that the software can be successfully installed by following the instructions in an installation manual (including the execution of any installation scripts), or by using an installation wizard. This includes exercising install options for different HW/SW-configurations and for various degrees of installation (e.g. full or partial)
- Testing whether failures which occur during installation (e.g. failure to load particular DLLs) are dealt with by the installation software correctly without leaving the system in an undefined state (e.g. partially installed software or incorrect system configurations)
- Testing whether a partial installation/de-installation can be completed
- Testing whether an installation wizard can successfully identify invalid hardware platform or operating system configurations

- Measuring whether the installation process can be completed within a specified number of minutes or in less than a specified number of steps
- Validation that the software can be successfully downgraded or de-installed

Functionality testing is normally conducted after the installation test to detect any faults which may have been introduced by the installation (e.g. incorrect configurations, functions not available). Usability testing is normally conducted in parallel to installability testing (e.g. to validate that users are provided with understandable instructions and feedback/error messages during the installation).

5.3.5.2 Co-Existence

Computer systems which are not related to each other are said to be compatible when they can run in the same environment (e.g. on the same hardware) without affecting each other's behavior (e.g. resource conflicts). Compatibility tests may be performed, for example, where new or upgraded software is rolled-out into environments (e.g. servers) which already contain installed applications.

Compatibility problems may arise where the application is tested in an environment where it is the only installed application (where incompatibility issues are not detectable) and then deployed onto another environment (e.g. production) which also runs other applications.

Typical compatibility testing objectives include:

- Evaluation of possible adverse impact on functionality when applications are loaded on the same environment (e.g. conflicting resource usage when a server runs multiple applications).
- Evaluation of the impact to any application resulting from the deployment of operating system fixes and upgrades.

Compatibility testing is normally performed when system and user acceptance testing have been successfully completed.

5.3.5.3 Adaptability Testing

Adaptability testing tests whether a given application can function correctly in all intended target environments (hardware, software, middleware, operating system, etc.). Specifying tests for adaptability requires that combinations of the intended target environments are identified, configured and available to the test team. These environments are then tested using a selection of functional test cases which exercise the various components present in the environment.

Adaptability may relate to the ability of software to be ported to various specified environments by performing a predefined procedure. Tests may evaluate this procedure.

Adaptability tests may be performed in conjunction with installability tests and are typically followed by functional tests to detect any faults which may have been introduced in adapting the software to a different environment.

5.3.5.4 Replaceability Testing

Replaceability focuses on the ability of software components within a system to be exchanged for others. This may be particularly relevant for systems which use commercial off-the-shelf software (COTS) for specific system components.

Replaceability tests may be performed in parallel to functional integration tests where more than one alternative component is available for integration into the complete system. Replaceability may be evaluated by technical review or inspection, where the emphasis is placed on the clear definition of interfaces to potential replacement components.

6. Reviews

Terms

Audit, IEEE 1028, informal review, inspection, inspection leader, management review, moderator, review, reviewer, technical review, walkthrough.

6.1 Introduction

A successful review process requires planning, participation and follow-up. Training providers will need to ensure that test managers understand the responsibilities they have for the planning and follow-up activities. Testers must be active participants in the review process, providing their unique views. They should have formal review training to better understand their respective roles in any technical review process. All review participants must be committed to the benefits of a well-conducted technical review. When done properly, inspections are the single biggest, and most cost-effective, contributor to overall delivered quality. An international standard on reviews is IEEE 1028.

6.2 The Principles of Reviews

A review is a type of static testing. Reviews frequently have as a major objective the detection of defects. Reviewers find defects by directly examining documents.

The fundamental types of reviews are described in section 3.2 of the Foundation Level Syllabus (version 2005) and are listed below in chapter 6.3.

All types of review are best executed as soon as the relevant source documents (documents which describe the project requirements) and standards (to which the project must adhere) are available. If one of the documents or standards is missing, then faults and inconsistencies across all documentation cannot be discovered, only those within one document can be discovered. Reviewers must be provided with the document to be reviewed in adequate time to allow them to become familiar with the contents of the document.

All types of documents can be subjected to a review, e.g. source code, requirements specifications, concepts, test plans, test documents, etc. Dynamic testing normally follows a source code review; it is designed to find any defects that cannot be found by static examination.

A review can lead to three possible results:

- The document can be used unchanged or with minor changes
- The document must be changed but a further review is not necessary
- The document must be extensively changed and a further review is necessary

The roles and responsibilities of those involved in a typical formal review are covered in the Foundation Syllabus, i.e. manager, moderator or leader, author, reviewers and scribe. Others who may be involved in reviews include decision makers or stakeholders, and customer or user representatives. An additional optional role sometimes used in inspections is that of a reader, who is intended to paraphrase sections of the work product in the meeting. In addition to review roles, individual reviewers may each be assigned a defect-based role to look for particular types of defect.

More than one of the review types may be employed on a single product. For example, a team may hold a technical review to decide which functionalities to implement in the next iteration. An inspection might then be performed on the specifications for the included functionalities.

6.3 Types of Reviews

The Foundation Syllabus introduced the following types of review:

- Informal review
- Walkthrough
- Technical review
- Inspection

Hybrids of these types of reviews may also occur in practice, such as a technical review using rule sets.

6.3.1 Management review and audit

In addition to the types mentioned in the Foundation Syllabus, IEEE 1028 also describes the following types of review:

- Management review
- Audit

The key characteristics of a management review are:

- Main purposes: to monitor progress, assess status, and make decisions about future actions
- Carried out by or for managers having direct responsibility for the project or system
- Carried out by or for a stakeholder or decision maker, e.g. a higher level manager or director
- Checks consistency with and deviations from plans, or adequacy of management procedures
- Includes assessment of project risks
- Outcome includes action items and issues to be resolved
- Participants expected to prepare, decisions are documented

Note that test managers should participate in and may instigate management reviews of testing progress.

Audits are extremely formal, and are usually performed to demonstrate conformance to some set of expectations, most likely an applicable standard or a contractual obligation. As such, audits are the least effective at revealing defects.

The key characteristics of an audit are:

- Main purpose: provide independent evaluation of compliance to processes, regulations, standards etc.
- A lead auditor is responsible for the audit and acts as the moderator
- Auditors collect evidence of compliance through interviews, witnessing and examining documents
- Outcome includes observations, recommendations, corrective actions and a pass/fail assessment

6.3.2 Reviews of particular work products

Reviews may be described in terms of the work products or activities that are subject to reviews, such as:

- Contractual review
- Requirements review
- Design review
  - o preliminary design review
  - o critical design review
- Acceptance review / qualification review
- Operational readiness review

A contractual review may be associated with a contract milestone, and would typically be a management review for a safety-critical or safety-related system. It would involve managers, customers and technical staff.

A requirement review may be a walkthrough, technical review or inspection, and may consider safety and dependability requirements as well as functional and non-functional requirements. A requirement review may include acceptance criteria and test conditions.

Design reviews are typically technical reviews or inspections, and involve technical staff and customers or stakeholders. The Preliminary Design Review proposes the initial approach to some technical designs and tests; the Critical Design Review covers all of the proposed design solutions, including test cases and procedures.

Acceptance reviews are to obtain management approval for a system. This is also referred to as a Qualification Review, and is normally a management review or audit.

6.3.3 Performing a formal review

The Foundation Syllabus describes six phases of a formal review: planning, kick-off, individual preparation, review meeting, rework and follow-up. The work product to be reviewed should be appropriate for the qualification or the reviewer, e.g. a Test Plan for a Test Manager, a business requirements or test design for a Test Analyst, or functional specification, test cases or test scripts for Technical Test Analyst.

6.4 Introducing Reviews

In order for reviews to be successfully introduced into an organization, the following steps should occur (not necessarily in this order):

- Securing management support
- Educating managers about the costs, benefits and implementation issues
- Selecting and documenting review procedures, forms and infrastructure (e.g. reviews metrics database)
- Training in review techniques and procedures
- Obtaining support from those who will be doing reviews and having their work reviewed
- Executing pilot reviews
- Demonstrating the benefit of reviews through cost savings
- Applying reviews to the most important documents, e.g. requirements, contracts, plans etc.

Metrics such as reducing or avoiding cost of fixing defects and/or their consequences may be used to evaluate the success of the introduction of reviews. Savings may also be measured in elapsed time saved by finding and fixing defects early.

Review processes should be continually monitored and improved over time. Managers should be aware that learning a new review technique is an investment – the benefits are not instant but will grow significantly over time.

6.5 Success Factors for Reviews

There are a number of factors that contribute to successful reviews. Reviews need not be difficult to perform, but they can go wrong in various ways if factors such as these are not considered.

Technical factors

- Ensure the defined process for the review type is followed correctly, particularly for more formal types of review such as inspection
- Record the costs of reviews (including time spent) and benefits achieved
- Review early drafts or partial documents to identify patterns of defects before they are built into the whole document
- Ensure that the documents or partial documents are review-ready before starting a review process (i.e. apply entry criteria)
- Use organization-specific checklists of common defects
- Use more than one type of review, depending on objectives, such as document cleanup, technical improvement, transfer of information, or progress management
- Review or inspect documents on which important decisions will be made, for example, inspect a proposal, contract or high level requirement before a management review authorizing major expenditure on the project
- Sample a limited subset of a document for assessment not clean-up
- Encourage finding the most important defects: focus on content not format
- Continuously improve the review process

Organizational factors

- Ensure managers allow adequate time to be spent on review activities, even under deadline pressure
- Remember, time and budget spent are not in proportion to the errors found.
- Allow adequate time for rework of defects identified by reviews
- Never use the metrics from reviews for individual performance appraisal
- Ensure that the right people are involved in the different types of review
- Provide training in reviews, particularly the more formal review types
- Support a review leader forum to share experience and ideas
- Ensure everyone participates in reviews and everyone has their own documents reviewed
- Apply the strongest review techniques to the most important documents
- Ensure a well-balanced review team of people with different skills and backgrounds
- Support process improvement actions must be supported to address systemic problems
- Recognize improvements gained through the review process

People issues

- Educate stakeholders to expect that defects will be found and to allow time for rework and re-review
- Ensure the review is a positive experience for the author
- Welcome the identification of defects in a blame-free atmosphere
- Ensure comments are constructive, helpful and objective, not subjective
- Do not review if the author does not agree or is not willing
- Encourage everyone to think deeply about the most important aspects of the documents being reviewed

For more information on reviews and inspection see [Gilb93] and [Weigers02].

