ISO/IEC JTC 1/SC 7
Software and systems engineering
Secretariat: SCC (Canada)

Document type: Text for CD ballot or comment

Title: ISO 29119-1 CD-3

Status:

Date of document: 2012-02-06

Source: WG26

Expected action: VOTE

Action due date: 2012-05-07

Email of secretary: witold.suryn@etsmtl.ca

Committee URL: http://isotc.iso.org/livelink/livelink/open/jtc1sc7

ISO/IEC TC JTC1/SC SC7 N 1

Date: 2012-02-02

ISO/IEC CD-3 29119-1

ISO/IEC TC JTC1/SC SC7/WG 26

Secretariat: ANSI

Systems and Software Engineering — Software Testing — Part 1: Concepts and Definitions

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International Standard
Document subtype:
Document stage: (20) Preparatory
Document language: E

C:\Users\St\Desktop\29119\ISO-IEC 29119-1 CD-3 WG26 (04_02_12).docx STD Version 2.1

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29119-1 was prepared by Technical Committee ISO/IEC/TC JTC1, *Information Technology*, Subcommittee SC SC7, *Software and Systems Engineering*.

ISO/IEC 29119 consists of the following parts, under the general title *Systems and Software Engineering — Software Testing*:

— Part 1: Concepts and Definitions

— Part 2: Test Process

— Part 3: Test Documentation

— Part 4: Test Techniques

Introduction

The purpose of ISO/IEC 29119 Software Testing is to define an internationally-agreed standard for software testing that can be used by any organization when performing any form of software testing.

This part, ISO/IEC 29119-1 Concepts and Definitions, facilitates the use of the other parts of the standard by introducing the vocabulary on which the standard is built and providing examples of its application in software testing practice. Part 1 is informative providing definitions, context and guidance for the other parts.

The document includes a description of the concepts of software testing and ways to apply the software testing process defined in this standard. Initially, general software testing concepts are discussed. The role of software testing in an organizational and project context is described. Software testing in a generic software life cycle is explained, introducing the way software test processes and sub-processes may be established for specific test items or with specific test objectives. It describes how software testing fits into different life cycle models. Annex A provides a collection of metrics that can be used to monitor and control testing. Annex B contains a set of examples showing how to apply the standard in particular situations. Note that uppercase is used for actual documents (e.g. Project Test Plan) whereas lowercase is used for parts of documents (e.g. project test strategy).

The test process model that this part of the standard is based on is defined in detail in ISO/IEC 29119-2 Test Process. ISO/IEC 29119-2 covers the software testing processes at the organizational level, test management level and for dynamic test levels. Since testing is a primary approach to risk treatment in software development, this standard defines a risk-based approach to testing. Risk-based testing is a best-practice approach to strategizing and managing testing as it allows testing to be prioritized and focused.

Templates and examples of test documentation that are produced during the testing process are defined in ISO/IEC 29119-3 Test Documentation. Software testing techniques that can be used during testing are defined in ISO/IEC 29119-4 Test Techniques.

Together, this four part standard aims to provide stakeholders with the ability to manage and perform software testing in any organization.

Systems and Software Engineering — Software Testing —Part 1: Concepts and Definitions

1 Scope

This standard covers definitions and concepts in systems and software testing. It provides definitions of testing terms and discussion of concepts key to the understanding of the ISO/IEC 29119 standard.

2 Conformance

ISO/IEC 29119 contains three principal areas where conformance may be claimed:

— test process;

— test documentation;

— test techniques.

Conformance is addressed in ISO/IEC 29119-2, ISO/IEC 29119-3 and ISO/IEC 29119-4.

3 Normative References

This standard does not require the use of any normative references. Standards useful for the implementation and interpretation of this standard are listed in the bibliography (Annex E).

4 Terms and Definitions

For the purposes of this document, the terms and definitions given in ISO/IEC/IEEE 24765 *Systems and software engineering — Vocabulary* and the following apply.

NOTE Use of the terminology in this standard is for ease of reference and is not mandatory for conformance with the standard. The following terms and definitions are provided to assist with the understanding and readability of parts 1, 2, 3 and 4 of this standard. Only terms critical to the understanding of this standard are included, this clause is not intended to provide a complete list of testing terms. The Systems and software engineering Vocabulary ISO/IEC/IEEE 24765 should be referenced for terms not defined in this clause. This source is available at the following web site: http://www.computer.org/sevocab.

4.1
abstract test case
a test case expressed as a set of logical conditions for inputs and expected results, rather than as concrete (implementation level) values. See also high-level test case

NOTE Actual input values and values of expected results are defined in concrete test cases.

4.2
actual result
test case expressed as a set of logical conditions for inputs and expected results, rather than as concrete (implementation level) values.

4.3
actual result
set of behaviours or conditions of a test item, or set of conditions of associated data or the test environment, observed as a result of test execution

4.4
base choice testing
combinatorial test design technique in which the most common (or default) value for the parameter in a P-V pair is used as the basis for deriving test cases

4.5
black-box testing
see specification-based testing

4.6
boundary value analysis
test design technique in which test cases are designed based on the boundary values of equivalence partitions

4.7
boundary value coverage
percentage of the set of boundary values identified within a test item, or at its interfaces, that are covered by a test set

4.8
boundary value testing
see boundary value analysis

4.9
branch testing
test design technique in which test cases are designed to force the execution of branch instructions in an executable test item

4.10
branch condition testing
form of condition testing where the test coverage items are the Boolean operand values within conditions

4.11
branch condition combination testing
form of condition testing where the test coverage items are the unique combinations of Boolean operand values within decisions

4.12
bug
(1) see defect
(2) see fault [BS 7925]

4.13
cause-effect graphing
test design technique which uses a model of the logical relationships between causes (e.g. inputs) and effects (e.g. outputs) for the test item, where each cause and each effect is expressed as a Boolean

4.14
classification tree method
test design technique which uses a model of the inputs to the test item in the form of a classification tree, that partitions the inputs (e.g. parameters) into equivalence classes

4.15
combinatorial test techniques
a class of test design techniques in which test cases are designed to exercise combinations of test conditions consisting of P-V pairs of the test item. See also base-choice testing, each-choice testing, pairwise testing,

4.16
completion criteria
see exit criteria

4.17
concrete test case
instantiation of an abstract test case by specification of actual (implementation-level) input values and expected results

NOTE A given abstract test case might be instantiated by numerous different concrete test cases.

4.18
condition testing
class of a test design techniques that use a model of the source code of the test item which identifies decisions (i.e. Boolean operand values). There are a number of forms of condition testing, all based on exercising these conditions, such as branch condition testing, branch condition combination testing, and modified condition decision coverage testing

4.19
conversion testing
test type which evaluates the transfer of data from an existing test item to a new (replacement) test item

NOTE For example, this could include assessing the ability to transfer data from one database to another.

4.20
data flow testing
class of test design techniques in which test cases are designed to evaluate the flow of information through a test item in terms of the definitions and uses of data

NOTE "Data" may include discrete variables, or data structures such as arrays, records, or files.

4.21
decision
decisions correspond to statements in the source code with two (or more) possible outcomes that determine which sub-paths will be taken by the control flow

NOTE Typical decisions are simple selections (e.g. if-then-else), to decide when to exit loops (e.g. while-loop), and in case (switch) statements (e.g. case-1-2-3-..-N). See also decision testing.

4.22
decision coverage
proportion of a given set of decision outcomes, that are covered by a test set

4.23
decision table testing
test design technique in which test cases are derived from the rules of a decision table

4.24
decision testing
test design technique in which test cases are designed to execute decision outcomes within the source code

4.25
defect
flaw in a work product that can cause it to fail to perform its required function

NOTE Defects may be found during, but not limited to, reviewing, testing, analysis, compilation, or use of software products or documentation.

NOTE Product failures might occur in respect of any of the quality characteristics of the work product, such as the usability of a document or of an interface.

4.26
dynamic testing
testing that requires the execution of program code

4.27
each choice testing
combinatorial test technique in which test cases are designed to execute each value of each P-V pair

4.28
equivalence class
see equivalence partition

4.29
equivalence class coverage
see equivalence partition coverage

4.30
equivalence partition
partition of the inputs (e.g. parameters) or outputs of the test item, such that all the values in the partition can reasonably be expected to be treated similarly (i.e. they may be considered "equivalent") by the test item

4.31
equivalence partition coverage
proportion of identified equivalence partitions of a test item that are covered by a test set

NOTE In many cases, the identification of equivalence partitions is subjective (especially in the sub-partitioning of "invalid" partitions), so a definitive count of the number of equivalence partitions in a test item may be impossible.

4.32
equivalence partitioning
test design technique in which test cases are designed to exercise equivalence partitions by using one or more representative members of each partition.

4.33
error guessing
test design technique in which test cases are derived on the basis of knowledge of past failures, or general knowledge of failure modes

NOTE The relevant knowledge may be gained from personal experience, or might be encapsulated in, for example, a defects database or a "bug taxonomy."

4.34
expected result
behaviour predicted by the specification, or another source, of the test item under specified conditions

4.35
experience-based testing
testing in which knowledge of the test item is augmented by personal or collective experience software or technologies

4.36
exploratory testing
approach to testing where the tester designs tests based on their knowledge and exploration of the test item and the results of previous tests

4.37
feature set
set of features of some item or work product

NOTE This may be the set of all features for the item (its full feature set) or a subset or for a specific purpose (the functional feature set, etc.).

4.38
high level test case
see abstract test case

4.39
incident report
documentation of the occurrence, nature, and status of an incident

NOTE Informally, incident reports may also be known as anomaly reports, bug reports, defect reports, error reports, issues, problem reports, , or trouble reports, amongst other terms.

4.40
multiple condition coverage
percentage of combinations of all single condition outcomes within one statement that have been exercised by a test procedure

4.41
multiple condition testing
test design technique in which test cases are designed to execute combinations of condition outcomes within a decision

4.42
negative testing
approach to testing aimed at showing that a test item does not behave as expected. Negative testing is related to the tester's attitude rather than a specific or test design technique, e.g. testing with invalid input values or exceptions

4.43
Organizational Test Policy
a short executive-level document that describes the purpose, goals, and overall scope of the testing within an organization; the Organizational Test Policy expresses why testing is performed

4.44
organizational test specification
document that provides information about testing for an organization, i.e. information that is not project-dependent. The most common examples of organizational test specifications are Organizational Test Policy and Organizational Test Strategy

4.45
Organizational Test Strategy
The Organizational Test Strategy expresses the generic requirements for the testing to be performed on all the projects run within the organization. It is aligned with the Organizational Test Policy providing detail on how the testing is to be performed.

4.46
orthogonal array
matrix with specific properties that can be used to identify a pair-wise set of test cases in combinatorial testing

4.47
P-V pair
combination of a test-item parameter with a value assigned to that parameter, used as a test condition and coverage item in combinatorial test techniques

4.48
pair-wise testing
form of combinatorial testing in which the test coverage items are unique pairs of P-V pairs, where each P-V pair is for a different test item parameter

4.49
partition testing
a class of test design techniques that divide (partition) the test item's data space into partitions, such that all data values within a partition are expected to exercise the same test coverage item (e.g. an equivalence partition, boundary values of an equivalence partition, a decision outcome, a state transition)

NOTE Most test design techniques fall into the category of partition testing techniques, because most test conditions offer a vast range of potential implementation-level values.

4.50
pass/fail criteria
decision rules used to determine whether a test item has passed or failed a test [IEEE 829]

4.51
path coverage
proportion of the set of all entry-exit paths through a test item that are covered by a test set

NOTE Path coverage measurements should specify whether they include infeasible paths.

4.52
random testing
test design technique in which test inputs are random or pseudo-random selections from the test item's expected input domain

NOTE True random inputs are appropriate for some purposes, but pseudo-random inputs may provide repeatability if seed values are stored

4.53
regression testing
testing with test cases that returned a pass result when last executed, following modifications to a test item or to its operational environment, to identify whether regressions occur

4.54
resumption criteria
the criteria which must be satisfied in order to restart testing that has been formally suspended. See also suspension criteria

4.55
retesting
testing that re-runs test cases that failed previously in order to verify the success of corrective actions

NOTE Retesting may also be combined with regression testing.

4.56
risk-based testing
testing in which the management, selection, prioritisation, and use of testing activities and resources is consciously based on corresponding types and levels of analyzed risk

4.57
scripted testing
testing performed based on a documented test script

NOTE This term normally applies to manually executed testing, rather than the execution of an automated script.

4.58
software test environment
facilities, hardware, software, firmware, procedures, and documentation needed to perform qualification or other testing of software [ISO/IEC 24765, Systems and Software Engineering Vocabulary]

4.59
software testing
see testing

4.60
specification-based testing
testing in which the principal test basis is a specification, rather than its implementation in source code or executable software; see black box testing

4.61
state transition testing
test design technique in which test cases are designed to execute specific aspects of a state model corresponding to the test item

4.62
statement coverage
the percentage of the set of all executable statements of a program that are covered by a test set

4.63
statement testing
test design technique in which test cases are constructed to force execution of individual statements in a program

4.64
static testing
testing in which a test item is examined against a set of quality or other criteria without being executed, e.g. reviews or static analysis

4.65
structural testing
see structure-based testing

4.66
structure-based testing
testing where the test inputs (of the test cases) are derived from an examination of the structure of the test item

NOTE Structure-based testing is not restricted to use at component level and can be used at all levels e.g., menu item coverage as part of a system test.

NOTE Forms include branch testing, decision testing, and statement testing. Synonyms for structure-based testing are structural testing, glass-box testing and white-box testing.

4.67
suspension criteria
criteria used to (temporarily) stop all or a portion of the testing activities

4.68
syntax testing
test design technique in which test cases are derived from a set of compositional rules (a syntax) for defining data objects (the inputs to the test item), and are designed to exercise options (valid variations) and mutations (invalid variations) of the defined syntax

4.69
test
(1) noun: activity by which a test item is evaluated and the outcomes reported
(2) verb: see testing

4.70
test basis
body of knowledge from which the required behaviour of a test item can be inferred

NOTE The test basis may take the form of documentation, such as a requirements specification, design specification, or module specification, but may also be an undocumented understanding of the required behaviour.

NOTE For specification-based testing the test basis is used to derive both test inputs and expected results, whereas for structure-based testing, the test basis is used solely for deriving expected results.

4.71
test case
set of test pre-conditions, inputs (including actions, where applicable), and expected results and expected test post-conditions, developed to achieve such an interaction

NOTE In exploratory testing especially, the results of executing a test case, and the resultant state of the system, are rarely documented.

4.72
test case precondition
state the test item must be in before the test case can be executed

4.73
test case specification
documentation of a set of one or more test cases

4.74
test completion criteria
criteria for determining when testing is complete

4.75
test completion report
report that provides a summary of the testing that was performed

4.76
test condition
a testable aspect of a component or system, such as a function, transaction, feature, quality attribute, or structural element identified as a basis for testing

NOTE Test conditions may be used to derive coverage items, or may themselves constitute coverage items.

4.77
test coverage
degree, expressed as a percentage, to which specified coverage items have been exercised by a test procedure or procedures

4.78
test coverage item
attribute or combination of attributes to be exercised by a test case that is derived from one or more test conditions by using a test design technique

4.79
test data
data created or selected to satisfy the pre-conditions for test execution, which may be defined in the Test Plan, Test Case or Test Procedure

NOTE Test data may be stored within the product under test (e.g., in arrays, flat files, or a database), or may be available from or supplied by external sources, such as other systems, other system components, hardware devices, or human operators.

4.80
test design specification
document specifying the features to be tested and their corresponding test conditions

4.81
test design technique
procedure used to identify test conditions for a test item, derive corresponding test coverage items, and subsequently derive or select test cases

4.82
test environment
see software test environment

4.83
test environment requirements
description of the necessary and desired properties of the test environment

NOTE All or parts of the Test Environment Requirements may reference where the information can be found, e.g. in the appropriate Organizational Test Policy and Test Strategy, Test Plan, and/or Test Specification.

4.84
test environment readiness report
document that describes the status of each environment requirement

4.85
test execution
process of running a test on the test item, producing actual result(s)

4.86
Test Execution Log
document that summarizes the course of the execution of one or more test procedures

4.87
test item
system, software item, or work product (e.g. requirements document, design specification) that is an object of testing

4.88
test level
specific instantiation of a test sub-process

NOTE For example the following are common test levels that can be instantiated as test sub-processes: component test level/sub-process, integration test level/sub-process, system test level/sub-process, acceptance test level/sub-process.

4.89
test management
planning, estimating, monitoring, reporting, and control of test activities

4.90
test method
see test design technique

4.91
test object
see test item

4.92
test outcome
consequence or outcome of the execution of a test. It includes outputs to screens, changes to data, reports, and communication messages sent

4.93
test phase
specific instantiation of test sub-process

NOTE Test phases are synonymous with test levels, therefore examples of test phases are the same as for test levels (e.g. system test phase/sub-process).

4.94
Test Plan
document that describes the plan to complete testing for a project

NOTE A project may have more than one test plan, for example there may be a project test plan that encompasses all testing activities on the project; further detail of particular test activities may be defined in one or more test sub-process plans (i.e. a system test plan or a performance test plan).

4.95
test procedure
sequence of test cases and any associated actions required to set up the preconditions for the test cases

NOTE Test procedures include detailed instructions for how to run a set of one or more test cases selected to be run consecutively, including set up of common preconditions, and providing input and evaluating the actual result for each included test case.

4.96
Test Procedure Specification
document describing the test sets, which are collections of test cases to be executed for a particular objective. The test cases in a test set are listed in their required order in the test procedure

NOTE A test procedure specification may be a separate document, a chapter in the test specification document, held in a database, or loosely scribbled. It is also known as a manual test script. A test procedure specification for an automated test run is usually called a test script.

4.97
test process
process used to provide information on the quality of a software product, often comprised of a number of activities, grouped into one or more test sub-processes

4.98
test result
indication of whether or not a specific test case has passed or failed, i.e. if the actual result corresponds to the expected result or if deviations were observed

4.99
test requirement
see test condition

4.100
test script
test procedure specification for manual or automated testing

4.101
test set
set of one or more test cases with a common constraint on their execution (e.g. a specific test environment, specialized domain knowledge or specific purpose)

4.102
test specification
complete documentation of the test design, test cases and/or test procedures for a specific test item

NOTE A test specification may be detailed in one document, in a set of documents, or in other ways, for example in a mixture of documents and database entries.