7. Incident Management

Terms

IEEE 829, IEEE 1044, IEEE 1044.1, anomaly, configuration control board, defect, error, failure, incident, incident logging, priority, root cause analysis, severity

7.1 Introduction

Test managers, and testers must be familiar with the defect management process. Test managers focus on the process, including methods to recognize, track and remove defects. Testers are primarily concerned with accurately recording issues found in their test areas. For each of the steps in the lifecycle, test analysts and technical test analysts will have a different orientation. Test analysts will evaluate the behavior in terms of business and user needs, e.g., would the user know what to do when faced with this message or behavior. The technical test analysts will be evaluating the behavior of the software itself and will likely do more technical investigation of the problem, e.g., testing the same failure on different platforms or with different memory configurations.

7.2 When can a Defect be detected?

An incident is an unexpected occurrence that requires further investigation. An incident is the recognition of a failure caused by a defect. An incident may or may not result in the generation of a defect report. A defect is an actual problem that has been determined to require resolution by changing the work item.

A defect can be detected through static testing. A failure can be detected only through dynamic testing. Each phase of the Software Life Cycle should provide a method for detecting and eliminating potential failures. For example, during the development phase, code and design reviews should be used to detect defects. During dynamic test, test cases are used to detect failures. The earlier a defect is detected and corrected, the lower the cost of quality for the system as a whole. It should be remembered that defects can exist in testware as well as in the test object.

7.3 Defect Lifecycle

All defects have a lifecycle, although some may skip some stages. The defect lifecycle (as described in IEEE 1044-1993) is composed of four steps:

- Step 1: Recognition
- Step 2: Investigation
- Step 3: Action
- Step 4: Disposition

Within each step are three information capture activities:

- Recording
- Classifying
- Identifying impact

7.3.1 Step 1: Recognition

The Recognition step occurs when a potential defect (incident) is discovered. This can occur in any phase of the Software Life Cycle. At the time of discovery, the data items that identify the defect are recorded. This includes such information as the environment in which the defect was observed, the originator, the description information, the time and the vendor (if applicable). The recognition is classified by identifying certain attributes of the potential defect, including project activity, project phase, suspected cause, repeatability, symptom(s) and product status as a result of the anomaly. With this information, the impact is assessed by rating the severity, project schedule impact and project cost impact.

7.3.2 Step 2: Investigation

After Recognition, each potential defect is investigated. This step is primarily used to find any related issues and propose solutions, which may include no action (e.g. potential defect is no longer considered an actual defect). Additional data is recorded at this step as well as a re-evaluation of the classification and impact information supplied in the previous step.

7.3.3 Step 3: Action

Based on the results of the Investigation, the Action step commences. Actions include those required to resolve the defect as well as any actions indicated for revising/improving processes and policies in order to prevent future similar defects. Regression testing and retesting, as well as progression testing must be performed for each change. Additional data is recorded at this step as well as a re-evaluation of the classification and impact information supplied in the previous step.

7.3.4 Step 4: Disposition

The anomaly then moves to the Disposition step where any additional data items are recorded and the disposition classification is set to either closed, deferred, merged or referred to another project.

7.4 Defect Fields

IEEE 1044-1993 specifies a set of mandatory fields that are set at various times in the defect's lifecycle. A large set of optional fields is also defined. According to IEEE 1044.1, when implementing IEEE 1044-1993, it is acceptable to create a mapping between the IEEE terms for defect fields and the associated names used by an individual company. This allows conformance with IEEE 1044-1993 without have to tightly adhere to the naming conventions. IEEE conformance allows comparison of defect information across multiple companies and organizations.

Regardless of whether IEEE conformance is a goal, the fields supplied for a defect are intended to provide enough information so the defect is actionable. An actionable defect report is:

- Complete
- Concise
- Accurate
- Objective

In addition to resolving the specific defect, information must also be supplied for accurate classification, risk analysis, and process improvement.

7.5 Metrics & Incident Management

Defect information needs to include enough information to assist in test progress monitoring, defect density analysis, found vs. fixed metrics and convergence metrics (open vs. closed). In addition, defect information needs to support process improvement initiatives by tracking phase containment information, root cause analysis and identifying defect trends to be used as input to strategic risk mitigation adjustments.

7.6 Communicating Incidents

Incident management includes effective communication that is free from accusations and supports the gathering and interpretation of objective information. The accuracy of the incident reports, proper classification and demonstrated objectivity are imperative to maintain professional relations among the people reporting defects and the people resolving the defects. Testers may be consulted regarding the relative importance of a defect and should provide available objective information.

Defect triage meetings may be conducted to assist in proper prioritization. A defect tracking tool should not be used as a substitute for good communication nor should triage meetings be used as a substitute for not using a good defect tracking tool. Both communication and adequate tool support are necessary for an effective defect process.

8. Standards & Test Improvement Process

Terms

Capability Maturity Model (CMM), Capability Maturity Model Integration (CMMI), Test Maturity Model (TMM), Test Maturity Model integration (TMMi), Test Process Improvement (TPI).

8.1 Introduction

Support for establishing and improving test processes can come from a variety of sources. This section first considers standards as a useful (sometimes mandatory) source of information for a number of test-related topics. Knowing which standards are available and where they may be applied is considered a learning objective for test managers and testers. Training providers should highlight those specific standards which are particularly relevant to the module being taught.

Once established, a test process should undergo continuous improvement. In section 8.3 generic improvement issues are first covered, followed by an introduction to some specific models which can be used for test process improvement. While test managers will need to understand all of the material in this section, it is also important that test analysts and technical test analysts, as key players in the implementation of improvements, are aware of the improvement models available.

8.2 Standards Considerations

In this, and in the Foundation Level Syllabus, some standards are mentioned. There are standards for a number of topics related to software such as:

- Software development lifecycles
- Software testing and methodologies
- Software configuration management
- Software maintenance
- Quality Assurance
- Project Management
- Requirements
- Software languages
- Software interfaces
- Defect management

It is not the purpose of this syllabus to list or recommend specific standards. The testers should be aware of how standards are created, and how they should be used within the user's environment.

Standards can come from different sources:

- International or with international objectives
- National, such as national applications of international standards
- Domain specific, such as when international or national standards are adapted to particular domains, or developed for specific domains

Some considerations apply when using standards. These are described further in this section.

8.2.1 General Aspects on Standards

8.2.1.1 Sources of Standards

Standards are created by groups of professionals and reflect the collective wisdom. There are two major sources of international standards: IEEE and ISO. National and domain specific standards are also important and available.

Testers should be aware of the standards that apply to their environment and context, whether formal standards (international, national or domain specific) or in-house standards and recommended practices. As standards evolve and change it is necessary to specify the version (publication date) of the standard to ensure that compliance is obtained. When a reference to a standard is specified without the date or version, then the latest version is considered to be referenced.

8.2.1.2 Usefulness of Standards

Standards can be used as an instrument to promote constructive quality assurance, which focuses on minimizing defects introduced rather than finding them through testing (analytical quality assurance). Not all standards are applicable to all projects; the information stated in a standard may be useful for a project, or may hinder it. Following a standard for the sake of following a standard will not help the tester find more defects in a work product [Kaner02]. However standards can provide some reference framework, and provide a basis on which to define test solutions.

8.2.1.3 Coherence & Conflicts

Some standards can lack coherence with other standards, or even provide conflicting definitions. It is up to the standards users to determine the usefulness of the different standards in his/her environment and context.

8.2.2 International Standards

8.2.2.1 ISO

ISO [www.iso.org] stands for International Standards Organization (recently changed to International Organization for Standardization) and is made up of members representing, for their country, the national body most representative of standardization. This international body has promoted a number of standards useful for software testers, such as:

- ISO 9126:1998, that is now split into the following standard and technical reports (TR):
  - ISO/IEC 9126-1:2001 Software engineering -- Product quality -- Part 1: Quality model
  - ISO/IEC TR 9126-2:2003 Software engineering -- Product quality -- Part 2: External metrics
  - ISO/IEC TR 9126-3:2003 Software engineering -- Product quality -- Part 3: Internal metrics
  - ISO/IEC TR 9126-4:2004 Software engineering -- Product quality -- Part 4: Quality in use metrics
- ISO 12207:1995/Amd 2:2004 Systems and Software Engineering -- Software Lifecycle Processes
- ISO/IEC 15504[1]-2:2003 Information technology -- Process assessment -- Part 2: Performing an assessment

This list is not exhaustive; other ISO standards may be applicable to a tester's context and environment.

[1] ISO 15504 is also known as SPICE, and derived from the SPICE project

8.2.2.2 IEEE

IEEE [www.ieee.org] is the Institute of Electrical and Electronics Engineer, a professional organization based in the USA. National representatives are available in more than one hundred countries. This organization has proposed a number of standards that are useful for software testers such as:

- IEEE 610:1991 IEEE standard computer dictionary. A compilation of IEEE standard computer glossaries
- IEEE 829:1998 IEEE standard for software test documentation
- IEEE 1028:1997 IEEE standard for software reviews
- IEEE 1044:1995 IEEE guide to classification for software anomalies

This list is not exhaustive; other IEEE standards may be applicable to your context and environment.

8.2.3 National Standards

Many countries have their own specific standards. Some of these standards are applicable and/or useful for software testing. One such British standard is BS-7925-2:1998 “Software testing. Software component testing” that provides information related to many of the test techniques described in this syllabus, including:

- Equivalence Partitioning
- Boundary Value Analysis
- State Transition Testing
- Cause-Effect Graphing
- Statement Testing
- Branch/Decision Testing
- Condition Testing
- Random Testing

BS-7925-2 also provides a process description for Component Testing

8.2.4 Domain Specific Standards

Standards are also present in different technical domains. Some industries tailor other standards for their specific technical domains. Here also are aspects of interest in software testing, software quality and software development.

8.2.4.1 Avionics System

RTCA DO-178B/ED 12B “Software Considerations in Airborne Systems and Equipment Certification” is applicable to software used in civilian aircrafts. This standard also applies to software used to create (or verify) such software used in aircrafts. For avionics software, the United States Federal Aviation Administration (FAA) and the international Joint Aviation Authorities (JAA) prescribe certain structural coverage criteria based on the level of criticality of the software being tested:

Criticality Level	**Potential Impact of Failure**	**Required Structural Coverage**
A Catastrophic	Prevent continued safe flight and landing	Condition determination, Decision, and Statement
B Hazardous / Severe-Major	Large reduction in safety margins or functional capabilities Crew can not be relied upon to perform their tasks accurately or completely Serious or fatal injuries to a small number of occupants	Decision and Statement
C Major	Significant reduction in safety margins Significant increase in crew workload	Statement

Criticality Level	Potential Impact of Failure	Required Coverage	Structural
	Discomfort to occupants possibly including injuries		
D Minor	Slight reduction of aircraft safety margins of functional capabilities Slight increase in crew workload Some inconvenience to occupants	None	
E No effect	No impact on the capabilities of the aircraft No increase in crew workload	None	

The appropriate level of structural coverage must be achieved depending on the criticality level of the software that will be certified for use in civilian aviation.

8.2.4.2 Space Industry

Some industries tailor other standards for their specific domain. This is the case for the space industry with the ECSS (European Cooperation on Space Standardization) [www.ecss.org]. Depending on the criticality of the software, the ECSS recommends methods and techniques which are consistent with the ISTQB® Foundation and Advanced Syllabus including:

- SFMECA - Software Failure Modes, Effects and Criticality Analysis
- SFTA - Software Fault Tree Analysis
- HSIA - Hardware Software Interaction Analysis
- SCCFA - Software Common Cause Failure Analysis

8.2.4.3 Food & Drug Administration

- For medical systems subject to Title 21 CFR Part 820, the United States Food and Drug Administration (FDA) recommends certain structural and functional test techniques.

The FDA also recommends testing strategies and principles which are consistent with the ISTQB® Foundation and Advanced Level Syllabus.

8.2.5 Other Standards

The number of standards available in the different industries is very large. Some are adaptations to specific domains or industries, some are applicable to specific tasks or provide explanation on how to apply one standard.

It is up to the tester to be aware of the different standards (including in-house standards, best practices, etc.) that are applicable within his domain, industry or context. Sometimes the applicable standards are specified, with hierarchical applicability to specific contracts. It is up to the test manager to be aware of the standards that have to be complied with, and ensure that adequate compliance is maintained.

8.3 Test Improvement Process

Just as testing is used to improve software, software quality processes are selected and used to improve the process of developing software (and the resulting software deliverables). Process improvement can also be applied to the testing processes. Different ways and methods are available to improve the testing of software and of systems containing software. These methods aim at improving the process (and hence the deliverables) by providing guidelines and areas of improvement.

Testing often accounts for a major part of the total project costs. However, only limited attention is given to the test process in the various software process improvement models, such as CMMI (see below for details).

Test improvement models such as the Test Maturity Model (TMM), Systematic Test and Evaluation Process (STEP), Critical Testing Processes (CTP) and Test Process Improvement (TPI) were developed to cover this aspect. TPI and TMM provide a degree of cross-organization metrics that can be used for “benchmark” comparisons. There are many improvement models available in industry today. In addition to those covered in this section, Test Organization Maturity (TOM), Test Improvement Model (TIM) and Software Quality Rank (SQR) should also be considered. There are also a large number of regional models that are in use today. Testing professionals should research all available models to determine the best fit for their situation.

The models presented in this section are not intended to be a recommendation for use but are shown here to provide a representative view of how the models work and what is included within them.

8.3.1 Introduction to Process Improvement

Process improvements are relevant to the software development process as well as for the testing process. Learning from one's own mistakes makes it possible to improve the process organizations are using to develop and test software. The Deming improvement cycle: Plan, Do, Check, Act, has been used for many decades, and is still relevant when testers need to improve the process in use today.

One premise for process improvement is the belief that the quality of a system is highly influenced by the quality of the process used to develop the software. Improved quality in the software industry reduces the need for resources to maintain the software and thus provides more time for creating more and better solutions in the future.

Process models provide a place to start improving, by measuring the organization’s maturity processes with the model. The model also provides a framework for improving the organization’s processes based on the outcome of an assessment.

A Process Assessment leads to a Process Capability Determination, which motivates a Process Improvement. This may invoke a Process Assessment later to measure the effect of the improvement.

8.3.2 Types of Process Improvement

There are two types of models: process reference models and content reference models.

1. The process reference model is used as a framework when an assessment is done, in order to evaluate an organization’s capability compared with the model, and to evaluate the organization within the framework.
2. The content reference model is used to improve the process once the assessment is done.

Some models may have both parts built in whereas others will only have one.

8.4 Improving the Test Process

The IT industry has started to work with test process improvement models as it seeks to reach a higher level of maturity and professionalism. Industry standard models are helping to develop cross-organization metrics and measures that can be used for comparison. The staged models, like TMMi and CMMI provide standards for comparison across different companies and organizations. The continuous models allow an organization to address its highest priority issues with more freedom in the order of implementation. Out of the need for process improvement in the testing industry, several standards have materialized. These include STEP, TMMi, TPI and CTP. These are each discussed further in this section.

All four of these test process assessment models allow an organization to determine where they stand in terms of their current test processes. Once an assessment is performed, TMMi and TPI provide a prescriptive roadmap for improving the test process. STEP and CTP instead provide the organization with means to determine where its greatest process improvement return on investment will come from and leave it to the organization to select the appropriate roadmap.

Once it has been agreed that test processes should be reviewed and improved, the process steps to be adopted for this activity should be as follows:

- **I**nitiate
- **M**easure
- **P**rioritize and Plan
- Define and **R**edefine
- **O**perate
- **V**alidate
- **E**volve

Initiate
In this activity the confirmation of the stakeholders, the objectives, goals, scope and coverage of the process improvements is agreed. The choice of the process model upon which the improvements will be identified is also made during this activity. The model could be either selected from those published or defined individually.

Before the process improvement activity starts, success criteria should be defined and a method by which they will be measured throughout the improvement activity should be implemented.

Measure
The agreed assessment approach is undertaken culminating in a list of possible process improvements.

Prioritize & Plan
The list of possible process improvements are put in order of priority. The order could be based upon return on investment, risks, alignment to organizational strategy, measurable quantitative or qualitative benefits.

Having established the priority order, a plan for the delivery of the improvements should then be developed and deployed.

Define & Redefine
Based upon the process improvement requirements identified, where new processes are required they are defined, and where existing processes require an update they are redefined and made ready for deployment.

Operate
Once developed, the process improvements are deployed. This could include any training or mentoring required, piloting of processes and ultimately the full deployment of them.

Validate
Having fully deployed the process improvements, it is key that any benefits that were agreed before the improvement(s) were made are validated e.g., benefit realization. It is also important that any success criteria for the process improvement activity have been met.

Evolve
Dependent on the process model used, this stage of the process is where monitoring of the next level of maturity starts and a decision is made to either start the improvement process again, or to stop the activity at this point.

The use of assessment models is a common method which ensures a standardized approach to improving test processes using tried and trusted practices. Test process improvement can however also be accomplished without models by using, for example, analytical approaches and retrospective meetings.

8.5 Improving the Test Process with TMM

The Testing Maturity Model is composed of five levels and is intended to complement CMM. Each of the levels contains defined process areas that must be completely fulfilled before the organization can advance to the next level (i.e. staged representation). TMM provides both a process reference model and a content reference model.

The TMM levels are:

Level 1: Initial

> The initial level represents a state where there is no formally documented or structured testing process. Tests are typically developed in an ad hoc way after coding, and testing is seen as the same as debugging. The aim of testing is understood to be proving that the software works.

Level 2: Definition

> The second level can be reached by setting testing policy and goals, introducing a test planning process, and implementing basic testing techniques and methods.

Level 3: Integration

> The third level is reached when a testing process is integrated into the software development lifecycle, and documented in formal standards, procedures, and methods. There should be a distinct software testing function that can be controlled and monitored.

Level 4: Management & Measurement

> Level four is achieved when the testing process is capable of being effectively measured, managed, and adapted to specific projects.

Level 5: Optimization

> The final level represents a state of test process maturity, where data from the testing process can be used to help prevent defects, and the focus is on optimizing the established process

To achieve a particular level a number of pre-defined maturity goals and sub-goals must be achieved. These goals are defined in terms of activities, tasks and responsibilities and assessed according to specified “views” for manager, developer/tester and customer/user. For more information on TMM, see [Burnstein03].

The TMMi Foundation [see www.tmmifoundation.org for details] has defined the successor of TMM: TMMi. TMMi is a detailed model for test process improvement based on the TMM framework as developed by the Illinois Institute of Technology and practical experiences of using TMM and positioned as being complementary to the CMMI.

The structure of the TMMi is largely based on the structure of the CMMI (e.g, process areas, generic goals, generic practices, specific goals, specific practices).

8.6 Improving the Test Process with TPI

TPI (Test Process Improvement) uses a continuous representation rather than the staged representation of TMM.

The test process optimization plan as outlined in [Koomen99] involves a set of key areas that are set within the four cornerstones of Lifecycle, Organization, Infrastructure and tools, and Techniques. Key areas can be evaluated at a level between A to D, A being low. It is also possible for very immature key areas not to achieve the initial level A. Some key areas may only be rated at A or B (such as Estimating and Planning) while others (such as Metrics) may be rated at A, B, C or D.