4.103
test specification technique
see test design technique

4.104
Test Status Report
report that provides up to date information about the status of the testing that is being performed

4.105
Test Strategy
part of the Test Plan that describes the approach to testing for the specific test project or test sub-process.

NOTE The test strategy is a distinct entity from the Organizational Test Strategy.

NOTE The test strategy describes the test sub-processes to be implemented (for a project test plan), the retesting and regression testing to be employed, the test design techniques and corresponding test completion criteria to be used, test data, test environment and testing tool requirements, and expectations for test deliverables.

4.106
test sub-process
test management and dynamic (and static) test processes used to perform a specific test level (e.g. system testing, acceptance testing) or test type (e.g. usability testing, performance testing) normally within the context of an overall test project

NOTE A test sub-process may comprise one or more test types. Depending on the life cycle model used, test sub-processes are also typically called test phases, test levels, test stages or test tasks.

4.107
test summary report
document summarizing testing activities and results

4.108
test technique
see test design technique

4.109
test traceability matrix
document or spread-sheet used to identify related items in documentation and software, such as requirements with associated tests

4.110
test type
group of testing activities that are focused on specific quality characteristics.

NOTE A test type may be performed in a single test sub-process or may be performed across a number of test sub-processes (e.g. performance testing completed at a component test sub-process and also completed at a system test sub-process).

NOTE Examples of test types are: security testing, functional testing, usability testing, and performance.

4.111
testing
set of activities conducted to facilitate discovery and/or evaluation of properties of one or more test items

NOTE Testing activities may include planning, preparation, execution, reporting, and management activities, insofar as they are directed towards testing.

4.112
traceability matrix
see test traceability matrix

4.113
use case testing
example of a scenario-based technique, where each test case is modelled as a use case. In use case testing, an activity diagram typically defines the scenario; it is then used to identify test conditions and test coverage items, from which use case test cases can be derived

4.114
white box testing
see structure-based testing

5 Software Testing Concepts

5.1 Introduction to Software Testing

This standard is written because:

— The test item(s) being tested don't always do what they are expected to do.

— The test item(s) being tested need to be verified.

— The test item(s) being tested need to be validated.

— Evaluation of the item(s) needs to be conducted throughout the lifecycle.

— Information on the quality characteristics (i.e. functionality, reliability, usability, etc.) of the item(s) being tested is required by decision makers.

It is a recognized condition of life that human beings are not perfect and hence inevitably make mistakes in things they create, including software. It is therefore necessary to test software systems before they are released to the users to reduce the risks of these mistakes having a negative impact when they are used. It is equally necessary to ensure that testing is performed well.

Mistakes or introduced defects are not usually made on purpose, and they may be largely unavoidable. A mistake or error, made by a human being, causes a defect to appear in the product that the person is working on; (e.g. a requirements specification or a software component). The defect causes no harm as long as it is not encountered. But if a defect is encountered under the right conditions during the use of the system, it may give rise to a failure in the system. A failure can be experienced by a user with potentially serious consequences, for example business reputation-wise, safety-wise, economically, security-wise, and/or environmentally.

The primary goals of testing are to: provide information about the quality of the test item in terms of conformance to requirements; to find defects in the test item prior to its release into an operational environment; to mitigate the risks to the stakeholders of poor product quality; and to provide information on any residual risk in relation to how much of the test item that has been tested, i.e. the test coverage.

This information can be used for several purposes; including:

— to improve the test item by having the defects removed;

— to improve management decisions by providing information about quality and risk as the basis for the decisions; and

— to improve the processes in the organization by highlighting processes that allow defects to appear and/or to remain undiscovered where they could have been discovered.

Quality is value that someone is willing to pay for. Qualities of software include: conformance with requirements; absence of errors; usability; etc. *ISO/IEC 25010-1, Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software quality models* defines eight quality characteristics that can be measured or assessed through test activity (see 5.5.3).

Annex B presents a number of test-related metrics that could be used to provide information from testing activities.

The sooner a defect is found and corrected, the lower the cost. Software testing should focus on, under given constraints, finding as many defects as possible, as early as possible in the development process. Testing attributes include:

— Testing can be described as a process. A process is a set of interrelated or interacting activities that transforms inputs into outputs. The objective of this standard is to present and describe a generic testing process (see part 2 of this standard for more details).

— Testing exists in an organizational context, because management at a given organizational level has the ultimate responsibility for what is going on in their organization, including, but not necessarily limited to, projects including testing. This standard includes the organizational test process (see part 2 of this standard for more details).

— Testing should be planned, monitored and controlled, be it a development project (using any type of development model), or maintenance for an existing system. This standard includes a test management process, and can be applied to traditional projects, agile projects and the management of exploratory testing.

— Testing processes and sub-processes can be applied to a phase or level of testing (e.g. system testing) or a type of testing (e.g. performance testing).

— Testing entails examining a test item. A test item may be any work product, for instance a software system component, or a fully integrated software system. For example, the test item could be a business requirements specification, and (rather than executing a program) static testing can be applied to the specification to carry out a review of the requirements.

— Testing can be carried out on a product without executing the product on a computer. This is called static testing in this standard and in many areas of the industry, although other standards (e.g. IEEE1028) may more specifically call this reviews, walkthroughs or inspections. For static testing this standard acknowledges and identifies the role of the tester in these activities even though they may be “owned” by other groups or standards within a project. This is because the static testing activities are considered highly important for complete lifecycle testing and test involvement has been shown to be critical for early defect detection, reduced overall project costs and an improved ability to meet schedule demands.

— Testing can also be carried out by running executable test items, which is called dynamic testing. This is much more than ‘just’ executing tests. The dynamic testing process included in this standard covers each of the activities to be performed in dynamic testing.

— Verification is confirmation, through the provision of objective evidence, that specified requirements have been fulfilled.

— Validation demonstrates that the system can be used by the users for their specific tasks.

5.1.1 The Role of Testing in Verification and Validation

This standard addresses only parts of the verification and validation activities. Other standards, e.g. ISO/IEC 12207, address other verification or validation activities. Testing, as described in this standard, addresses both dynamic and static testing as well as associated sub processes and activities. This standard does not address, except in passing, demonstration activities or validation and verification. To provide complete validation and verification of a product an organization will need to use this standard in conjunction with other standards as part of a comprehensive engineering program. See Annex A for a diagram of verification and validation activities.

5.1.2 Exhaustive Testing

Due to the complexity of systems and software it is not possible to exhaustively test every single aspect of any given test items. Testers should recognise that exhaustive testing is not possible and that test activities should be targeted to best fulfil the test objectives for a test item. Risk Based Testing is an approach that uses risk to direct test effort. See section 5.4 Risk Based testing.

5.1.3 Testing as a Heuristic

In engineering (and software engineering) a heuristic is an experience (trial and error) based method that can be used as an aid to problem solving and design. However, while heuristics can be used to solve problems, they are fallible in that they may sometimes not solve, or only partially solve, a problem. Much of systems and software testing is based on heuristics. For instance, they are useful to create models of the system to be tested; however, they may fail to fully model the system and so defects in the system may not be found even though the testing appears to be complete. Recognition that the manner in which testing is undertaken may be fallible allows us to mitigate the risk of an ineffective test lifecycle by employing multiple test strategies.

5.2 Software Testing in an Organizational and Project Context

Modern businesses involved in system development have an interest in developing effective, efficient and repeatable processes. To accomplish this they generally develop a robust set of system lifecycle processes that are applied to the development projects that they perform. This standard is intended to be useful for adoption by an organization and on specific projects. An organization would adopt the standard and supplement it with appropriate procedures, practices, tools and policies. A software or system development project of the organization would typically conform to the organization's processes rather than conform directly to this standard. In some cases a project may be executed by an organization that does not have an appropriate set of processes in place at an organizational level. Such a project may apply the provisions of this standard directly to that project.

In every type of software producing organization, be it a multi-national organization with thousands of testers or a one person company, software testing should have the commitment of the highest level of organizational management, be it the CEO, open-source steering committee or a department manager. This commitment is preferably expressed in an Organizational Test Policy and one or more Organizational Test Strategies, serving as the basis for all the software testing being performed within the organization. Test policies and strategies are usually only seen in more mature organizations. Testing can be, and is, performed without test policies and test strategies in organizations of low maturity, but it gives less coherence to the testing within the organization and makes the test work in the projects harder.

Software testing is performed in a management context. This means it should be planned, monitored and controlled. The context could be a development project (ranging from a multi-person, multi-year formal development project to few person hours of informal development) or the on-going maintenance of an operational system. Some considerations in understanding the context of testing for a work item are: overall budget; schedule demands; risk; organizational culture; customer/user expectations; availability of infrastructure environments for testing; scope of the project; criticality of the project; etc. The experience of the industry is that no single test strategy, plan, method or process will work in all contexts. Hence organizations and projects should tailor and refine the details of testing with reference to standards such as this. The overall management plan for a project should include consideration of the test activities to be performed as a part of the project. A Project Test Plan should reflect both the Organizational Test Policy and Organizational Test Strategy and deviations from organizational guidelines. It should also account for the constraints given in the overall management plan. A Project Test Plan includes a project test strategy and the project-specific decisions based on this strategy.

The testing for a project will often be performed across a number of test sub-processes; each test sub-process will have a corresponding Test Plan (a Test Sub-process Plan e.g. a System Test Plan or a Performance Test Plan) consisting of a test sub-process strategy aligned with the project test strategy, and test sub-process specific detail.

Figure 1 shows how testing fits into a multi-layered context. Testing is founded on the regulatory context that an organization is in, if any. This context is made up of laws, regulations and industry standards. Inside this regulatory context the organization develops the necessary policies and procedures it requires in order to be successful in its context. The Organizational Test Policy operates at this level. Within the organizational context each project is instantiated to meet some need or opportunity the organization has identified. The project establishes a context within which a lifecycle model is chosen. At this level, based on the context of the project and lifecycle model, an approach to testing is determined. The project plan and approach to testing form the basis for the Project Test Plan.

The Project Test Plan identifies the overall approach to testing and the test process to be used. It establishes the context of testing for the project by determining the objectives, strategy, resources, and schedule; it also identifies the applicable test sub-processes (e.g. system testing, performance testing). The identified sub-processes are then described in the sub-process test plan (e.g. System Test Plan, Performance Test Plan). The test plan also describes the appropriate test techniques (static or dynamic) to use in order to complete the testing required by the particular sub-process plan. For more on test techniques see clause 5.5.5 of this document and Part Four of this standard.

Each sub-process test plan may address more than one test level (e.g. the security test plan may address several test phases) and may address more than one test type (e.g. a System Test Plan that addresses functional and performance testing at the system test level). The sub-process test plan will also describe the approach to test execution (e.g. scripted, unscripted, or, more likely, a mixture of both).

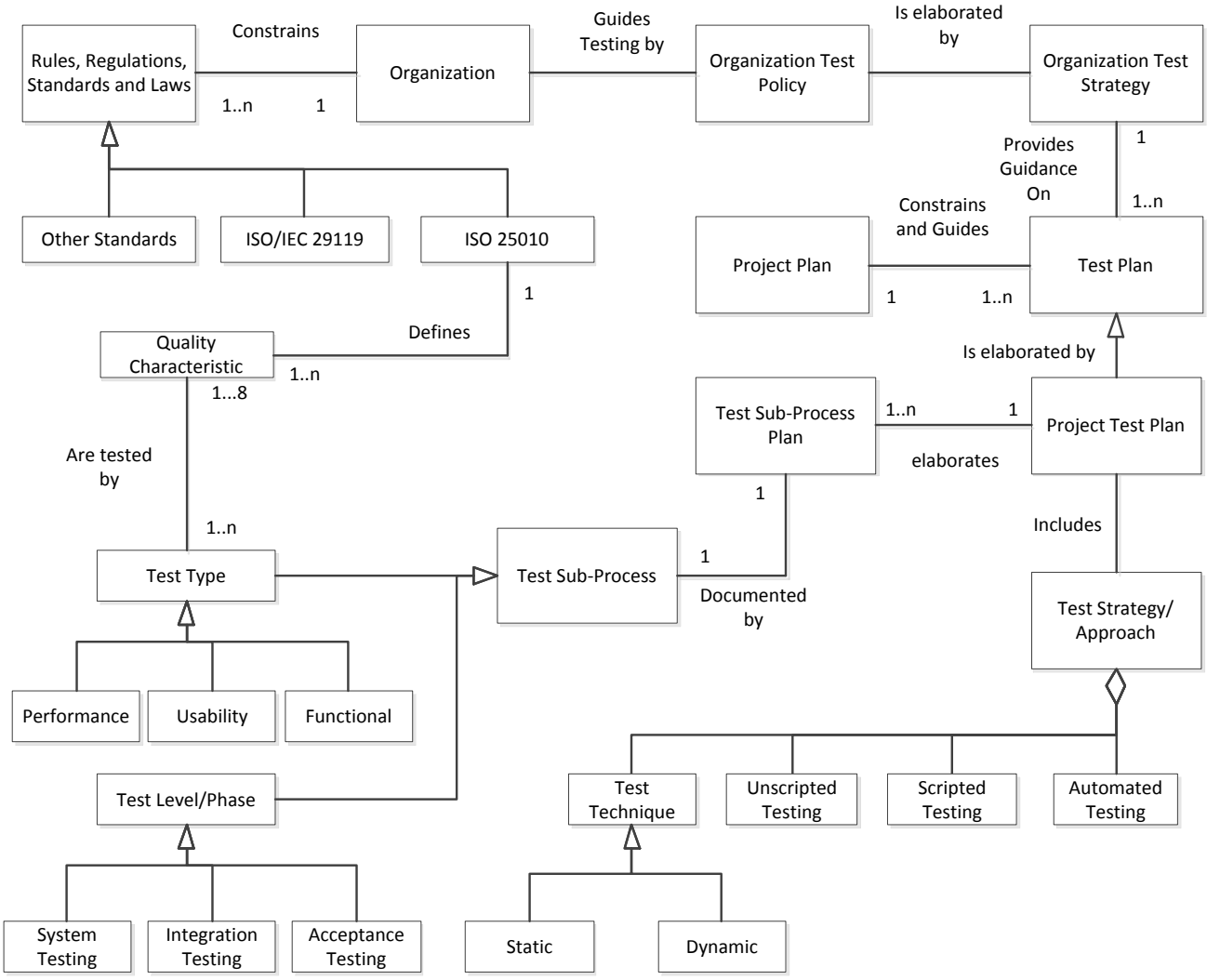

Figure 1 — Multi-layered text context Diagram

The relationship between the generic test process, generic test sub-processes, test levels and test types is described in more detail in Figure 2. This figure shows that the application of generic test sub-processes into particular test levels and/or test types. The generic test process can be applied in the following ways:

— As a test level. That is each test level is a specific application of the generic test sub-process (e.g. component test sub-process, acceptance test sub-process);

— As a test type. That is each test type is a specific application of the generic test sub-process (e.g. performance test sub-process, usability test sub-process); and

— A test sub-process associated with a test level may contain more than one test type sub-process (e.g. functional and performance test sub-processes at the system test level);

— The project test process may be comprised of a hierarchy of test sub-processes (e.g. a component test sub-process, an integration test sub-process, a system test sub-process and an acceptance test-sub-process)

The diagram also makes explicit the relationship between test types and quality characteristics (as defined in the *ISO/IEC 25010-1, Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software quality models* standard). Each test type targets one particular quality characteristic. For more on the relationship between test types and quality characteristics see clause 5.5.3 of this document.

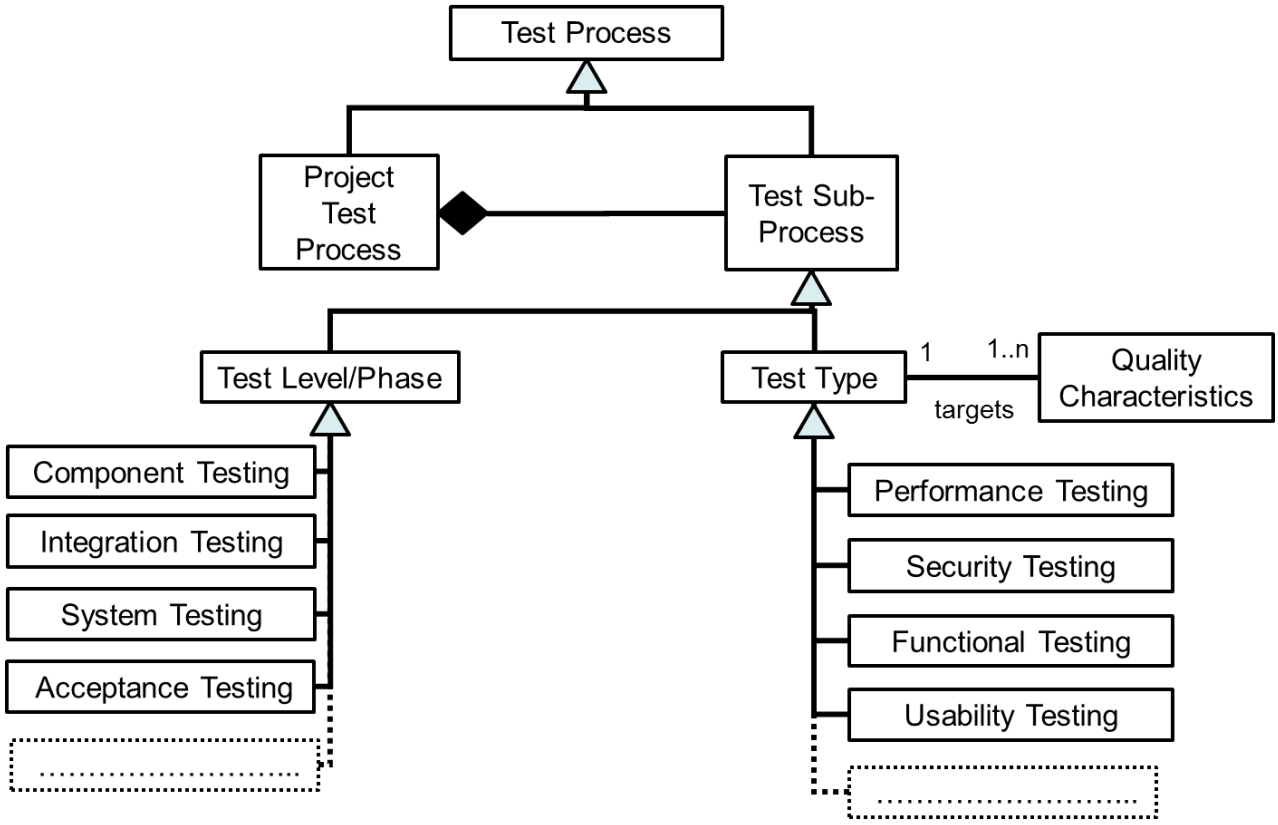

Figure 2 — The relationship between the generic test sub-process, test levels and test types.