The level achieved for a given key area is assessed by evaluating checkpoints defined in the TPI model. If, for example, all checkpoints for key area "reporting" are answered positively at level A and B, then level B is achieved for this key area.

The TPI model defines dependencies between the various key areas and levels. These dependencies ensure that the test process is developed evenly. It is not possible, for example, to achieve level A in key area "metrics" without also achieving level A in key area "reporting" (i.e. what is the use of taking metrics if they are not reported). The use of dependencies is optional in the TPI model.

TPI is primarily a process reference model.

A Test Maturity Matrix is provided that maps the levels (A, B, C or D) for each key area to an overall test process maturity level. These overall levels are

- Controlled
- Efficient
- Optimizing

During a TPI assessment, quantitative metrics and qualitative interviews are used to establish the level of test process maturity.

8.7 Improving the Test Process with CTP (CTP)

As described in [Black02], the basic premise of the Critical Testing Process assessment model is that certain testing processes are critical. These critical processes, if carried out well, will support successful test teams. Conversely, if these activities are carried out poorly, even talented individual testers and test managers are unlikely to be successful. The model identifies twelve critical testing processes.

CTP is primarily a content reference model.

The critical testing processes model is a context-sensitive approach that allows for tailoring the model including:

- Identification of specific challenges
- Recognition of attributes of good processes
- Selection of the order and importance of implementation of process improvements

The critical testing processes model is adaptable within the context of all software development lifecycle models.

Process improvements using CTP begin with an assessment of the existing test process. The assessment identifies which processes are strong and which are weak, and provides prioritized recommendations for improvement based on organizational needs. While the assessments vary depending on the specific context in which they are performed, the following quantitative metrics are commonly examined during a CTP assessment:

- Defect detection percentage
- Return on the testing investment
- Requirements coverage and risk coverage
- Test release overhead
- Defect report rejection rate

The following qualitative factors are commonly evaluated during a CTP assessment:

- Test team role and effectiveness
- Test plan utility

- Test team skills in testing, domain knowledge, and technology
- Defect report utility
- Test result report utility
- Change management utility and balance

Once an assessment has identified weak areas, plans for improvement are put into place. Generic improvement plans are provided by the model for each of the critical testing processes, but the assessment team is expect to tailor those heavily.

8.8 Improving the Test Process with STEP

STEP (Systematic Test and Evaluation Process), like CTP and unlike TMMi and TPI, does not require that improvements occur in a specific order.

Basic premises of the methodology include:

- A requirements-based testing strategy
- Testing starts at the beginning of the lifecycle
- Tests are used as requirements and usage models
- Testware design leads software design
- Defects are detected earlier or prevented altogether
- Defects are systematically analyzed
- Testers and developers work together

STEP is primarily a content reference model.

The STEP methodology is based upon the idea that testing is a lifecycle activity that begins during requirements formulation and continues until retirement of the system. The STEP methodology stresses “test then code" by using a requirements-based testing strategy to ensure that early creation of test cases validates the requirements specification prior to design and coding. The methodology identifies and focuses on improvement of three major phases of testing:

- Planning
- Acquisition
- Measurement

During a STEP assessment, quantitative metrics and qualitative interviews are used. Quantitative metrics include:

- Test status over time
- Test requirements or risk coverage
- Defect trends including detection, severity, and clustering
- Defect density
- Defect removal effectiveness
- Defect detection percentage
- Defect introduction, detection, and removal phases
- Cost of testing in terms of time, effort, and money

Quantitative factors include:

- Defined test process utilization
- Customer satisfaction

In some cases the STEP assessment model is blended with the TPI maturity model.

8.9 Capability Maturity Model Integration, CMMI

The CMMI can be implemented via two approaches or representations: the staged representation or the continuous one. In the staged representation there are five “levels of maturity”, each level building upon the process areas established in the previous levels. In the continuous representation the organization is allowed to concentrate its improvement efforts on its own primary areas of need without regard to predecessor levels.

The staged representation is primarily included in CMMI to ensure commonality with CMM, while the continuous representation is generally considered more flexible.

Within CMMI the process areas of Validation and Verification reference both static and dynamic testing test process.

9. Test Tools & Automation

Terms

Debugging tool, dynamic analysis tool, emulator, fault seeding tool, hyperlink test tools, keyword-driven testing, performance testing tool, simulator, static analyzer, test execution tool, test management tool, test oracle

9.1 Introduction

This section expands on the Foundation Level Syllabus by first covering a number of general concepts and then discussing some specific tools in further detail.

Even though some of the concepts covered may be more relevant to either test managers, test analysts or technical test analysts, a basic understanding of the concepts is required by all testing professionals. This basic understanding may then be expanded on where appropriate.

Tools can be grouped in a number of different ways, including a grouping according to their actual user, such as test managers, test analysts and technical test analysts. This particular grouping, which reflects the modules of this syllabus, is used for the remaining sections of this chapter. In general, the tools discussed in these sections are primarily relevant to a specific module, although certain tools (e.g., test management tools) may have wider relevance. Where this is the case, examples of the tool's application in a specific context will be given by the Training Provider.

9.2 Test Tool Concepts

Test tools can greatly improve the efficiency and accuracy of the test effort, but only if the proper tools are implemented in the proper way. Test tools have to be managed as another aspect of a well-run test organization. Test automation often is assumed to be synonym with test execution, but most manual labor has different forms of test automation, meaning that most areas within test could be automated to some degree if the right tools were present.

Any test tool version, test script or test session should be, as any test basis, under configuration management and bound to a particular software version where it has been used. Any test tool is an important part of the testware and should be managed accordingly, such as:

- Creating an architecture prior to the creation of the test tool
- Ensuring proper configuration management of scripts & tool versions, patches etc. including version information
- Creating and maintaining libraries (re-use of similar concepts within test cases), documenting the test tool implementation (e.g. process of how the tool is used and utilized in the organization)
- Planning for the future by structuring test cases for future development, e.g. making them expandable and maintainable

9.2.1 Cost benefits and Risks of Test Tools and Automation

A cost-benefit analysis should always be performed and show a significant positive return on investment. The main characteristics of a cost-benefit analysis should encompass the following cost items by comparing actual cost for both manual (non-tool usage) and tool-usage in terms of costs (hours translated into cost, direct costs, recurring and non-recurring costs):

- Initial costs

    - Knowledge acquisition (learning curve for tool)
    - Evaluation (tool comparisons) when appropriate
    - Integration with other tools
    - Consideration of initial costs for purchase, adaptation or development of tool
- Recurring costs
    - Cost of tool ownership (maintenance, licensing fees, support fees, sustaining knowledge levels)
    - Portability
    - Availability and dependencies (if lacking)
    - Continual cost evaluation
    - Quality improvement, to ensure optimal use of the selected tools

Business cases based only on pilot automation projects often miss important costs such as the cost of maintaining, updating and expanding the test scripts when the system changes. Durability of a test case is how long it will remain valid without rework. The time required to implement the first version of automated test scripts is often far beyond the time to run them manually, but will enable the possibility to create many more similar test scripts much faster and easily expand the number of good test cases over time. In addition, significant test coverage and test efficiency improvements will be seen on future uses of the automation after the implementation period. The business case for tool implementation must be based on the long term business case.

On a specific level each test case must be considered to see if it merits automation. Many automation projects are based on implementation of readily available manual test cases without reviewing the actual benefit of the automation of each particular case. It is likely that any given set of test cases (a suite) may contain manual, semi-automated and fully automated tests

In addition to the topics covered in the Foundation Level Syllabus, the following aspects should be considered:

Additional Benefits:

- Automated test execution time become more predictable
- Regression testing and defect validation are faster and safer late in the project when the test cases are automated
- Use of automated tools can enhance the status and technical growth of the tester or test team
- Automation can be used in parallel, iterative and incremental development to provide better regression testing for each build
- Coverage of certain test types which cannot be covered manually (e.g., performance and reliability)

Additional Risks:

- Incomplete or incorrect manual testing is automated as is
- The testware is difficult to maintain, requiring multiple changes when the software under test is changed
- Loss of direct tester involvement in the execution can reduce defect detection as only automated, scripted, tests are performed

9.2.2 Test Tool Strategies

Test tools are usually intended to be used for more than one particular project. Depending on the investment and length of a project, it might not give an adequate return on investment within the project, but would on subsequent versions of the software. For example, since the maintenance phase is often test intensive (for every correction a large regression suite must be executed), it can sometimes be cost-beneficial to automate a system that is in the maintenance phase if the system life span can make it economical. For another example it is easy for humans to make mistakes while doing manual testing (such as typing errors), thus the cost-benefit of automating input of data and comparison of output data to data from an oracle (e.g. test result comparison to expected result) in a test suite is beneficial.

For companies that use and are dependent on many test tools (for different phases and purposes) a long-term test tool strategy is advisable to aid in decisions of phase-in and out of different tool versions and tool support. For larger companies with a tool intensive domain, it can be advisable to provide general guidelines for tool acquisition, strategies, tool paradigms or script languages to use.

9.2.3 Integration & Information Interchange Between Tools

Usually there is more than one test tool being used within the test (and development) process. Let's take an example of a company using a static analysis tool, a test management and reporting tool, a configuration management tool, an incident management tool and a test execution tool all at the same time. It is important to consider if the tools can be integrated and exchange information with each other in a beneficial way. For example, it would be beneficial if all test execution status was reported directly to the test management system to provide immediate updates of progress, and direct traceability from requirements to a particular test case. It is more effort and more error-prone to store test scripts both in a test management database and in the configuration management system. If a tester wants to launch an incident report in the middle of test case execution, the defect tracking and test management systems have to be integrated. While static analysis tools may be separate from the other tools, it would be far more convenient if the tool could report incidents, warnings and feedback directly to the test management system.

Buying a suite of test tools from the same vendor does not automatically mean that the tools work together in this manner, but is a reasonable requirement. All these aspects should be evaluated from the cost of automating the information interchange compared to the risk of tampering with or losing information with sheer manual labor, assuming the organization has the time for the work this will entail.