5.2.1 The Test Process

This standard uses a three layer process model, which is described in detail in Part 2 of this standard, and illustrated at a high level in figure 3. The process model starts with an organizational layer managing high level (organizational) test specifications, such as Organizational Test Policy and Organizational Test Strategy. The middle layer moves into test management (project test management, phase test management, type test management), while the bottom layer defines a number of dynamic test processes used for dynamic testing.

The 3-layer process model is shown in figure 3.

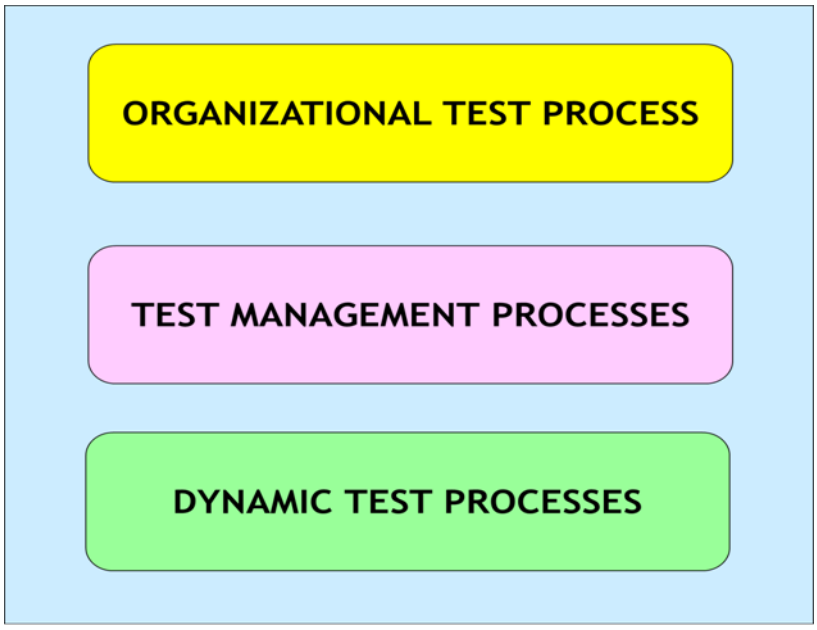

Figure 3 — The multi-layer relationship between test processes

The Organizational Test Policy expresses the organization's management expectations and approach to software testing in business terms. Although it would also be useful to anyone involved with testing, it is aimed at executives and senior managers. The creation, implementation and maintenance of the Organizational Test Policy are described by the Organizational Test process.

The Organizational Test Strategy expresses the generic requirements for the testing to be performed on all the projects run within the organization (assuming they are not too dissimilar). It is aligned with the Organizational Test Policy providing detail on *how* the testing is to be performed. The creation, implementation and maintenance of the Organizational Test Strategy are also defined by the Organizational Test process.

The management of the testing to be performed is described by the Test Management processes. Based on an analysis of identified risks, and taking account of the Organizational Test Strategy, a project-related test strategy is developed. This strategy is elaborated in terms of defining the static and dynamic testing to be performed, the overall staffing, and the schedule, balancing the given constraints (resources and time) with the defined quality of the test work to be done. This is documented in a Project Test Plan. The test activities performed are monitored as they progress and the plan updated and controls applied when necessary. The overall result of the testing for the project is documented in the Project Test Completion Report.

The Test Management Processes are shown in figure 4.

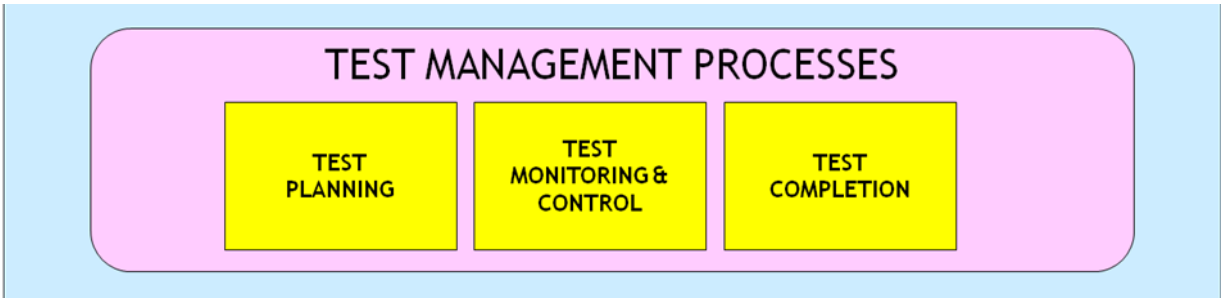

Figure 4 — The test management processes

The overall testing for a project is usually broken down into smaller test sub-processes (e.g. component testing, system testing, usability testing, performance testing), and these should also be managed, executed, and reported on in a manner similar to the overall test project. The test management processes can also be

applied to test sub-process. Examples of test sub-process plans are a System Test Plan, Acceptance Test Plan, or a Performance Test Plan.

Test sub-processes may include both static testing and dynamic testing. The Dynamic Test process is outlined in figure 5 and fully described in Part 2 of this Standard. Processes for Static Testing are described in other published standards (e.g. IEEE 1028).

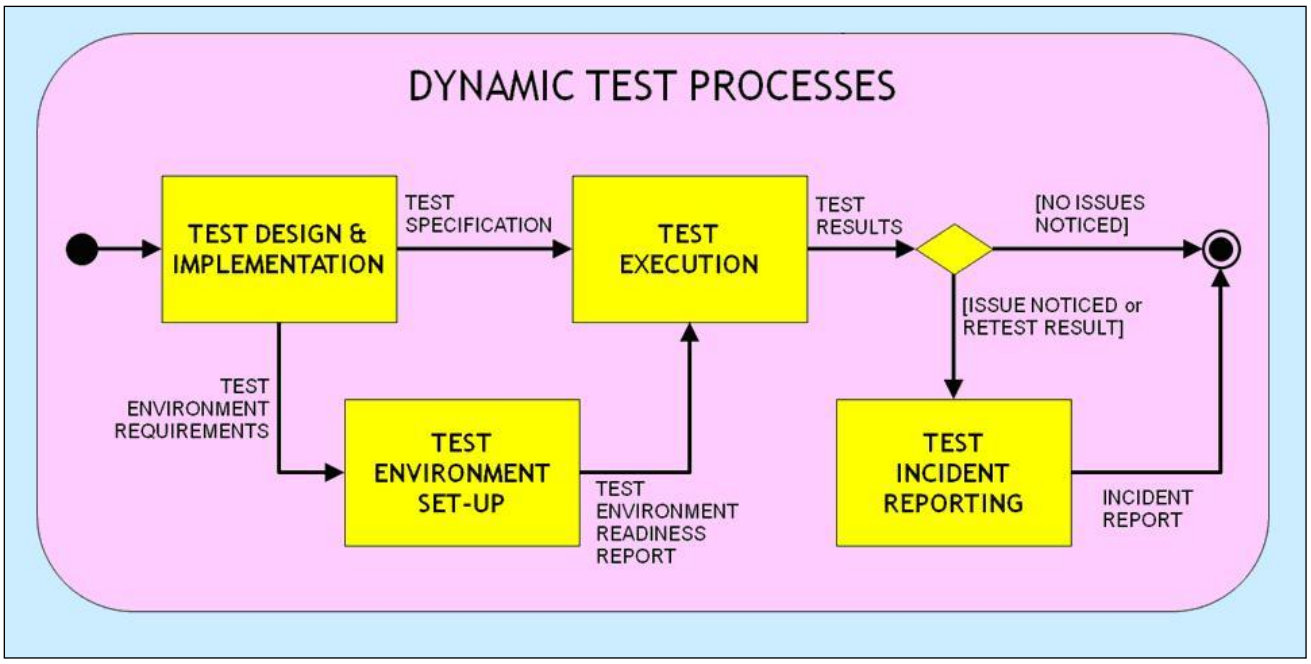

Figure 5 — Dynamic test processes

For more information on any of the testing processes, including, the Organizational Test Process, the Test Management Process, and the Dynamic Test Process, refer to Part 2 of this standard.

5.3 Generic Testing Processes in the System Life Cycle

Systems have an expected life cycle from their initial conception to their eventual retirement. An example system life cycle is illustrated in figure 6.

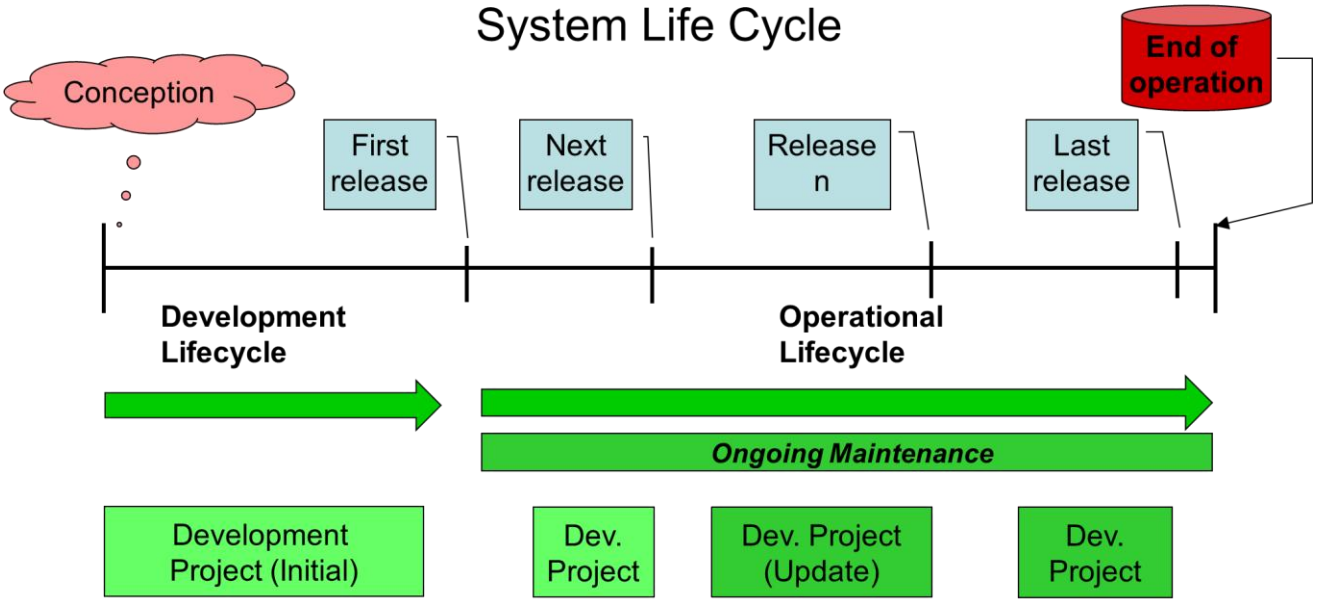

Figure 6 — An example System Life Cycle

A system lifecycle is generally comprised of multiple sub-lifecycles. Figure 6 shows that a system lifecycle is often made up of one or more development lifecycles and one or more operational lifecycles.

The period of time from conception to the initial release is known as the development life cycle, which is a subset of the system life cycle. A development life cycle is managed and controlled in a development project.

From the time the system is first released it enters operation. A system remains in operation until it is retired; this may be a period of a few hours to several decades. The operational period often consists of periods of time where a specific version of the system is used, while a new version is being developed for release. Any new version being developed should be treated as a development project in its own right with the corresponding testing this entails. Usually on-going maintenance is also set up to keep the system available and behaving as expected.

It can also be the case that testing occurs on an operational system without a corresponding development project - for example “dry run” Disaster Recovery tests. The process from this standard can also be applied in situations such as this.

5.3.1 Development Project Sub-processes and their Results

System development typically consists of a few common building blocks. These include:

— Defining the parameters of the problem in terms of business objectives, operational environment and constraints (business requirements engineering);

— Defining a solution product at a high level as a set of system functions, interfaces, and quality characteristics (system requirements engineering);

— Defining the system at an intermediate level as a set of interacting components (architectural design);

— Defining how each system component should implement its assigned behaviour (detailed design),

— Implementing the required behaviour, typically as code in a programming language or by assembling reusable components (development);

— Assembling the executable or interpretable components (gradually) into a complete system;

— Verifying that the completed system satisfies the system requirements, and validating its fitness for use (system testing);

— Gaining assurance that the system adequately resolves the business problem, or meets the business opportunity, and is acceptable for production use (acceptance testing).

The particular approach to system development adopted by an organization or project will determine how these building blocks are arranged. In a sequential development project, requirements analysis may be an initial process in its own right. In an agile development project, requirements will be identified for each incremental delivery, perhaps every few weeks.

In the software industry these building blocks are typically referred to as ‘phases’, ‘stages’, ‘steps’, ‘levels’, or more generically as ‘development sub-processes’.

In each of the development sub-processes something is produced; it may be a highly-structured detailed document or it may be informally documented or undocumented decisions. The following phases are usually completed prior to commencing the coding (or other production activities, such as writing manuals):

— Requirements analysis produces requirements, for example in the form of a Product Backlog for Agile development or a formal System Requirements Specification;

— Architectural design produces a logical model of how the system will be implemented in terms of a number of interacting components (or units) along with the use of technology such as network infrastructure, operating system and middleware for example in the form of a class diagram and/or other diagrams or a formal Architectural Design Specification;

— Detailed design produces a description of how each component should be implemented, for example in the form of pseudo-code, a flow diagram, or a formal Detailed Design Specification.

The system is produced in the following way based on the available information:

— The software system’s components are produced in the coding phase where the source code and other necessary components are written.

— Sub-systems are produced when components are integrated into larger sub-systems, until the software system is completed at the end of the integration phase.

— The completed system is accepted by the customer.

In any given development project the individual development sub-processes may be performed once or repeated as often as required. The generic development sub-processes and related work products, sub-systems and completed software system are shown in figure 7.

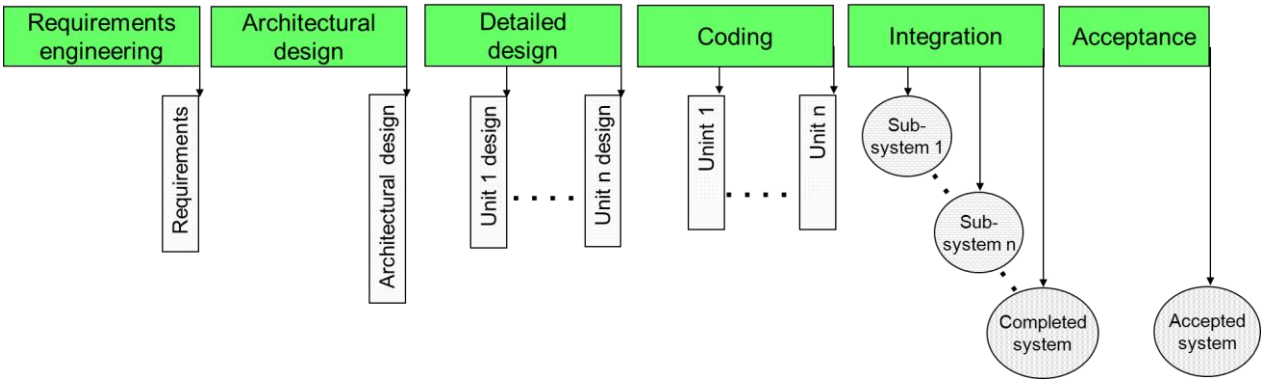

Figure 7 — Example of development sub-processes

Each work product, software system component, and the complete software system is a potential test item.

Note that it is out of the scope of this Standard to define a development model. The phases depicted in figure 7 are only examples needed to illustrate how testing applies to development.

5.3.2 On-going Maintenance and its Results

On-going maintenance takes place during the operational part of the system life cycle, i.e. after the initial development, and it may be managed in the context of an Application Management or an IT Service Management process framework. A project may be set up for continuous maintenance, and this is often quite different from a development project, for example the financial model may be different. One common financial model is that an amount of money is budgeted for maintenance in a specific period; this may be stated in a service level agreement (SLA) between the customer and the maintenance organization.

The on-going maintenance is generally concerned with keeping a specified level of reliability and availability of the system. This involves production of new versions of the system with corrections of high priority defects found in operation and possibly the introduction of high priority minor changes to the functionality. Such versions may be released as the 'live' system on an ad hoc basis, when selected corrections and changes are completed, and/or on a fixed frequency basis, for example every 3 months. If a maintenance release period is ad hoc, the corrections and/or changes to be included are selected, and their implementation and release is usually performed as quickly as possible. If a fixed length maintenance release period is used, as many corrections and/or changes that can be implemented in that timeframe are made, with the implementation order based on an agreed prioritisation. The primary outputs of such a release period are shown in figure 8.

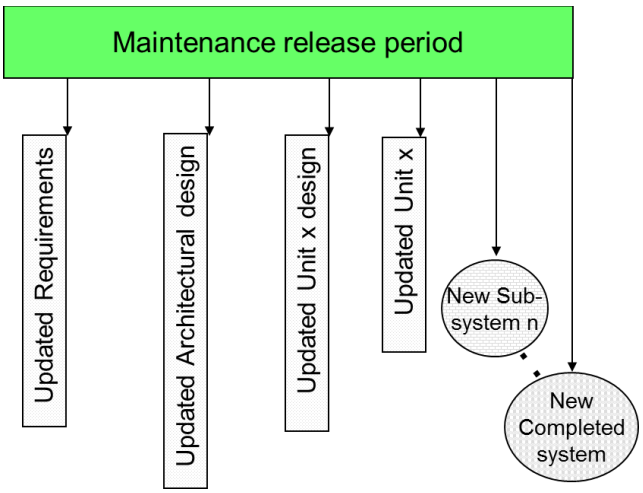

Figure 8 — Example Maintenance Release Period

Depending on the purpose of the release, each of the outputs shown in figure 8 may be produced in a single release period, or it may be that only a subset is needed. For instance It may be that an ad hoc correction

does not affect the requirements and the design, but only the code and the completed system; or it may be that all the work-products are affected more than once before the system is ready for release.

Maintenance release periods are repeated as necessary during the operational lifetime of the system.

5.3.3 Support Processes for the Software Development Life Cycle

Within an organization, support processes are required to aid the system development lifecycle. Some of these are:

— Quality assurance;

— Project management;

— Configuration management;

— Process improvement.