New concepts like the integrated development environments like Eclipse aim to ease the integration and usage of different tools in the same environment by providing a common interface for development and test tools. A tool vendor can become Eclipse "compliant" by creating a plug-in to the Eclipse framework, making it have the same look and feel as any other tool. This creates a good advantage for the user. Note: this does not automatically mean that even if the user interface is similar, that the tools automatically provide integration and information interchange between themselves.

9.2.4 Automation Languages: Scripts, Script Language

Scripts and script languages are sometimes used to better implement and expand the test conditions and test cases. For example when testing a web application, a script might be used to bypass the user interface to more adequately test the API (application programming interface) itself. Another example would be the case where the testing of a user interface is automated to allow all possible combinations of inputs which would be infeasible with manual testing.

The capability of scripting languages varies widely. Note that scripting languages can range from normal programming languages, to very specific standard notations, e.g. signaling for protocols like TTCN-3.

9.2.5 The Concept of Test Oracles

Test oracles are generally used to determine expected results. As such, they perform the same function as the software under test and so are rarely available. They may be used, however, in situations where an old system is being replaced by a new system with the same functionality, and so the old system can be used as an oracle. Oracles may also be used where performance is an issue for the delivered system. A low performance oracle may be built or used to generate expected results for the functional testing of the high performance software to be delivered.

9.2.6 Test Tool Deployment

Every automated tool is software in its own right and may have hardware or software dependencies. A tool should be documented and tested itself regardless of whether it is purchased as-is, adapted or created in house. Some tools are more integrated with the environment, and other tools work better as stand-alone tools.

When the system under test runs on proprietary hardware, operating systems, embedded software or uses non-standard configurations, it may be necessary to create (develop) a tool or adapt a tool to fit the specific environment. It is always advisable to do a cost-benefit analysis that includes initial implementation as well as long-term maintenance.

During deployment of a test automation tool it is not always wise to automate manual test cases as is, but to redefine the test cases for better automation use. This includes formatting the test cases, considering re-use patterns, expanding input by using variables instead of using hard-coded values and utilizing the benefits of the test tool, which has abilities to traverse, repeat and change order with better analysis and reporting facilities. For many test automation tools, programming skills are necessary to create efficient and effective test programs (scripts) and test suites. It is common that large test suites are very difficult to update and manage if not designed with care. Appropriate training in test tools, programming and design techniques is valuable to make sure the full benefits of the tools are leveraged.

Even when manual test cases have been automated, it is important to periodically execute the test cases manually to retain the knowledge of how the test works and to verify correct operation.

When a tool gets into use and the number of test scripts grows, there may be a need to add on features that could be provided by other tools. This is not always possible since tools do not always have open interfaces and sometime use proprietary non-standard scripting languages. It is wise to use tools that have plug-ins to open frameworks or API (Application Programming Interface,. This will guarantee a better future-proofing of the test scripts as testware.

For each type of tool, regardless of the phase in which it is to be used, consider the characteristics listed below. These characteristics can be used both in tool evaluations and when building a tool. In each of these areas, a tool can be weak or strong. A list such as this is useful when comparing the capabilities of similar tools.

- Analysis (understanding of concept, input, information provided manually or automatically)
- Design (manual, automatically generated)
- Selection (manual, automatically selected according to a range of criteria, e.g. coverage)
- Execution (manual, automatic, driving, restarting etc.)
- Evaluation (e.g. test oracle) and presentation. These are often referred to as logging or reporting functions (manual, automatic e.g., comparative, against a form, standard, generated to a criteria)

9.2.7 Usage of Open Source Test Tools

Tools used to test safety critical systems must be certified to comply with the intended purpose against the corresponding standards. It is not recommended to use open-source tools in safety-critical systems, unless they have obtained the appropriate level of certification.

The quality of open-source software is dependent on the exposure, history and usage of software in question, and should not be assumed to be more (or less) accurate than any commercially available tool.

An assessment of the quality should always be conducted for any test tool to assess the accuracy of the tool. For some tool types it is more easy to confuse a positive evaluation result with a wrong tool execution (e.g. it skipped executions and did not report what was skipped). Careful consideration should be give to license fees. There may also be an expectation that code modifications made to enhance the tool will be shared.

9.2.8 Developing Your Own Test Tool

Many test tools are developed out of the need for an individual tester or designer to speed up their own work. Other reasons for self-developed test tools are the lack of suitable commercial tools or the use of proprietary hardware or test environment. These tools are often efficient to do the task they are supposed to do, but are often very dependent on the person creating the tool. These tools should be documented in a way that they can be maintained by others. It is also important to review the purpose, aim, benefits and possible downside before spreading it across an organization. Often these tools get new requirements and are expanded far beyond the initial use, which might not be beneficial.

9.2.9 Test Tool Classification

In addition to tools being divided into what activity they support (which is the concept used at the Foundation Level), there are other tool classification schemes, including:

- Tools grouped by what level of testing is performed (component, integration, system, acceptance)
- Tools grouped by what faults they process and support
- Tools based on test approach or test technique (see further discussion below)
- Tools for testing different purposes, e.g. measurement, drivers, logging, comparisons
- Tools specific to certain domains, e.g. traffic simulation & signaling, networks, protocol, transactions, TV-screens, expert systems
- Tools for supporting different areas within testing e.g. data input, environment, configuration or other conceptual areas
- Tools based on how the tool is applied: Off the shelf, frame-work (for adaptation), plug-in adaptation (i.e. Eclipse), standard or certification test suite, in house development of tool

And finally, tools can be grouped according to their actual user, such as test managers, test analysts and technical test analysts. This grouping, which reflects the modules of this syllabus, is used for the remaining sections of this chapter. The Foundation Level Syllabus includes a section regarding tools. The sections below are additional aspects of these tools.

9.3 Test Tools Categories

This section has the following objectives:

- providing additional information on tools categories already introduced in the ISTQB® Foundation Level Syllabus section 6, such as Test Management Tools, Test Execution Tools and Performance Testing Tools
- introducing new tools categories

Please refer to the ISTQB® Foundation Level Syllabus section 6 for general information concerning the other tools categories not included in this section

9.3.1 Test Management Tools

For general information concerning test management tools, please refer to the ISTQB® Foundation Level Syllabus section 6.1.2.

Test management tools should have the ability to track the following information:

- Traceability of test artifacts
- Capture of test environment data in complicated environments
- Data regarding the execution of concurrent test suites on different test environments in the same test session across multiple sites (test organizations)
- Metrics such as:
    - Test conditions
    - Test cases
    - Time to execute (e.g. a test case, a suite, a regression suite) and other important timing, including averages that can contribute to management decisions
    - Numbers of test cases, test artifacts and test environments
    - Pass/fail rates
    - Number of pending test cases (and reasons for their not being executed)
    - Trends
    - Requirements
    - Relationships and traceability between test artifacts
- Concepts supported by the test management tools, such as:
    - Organizer of test artifacts, repository and driver of test cases
    - Test conditions and test environments
    - Regression suites, test sessions
    - Logging and failure handling information
    - Environment restart (and re-initialization)
    - Test metrics about the tests artifacts (test documentation) to document test progress

Test Management tools are used by test managers, test analysts and technical test analysts.

9.3.2 Test Execution Tools

For general information concerning Test Management tools, please refer to the ISTQB® Foundation Level Syllabus section 6.1.5.

Test Execution tools are mostly used by Test analysts and Technical Test analysts at all levels of testing, to run tests and check the outcome of the tests. The objective of using a Test Execution tool is typically one or more of the following:

- to reduce costs (effort or time),
- to run more tests,
- to make tests more repeatable.

Test Execution tools are most often used to automate regression tests.

Test Execution tools work by executing a set of instructions written in a programming language, often called a scripting language. The instructions to the tool are at a very detailed level that specifies individual button presses, key hits and mouse movements. This makes the detailed scripts very susceptible to changes in the software under test (SUT), particularly changes to the graphical user interface (GUI).

The starting point for a script may be a recording (done with capture replay) or a constructed or edited script using existing scripts, templates or keywords. Scripting is a program, and is working exactly like any software. Capturing (or recording) can be used to record an audit trail for non-systematic testing. Most test execution tools include a comparator, the ability to compare an actual result to a stored expected result. The tendency in testing (as in programming) is moving from detailed low-level instruction to more “high-level” languages, here again libraries, macros and sub-programs are utilized. A series of instructions are labelled with one name – in testing called key-word driven or action-word driven. The main advantage is separating instructions from data. It is the same concept as utilizing templates when scripting to minimize user effort.

The main reason why some test tools fail is due to the low skills in programming and understanding that a test tool just solves part of the problems in automating test execution. It is important to note that any test execution automation takes management, effort, skills and attention, e.g. test architecture and configuration management. This also means that test scripts can have defects. The use of testware architecture might give independence from a particular tool vendor. When a tool is procured, there is a tendency to think that the tool's standards must be followed, for example for the structure and naming conventions of scripts. However, the automating of test set-up can form an interface between your own best way of organizing tests and where the tool needs to find them in order to run them.

9.3.3 Debugging & Troubleshooting Tools

Troubleshooting may employ tools to narrow down the area where a fault occurs. This might also be needed in a system where it is not evident what fault caused the exhibited failure. Troubleshooting tools include traces and simulated environments used to interact with the software or extract information from the system to narrow down the location of the fault.

Programmers reproduce faults and investigate the state of programs by using debugging and tracing tools. Debuggers and traces enable programmers to:

- Execute programs line by line
- Halt the program at any program statement
- Set and examine program variables.

It should be made clear that debugging (and debugging tools) are related to testing but are not testing (or testing tools). Debugging and tracing tools may be used for trouble-shooting purposes by testers to better locate a fault origin and help determine where to send a defect report. Debugging, tracing and troubleshooting tools are mainly used by Technical Test Analysts.