5.3.3.1 Quality Assurance and Testing

Quality Assurance is a set of planned and systematic supporting processes and activities required to provide adequate confidence that a process or work product fulfils established technical or quality requirements. During the process of testing, numerous work products may be produced, such as an Organizational Test Policy, Organizational Test Strategy, Test Plans for the test projects (Project Test Plan) and for specific test sub-processes (e.g. system testing, performance testing), as well as test specifications and test environments, are created. Measures should be collected during testing, as they can provide information about the quality of test processes and the effectiveness of their application on each project.

5.3.3.2 Project Management and Testing

Project management refers to the support processes that are used to plan and control the course of a project.

Project management also includes the management of the test project within the overall project. This can be done in several ways, ranging from delegating the entire responsibility for the test budget, test time frame, and expected quality of testing to a test manager, to having the project manager performing all the test management.

Regardless of who has responsibility for the individual processes, project management and test management processes are closely related, as shown in figure 9.

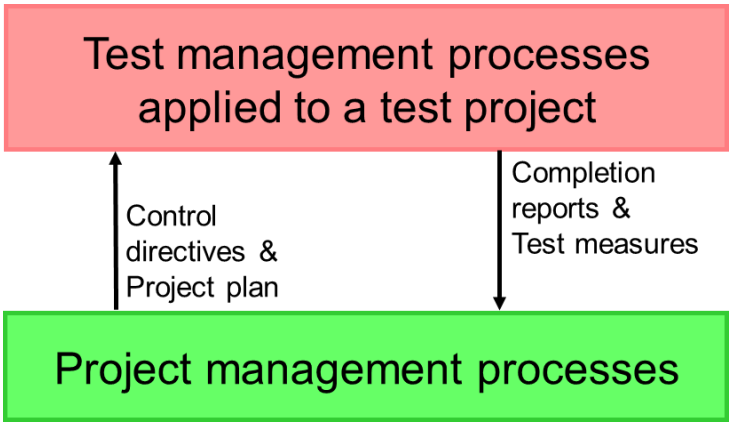

Figure 9 — Relationship between the overall project and the test project

The estimation, risk analysis, and scheduling of the test activities should be consolidated with the overall project planning. The project plan, which is an information item from the project management process, is therefore an input to the test management process when it is used for managing the test project.

During the course of the test project, measures collected from detailed test activities are analyzed by the test manager and communicated to the project manager for analysis in the project context. This may result in changes to the project plan affecting the test project, updated project plans, and appropriate directives to ensure that the test project is kept under control, which will need to be issued to the test project.

When a test sub-process or the test project is completed a completion report summarizing the course and results of the test sub-process or the test project is provided to the project manager.

5.3.3.3 Configuration Management and Testing

Configuration management is another set of support processes with which testing interacts. Configuration management focuses on establishing and maintaining consistency of a system or product's performance and its functional and physical attributes with its requirements, design, and operational information throughout its life. The purpose of configuration management is to establish and maintain the integrity of work products.

Configuration management processes include unique identification, controlled storage, release audit, change control, and status reporting for selected work products, system components, and systems during the entire lifetime of the product. An object under configuration management is called a configuration item. The configuration management processes are event driven, that is they are all initiated independently from each other depending on the requirements of other processes.

Work products from the testing process, which can be placed under configuration management, include:

— Organizational test specifications, e.g. Test Policy and Organizational Test Strategy;

— Test plans;

— Test specifications and procedures;

— Test environment configuration items such as: operating systems, middleware, tools, test data(base), drivers and stubs.

All three layers in the test process model defined in this standard can interact with configuration management. The configuration items provided to a test process are the work products the process needs as input and which are under configuration management. The configuration items delivered from a test process are the work products produced by the test process which need to be placed under configuration management.

The Organizational Test Process may, for example, produce an Organizational Test Policy and an Organizational Test Strategy, which are placed under configuration management. The project test manager may extract a copy of the Project Plan from configuration management and use that as the basis for the Project Test Plan, which is subsequently placed under configuration management. Those performing a dynamic test sub-process may extract a copy of the requirements specification from configuration management and use that as the basis for a test specification which is subsequently placed under configuration management.

The configuration management reports for a test process should provide detailed measures which may be needed for analysing the progress and status of incidents. Where an effective configuration management system is in the place this is where incidents arising in any process in the organization, including testing, should be handled.

5.3.3.4 Process Improvement and Testing

Process improvement takes actions to change an organization’s processes so that they more effectively and efficiently meet the organization’s business goals.

The test process and the process improvement process interact in two ways:

1. The test processes deliver information on which process improvement actions can be based (along with information obtained from other sources);

2. The test processes can themselves be subject to process improvement.

When testing delivers information to process improvement the interaction is typically between the test management process applied at the project level and the process improvement process. Test measures can be communicated directly from the test management processes. The process improvement process can also obtain some test related metrics from configuration management, if this is in place and covers change control.

In the case of the test processes being subjected to process improvement, this can be in the scope of an organization-wide improvement or it can be in the scope of improving test processes independently of the overall organization. In either case, process improvement processes should obtain information from many sources to diagnose which test processes to improve and how, and to define the improved test processes. This will result in a new version of each of the selected test processes, which will then have to be rolled out to the appropriate users.

5.4 Risk-based Testing

It is impossible to test a software system exhaustively, thus testing is a sampling activity. A variety of testing concepts (techniques, types and methods) exist to aid in choosing an appropriate sample to test and these are discussed and outlined in this standard. However a key premise of this standard is the idea of performing the 'best possible' test within the given constraints and context by identifying the relative importance of different tests in terms of the risks they mitigate for the stakeholders of the completed system and for stakeholders developing the system. Carrying out risk-based testing (including risk-based test planning) ensures that the risks with the highest priority are paid the highest attention during testing.

A risk has two aspects:

1. Impact (or effect or consequence);

2. Likelihood (or probability and/or frequency).

The two aspects of risk can be combined into the risk exposure, normally calculated as the product of the likelihood of the event happening and the impact of the event if it occurs. Thus it is not a risk if there is no impact from the event or if there is no likelihood that the event will happen. It is not a risk either if the likelihood of an event is 100%; in this case it is an issue.

Risks can be categorized corresponding to where they may cause harm – or what they are threatening. In this standard two categories of risks are used, namely risks concerning:

1. The product;

2. The project.

Product risks are the risks of defects remaining in the product when it is delivered in relation to the effect possible failures caused by these defects may have. Product risks may originate in the functional and non-functional requirements (e.g. missing, ambiguous, or misunderstood requirements), the design, or the implementation. Product risks jeopardize the customer satisfaction, business reputation, user experience and, for some systems, can endanger user's lives and livelihoods. Examples of product risks are related to a specific product, but may include:

— Overlooked functionality that is consequently not implemented;

— Incorrect results due to an algorithm that is incorrectly implemented; and

— The possibility of revealing confidential customer information through a loop-hole in a reporting function.

Testing can, allied with defect removal, reduce product risk by reducing the number of defects remaining in the product when it is released. Testing can also reduce the perceived risk as passed tests will increase confidence in the product thus decreasing the perceived probability of failure. Tests should be designed and built to reduce the likelihood of product risks manifesting as failures.

Project risks are related to successful completion of the project. A project could be the overall development project or the test project. Risks identified in the overall project may be further identified as test project risks and referred to the person responsible for the test project (e.g. the project test manager). Likewise risks identified in the test project may be risks that need to be managed at a higher level and thus referred to the overall project (e.g. the project manager). Project risks may originate in, for example, people assigned to the project (their availability, adequate skills and knowledge, and personalities), available time and money, environment including tools, and customer / supplier relationships. Project risks jeopardize the project's progress and successful completion according to the plan. Examples of test project risks include:

— The necessary test analysts are not available when the test design and implementation activity is expected to start;

— The testers are not adequately trained in testing techniques, so testing requires more resources than expected; and

— The integration is more time-consuming than expected and so testing timelines have to be adjusted to account for starting integration testing later than expected.

The planning, monitoring, and control of the test project can mitigate identified and analyzed project risks if an appropriate test strategy is applied, especially if testing is started early.

Project risks and product risks can influence and be the cause of each other. A project risk may cause a product risk, and a product risk may cause a project risk. If, for example, a project risk results in time being cut from component testing, this will result in an increase in the likelihood of defects remaining in the components that are not tested or not tested sufficiently. This may further increase the likelihood of the project risk of there being insufficient time to perform a complete system test because too many failures related to defects in components are encountered in the system test.

Testing and product risk management are tightly related as they support each other. Testing is based on the results of risk analysis and test results inform the on-going risk analysis.

The product risks can be used in the test planning to make the testing as effective as possible. It can be used to target the testing effort, since different test sub-processes are most effective for mitigating different risks (e.g. a risk of slow response times can be mitigated by performance testing). Functional component testing is, for example, more effective for testing a product risk related to calculation algorithms than non-functional system testing.

The risk estimation results can also be used to prioritize and distribute the test effort. The features with higher risk exposures can be scheduled first and given more time.

Test results can be used to inform the on-going risk analysis. If, for example, a feature has been tested then the test results can inform the risk analysis of changes in the likelihood of the risks for that feature. There could be a decrease in risk if fewer defects than expected are found and/or when defects are successfully removed; it could be an increase in risk if more defects than expected are found and/or there are difficulties related to the removal of defects. It could also be the case that the test results show that a risk is eliminated.

Product risks and their related exposure can be included in the completion criteria for testing, and risk analysis can contribute to the decision to stop testing and release the test item.

5.5 Test Sub-process

A test project is usually structured as a number of test sub-processes based on the project test strategy. A test sub-process can be associated with a system life cycle phase and/or be focused on a specific quality attribute. A test sub-process may include both static testing and dynamic testing.

Each test sub-process is managed by applying the test management process to it. A test sub-process starts with test planning. The test monitoring and control activity is continuous during the entire course of the testing planned for the test sub-process, and information from the monitoring activity may cause the planning to be revisited as appropriate.

The test planning involves identifying the test objective, test scope, and the risks associated with the test sub-process in question. The result of this guides the strategy for the test sub-process, including the static testing and dynamic testing to be planned for in the test sub-process. In some cases what is outlined in the project test strategy can be used directly in the strategy for the test sub-process, in other cases specific decisions for a test sub-process will have to be made based on the specific risks to be handled.

When describing a test sub-process it is important to specify what features of the test item are to be tested and which are not. This is to ensure that the expectations concerning the scope of the testing are clearly and properly understood.

It is unusual for a test sub-process to include only one round of dynamic test or static test. If only one round of each type of testing is defined it may be necessary to repeat it, for example to meet the test completion criteria. Repeated dynamic test processes are usually referred to as retesting and/or regression testing. Regression testing and retesting are activities performed to ensure that changes made to a work product in order to correct defects have not caused further defects (for more information see clause 5.5.5). An example test sub-process may hence be illustrated as shown in figure 10.

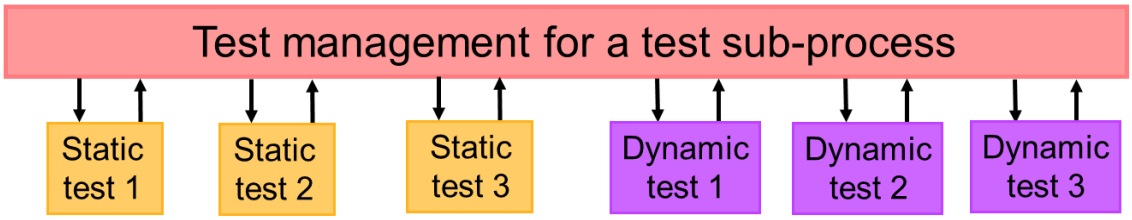

Figure 10 — Generic example test sub-process

The number of test sub-processes in a test project depends on the test strategy and the lifecycle phases defined for the overall project. It does not, however, depend on the development life cycle.

Examples of test sub-processes are detailed in Annex C.

The test objective, test item, test basis, and risks are specific to a test sub-process, and these guide the choice of test activities to perform in a test sub-process as well as the test techniques to use. Examples of test objectives, test items, test basis, and test techniques are provided below.

5.5.1 Test Objectives

A test is performed to achieve one or more objectives. The test objectives covered by this standard include:

— Provision of information to the risk management activity;

— Provision of information about the qualities of the system;

— Assessment of whether the product has met stakeholder expectations;

— Assessment that defects have been correctly removed with no adverse side effects;

— Assessment of correct change implementation with no adverse side effects; and

— Assessment of fulfilment of regulatory requirements.

Testing is completed to fulfil the test objectives for a feature or feature set. The type of feature to be tested will determine the kind of testing that will be required. Features have quality characteristics that they are intended to fulfil. The composition of the quality characteristics for a feature or feature set allows the tester to determine what test types may be used to fulfil the test objectives.

It is possible that only some of the test objectives are relevant for a particular test sub-process or product type. Deciding the relevant test objectives for a product can aid in determining the correct test sub-processes to apply. For example, in testing a commercial off the shelf product the test objective of assessing that defects have been fixed may not be relevant as it is expected that the vendor has already completed robust testing to find and fix defects. Therefore the test sub-process may be applied at the acceptance testing level to meet the test objective that the change is implemented with no adverse side effects.

5.5.2 Test Item

A test is performed on a test item against what is expected of the test item; this expectation is described by the test basis (see 5.5.4). A test item is the result of an activity, such as management, development, maintenance, test itself, or other supporting processes.

Examples of test items include:

- Code-related test items:
  - An executable component;
  - A sub-system;
  - A complete system.
- Document-related test items:
  - A plan, for example project plan, Test Plan, or configuration management plan;
  - A requirements specification;
  - Architectural design and detailed design;
  - Source code;
  - A manual, for example user manual or installation manual;
  - Test specification and test procedures.

The last entry in the list above indicates that although test specifications and test procedures are testware, they should also be subject to testing (static testing) as they may not be defect-free.

5.5.3 Testing of Quality Characteristics

The *ISO/IEC 25010-1, Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software quality models* outlines a model for software quality. This model references eight quality characteristics, which define the quality attributes of a test item. Testing is an activity that measures these quality characteristics in a given test item. The quality characteristics are;

- Functional Suitability: the degree to which the product provides functions that meet stated and implied needs when the product is used under specified conditions
- Performance Efficiency: the performance relative to the amount of resources used under stated conditions.

— Compatibility: the degree to which two or more systems or components can exchange information and/or perform their required functions while sharing the same hardware or software environment.

— Usability: the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

— Reliability: the degree to which a system or component performs specified functions under specified conditions for a specified period of time.

— Security: the degree to which information and data are protected so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them

— Maintainability: the degree of effectiveness and efficiency with which the product can be modified.

— Portability: the degree to which a system or component can be effectively and efficiently transferred from one hardware, software or other operational or usage environment to another.

In order to test a quality characteristic a test sub-process may need to be instantiated. For example planning for and executing testing to measure the security quality characteristic may require a security test sub-process to be implemented.

Each of these quality characteristics has a number of sub-characteristics that can be tested to provide an overall view of the characteristic's quality. It should also be remembered that not all quality characteristics are applicable to all systems, e.g. portability may not be important for a one-off embedded system. Note the above quality characteristics are not necessarily exhaustive and it may be appropriate to define further quality characteristics for a particular test item.

It is common among testing practitioners to refer to the testing of the functional quality characteristic as "functional testing" and to refer to the testing of the other quality characteristics as "non-functional" testing. Test types used to measure the quality characteristics other than functional suitability are typically referred to as non-functional test types and may include test types such as load testing, stress testing, penetration testing, and usability testing, etc.

When developing a Test Plan the test manager should consider all quality characteristics. The emphasis placed on the testing of each different quality characteristics and its sub-characteristics is likely to change depending on variables such as:

— The risk profile of the system being developed; for example, a safety-critical application may emphasise reliability; and/or

— The industry sector for which the system is being developed; for example, a banking application may emphasise security.

5.5.4 Test Basis

In this standard the term "Test Basis" is used for the body of knowledge (in whatever form) from which the requirements for a test item can be inferred. The nature of the test basis also varies over the phases of the development life cycle.

Examples of test basis may include:

— Expectations of format and contents of documentation, typically in the form of standards and/or checklists;

— Customer/user expectations of a software system, new or existing are typically in the form of written requirement specifications. These may be presented as functional/non-functional descriptions with "shall" statements, use cases, user stories or other forms of informally or formally written requirements. This may also include regulatory standards to be complied with for certain types of products, for example safety-critical software for the pharmaceutical industry or for transportation systems such as trains or aircraft;

— Expectations of direct and/or indirect interfaces between software system components and/or for co-existence of software system components, typically in the form of an architectural design such as diagrams and/or formally written protocol descriptions;

— Expectations of the implementation of software system components in code, typically in a form of detailed design.

Note that the same item may appear as a test item in one test sub-process, and as a test basis in another test sub-process e.g. a requirements specification may be the test item of a static test sub-process and the test basis of a system test sub-process.

Requirements can be classified into two main categories, namely:

— Functional requirements – specifying what the item should do aligning to the Functional Suitability quality characteristic outlined in ISO 25010; and

— Non-functional requirements – specifying *how* the functionality should present itself and behave and aligning to the other quality characteristics outlined in ISO 25010. Non-functional requirements are related to some or all of the functionality; and typically functional requirements are associated with appropriate non-functional requirements, either individually or in groups.

5.5.5 Retesting and Regression Testing

Retesting is testing to evaluate whether a solution to an incident found in testing has remedied the original issue. Retesting is often performed by rerunning the test case that produced the unexpected result however in order to retest effectively new test conditions may be identified, analysed and new test cases written.

Regression testing is selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its original requirements. The intent of regression testing is to ensure that when a defect is fixed that it has caused no further defects in unchanged parts of the system.

It is important when planning testing to consider both regression testing and retesting as both are usually required in order to complete testing for a sub-process. Time should be allowed in the test execution schedule for both activities.

5.5.6 Test Techniques

Test techniques exist for static testing and for dynamic testing. The purpose of a test technique is to assist testers in finding defects as effectively and efficiently as possible.

5.5.6.1 Static Test Techniques

For static testing, the test techniques provide support for the entire static test process. Static test techniques consist of static (code) analysis and reviews. Review techniques are defined in the IEEE 1028–2008, Standard for Software Reviews and Audits.

The main purpose of any of the static test techniques is to find defects, but they also have secondary purposes, for example: walkthroughs may be used for knowledge exchange, and technical reviews for gaining consensus. The type(s) of static test technique(s) to choose in any particular situation does not so much depend on the type of the test item, as on the risks involved (the higher the risk, the more formal static test technique is recommended), and the secondary purpose of the static test technique.