9.3.4 Fault Seeding & Fault Injection Tools

Fault seeding and fault injection are two different techniques that can be used in testing. Fault seeding will utilize a tool similar to a compiler to create single or limited types of code faults in a systematic way. These tools are also often used in conjunction with the mutation test technique and are sometimes called mutation test tools.

Fault injection is aimed at changing the actual interfaces to test components (when source code is not available), but could also be deliberately (re-)injecting a particular fault to check if 1) the software can cope with it (fault tolerance) or 2) to evaluate that a test in a test suite finds the deliberately inserted fault. Fault seeding and fault injection tools are mainly used at the code level by Technical Test Analysts, but it is also possible for a test analyst to manipulate data in a data base or inject faults into the data stream to test the system behavior.

9.3.5 Simulation & Emulation Tools

Simulators are used to support tests where code or other systems are unavailable, expensive or impracticable to use (e.g. testing software to cope with nuclear meltdowns). Some simulators and test harness tools can also support or mimic fault behavior, so error states or error scenarios can be checked. The main risk with using these tools is that resource-related errors like timing issues may not be found which are very important for some type of systems.

Emulators are a particular category of simulators, and consist of software written to mimic the hardware. The advantage of using emulators is that more elaborate testing may be possible. One particular advantage with emulators is that tracing, debugging and time-dependent causes can be re-created, which might be impossible in a real system. Emulators are costly to create, but the advantage of analysis where the system can be run in "slow-motion" are invaluable for some parallel, time-dependent and complex systems.

Test Analysts and Technical Test Analysts, depending on the type of emulation required, use these tools.

9.3.6 Static and Dynamic Analysis Tools

For general information concerning test static and dynamic analysis tools, please refer to the ISTQB® Foundation Level Syllabus section 6.1.6 "Tools for performance and monitoring".

9.3.6.1 Static analysis tools

Static Analysis tools can be used at any time in the software lifecycle and also at all levels/phases of the software development, depending on the measurements provided by the tool.

Static Analysis tools report their findings in terms of warnings. The warnings that are unsubstantiated are called false positives. True positives are real faults that could lead to failures during execution. It can be difficult and time-consuming to discern false from true positives since it requires proper trouble-shooting. More recent static analysis tools can use information in the dynamic binding during compilation, and are therefore more powerful in finding real faults with less false positives. Technical Test Analysts use these tools.

9.3.6.2 Dynamic analysis tools

Dynamic Analysis tools provide run-time information on the state of the executing software. These tools are most commonly used to identify unassigned pointers, check pointer arithmetic, monitor the allocation, use and de-allocation of memory to flag memory leaks and highlight other errors difficult to find 'statically'. Memory tools should be re-used at more levels than one in large complex systems, since memory problems are dynamically created. Note that different commercial test tools might be implemented differently, and thus target and report different types of memory or resource (stack, heap) problems. The conclusion is that two different memory tools could identify different problems. Memory tools are particularly useful for some programming languages (C, C++) where memory management is left to the programmer. Technical Test Analysts use these tools.

9.3.7 Keyword-Driven Test Automation

Keywords (sometimes referred to as "Action Words") are mostly (but not exclusively) used to represent high-level business interactions with a system (e.g. "cancel order"). Each keyword is typically used to represent a number of detailed interactions with the system under test. Sequences of keywords (including relevant test data) are used to specify test cases.[Buwalda01]

In test automation a keyword word is implemented as one or more executable test scripts. Tools read test cases written with keywords and call the appropriate test scripts which implement them. The scripts are implemented in a highly modular manner to enable easy mapping to specific keywords. Programming skills are needed to implement these modular scripts.

The primary advantages of keyword-driven test automation are:

- Keywords can be defined by domain experts which relate to a particular application or business domain. This can make the task of test case specification more efficient.
- A person with primarily domain expertise can benefit from automatic test case execution (once the keywords have been implemented as scripts).
- Test cases written using keywords are easier to maintain because they are less likely to need modification if details in the software under test change.

- Test case specifications are independent of their implementation. The keywords can be implemented using a variety of scripting languages and tools.

Keyword-based test automation is mostly used by domain experts and Test Analysts.

9.3.8 Performance Testing Tools

For general information concerning test performance tools, please refer to the ISTQB® Foundation Level Syllabus section 6.1.6 “Tools for performance and monitoring”

Performance test tools have two main facilities:

- Load generation
- Measurement and analysis of system response to a given load

Load generation is performed by implementing a pre-defined operational profile (see section 5.3.3) as a script. The script may initially be captured for a single user (possibly using a capture/replay tool) and then implemented for the specified operational profile using the performance test tool. This implementation must take into account the variation of data per transaction (or sets of transactions).

Performance tools generate a load by simulating large numbers of multiple users (“virtual” users) with specific volumes of input data. In comparison with capture/replay tools, many performance testing scripts reproduce user interaction with the system at the communications protocol level and not by simulating user interaction via a graphical user interface. A limited number of load generation tools can generate the load by driving the application using its user interface.

A wide range of measurements are taken by a performance test tool to enable analysis during or after execution of the test. Typical metrics taken and reports provided include:

- Numbers of simulated users
- Number and type of transactions generated by the simulated users
- Response times to particular transaction requests made by the users
- Reports based on test logs, and graphs of load against response times.

Significant factors to consider in the implementation of performance test tools include:

- The hardware and network-bandwidth required to generate the load
- The compatibility of the tool with the communications protocol used by the system under test
- The flexibility of the tool to allow different operational profiles to be easily implemented
- The monitoring, analysis and reporting facilities required

Performance test tools are typically acquired due to the effort required to develop them. It may, however, be appropriate to develop a specific performance tool if technical restrictions prevent a product being used, or if the load profile and facilities to be provided are relatively simple. Performance test tools are typically used by Technical Test Analysts.

Note: performance related defects often have deep ranging impact on the SUT. When performance requirements are imperative, it is often useful to performance test the critical components (via drivers and stubs) instead of waiting for system tests.

9.3.9 Web Tools

Hyperlink test tools are used to scan and check that no broken or missing hyperlinks are present on a web site. Some tools also provide additional information such as a graph of the architecture (arborescence of the site), the speed and size of download (per URL), hits and volumes. These tools may also be helpful for monitoring SLA (Service Level Agreements) compliance. Test Analysts and Technical Test Analysts use these tools.

10. People Skills – Team Composition

Terms

Independence of testing.

10.1 Introduction

All testing professionals should be aware of the individual skills required to perform their specific tasks well. This section focuses initially on those individual skills and then continues with a number of issues specific to Test Managers, such as team dynamics, organization, motivation and communication.

10.2 Individual Skills

An individual's capability to test software can be obtained through experience or training in different work areas. Each of the following can contribute to the tester's knowledge base:

- Use of software systems
- Knowledge of the domain or business
- Activities in various phases of the software development process activities including analysis, development and technical support
- Activities in software testing

The users of software systems know the user side of the system well and have a good knowledge of how the system is operated, where failures would have the greatest impact, and what should be the expected reaction of the system. Users with domain expertise know which areas are of most importance to the business and how those areas affect the ability of the business to meet its requirements. This knowledge can be used to help prioritize the testing activities, create realistic test data and test cases, and verify or supply use cases.

Knowledge of the software development process (requirements analysis, design and coding) gives insight into how errors can be introduced, where they can be detected and how to prevent their introduction. Experience in technical support provides knowledge of the user experience, expectations and usability requirements. Software development experience is important for the use of the high end test automation tools that require programming and design expertise.

Specific software testing skills include the ability to analyze a specification, participate in risk analysis, design test cases, and the diligence for running tests and recording the results.

Specifically for Test Managers having knowledge, skills and experience in project management is important since test management is like running a project, e.g. making a plan, tracking progress and reporting to stakeholders.

Interpersonal skills, such as giving and receiving criticism, influencing, and negotiating are all important in the role of testing. A technically competent tester is likely to fail in the role unless they possess and employ the necessary interpersonal skills. In addition to working effectively with others, the successful test professional must also be well-organized, attentive to detail and possess strong written and verbal communication skills.

10.3 Test Team Dynamics

Staff selection is one of the most important functions of a management role in the organization. There are many items to consider in addition to the specific individual skills required for the job. When selecting an individual to join the team, the dynamics of the team must be considered. Will this person complement the skills and personality types that already exist within the test team? It is important to consider the advantages of having a variety of personality types on the testing team as well as a mix of technical skills. A strong test team is able to deal with multiple projects of varying complexity while also successfully handling the interpersonal interactions with the other project team members.

New team members must be quickly assimilated into the team and provided with adequate supervision. Each person should be given a defined role on the team. This can be based on an individual assessment process. The goal is to make each individual successful as an individual while contributing to the overall success of the team. This is largely done by matching personality types to team roles and building on the individual's innate skills as well as increasing their skill portfolio.

An important point to remember is that the perfect individual will rarely be available, but a strong team can be built by balancing the strengths and weaknesses of the individuals. Cross-training within the team is required to maintain and build the team knowledge and increase flexibility.

10.4 Fitting Testing Within an Organization

Organizations vary widely in how testing fits into the organizational structure. While quality is everyone's responsibility throughout the software development lifecycle, an independent test team can contribute largely to a quality product. Independence of the testing function varies widely in practice, as seen from the following list, ordered from least to most independence:

- No independent testers
  - In this case there is no independence and the developer is testing his own code
  - The developer, if allowed time to do the testing, will determine that the code works as he intended, which may or may not match the actual requirements
  - The developer can fix any found defects quickly.
- Testing is done by a different developer than the one who wrote the code
  - There is little independence between the developer and the tester
  - A developer testing another developer's code may be reluctant to report defects
  - A developer mind set toward testing is usually focused on positive test cases
- Testing is done by a tester (or test team) being part of the development team
  - The tester (or test team) will report to project management
  - The tester mind set is focused more on verifying adherence to requirements
  - Because the tester is a member of the development team, he may have development responsibilities in addition to testing
- Testers from the business organization, user community, or other non-development technical organization
  - Independent reporting to the stakeholders
  - Quality is the primary focus of this team
  - Skills development and training are focused on testing
- External test specialists perform testing on specific test targets
  - Test targets could be usability, security or performance
  - Quality should be the focus of these individuals, but that may depend on reporting structure
- Testing is done by an organization external to the company
  - Maximum independence is achieved
  - Knowledge transfer may not be sufficient
  - Clear requirements and a well defined communication structure will be needed
  - Quality of the external organization must be audited regularly

There are varying degrees of independence between the development and test organizations. It is important to understand that there may be a tradeoff where more independence results in more isolation and less knowledge transfer. A lower level of independence may increase knowledge, but can also introduce conflicting goals. The level of independence will also be determined by the software development model being used, e.g. within agile development the testers are most often part of the development team.

Any of the above options can be mixed in an organization. There may be testing done within the development organization as well as by an independent testing organization and there may be final certification by an external organization. It is important to understand the responsibilities and expectations for each phase of testing and to set those requirements to maximize the quality of the finished product while staying within schedule and budget constraints.

Outsourcing is one of the forms of using an external organization. The outsource group can be another company that provides testing services which may reside at your location, external to your location but within your country or in another country (sometimes referred to as off-shore). Outsourcing brings challenges, particularly when external to your country. Some of the items to be considered include the following:

- Cultural differences
- Supervision of outsource resources
- Transfer of information, communication
- Protection of intellectual property
- Skills sets, skill development and training
- Employee turnover
- Accurate cost estimation
- Quality

10.5 Motivation

There are many ways to motivate an individual in a testing position. These include:

- Recognition for the job accomplished
- Approval by management
- Respect within the project team and among peers
- Adequate rewards for the work done (including salary, merit increases and bonuses)

There are project influences that can make these motivational tools difficult to apply. For example, a tester can work very hard on a project that has an impossible deadline. The tester can do everything in his/her power to drive the quality focus of the team, put in extra hours and effort, and yet the product may ship before it should, due to external influences. The result may be a poor quality product despite the best efforts of the tester. This can easily be a demotivator if the tester's contribution is not understood and measured, regardless of whether the end product is successful.

The test team must ensure that it is tracking the appropriate metrics to prove that a good job was done to accomplish the testing, mitigate risks and accurately record the results. Unless this data is gathered and published, it is easy for a team to become demotivated when they don't receive the recognition they feel is due for a job well-done.

Recognition is not just determined in the intangibles of respect and approval, it is also apparent in promotional opportunities, salary scale and career paths. If the test group is not respected, these opportunities may not be available.

Recognition and respect are acquired when it is clear that the tester contributes to the incremental value of the project. In an individual project this is most rapidly achieved by involving the tester at the conception of the project and keeping that involvement throughout the lifecycle. In the course of time the testers will win recognition and respect by their contribution to the positive development of the project, but this contribution should also be quantified in terms of cost of quality reductions and risk mitigation.

10.6 Communication

Test team communication primarily occurs on three levels:

- Documentation of test products: test strategy, test plan, test cases, test summary reports, defect reports, etc.
- Feedback on reviewed documents: requirements, functional specifications, use cases, component test documentation, etc.
- Information gathering and dissemination: interaction with developers, other test team members, management, etc.

All communication must be professional, objective and effective in order to build and maintain respect for the test team. Diplomacy and objectivity are required when providing feedback, particularly constructive feedback, on the work products of others.

All communication should be focused on achieving test objectives and on improving quality both in products and the processes used to produce the software systems. Testers communicate with a wide audience, including users, project team members, management, external test groups and customers. Communication must be effective for the target audience. For example a defect trending report designed for the development team might be too detailed to be appropriate for an executive management briefing.

11. References

11.1 Standards

This section lists the standards mentioned in this syllabus.

11.1.1 Per chapter

The following chapters refer to these standards

- Chapter 2
  BS-7925-2, IEEE 829, DO-178B/ED-12B.
- Chapter 3
  IEEE829 D0-178B/ED-12B.
- Chapter 4
  BS 7925-2.
- Chapter 5
  ISO 9126.
- Chapter 06
  IEEE 1028.
- Chapter 7
  IEEE 829, IEEE 1044, IEEE 1044.1.

11.1.2 Alphabetical

The following standards are mentioned in these respective chapters

- [BS-7925-2] BS 7925-2 (1998) Software Component Testing
  Chapter 02 and 04
- [IEEE 829] IEEE Std 829™ (1998/2005) IEEE Standard for Software Test Documentation (currently under revision)
  Chapter 02 and 03
- [IEEE 1028] IEEE Std 1028™ (1997) IEEE Standard for Software Reviews
  Chapter 06
- [IEEE 1044] IEEE Std 1044™ (1993) IEEE Standard Classification for Software Anomalies
  Chapter 07
- [ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality
  Chapter 05
- [ISTQB] ISTQB Glossary of terms used in Software Testing, Version 2.0, 2007
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne systems and Equipment certification, RTCA/EUROCAE ED12B.1992.
  Chapter 02 and 03

11.2 Books

[Beizer95] Beizer Boris, “Black-box testing”, John Wiley & Sons, 1995, ISBN 0-471-12094-4

[Black02]: Rex Black, “Managing the Testing Process (2nd edition)”, John Wiley & Sons: New York, 2002, ISBN 0-471-22398-0

[Black03]: Rex Black, “Critical Testing Processes”, Addison-Wesley, 2003, ISBN 0-201-74868-1

[Black07]: Rex Black, “Pragmatic Software Testing”, John Wiley and Sons, 2007, ISBN 978-0-470-12790-2

[Burnstein03]: Ilene Burnstein, “Practical Software Testing”, Springer, 2003, ISBN 0-387-95131-8

[Buwalda01]: Hans Buwalda, “Integrated Test Design and Automation” Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Test Design”, Artech House, 2003, ISBN 1-58053-791-X

[Craig02]: Craig, Rick David; Jaskiel, Stefan P., “Systematic Software Testing”, Artech House, 2002, ISBN 1-580-53508-9

[Gerrard02]: Paul Gerrard, Neil Thompson, “Risk-based e-business testing”, Artech House, 2002, ISBN 1-580-53314-0

[Gilb93]: Gilb Tom, Graham Dorothy, “Software inspection”, Addison-Wesley, 1993, ISBN 0-201-63181-4

[Graham07]: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black “Foundations of Software Testing”, Thomson Learning, 2007, ISBN 978-1-84480-355-2

[Grochmann94]: M. Grochmann (1994), Test case design using Classification Trees, in: conference proceeedings STAR 1994

[Jorgensen02]: Paul C.Jorgensen, “Software Testing, a Craftsman’s Approach second edition”, CRC press, 2002, ISBN 0-8493-0809-7

[Kaner02]: Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4

[Koomen99]: Tim Koomen, Martin Pol, “Test Process Improvement”, Addison-Wesley, 1999, ISBN 0-201-59624-5.

[Myers79]: Glenford J.Myers, “The Art of Software Testing”, John Wiley & Sons, 1979, ISBN 0-471-46912-2

[Pol02]: Martin Pol, Ruud Teunissen, Erik van Veenendaal, “Software Testing: A Guide to the Tmap Approach”, Addison-Wesley, 2002, ISBN 0-201-74571-2

[Splaine01]: Steven Splaine, Stefan P.,Jaskiel, “The Web-Testing Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0

[Stamatis95]: D.H. Stamatis, “Failure Mode and Effect Analysis”, ASQC Quality Press, 1995, ISBN 0-873-89300

[vanVeenendaal02]: van Veenendaal Erik, “The Testing Practitioner”, UTN Publsihing, 2002, ISBN 90-72194-65-9

[Whittaker03]: James Whittaker, “How to Break Software”, Addison-Wesley, 2003, ISBN 0-201-79619-8

[Whittaker04]: James Whittaker and Herbert Thompson, “How to Break Software Security”, Peason / Addison-Wesley, 2004, ISBN 0-321-19433-0

11.3 Other references

The following references point to information available on the Internet.

Even though these references were checked at the time of publication of this Advanced Level Syllabus, the ISTQB can not be held responsible if the references is not available anymore.

- Chapter 05
  - www.testingstandards.co.uk
- Chapter 06
  - Bug Taxonomy: www.testingeducation.org/a/bsct2.pdf
  - Sample Bug Taxonomy based on Boris Beizer's work: inet.uni2.dk/~vinter/bugtaxst.doc
  - Good overview of various taxonomies: testingeducation.org/a/bugtax.pdf
  - Heuristic Risk-Based Testing By James BachJames Bach, Interview on What IsTesting.com. www.whatistesting.com/interviews/jbach.htm
  - www.satisfice.com/articles/et-article.pdf
  - From "Exploratory & Risk-Based Testing (2004) www.testingeducation.org"
  - Exploring Exploratory Testing , Cem Kaner and Andy Tikam , 2003
  - Pettichord, Bret, "An Exploratory Testing Workshop Report", www.testingcraft.com/exploratorypettichord
- Chapter 09
  - www.sei.cmu.edu/cmmi/adoption/pdf/cmmi-overview06.pdf
  - TMMi www.tmmifoundation.org/

12. Appendix A – Syllabus background

Objectives of the Advanced Certificate qualification

- To gain recognition for testing as an essential and professional software engineering specialization.
- To provide a standard framework for the development of testers' careers.
- To enable professionally qualified testers to be recognized by employers, customers and peers, and to raise the profile of testers.
- To promote consistent and good testing practices within all software engineering disciplines.
- To identify testing topics that are relevant and of value to industry.
- To enable software suppliers to hire certified testers and thereby gain commercial advantage over their competitors by advertising their tester recruitment policy.
- To provide an opportunity for testers and those with an interest in testing to acquire an internationally recognized qualification in the subject.

Entry requirements for this qualification

The entry criteria for taking the ISTQB Advanced Certificate in Software Testing examination are:

- Holding a Foundation Level certificate, issued by an ISTQB-recognized Exam Board or Member Board.
- Have an appropriate number of years' experience in software testing or development, as determined by the Exam Board or Member Board granting the Advanced certification
- Subscribing to the Code of Ethics in this Syllabus.