5.5.6.2 Dynamic Test Techniques

Test techniques for dynamic testing are techniques for identifying test conditions, test coverage items and subsequently test cases to be executed on a test item running on a computer. The dynamic test techniques are classified into three main categories based on how the test inputs are derived. These categories are

specification-based, structure-based and experience-based techniques. Part 4 of this standard describes each of the dynamic test techniques in more detail.

Specification-based test techniques are used to derive test cases from a test basis describing the expected behaviour of the test item. With these techniques both the test input part of the test case and the expected result are derived from the test basis. The choice of which specification-based test technique(s) to use in any particular situation depends on the nature of the test basis and on the risks involved. Specification-based test techniques covered in this standard are, in alphabetic order:

— Boundary Value Analysis

— Cause-Effect Graphing

— Classification Tree Method

— Combinatorial Test Techniques

— Decision Table Testing

— Equivalence Partitioning

— Random Testing

— Scenario Testing

— State Transition Testing

— Syntax Testing

Structure-based test techniques are used to derive test cases from a structural feature, for example the structure of the source code or a menu structure. With these techniques the test input part of a test case is derived from the structural feature and the expected result is derived from the test basis. The choice of which structure-based test technique(s) to use in any particular case depends on the nature of the test basis and on the risks involved. Structure-based test techniques covered in this standard are, in alphabetic order:

— Branch Testing

— Branch Condition Testing

— Branch Condition Combination Testing

— Data Flow Testing

— Decision Testing

— Modified Condition Decision Testing

— Statement Testing

In experience-based testing the basis for deriving test cases is the tester’s experience with similar test items or the test item itself rather than explicit descriptions of the test item. Alternatively, experiences may be documented and collated in checklists or other summary forms of descriptions. The choice of which experience-based test technique(s) to use in any particular situation depends on: the person(s) performing the test; the risks involved; familiarity with the test item; and the type of defects expected in the test item. Experience-based test techniques covered in this standard are, in alphabetic order:

— Error Guessing

— Exploratory Testing

Annex A
(informative)

The Role of Testing in Verification and Validation

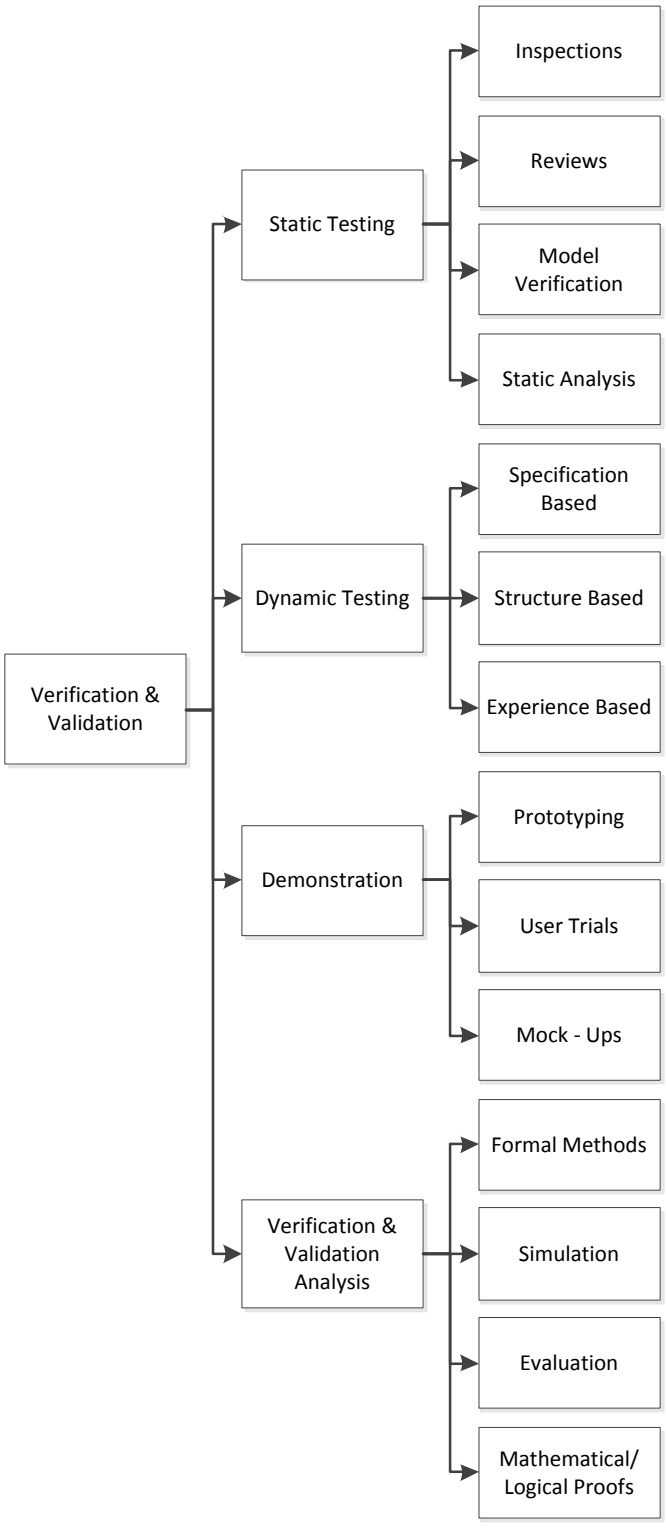

Figure 11 — Generic example test sub-process

Figure 11 defines the full nature of verification and validation (V&V) activities. The ISO/IEC 29119 standard addresses only the Dynamic and Static testing (fully or via reference) parts of the verification and validation model presented as well as associated sub boxes and elements, e.g. documentation, management, process.

Annex B
(informative)

Metrics and Measures

B.1 Metrics and Measures

The primary aim of testing is to provide information to help manage risks. To be able to do that it is necessary to define what information to provide and how to obtain and present it. It is not possible to assess, evaluate and improve something without measuring it. Thus all test efforts need to use and provide metrics and measurements. This section provides an introduction to this topic for testing.

B.1.1 Measuring in General

The provision of information is constrained by two factors:

— What data is available

— Time and effort available to perform measurement

The availability of data can be determined from raw data sources, such as documents, time registrations, and Incident Reports. Annex B.2 presents a list of raw data available in a typical testing project.

Determining what information is needed is based on the defined goals, and questions related to those, such as:

— what factors are constraining and defining the setting of the goals;

— what facts are needed to determine if a goal has been achieved; and

— if the goal has not been achieved, what has been achieved and what is still outstanding?

This information can be expressed as a metric that is a definition of what to measure including data type, scale, and component; for example:

C = percentage (as an integer) of requirements covered by the passing test procedures

The actual value of a metric at a given point in time is the measure. Measures can be extracted directly from the raw data, for example by counting the number of log sheets for passing test procedures and counting the number of incident reports. Most metrics are, however, expressed as relations between direct measures, and hence the value is calculated as an indirect metric combining two or more direct measures.

C above is an example of a metric whose value must be calculated as

C = (number of requirements covered by passing test procedures * 100) / total number of requirements

The total number of requirements and the total number of requirements covered by passing test procedures are the measures used to provide input into the metric.

A measure that can be obtained independently of human opinions is objective; when human opinions are involved the measure is subjective. Both types have value, if used with care. Annex B.3 presents a number of lists of test-related metrics.

Information gathering should be planned and documented, for example in the Project Test Plan and/or in test sub-process plans. This measurement plan should include definitions of metrics for which measures should be

obtained and how these measures should be obtained, as well as, most importantly, how the measures will be analysed, presented and used.

B.1.2 Planning Measuring

The following is a checklist to keep in mind when producing a measurement plan. Aim for:

— agreed metrics – definitions (for example, what is a line of code), scale (for example, is 1 highest or lowest), and components (for example, seconds or hours) should be agreed and understood

— required metrics – what is it you want to know, to monitor, and to control?

— repeatable measuring – same time of measuring and same instrument must give the same measure (within accepted limits of tolerance)

— precise measuring – valid scale and known source must be used

— comparable measures – for example over time or between sources

— economical metrics – practical to collect and analyse compared to the value of the analysis results

— using already existing measures – analyse the raw data in a different manner

B.1.3 Presenting the Measurements

Part 2 of this standard has a process for reporting and Part 3 of this standard has templates for reporting. Test status reports are used to communicate test progress. These reports should be tailored to the different recipients or stakeholders. The usual groups of stakeholders for test measures are the customer, the project and/or product manager (or higher), the test manager, and the testers.

The customer and the management above the test manager should be sent test completion reports when the tests defined by the associated test plan have completed, whether for a test sub-process or for the whole test project. The test manager needs information on a continuous basis from test sub-processes to keep in control. The testers will benefit from being kept informed on progress on a very regular basis, maybe even daily when the activities are at their peak.

In risk-based testing one of the important ways to report progress is in terms of eliminated risks. At any point in time it should be possible to show which risks are still open, and continue to present a threat to the system, and which are closed and no longer present any threat to the system.

One of the best ways to present progress information for testing is by using graphics. This holds true for all stakeholders. Graphics used in the right way give an immediate overview of the state of the testing. Examples of graphical expression methods are:

— S-Curves

— Pie Charts

— Check Sheets

— Statistical Reporting

B.1.4 Potential Issues with Metrics and Measures

Any measure and metric program needs to recognise that the process of measuring and generating metrics can cause issues in the project requiring the metrics. Some of these include:

— Validity: does the chosen measure/metric provide the information it is expected to provide?

— Dysfunction: what checks and balances are in place to prevent a measure/metric being falsified to ‘look good’?

— Resource Impact: does the team have the resources (i.e. time, personnel, budget) to effectively implement the measure and metric programme?

— Goal: does each measure and metric in the program have a goal and purpose within the organizational and/or project context?

The metrics listed below, and others, should be considered with these concerns and the local project context. It is likely that organizations and/or projects will not implement all of the measures and metrics listed; and those they do use will need to be tailored to their particular context.

B.2 Raw Data

Test work product sources available for obtaining test related measures include:

— Product risk log

— Budget

— The Project Test Plan with estimates

— The test sub-process plans with risks, estimates, and schedule

— Test specifications for each test sub-process including

  — test design

  — test cases

  — test procedures

  — traceability matrices

— Test execution logs

— Incident reports

— Completed time sheets

— Invoices for expenditures

Other work product sources available for obtaining test related measures from the project include:

— The project plan with risks, estimates, and schedule

— Requirements specifications

— Overall design specification

— Detailed design specification

— Source code

— Scripts

— Other specifications

B.3 Test-Related Metrics

Test related metrics can be divided into groups according to aspects of the test processes. The groups include:

— Metrics to aid planning

— Metrics for monitoring progress in general

— Metrics concerning coverage

— Metrics for test results

— Metrics related to confidence

— Metrics for test effectiveness and efficiency

B.3.1 Metrics to Aid Planning

Planning is supported by estimates of the time it will take to perform defined tasks. Estimations can be based on measures derived from risk analysis and test basis documentation, and on estimations of size of test work products. Planning metrics are quantitative.

Metrics from risk analysis and test basis may include:

— Number of risks

— Specification size (for example number of requirements, number of use cases, number of data entities, number of entity relationships, number of forms)

— Design size (for example number of components)

— Design complexity (for example number of nested levels, fan-in, fan-out, coupling, complexity of structure)

— Source code size (for example number of lines of code, number of comment lines, number of statements)

— Code complexity (for example McCabe, Halstead)

— Size of other documents to undergo static test (for example pages in a plan)

Estimated test related metrics may include:

— Size of test specification to be produced (for example number of test designs, number of test cases)

— Size of test data to be produced (for example number of data tables, number of entries in tables, number of concurrent users)

— Size and/or complexity of test environment(s) to be established (often qualitative)

People related metrics related to planning may include:

— Availability of people to perform tasks

— Effectiveness of people to perform the tasks

— Usage information (types of users, number of users, usage patterns)

The result of the planning process provides metrics including:

— Number of tasks

— Estimated time to complete tasks

— Estimated cost of completing tasks

— Estimated other expenditures

B.3.2 Metrics for Monitoring Progress

Monitoring progress requires information about how planned tasks are progressing and relating this to the estimates in the plan. Progress metrics are quantitative.

Metrics for task progress may include:

— Number of commenced tasks (for example test design, execution of test procedures, reporting)

— Number of tasks not able to be completed (for example blocked test procedures)

— Number of repeated tasks (for example execution of test procedures or test cases)

— Number of completed tasks

— Number of specified and approved test work products (for example test design, test cases, test procedures, test data, test environments)

— Actual size of test work products

Metrics concerning scheduling and cost may include:

— working hours spent

— elapsed time passed

— expenditure for people (for example fixed salary, time related cost)

— direct expenditure (for example purchase of tools, license fees)

The measures can be collected at various points in time and compared to the associated estimates to obtain measures related to progress over time, and to inform re-analysis of risks and re-planning.

B.3.3 Metrics Concerning Coverage

Coverage is the percentage of a specified coverage element that a given test is expected to exercise when it is being designed or have exercised when it has been run. Coverage is an indirect measure, and quantitative.

Coverage elements to define coverage metrics may include:

— Identified risks

— Business processes

— Requirements (in various forms such as statements or use cases)

— Equivalence partitions

— Boundary values

— Classification tree leaves

— Decision combinations

— Sequences of transitions

— Combination pairs

— Statements

— Decision outcomes

— Conditions

— Condition combinations

B.3.4 Metrics for Test Results

Test results are measured in terms of executed tests and observed incidents. Test result metrics are quantitative.

Metrics related to execution of tests may include:

— number of initiated test procedures

— number of passed test procedures

— number of failed test procedures

— number of halted test procedures

— number of blocked test procedures

— number of passed re-tests

— number of reopened incidents

— number of test procedures run as regression testing

The measures may be collected at specific points in time to obtain measure for progress over time.

Metrics related to incidents may include:

— number of reported incidents

— number of open incidents

— number of incidents ready for re-test

— number of closed incidents

— number of eliminated risks

Incidents may be categorized in a number of ways (for example by severity, by activity where the incident was registered, by activity where the associated defect was inserted, by source of the underlying defect, by complexity or size of the source) and metrics like the above can be defined for incidents in a specific category. Such metrics may include:

— number of incidents found during review of the requirements

— number of incidents found during component testing

— number of incidents where the source was the specification

— number of incidents where the type was a data problem

— incident distribution over test item areas

— incident density in relation to complexity

Incident measures may be collected at specific points in time along with other measures and metrics for defined rates. Such metrics may include:

— rate of executed test procedures

— rate of incidents found per time component

— rate of incidents per number of transactions

— incidents found per attempted test procedure

These metrics may be combined with the category metrics above.

B.3.5 Metrics Related to Confidence

Confidence is related to the maturity or reliability of a test item. Confidence may be measured prior to release or after release. Confidence metrics may be quantitative or qualitative.

Quantitative metrics related to confidence may include:

— Mean time between failures

— Failures per time component

— Failures per number of transactions

Specifically for web-based systems confidence metrics may include:

— Number of broken links

— Number of incomplete downloads

— Number of occurrences of deteriorating performance

— Number of orphaned files

Qualitative metrics related to confidence may include:

— Perceived levels of remaining risks

— Subjective statements about confidence from different stakeholders

B.3.6 Measures for Test Effectiveness and Efficiency

Measurements of effectiveness and efficiency of testing are necessary to be able to identify potential targets for process improvement.

Quantitative metrics related to effectiveness and efficiency may include:

— Defect Detection Percentage (for example for various test sub-process and for a number of periods)

— Return on the testing investment (for example in terms of saved cost of correcting defects found by the users in relation to cost of finding and correcting defects before release)

Qualitative metrics related to effectiveness and efficiency may include:

— Subjective statements from different stakeholders as to whether or not a test sub-process seems worthwhile

Annex C
(informative)

Testing in Different Life Cycle Models

This annex gives examples of how testing might fit into difference software development lifecycle models. For a given project, the necessary development phases and/or activities are identified and so is the way these are going to be performed in relation to each other during the course of the development life cycle. The set of development phases and their relationship is called the “development life cycle model” or, more simply, the “life cycle model”.

A number of life cycle models have been used by the software industry over the years. A sample of typical lifecycle models are classified here, in alphabetical order, though this is not a comprehensive classification of all development and test lifecycles.

— Agile

— Evolutionary

— Sequential (i.e. the waterfall model)

Incidentally, the above list shows the life cycle model categories in the reverse order of their coming into existence.

The development activities performed are more or less the same in the different categories of life cycle models; the main differences lie in the definition of their scopes, and the frequency with which they are repeated during the course of the development life cycle.

All development projects are performed in an organizational context. The organizational test specifications, typically in the form of an Organizational Test Policy and one or more Organizational Test Strategies, can be developed following the organizational test process.

An Organizational Test Policy in an organization covers all types of testing in an organization as it expresses the very top-level management’s expectations of the testing done in the organization, so its contents are independent of the development model(s) used in the organization.

C.1 Agile Development and Testing

Agile Development Principles

Agile development typically follows these basic principles:

— Incremental development – each cycle delivering useful and usable products;

— Iterative development – allowing requirements to evolve (i.e. change and be added to) as the project develops;

— People-oriented development – relying on the quality of the project team (i.e. developers and testers) rather than well-defined processes; allowing the agile team to manage themselves; expecting daily interaction between the development team and the business stakeholders; and

— Technical and engineering excellence – achieved through disciplined approaches to the development, integration and testing of products.

There are a number of Agile development methods and frameworks including: Extreme Programming (XP), Scrum, Crystal, Kanban and Feature-Driven Development. While they have different approaches they all share the basic Agile principles as expressed in the Agile Manifesto (see http//:agilemanifesto.org/). It is not possible to provide examples of this standard being implemented with each different Agile method therefore this standard will use the Scrum framework as an example. Scrum is not a development methodology (i.e. it does not tell you the best method for describing requirements or how to write code) and is better described as a project management framework within which an agile methodology is applied by the developers (XP is often used).

A Scrum project comprises a number of iterations called sprints, with each sprint resulting in new functionality that can be delivered to the customer (see figure 12). This new functionality may be enhancements to existing functionality or the introduction of new functionality. Each sprint typically lasts between one and four weeks. Often the number of sprints is unknown at the start because in typical agile projects the requirements are not fully understood at the beginning of the project. Requirements evolve as the project progresses. Customer requirements are collected in a Product Backlog, typically expressed as user stories.