It is also recommended that candidates take a course that has been accredited by an ISTQB Member Board. However, training is not required to take any ISTQB examination.

An existing Practitioner or Advanced Certificate in Software Testing (from an ISTQB-recognized Member Board or Exam Board) awarded before this International Certificate was released will be deemed to be equivalent to the International Certificate. The Advanced Certificate does not expire and does not need to be renewed. The date it was awarded is shown on the Certificate.

Within each participating country, local aspects are controlled by an ISTQB-recognized Member Board. Duties of the Member Boards are specified by the ISTQB, but are implemented within each country. The duties of the Member Boards include accreditation of training providers and arranging for exams, directly or indirectly through one or more contracted Exam Board.

13. Appendix B – Notice to the Readers

13.1 Examination Boards

This Advanced Level Syllabus requires knowledge of the content of the “Certified Tester Foundation Level Syllabus of the ISTQB”, version 2005, with the knowledge level specified in the Foundation Level Syllabus.

ISTQB-recognized examination bodies can generate questions based on any topic mentioned in the syllabus.

It is recommended that the questions generated be assigned different values according to the learning objectives of their respective topics. As an example: a question pertaining to a K1 knowledge level may be awarded fewer points than one pertaining to a K3 knowledge level, while a question on a K4 knowledge level would be awarded even more points.

13.2 Candidates & Training Providers

To receive Advanced Level certification, candidates must hold the Foundation Certificate and satisfy the Exam Board which examines them that they have sufficient practical experience to be considered Advanced Level qualified. Refer to the relevant Exam Board to understand their specific practical experience criteria. The ISTQB suggests a minimum of 5 years of practical experience in software engineering as a pre-requisite for holding the Advanced level, or 3 years of practical experience if the candidate holds a baccalaureate or equivalent degree in science or engineering.

Attaining the adequate level of proficiency to reach Advanced Level status in the Software Testing profession requires more than just knowledge of the content of this syllabus. Candidates and training providers are encouraged to spend, in reading and research, more time than indicated in this syllabus.

This syllabus provides a list of references, books and standards that candidates and training providers may wish to read in order to understand the specified topics in more detail.

14. Appendix C – Notice to Training Providers

14.1 Modularity

This syllabus provides content for three modules, called respectively:

- Advanced Level Test Manager
- Advanced Level Test Analyst
- Advanced Level Technical Test Analyst

Passing all modules allows the candidate to obtain the "Full Advanced Level Testing Professional" certification.

14.2 Training Times

14.2.1 Training per module

The recommended time necessary to teach the 3 different roles are for each role as follow:

• Advanced Level Test Manager	5 days
• Advanced Level Test Analyst	5 days
• Advanced Level Technical Test Analyst	5 days

This duration is based on the number of chapters per module and the specific learning objectives for each chapter. Specific minimum duration is listed for each chapter, for each role.

Training providers may spend more time than is indicated and candidates may spend more time again in reading and research. A course curriculum does not have to follow the same order as the syllabus.

The courses do not have to be contiguous days. Training providers may elect to organize their course differently, such as 3+2 days for Test Management, or 2 common days followed by 3 days each for Test Analysts and Technical Test Analysts.

14.2.2 Commonality

Training providers may elect to teach common topics only once, thereby reducing the overall time and avoiding repetitions. Training providers are also reminded that sometimes the same topic should be seen from different angles depending on the module.

14.2.3 Sources

The syllabus contains references to established standards which must be used in the preparation of training material. Each standard used must be the version quoted in the current version of this syllabus. Other publications, templates or standards not referenced in this syllabus may also be used and referenced, but will not be examined.

14.3 Practical Exercises

Practical work (short exercises) should be included for all aspects where candidates are expected to apply their knowledge (Learning Objective of K3 or higher). The lectures and exercises should be based on the Learning Objectives and the description of the topics in the content of the syllabus.

15. Appendix D – Recommendations

As these recommendations apply to various chapters of this syllabus, they have been compiled and merged in one appendix. The items listed below are examinable.

This list provides a number of helpful recommendations to meet testing challenges and builds on the list of seven basic testing principles introduced at the Foundation Level. The list provided in this appendix is not intended to be complete, but instead provides a sample of “lessons learned” which should be considered. Training providers will choose the points from this list which are most relevant to the module being taught

15.1 Recommendations for Industrialization

When implementing tests in an industrialized fashion, one is faced with a number of challenges. Advanced testers should be able to put the different recommendations described in this syllabus in focus within the context of their organization, teams, tasks and software components. The following list provides some areas that have proven to **negatively** impact performance of test efforts. Please note that the list is not intended to be complete.

- Generate test plans biased toward functional testing
  Defects are not limited to functional aspects, or with only a single user. Interaction of multiple users may have impact on the software under test.
- Not enough configuration testing
  If multiple types of processors, operating systems, virtual machines, browsers, and various peripherals can be combined into many possible configurations, limiting testing to just a few of these configurations may leave a large number of potential defects undiscovered.
- Putting stress and load testing off to the last minute
  The results of stress and load testing may require major changes in the software (up to and including the basic architecture). Since this may require considerable resources to implement, this may be highly negative for the project if these tests are conducted just before the planned introduction into production.
- Not testing the documentation
  Users are provided with the software and with documentation. If the documentation does not fit the software, the user will not be able to utilize the full potential of the software, or may even discard the software entirely.
- Not testing installation procedures
  Installation procedures, as well as backup and restore procedures, are done a very limited number of times. These procedures are, however, more critical than the software; if the software can not be installed, it will not be used at all.
- Insisting on completing one testing task fully before moving on to the next
  Even though some software development lifecycle models suggest the sequential execution of tasks, in practice many tasks often need to be performed (at least partly) concurrently.
- Failing to correctly identify risky areas
  Some areas may be identified as risky and may as a result be tested more thoroughly. However, the areas left with minimal or no tests may subsequently turn out to be of higher risk than originally estimated.
- Being too specific about test inputs and procedures
  By not giving testers enough scope for their own initiative in terms of defining test inputs and procedures, the tester may not be encouraged to examine areas that may look promising (in terms of possible hidden defects)
- Not noticing and exploring “irrelevant” oddities
  Observations or results that may seem irrelevant are often indicators for defects that (like icebergs) are lurking beneath the surface

- Checking that the product does what it's supposed to do, but not that it doesn't do what it isn't supposed to do
  By limiting oneself only to what the product is supposed to do, it is possible to miss aspects of the software which it is not supposed to do (additional, undesired functions for example)
- Test suites that are understandable only by their owners
  Testers may move to other areas of responsibility. Other testers will then need to read and understand previously specified tests. Failing to provide readable and understandable test specifications may have a negative impact because the test objectives may not be understood or the test may be removed altogether.
- Testing only through the user-visible interface
  The interfaces to the software are not limited to the user-interface. Inter-process communications, batch execution and other interrupts also interact with the software, and can generate defects.
- Poor bug reporting and configuration management
  Incident reporting and management, as well as configuration management are extremely important to ensure the success of the overall project (which includes development as well as testing). A successful tester may be considered one who gets defects fixed rather than one who finds many defects but fails to report them well enough to be corrected.
- Adding only regression tests
  Evolution of test suites over time is not limited to checking that no regression defects occur. The code will evolve over time and the additional tests will need to be implemented to cover these new functionalities, as well as to check for regressions in other areas of the software.
- Failing to take notes for the next testing effort
  The testing tasks do not end when the software is provided to the user or distributed to the market. A new version or release of the software will most likely be produced, so knowledge should be stored and transferred to the testers responsible for the next testing effort.
- Attempting to automate all tests
  Automation may seem like a good idea, but automation is a development project on its own. Not all tests should be automated: some tests are faster to do manually than to automate.
- Expecting to rerun all manual tests
  When rerunning manual tests, it is often unrealistic to expect that all tests will be rerun. Testers' attention span will waver, and testers will tend to focus on particular areas of the software, either consciously or not.
- Using GUI automation tools to reduce test creation cost
  A GUI capture/replay tool is an important initial investment and should only be used to support a defined strategy, with all the associated costs understood and evaluated.
- Expecting regression tests to find a high proportion of new bugs
  Regression tests generally do not find a large proportion of defects, mostly because they are tests which have already been run (e.g., for a previous version of same software), and defects should have been detected in those previous runs. This does not mean that regression tests should be eliminated altogether, only that the efficiency (capacity to detect new defects) of regression tests is lower than other tests.
- Embracing code coverage with the devotion that only simple numbers can inspire
  Code coverage and metrics may seem very interesting from a management point of view, based on the numbers and graphs provided, but numbers can not reflect the efficiency or pertinence of a test. Example: 100% is a nice target for code coverage, but is it a realistic one, is it the adequate one (i.e. is it instruction, condition, or MCDC coverage)?
- Removing tests from a regression test suite just because they don't add coverage
  In a regression test suite, some tests may be (should be) removed, and some others added. The reason for removal should not be based only on whether the test adds coverage, as pertinence of a test case (the type of defect it checks for) may have no impact on coverage. Example: coverage of code is not the only type of coverage; tests may have been created for other reasons (such as specific values or sequence of events) than simple coverage.

- Using coverage as a performance goal for testers
  Coverage is a measure of completeness, not of performance or efficiency of personnel. Other metrics may be defined to evaluate tester efficiency in the context of their organization and project. The use of such metrics must be very carefully thought through to avoid undesired effects ("dysfunctions").
- Abandoning coverage entirely
  Different types of coverage are available (e.g code statements, conditions, modules, functions etc), and the workload to obtain the adequate metrics may be significant. This is however not a reason to abandon coverage metrics altogether, since these may be very useful to the testing task.

Many of these points are dealt with within the context of specific sections of this syllabus.

When introducing testing measures and practices into your organization, it may be useful to consider not just the list of points above, but to also consider additional sources of information such as:

- The ISTQB syllabi (Foundation and Advanced)
- Books included in the reference list of this syllabus
- Reference models such as those covered in section 8.3.

16. Index