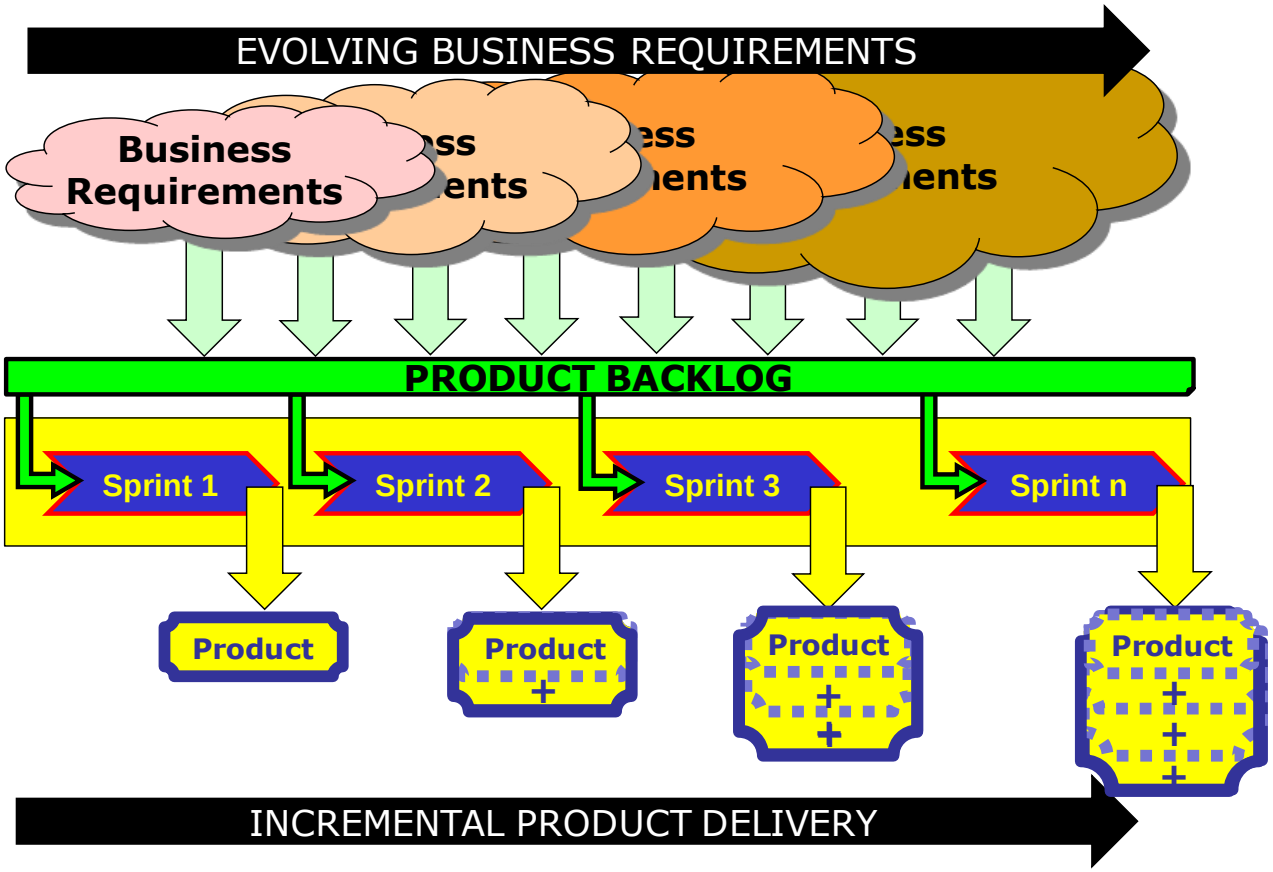

Figure 12 — Example Scrum (Agile) project lifecycle

The testing process model defined in this standard can be applied to the testing for development following an agile development model.

Test Management in Agile Development

The Organizational Test Strategy should reflect the fact that the development follows an agile development model. In an organization where development is performed using a mixture of development models on different projects including agile, there will typically be a specific Organizational Test Strategy covering the agile development. The Organizational Test Strategy for projects using agile development should use the specific agile vocabulary, including the concepts of backlogs, sprints, and daily scrums, but apart from that the contents of any Organizational Test Strategy depends on the risk profile for the projects and products it covers, and not on the development model used.

An agile project is often managed by a project manager, and the sprints are facilitated by a scrum master (these roles may be carried out by the same person). The test management in an agile project is performed as an integrated part of the management of the product backlog, the individual sprints, and the daily scrums.

At the start of the Sprint, the Scrum team and the customer (Product Owner) agrees which user stories from the Product Backlog should be implemented in this sprint. The selected stories comprise the Sprint Backlog. The team then plans the sprint by scheduling the development and test activities and assigning the team members' roles and responsibilities. Agile follows the same generic process across all product development and testing as can be seen in an example Scrum sprint in figure 13 below:

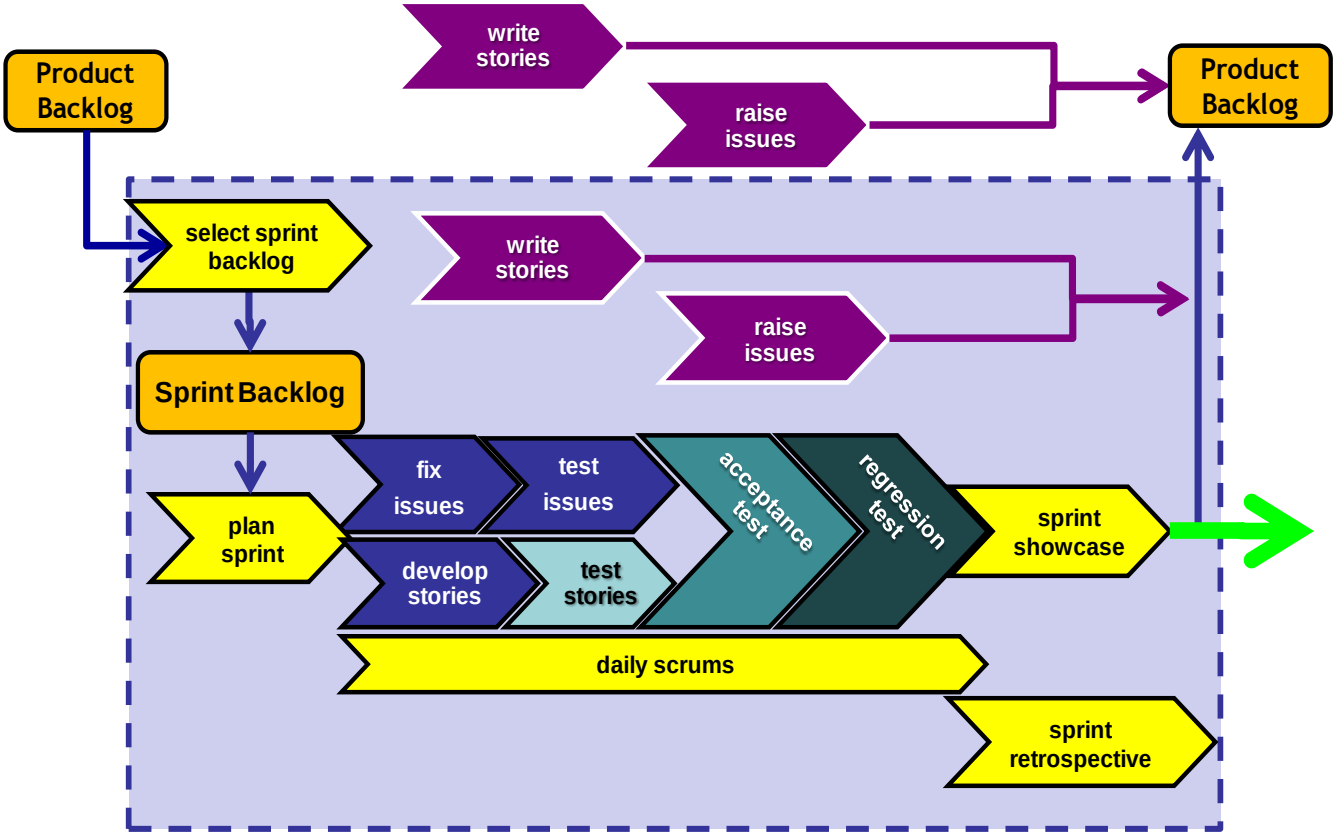

Figure 13 — Example Agile Sprint lifecycle

The deliverable from the sprint is demonstrated to the customer at the sprint showcase meeting where the team have the opportunity to show stakeholders what they have created. The final activity in the sprint is the sprint retrospective in which the team review how well they performed and identify where improvements could be made for future sprints – thus process improvement is built into the Scrum framework.

Throughout the sprint daily stand-up scrum meetings are held at the start of each day to ensure that the whole team is aware of what is being done that day. They are also for the scrum master to determine what impediments need to be removed to allow the team to progress most efficiently.

Key considerations in an agile project are managing the risk of regression defects (as each sprint builds on the previous ones), and managing the changing nature of requirements and their impact on test artefacts. Typically test automation is used to manage the regression risk and exploratory testing may be used to manage the impact of poorly-documented requirements.

Test Sub-processes in Agile Development

Ideally test activity is an integrated part of an agile development project, as shown in figure 13, with testing being performed on an on-going basis throughout the sprint. Test sub-processes can be defined and performed using the processes in this standard for testing the stories and the evolving system being delivered.

Typical test practices used by the sprint team are:

Test Driven Development (TDD); this is where tests for the code are written in advance of the code being written. The tests are based on the user story and may be developed by the tester and developer working together. These tests are typically implemented using automated component testing tools and this can result in TDD being seen as a form of programming.

Automated builds and continuous integration; this is where the system is continuously updated and regression tested as code is checked in. It is used to ensure the timely identification and correction of integration and regression issues.

System testing of all quality characteristics (i.e. both functional and non-functional) is performed against both the users stories and any high level requirements that exist. System testing is typically followed by acceptance testing which should involve end users ensuring the delivered functionality meets their needs.

Regression testing is generally required to ensure that any changes in the current sprint have had no adverse side effects on the existing functions and features of the product.

At the end of an 'ideal' sprint the features are ready for use by the users meaning that all the aforementioned testing is performed within the sprint (as shown in figure 13). In practice many projects find this difficult which leads to the adoption of compromise alternatives, such as testing being performed as a parallel but offset activity or testing being handled in an occasional testing focussed sprint.

C.2 Evolutionary Development and Testing

Evolutionary Development Principles

Evolutionary development is based on the two basic principles of iteration and incremental delivery. Iteration allows developers to develop the system as a series of smaller pieces and they can then use feedback on both the developed product and their development practices to improve the next iteration. Iteration allows risks to be addressed earlier than with traditional sequential cycles, thus providing increased time to address them. With incremental development the results of each iteration are released to the customer, meaning that the users receive the system in a series of deliveries with increasing functionality.

An iteration consists of all or some of the standard development activities. An iteration will include an acceptance phase if the result of the iteration is being delivered to the users; otherwise an acceptance phase will typically only be performed in the last iteration.

The testing process model defined in this standard can be applied to the testing for development following an evolutionary development model.

Test Management in Evolutionary Development

The Organizational Test Strategy should reflect the fact that the development follows an evolutionary development model. In an organization where development is performed using a mixture of development models on different projects including evolutionary, there will typically be one or more specific organizational test strategies covering the used development model(s). The Organizational Test Strategy should use the vocabulary used by the project type it covers, but apart from that the contents of any Organizational Test Strategy depend on the risk profile for the projects and products it covers, and not on the development model used.

An evolutionary project is managed by a project manager. On most projects the roles of development manager and test manager are also defined. Depending on the size of the project these roles may be carried out by different people, or two or all of the roles may be carried out by the same person.

Plans in evolutionary development are produced for the entire project and for each iteration. A Project Test Plan should also be produced, and may either be in the form of an individual document or as part of the overall project plan. A smaller iteration Test Plan may also be produced specifying the testing to be performed in an iteration. Plans may be more or less formally documented in evolutionary development, but they will usually be maintained in a document and/or in a planning tool. The iteration Test Plans and the related sub-process Test Plans will often be reused with necessary corrections from iteration to iteration.

The test progress is monitored during an iteration and necessary corrective actions decided at planned or ad hoc points in time and made available to the stakeholders by updating the relevant Test Plan(s).

Test Sub-processes in Evolutionary Development

In each iteration the work products and the completed system can be tested. Test sub-processes can be defined and performed using the processes in this standard for testing the work products and the system being produced. Note that the names of the test sub-processes used here are only examples.

Figure 14 shows a few of the test sub-processes that can be defined for one iteration in evolutionary development. Note that the iteration depicted in figure 14 includes acceptance (and possible delivery) of the system at the end of the iteration.

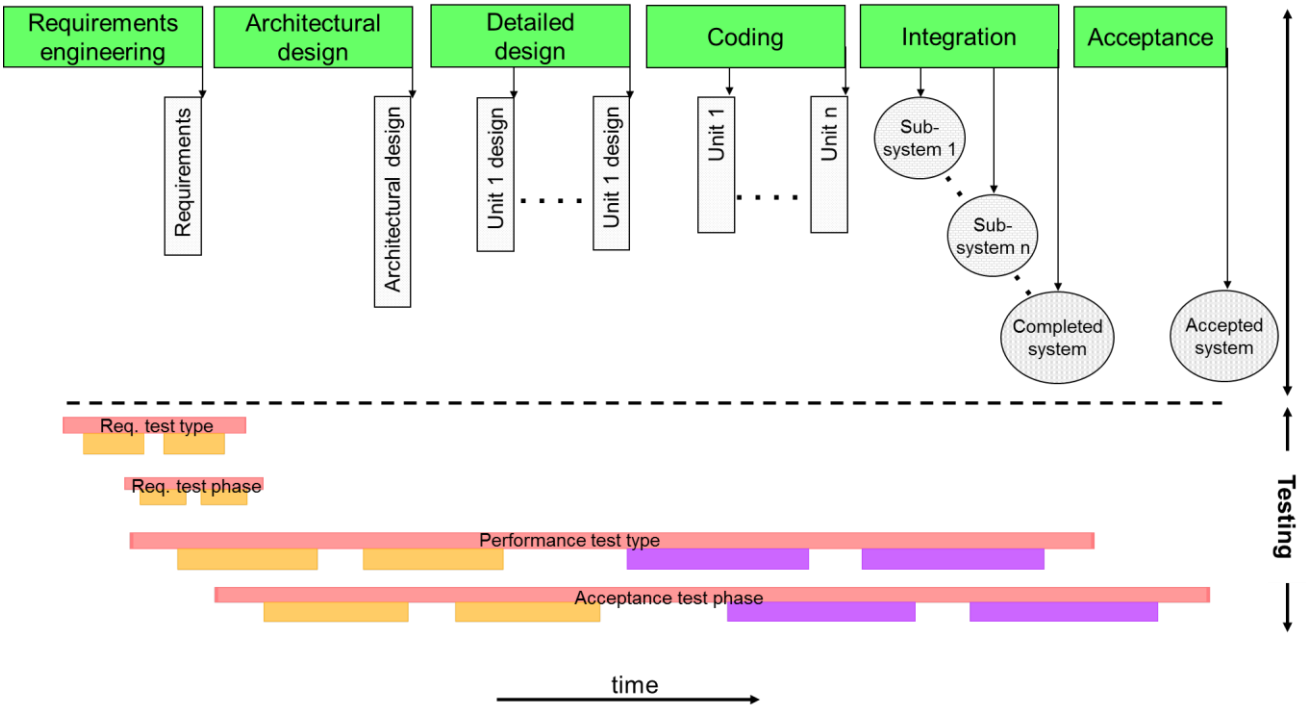

Figure 14 — Example test sub-processes in an Evolutionary iteration

The first development phase in this example is requirements engineering, where the system requirements are being specified. Figure 14 only shows the test sub-process related to the first phase (Req. test phase) to avoid cluttering the picture. The testing of the requirements as they are being specified can be considered as a sub-process consisting of static tests. The formality of this test sub-process depends on the risk profile for the product, but since the requirements are the foundation for the work in the iteration, the static test techniques should be chosen from the more formal end of the scale.

Similar test sub-processes can be defined for the two design phases, as well as for the coding phase. For the development model example shown in figure 14 this might include:

— Architectural design test sub-process;

— Detailed design test sub-process;

— Source code test sub-process.

These test sub-processes usually stretch over most of the corresponding development phase, and they concentrate on one type of test item, for example requirements, and include different ways of testing the test item, for example walkthroughs and technical reviews. The requirements test sub-process will not be completed until all developed requirements have been tested according to the test sub-process plan. There is usually a more or less formal milestone to mark the conclusion of the requirements engineering phase, and this will usually depend on the result of the requirements test sub-process.

Similarly the design and source code test sub-processes will not be completed until all developed test items have been tested according to the test sub-process plan, and the corresponding development milestones will depend on the results from the test sub-process.

The acceptance test sub-process is shown in figure 14. This test sub-process starts during the requirements engineering phase because the set of requirements is the test basis. The planning and preparation activities in the acceptance test sub-process can start as soon as the risk of the requirements (for this iteration) changing significantly is lower than the benefit of starting the test sub-process. Test design will unveil defects in the requirements and, in this way, contribute to providing information about the requirements. Similar test sub-processes can be defined for other development milestones where dynamic test is involved. For the development model example shown in figure 13 this might include the following sub-processes, which are similar to the acceptance sub-process:

— Component test sub-process;

— Integration test sub-process;

— System test sub-process.

In the example in figure 14 a performance test sub-process has also been defined. Specific test sub-processes may be defined to cover specific categories of requirements or specific areas of the system described by the requirements, usually depending on the risk profile for the system. Such specific test sub-process may stretch over a number of development phases, and hence may have a variety of test items and associated test bases, though there is only one focus of the test sub-process, in this case performance requirements and how they are being described, designed, and implemented in the system.

All the testing described above can be performed for each subsequent iteration.

Since the product is constantly expanding, extensive regression testing of what has previously been accepted is imperative in each iteration. A regression test sub-process should be defined for each iteration, except the first. The regression test sub-process may include regression testing of all the items being expanded in an iteration, including the requirements, the design, the source code, and the system, or it may cover only a selection of these depending on the risk profile. Separate regression test sub-processes may also be defined for each type of item being expanded to facilitate more flexibility in the regression testing from iteration to iteration.

C.3 Sequential Development and Testing

Sequential Development Principles

Historically, sequential life cycle models have been around the longest and are still widely used. The basic (and original) sequential model is known as the waterfall model, and is depicted by development phases ordered in sequence preceding a testing phase, with a final operations phase at the end.

A sequential life cycle model is characterized by including no explicit repetition of the phases other than those enforced as absolutely necessary by feedback from subsequent phases.

The testing process model defined in this standard can be applied to the testing for development following a sequential life cycle model.

Test Management in Sequential Development

The Organizational Test Strategy should reflect the fact that the development follows a sequential development model. In an organization where development is done using a mixture of development models on different projects including sequential, there will typically be one or more specific organizational test strategies covering the used development model(s). The Organizational Test Strategy should use the vocabulary used by the project type it covers; otherwise the content of an Organizational Test Strategy depends on the risk profile for the projects and products it covers, and not on the development model used.

A sequential project is managed by a project manager. On most projects the roles of development manager and test manager are also defined. Depending on the size of the project these roles may be carried by different people, or two or all of the roles may be carried out by the same person.

Plans in sequential development are produced for the entire project, though they should develop during the course of the project. Sequential development projects may be relatively large, and it may not be possible to plan at the same level of detail for the entire project at the start. A project Test Plan should also be produced. This is usually in the form of an individual document, but may be part of the overall project plan. Individual test sub-process plans may be produced if warrented for the test sub-processes specified to be performed in the project Test Plan. Plans are usually formally documented in sequential development.

The test progress is monitored during the course of the project and necessary corrective actions decided at a planned time, usually the beginning of a new development or testing phase, or ad hoc points in time, and made available to the stakeholders by updating the relevant Test Plan(s).

Test Sub-processes in Sequential Development

The test sub-processes described for evolutionary development in annex C 2 are also relevant for testing in a sequential project, although they are only performed once (there is only one iteration in a sequential development model).

A few examples of test sub-processes not included in annex C 2 are shown in figure 15.

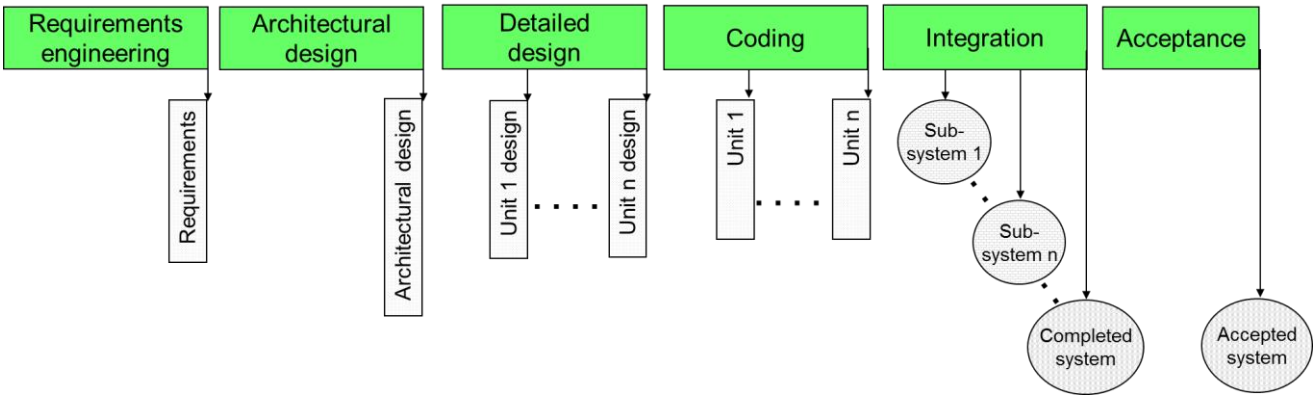

Figure 15 — Test sub-process examples in Sequential Development life cycle

Note that figure 15 is not to scale; the relative size of the test sub-processes shown is not related to the actual size of the test sub-processes in terms of, for example, elapsed time, effort, or cost.

Architectural Design, Detailed Design and Source Code Test sub-processes are defined. For each of these the corresponding development phase may be signed off on the basis of the results of the completed test sub-process, i.e. when all the detailed developed items have been tested individually.

For the coding phase there are two different test sub-process. The first is a source code test sub-process, constituted of static testing of the source code, the second is a component test sub-process, constituted of

dynamic testing of the executable or interpretable code. These may be combined, and the dynamic testing of a component made dependent on the successful static testing of the source code.

The ultimate result of the integration phase is a completed system. At this point the execution activities for the system test, that has been planned and prepared on the basis of the requirements, can start.

The example test sub-processes shown here are further described in annex E.

Annex D
(informative)

The Hierarchy of Test Documentation

This Annex provides two figures that demonstrate the hierarchy of test documentation described in ISO/IEC 29119-3 Test Documentation.

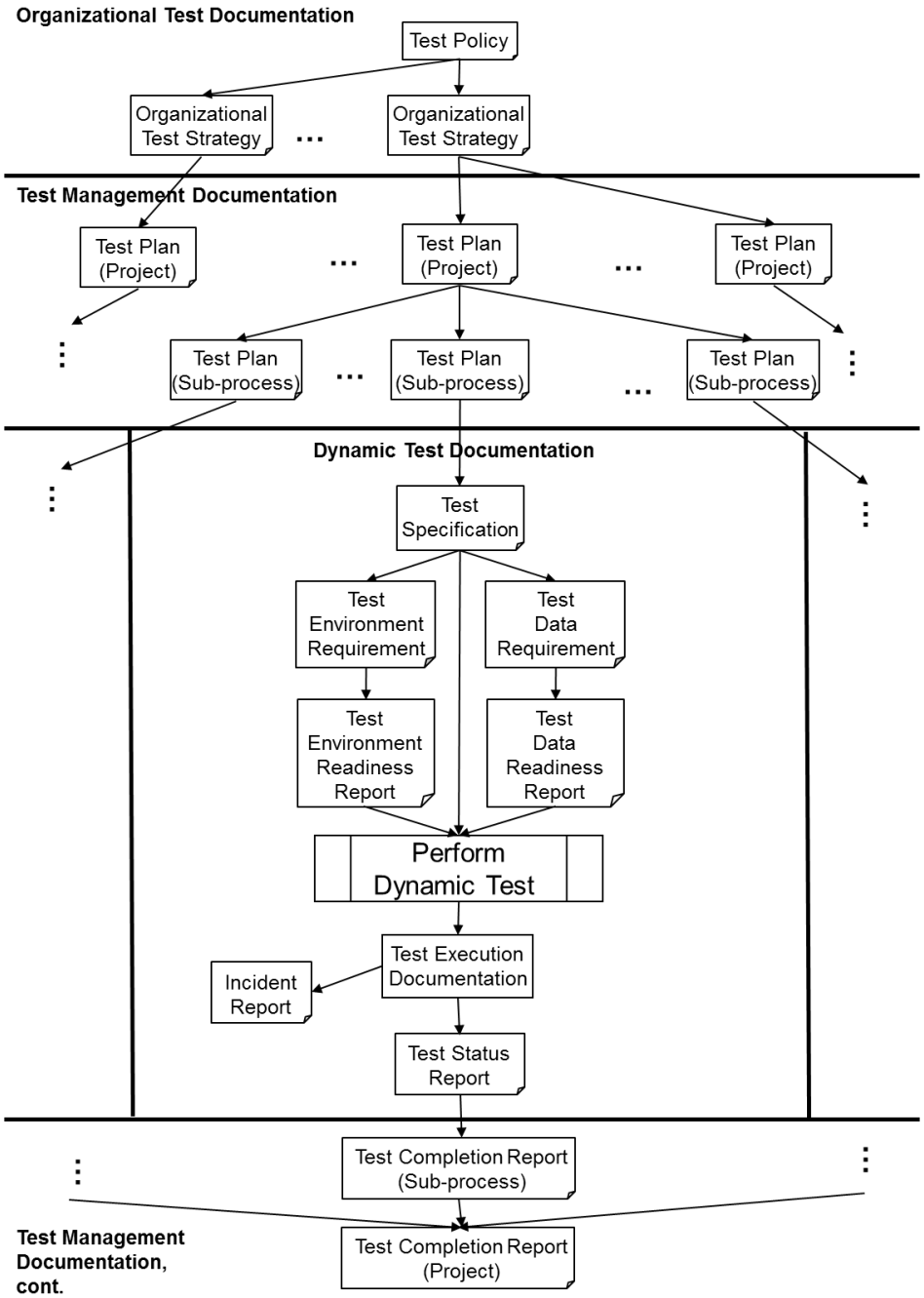

Figure 16 — The hierarchy of test documentation

Figure 16 shows that context for test documentation is set by the Organisational Test Policy. The Dynamic Test documentation is developed within the context of the test management documentation for a particular project.

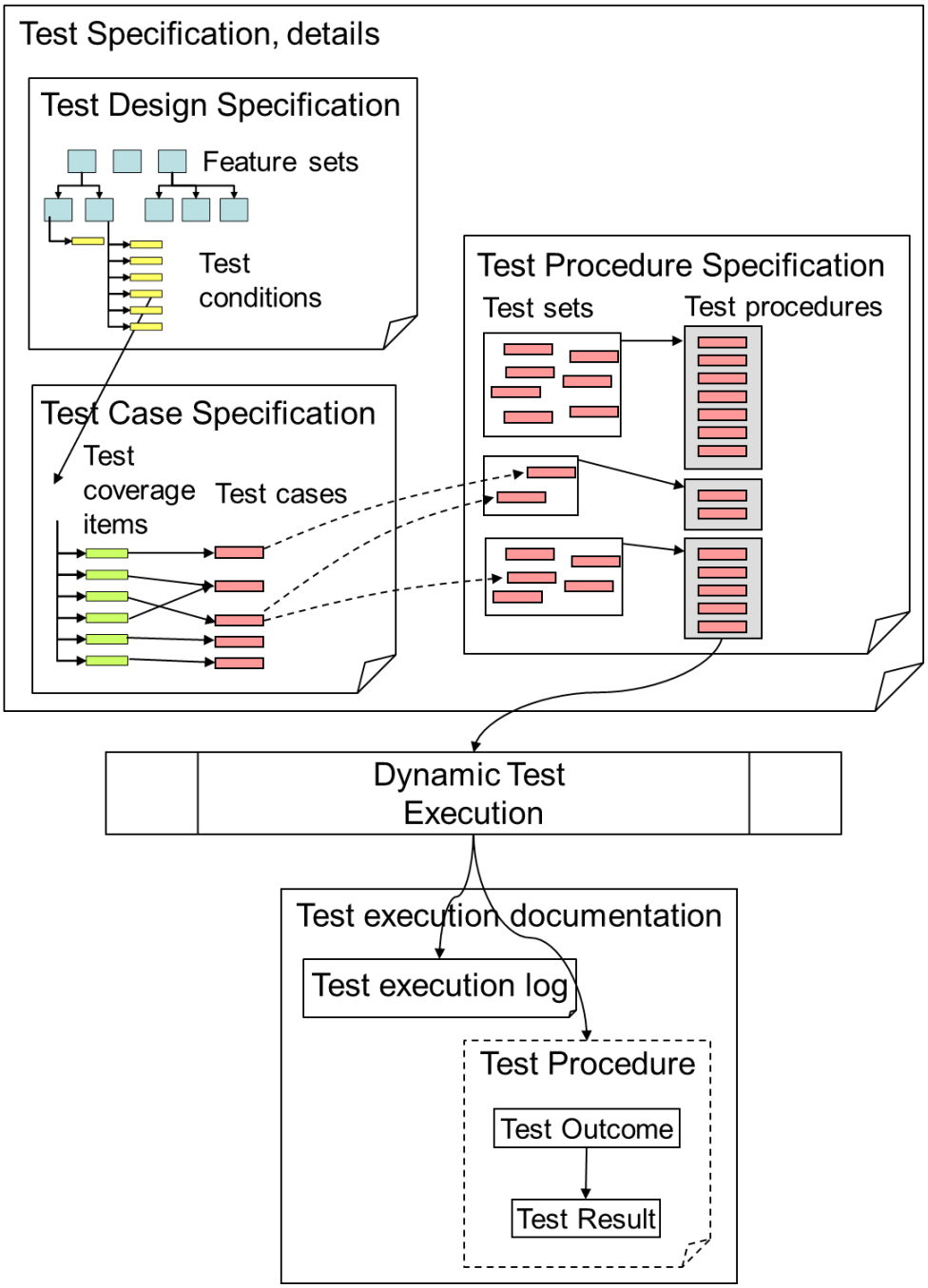

Figure 17 — The hierarchy of test design and implementation documentation

Figure 17 shows hierarchy between the documents produced in completing the *Test Design and Implementation* process outlined in Part 2.

Annex E
(informative)

Detailed Test Sub-process Examples

This annex outlines examples of test sub-processes. The examples are provided in alphabetic order. This list only shows a few examples; many other specific sub-processes may be needed in a specific test project.

The examples are:

— Acceptance test sub-process;

— Architectural design test sub-process;

— Coding test sub-process;

— Detailed design test sub-process;

— Integration test sub-process;

— Performance test sub-process;

— Regression test sub-process;

— Requirements test sub-process;

— Requirements validation test sub-process;

— Retest test sub-process;

— Showcase test sub-process;

— Story acceptance test sub-process;

— Story set test sub-process;

— Story test sub-process;

— System test sub-process;

— Component test sub-process.

Note that a regression test sub-process and a retest test sub-process have been included as specific test sub-processes. These may be included in any of the other test sub-processes as appropriate.

For each example test sub-process the description includes:

— Test sub-process objective;

— Planned testing sub-process content – the static and/or dynamic test processes to be performed; and

for each static or dynamic test process planned for the test sub-process:

— Test objective;

— Test item;

— Test basis;

— Applicable test processes;

— Suggested test technique(s), if applicable.

Note that the examples provided are examples only. In each actual situation the choices should be made in accordance with the risk profile for the product.

E.1 Acceptance Test Sub-process

This example represents the test sub-process associated with the acceptance phase of the development life cycle.

Test sub-process objective:	To present the final system to support acceptance
Planned testing sub-process content:	Dynamic test 1: 'Dress rehearsal', Dynamic test 2: Presentation

Dynamic test 1: 'Dress rehearsal'

Test objective:	To ensure that the final execution in the presence of the customer will be successful.
Test item:	The completed system
Test basis:	The user requirements
Dynamic test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.
Test techniques(s):	Use case testing, other techniques depending on the nature of the requirements.

The design and implementation of test cases may start as soon as the user requirements are stable. Though the assumption for the acceptance test is that the system works, most organizations prepare and execute the tests before the customer is present to witness the official presentation of the system; this is known as the 'dress rehearsal'. Retest and regression test processes may be planned to cater for any 'last minute' defect removal as a result of this test.

Dynamic test 2: Presentation

Test objective:	To present the completed system to the customer.
Test item:	The completed system
Test basis:	n/a
Dynamic test processes:	Test environment set-up and maintenance; test execution; and test incident reporting.
Test techniques(s):	n/a

The environment will need to be reset after the final 'dress rehearsal', but otherwise the test execution is performed according to the procedures and on the environment prepared in the 'dress rehearsal'.

E.2 Architectural Design Test Sub-process

This example represents the test sub-process associated with the architectural design phase in the development life cycle.

Test sub-process objective:	To provide information about the quality of the architectural design.

Planned test sub-process content: A number of static tests can be included depending on how formal the testing of the architectural design should be. Only a few are included here:
Static test 1: Architectural design documentation,
Static test 2: Architectural design quality attributes,
Static test 3: Architectural design usability, (not usability of the system!),
Static test 4: Architectural design completeness.

The static tests planned for this test sub-process can be performed during the course of the architectural design phase as the design is produced.

Static test 1: Architectural design documentation

Test objective: To provide information about the way the architectural design is documented

Test item: Architectural design (or selected parts)

Test basis: Internal and/or external checklists, for example regarding a specific architectural design model.

Test techniques(s): Technical review or inspection

The architectural design may be expressed using a variety of methods and/or models, and this test can be defined for each type using for example different checklists depending on the documentation method or model.

Static test 2: Architectural design quality attributes

Test objective: To provide information about the architectural design's compliance to design rules

Test item: Architectural design

Test basis: Checklist related to applicable design rules, for example regarding reuse, maintainability, and error handling

Test techniques(s): Technical review

Static test 3: Architectural design usability

Test objective: To provide information about the usefulness of the architectural design regarding for example detailed design or test

Test item: Architectural design

Test basis: n/a

Test techniques(s): Walkthrough for example with either programmers or testers

Static test 4: Architectural design completeness

Test objective: To provide information about the completeness of the architectural design

Test item: Architectural design

Test basis: Checklists and/or traceability information for requirements

Test techniques(s): Technical review

E.3 Coding Test Sub-process

This example represents the test sub-process associated with the coding phase of the development life cycle.

Test sub-process objective:	To provide information about the quality of the source code
Planned test sub-process content:	Static test: Source code quality attributes

In the coding phase several source code items are developed, depending on the architectural design. Each of these can be subjected to the static test defined for this test sub-process, thus an instance of the static test can be planned with the test items defined as a specific piece of source code. The coding test sub-process is only completed when all planned tests are completed (or abandoned as the case may be).

Static test: Source code quality attributes

Test objective:	To provide information about the quality of a specific piece of source code
Test item:	Specific piece of source code
Test basis:	Internal and/or external checklists, for example regarding a specific code writing style, and/or coding abnormalities such as use of a variable before it is declared.
Detailed test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.
Test techniques(s):	Technical review or static analysis.

E.4 Detailed Design Test Sub-process

This example represents the test sub-process associated with the detailed design phase of the development life cycle.

Test sub-process objective:	To provide information about the quality of the detailed design
Planned test sub-process content:	Static test 1: Detailed design item documentation Static test 2: Detailed design item usability, (not usability of the system!), Static test 3: Detailed design item completeness.

In the detailed design phase several design items are developed, depending on the architectural design. Each of these can be subjected to the static tests defined for this test sub-process, so instances of the static test can be planned with the test items defined as a specific detailed design item. The detailed design test sub-process is only completed when all planned tests are completed (or abandoned as the case may be).

Static test 1: Detailed design item documentation

Test objective:	To provide information about the way a detailed design item is documented
Test item:	A detailed design item
Test basis:	Internal and/or external checklists specifying detailed design documentation rules.
Test techniques(s):	Technical review or inspection.

Static test 2: Detailed design item usability

Test objective:	To provide information about the usefulness of a detailed design item regarding for example coding or test
Test item:	A detailed design item
Test basis:	n/a
Test techniques(s):	Walkthrough for example with either programmers or testers

Static test 3: Detailed design item completeness

Test objective:	To provide information about the completeness of a detailed design item
Test item:	A detailed design item
Test basis:	Traceability information for higher level design and/or requirements
Test techniques(s):	Technical review

E.5 Integration Test Sub-process

This example represents the test sub-process associated with the integration phase in the development life cycle, where (tested) components are gradually being integrated.

During the integration phase a number of similar test items are produced, that is two components being integrated. The dynamic tests in this example are generic and can be performed for any two components being integrated, from the very first two components being integrated into a component and until the last two components are integrated into the complete system. The integration test sub-process is only completed when all planned tests are completed (or abandoned as the case may be).

Integration may take place at different levels. It may be integration of a source code component with a component into a (larger) component ending with a complete system, different types of sub-systems (hardware, software, data, training material etc.) being integrated into one complete system, or complete systems being integrated into a (larger) component ending with a system of systems. The principles are the same, though the formality depends on the risk profile.

Test sub-process objective:	To provide information about the interaction of integrated components.
Planned test sub-process content:	Static test: Direct interface, Dynamic test 1: Direct interface, Dynamic test 2: Indirect interface, Dynamic test 3: Co-existence

Static test: Direct interface

Test objective:	To provide information about direct interface between two integrated components, for example in the form of a parameter list
Test item:	The source code for the interface between the components being integrated
Test basis:	Architectural design
Detailed test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.
Test techniques(s):	Technical review or inspection depending on the risk profile

Dynamic test 1: Direct interface

Test objective:	To provide information about direct interface between two integrated components, for example in the form of a parameter list
Test item:	The interface between the components being integrated
Test basis:	Architectural design
Detailed test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.

Test techniques(s): techniques as appropriate

Dynamic test 2: Indirect interface

Test objective: To provide information about indirect interface between two integrated components, for example through a database

Test item: The integrated component

Test basis: Architectural design

Detailed test processes: Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting. It may be possible to reuse test procedures from earlier (i.e. component) sub-process, if this is the case test design and implementation can be minimised or omitted.

Test techniques(s): As appropriate

Dynamic test 3: Co-existence

Test objective: To provide information about co-existence of an integrated component (or complete system) with other existing systems in the environment

Test item: The integrated component (or complete system) and existing systems in the environment

Test basis: Architectural design

Detailed test processes: Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting. It may be possible to reused test procedures from other tests minimizing the need for test design and implementation.

Test techniques(s): As appropriate, and possibly supplemented with experience based and/or exploratory test

E.6 Performance Test Sub-process

This example represents a test sub-process focusing on the performance of the system.

Test sub-process objective: To provide information concerning the fulfilment of performance requirements for the system.

Planned testing sub-process content: Static test 1: Performance requirements documentation,
Static test 2: Performance requirements completeness,
Static test 3: Architectural design with regard to performance,
Static test 4: Detailed design with regard to performance,
Dynamic test 1: Applicable sub-system with regard to performance,
Dynamic test 2: The completed system with regard to performance.

This test sub-process is not connected to a specific phase in the development life cycle. The tests can be planned to be performed as the applicable test items are being developed. For the static test the preparation can start as soon as the development of the test item is planned and the examination and follow-up can take place as soon as the test item is declared ready for static testing. For the dynamic tests the design and implementation can start as soon as the test basis is stable, and the execution can be performed as soon as the test item is declared ready.

This clause is an example of possible test sub-processes regarding quality attributes. Similar test sub-processes may be defined for example for functionality, operability, and portability.

Static test 1: Performance requirements documentation

Test objective:	To provide information about the quality of the performance requirements
Test item:	The set of performance requirements
Test basis:	Internal and/or external checklists for documentation of performance requirements, for example regarding testability.
Test techniques(s):	Technical review or inspection (remember that an inspection should be preceded by an informal or technical review to ensure a certain maturity of the requirements to be inspected).

Static test 2: Performance requirements completeness

Test objective:	To provide information about the completeness of the performance requirements (all functional requirements should have associated performance requirement(s))
Test item:	All requirements
Test basis:	Vertical traceability information between functional requirements and performance requirements
Test techniques(s):	Technical review

Static test 3: Architectural design with regard to performance

Test objective:	To provide information about how the performance requirements have been incorporated in the architectural design
Test item:	Architectural design
Test basis:	Performance requirements and relevant checklists
Test techniques(s):	Walkthrough, technical review, or inspection (remember that an inspection should be preceded by an informal or technical review to ensure a certain maturity of the requirements to be inspected).

Static test 4: Detailed design with regard to performance

Test objective:	To provide information about how the performance requirements have been incorporated in the detailed design
Test item:	One or more detailed design items as appropriate
Test basis:	Performance requirements and relevant checklists
Test techniques(s):	Walkthrough, technical review, or inspection (remember that an inspection should be preceded by an informal or technical review to ensure a certain maturity of the requirements to be inspected).

Dynamic test 1: Applicable sub-system with regard to performance

Test objective:	To provide information about the performance of a specific sub-system
Test item:	One or more relevant sub-systems
Test basis:	Performance requirements and possibly identified performance risks

Detailed test processes: Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting. Some test procedures for integration testing may be reused

Test techniques(s): Applicable techniques

This or these tests are usually performed during the integration test sub-process.

Dynamic test 2: The completed system with regard to performance

Test objective: To provide information about the performance of the completely integrated system

Test item: The completed system

Test basis: Performance requirements and possibly identified performance risks

Detailed test processes: Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.Some test procedures for system test may be reused

Test techniques(s): Applicable techniques

This test is usually performed during the system test sub-process.

E.7 Regression Test Sub-process

This example represents a generic test sub-process to be performed following an implemented change in an item related to the test item and/or a change to the environment in which the test item is running. The test sub-process is to be used for a test item having previously passed one or more tests, and assessed to be unaffected by the implemented change(s).

The test sub-process can be performed as part of any other test sub-process and on any test item. The selection of the test item depends on the nature of the change(s) and the risk profile. The number of regression tests necessary for a test sub-process depends on the initial quality of the test item and the test completion criteria.

Test sub-process objective: To provide information about possible effects on an unchanged test item following a change in a related item, items, or in the environment.

Planned test sub-process content: Re-execution of previous static examination(s) or executed dynamic test(s) as appropriate.

Regression test:

Test objective: To provide information about the quality of the changed test item on the unchanged test items.

Test item: The test item in question

Test basis: As appropriate

Detailed test processes: Test environment set-up and maintenance; test execution; and test incident reporting. Test design and implementation is not required, since this test is performed using selected test procedures that have already passed.

Test techniques(s): As appropriate

E.8 Requirements Test Sub-process

This example represents the test sub-process associated with a requirements development phase in the development life cycle.

Requirements may be developed at different levels, for example business requirements, user requirements, and system requirements. The principles for the testing are the same, though the formality depends on the risk profile.

Test sub-process objective:	To provide information about the quality of the requirements
Planned test sub-process content:	A number of static tests can be included depending on how formal the testing of the requirements should be. Only a few are included here: Static test 1: Requirements documentation, Static test 2: Requirements usability, (not usability of the system!) Static test 3: Requirements completeness.

The static tests planned for this test sub-process can be performed during the course of the requirements development phase as the requirements are produced. Some of the suggested static tests may be repeated several times for different requirements or different groups of requirements. The requirements test sub-process is only completed when all planned tests are completed (or abandoned as the case may be).

Static test 1: Requirements documentation

Test objective:	To provide information about the way the requirements are documented
Test item:	Selected requirements (either all or a group)
Test basis:	Internal and/or external checklists, for example regarding a specific requirements documentation style, and/ or regarding associated information such as unique identification, priority, and originator.
Test techniques(s):	Technical review or inspection (remember that an inspection should be preceded by an informal or technical review to ensure a certain maturity of the requirements to be inspected).

Static test 2: Requirements usability

Test objective:	To provide information about the usefulness of requirements regarding design or test
Test item:	Selected requirements (either all or a group)
Test basis:	n/a
Test techniques(s):	Walkthrough for example with either designers or testers

Static test 3: Requirements completeness

Test objective:	To provide information about the completeness of a set of requirements
Test item:	Selected requirements (either all or a group)
Test basis:	Checklists and/or traceability information for higher level requirements
Test techniques(s):	Technical review

E.9 Requirements Validation Test Sub-process

This example represents a test sub-process related to validating if the requirements express what the users need (fit for purpose).

This test sub-process is performed to investigate if the future users are able to use the specified system to solve their business objectives. The specification may for example be in the form of requirements concerning the business flows and/or the user interface. The results of performing this test sub-process may give inspiration and/or feed-back to requirements (for example operability requirements) and could be performed a number of times during the course of the requirements development phase.

Test sub-process objective:	To provide information about the requirements in relation to the users' needs.
Planned test sub-process content:	Static test 1: Operability assessment Static test 2: Work flow assessment

Static test 1: Operability assessment

Test objective:	To provide information about the operability of a proposed user interface
Test item:	Described user interface, for example in the form of a prototype
Test basis:	n/a
Test techniques(s):	Operability assessment

Static test 2: Work flow assessment

Test objective:	To provide information about the suitability of the proposed work flows to the business processes
Test item:	System requirements specification, for example in the form of use cases
Test basis:	n/a
Test techniques(s):	Role play, dry run

E.10 Retest Test Sub-process

This example represents a generic test sub-process to be performed following an implemented change in an item caused by a defect found in a previous test execution, i.e. for a test item having previously failed a test.

The test sub-process can be performed as part of any other test sub-process or test sub-process and on any test item.

Test sub-process objective:	To provide information about a defect being reported as removed.
Planned test sub-process content:	Re-execution of previously failed static examination or executed dynamic test as appropriate.

Retest test:

Test objective:	To provide information about the quality of the changed test item
Test item:	The test item in question
Test basis:	As appropriate
Detailed test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting. Test design and implementation are

not required since this test is performed using test procedures that have previously failed.

Test techniques(s): As appropriate

E.11 Showcase Test Sub-process

This example represents the presentation of a sprint result in the form of a new version of the system with implemented stories, corrections, and/or changes to the end users at the showcase meeting. This is where the end users and the developers agree if the new system can be delivered and the sprint signed-off.

Test sub-process objective: Determine quality of new system

Planned test sub-process content: Dynamic test: Acceptance.

Dynamic test: Acceptance

Test objective: To assess the quality of the end-of-sprint new system

Test item: Completed system

Test basis: Stories, and corrections and changes if any, for the sprint

Dynamic test processes: Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.

Test techniques(s): Exploratory testing guided by stories, corrections, and changes if any

E.12 Story Acceptance Test Sub-process

This example represents a test sub-process associated with the acceptance of the system completed in the daily scrum as a result of the daily build. The test sub-process may not be performed every day in the sprint, but it will usually be performed more and more frequently towards the end of the sprint.

Test sub-process objective: Presentation of the completed system to support acceptance

Planned test sub-process content: Dynamic test: Presentation

Dynamic test: Presentation

Test objective: To present the completed system to the customer.

Test item: The completed system

Test basis: The completed stories

Dynamic test processes: Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.

Test techniques(s): n/a

The test execution may be performed reusing test procedures from the dynamic part of the story test sub-process, or new test procedures may be designed.

E.13 Story Set Test Sub-process

This example represents a test sub-process associated with the selection of the backlog to handle in a specific sprint based on the current project backlog.

Test sub-process objective: To provide information about the quality of the set of stories to work on in a particular sprint.

Planned test sub-process content: Static test: Stories feasibility.

Static test: Stories feasibility

Test objective:	To provide information about the set of selected stories
Test item:	Selected stories
Test basis:	Checklists, for example regarding the understanding of the stories and estimated development time, as well as the dependencies and consistency between stories.
Test techniques(s):	Informal review or technical review.

E.14 Story Test Sub-process

This example represents a test sub-process associated with completion of the story implementation prior to including the story in the daily build (or at the rate decided for making new builds).

During the development of the stories in the sprint backlog a number of similar test items are produced, that is the implementation of a story. The entire story test sub-process can therefore cover the testing of one or more stories, depending on how many stories that are implemented before a build. The implemented stories are tested individually in similar ways. The dynamic test in this example is generic and can be performed for any story. The story test sub-process is only complete when all planned tests are completed (or abandoned as the case may be).

Test sub-process objective:	To provide information about the implemented story prior to including it in a build
Planned test sub-process content:	Static test: Source code quality attributes Dynamic test: Story test

Static test: Source code quality attributes

Test objective:	To provide information about the quality of source code
Test item:	Source code created or affected of story implementation
Test basis:	Internal and/or external checklists, for example regarding a specific code writing style, and/or coding abnormalities such as use of a variable before it is declared.
Test techniques(s):	Technical review or static analysis.

Dynamic test: Story test

Test objective:	To provide information about the quality of the implementation of the story
Test item:	The implemented story in an environment isolated from the public build
Test basis:	The story
Detailed test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.
Test techniques(s):	Appropriate techniques, supplemented with techniques as appropriate to achieve required coverage.

E.15 System Test Sub-process

This example represents a test sub-process associated with the completion of the integration phase. It may be associated with a phase defined as the maturing phase to be completed before the system can be declared ready for acceptance test.

Test sub-process objective: To provide information about the quality of the fully integrated system.

Planned test sub-process content: Dynamic test: System test

Dynamic test: System test

Test objective:	To assess the quality of the complete system after integration
Test item:	Fully integrated system
Test basis:	System requirements
Detailed test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.
Test techniques(s):	Equivalence partitioning, boundary value analysis, state transition testing, and classification tree method as appropriate

The objective of the system testing is to find defects in features of the system compared to the way it has been defined in the software system requirements.

It should be foreseen that a number of retest sub-processes and regression test sub-processes will be necessary to achieve the completion criteria for the system test.

E.16 Component Test Sub-process

This example represents the test sub-process associated with the coding phase of the development life cycle.

During the coding phase a number of similar test items are produced, that is source code components that may be directly interpretable components or be compiled and linked into executable components. The entire component test sub-process can therefore cover the testing of all or some of these components individually in similar ways. The dynamic test in this example is generic and can be performed for any component. The component test sub-process is only complete when all planned tests are completed (or abandoned as the case may be).

Test sub-process objective: To provide information about the quality of the component

Planned test sub-process content: Dynamic test: component test

Dynamic test: Component test

Test objective:	To provide information about the quality of a component
Test item:	A component in isolation from other components in the system (may require driver and stubs)
Test basis:	The detailed design for the component, including traceability to requirements
Detailed test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.
Test techniques(s):	Appropriate techniques, supplemented with techniques as appropriate to achieve required coverage.

Annex F
(informative)

Roles and Responsibilities in Testing

F.1Testing Roles

There are a variety of names for the different roles within the test profession therefore this standard does not provide a thorough list of different roles and responsibilities intended to be representative of the global testing profession. Instead, the following roles are outlined in so far as the person who might fill that role would be responsible for completing some aspect of the testing process outlined in this standard. More than one person may have responsibility for each of the roles below.

Test Strategist

Establishes, and ensures conformance to, the Organizational Test Process.

Test Manager

Develops, manages and ensures the conformance with the Test Management Process. The Test Manager also plans and controls the Dynamic Test Process.

Tester

Develops the test deliverables and completes the processes associated with the Dynamic Test Process.

In reality the processes outlined in this standard are likely to be completed by a range of people with a range of job titles.

F.1.1 Communication in Testing

Testers need to communicate with people at appropriate levels of the organization as well as with stakeholders in external organizations. Communication may be based on written documentation like the Organizational Test Strategy, Test Plans, test status reports, and test completion reports. Written documentation can be accompanied by oral presentations, as is often the case in more formal sequential or evolutionary development. In more informal development regimes, such as agile, communication may predominantly be oral, supported by a minimal amount of written documentation.

F.1.2 Independence in Testing

Testing should be as objective as possible. The closer the test analyst is to the producer of the test item, the more difficult it is to be objective. It is generally accepted that is more difficult for a producer to find defects in their own work than it is for an independent tester to find the same defects. Independence in product assessment is common in many industries, e.g. publishing, where the editor performs the assessment; manufacturing where there is Quality Control; and home construction where there are building inspectors.

The kind of independence increases between the producer and the tester with the ‘distance’, budget and so on. For example, from a view of ‘distance six kinds:

a) Producer tests his or her own product;

b) Tests are designed by a different person than the producer, but one with the same responsibilities, typically another producer;

c) Tests are designed by a tester who is a member of the same organizational unit as the producer, reporting to the same manager;

d) Tests are designed by testers independent of the producing organizational unit, though still in-house;

e) Tests are designed by testers belonging to an external organization working in the production organization (consultants);

f) Tests are designed by testers in an external organization (third party testing).

The intention is to achieve as much independence between those who design the tests and those who produced the test item as possible, within the project's constraints of time, budget, quality and risk. The Organizational Test Strategy should determine the necessary degree of independence in the organization, and this should be inherited in the Project Test Plan and the plan for the test sub-processes at hand. It is normally assumed that higher risk situations should lead to a higher degree of independence, however the success rates for agile projects, where the testers are integrated into the same team as the developers appears to question this assumption. The IEEE 1012–2004, *IEEE Standard for Software Verification and Validation* addresses the concept of independence in verification and validation activities, including testing.

The degree of independence usually varies for the different test sub-processes. In dynamic component testing the lowest level of independence (i.e. no independence) is often seen, even though the same degree of independence applied in peer reviews (static testing done by developers) is not usually considered acceptable.

Annex G
(informative)

Bibliography

G.1 Bibliography

The following documents were used during the development of this standard.

BS 7925-1, *Software testing - vocabulary*

BS 7925-2, *Software testing - software component testing*

IEC 60300-3-9:1995, *Risk Analysis of technological systems*

IEEE 601.12–1990, IEEE Standard Glossary of Software Engineering Terminology

IEEE 829–2008, *IEEE Standard for Software Test Documentation*

IEEE 1008–1987, *IEEE Standard for Software Unit Testing*

IEEE 1012–2004, *IEEE Standard for Software Verification and Validation*

IEEE 1028–2008, *IEEE Standard for Software Reviews and Audits*

ISO/IEC 12207:2008, *Systems and software engineering – Software life cycle processes*

ISO/IEC 16085:2006, *IT – Systems and software Engineering – Lifecycle Processes – Risk Management*

ISO/IEC 24765, *Systems and Software Engineering Vocabulary*

ISO/IEC 25000:2005, *Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE*

ISO/IEC 25010-1, *Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software quality models*

ISO/IEC 25051:2006, *Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing*

ISO/IEC 90003:2004, *Software engineering — Guidelines for the application of ISO 9001:2000 to computer software*

International Software Testing Qualifications Board (ISTQB) Glossary Working Party, *Standard glossary of terms used in Software Testing*