INTERNATIONAL STANDARD

ISO/IEC/IEEE 29119-1

First edition
2013-09-01

Software and systems engineering — Software testing —

Part 1: Concepts and definitions

Ingénierie du logiciel et des systèmes — Essais du logiciel —

Partie 1: Concepts et définitions

Reference number
ISO/IEC/IEEE 29119-1:2013(E)

ISO copyright office
Case postale 56
CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
E-mail inmail@iec.ch
Web www.iec.ch

Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York
NY 10016-5997, USA
E-mail stds.ipr@ieee.org
Web www.ieee.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

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

The main task of ISO/IEC JTC 1 is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is called to the possibility that implementation of this standard may require the use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. ISO/IEEE is not responsible for identifying essential patents or patent claims for which a license may be required, for conducting inquiries into the legal validity or scope of patents or patent claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance or a Patent Statement and Licensing Declaration Form, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from ISO or the IEEE Standards Association.

ISO/IEC/IEEE 29119-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*, in cooperation with the Software & Systems Engineering Standards Committee of the IEEE Computer Society, under the Partner Standards Development Organization cooperation agreement between ISO and IEEE.

ISO/IEC/IEEE 29119 consists of the following standards, under the general title *Software and systems engineering — Software testing*:

— *Part 1: Concepts and definitions*

— *Part 2: Test processes*

— *Part 3: Test documentation*

— *Part 4: Test techniques*

Introduction

The purpose of the ISO/IEC/IEEE 29119 series of software testing standards is to define an internationally-agreed set of standards for software testing that can be used by any organization when performing any form of software testing.

It is recognized that there are many different types of software, software organizations, and methodologies. Software domains include information technology (IT), personal computers (PC), embedded, mobile, and scientific and many other classifications. Software organizations range from small to large, co-located to world-wide, and commercial to public service-oriented. Software methodologies include object-oriented, traditional, data driven and agile. These and other factors influence software testing. This series of international standards can support testing in many different contexts.

This part of ISO/IEC/IEEE 29119 facilitates the use of the other ISO/IEC/IEEE 29119 Software Testing standards by introducing the vocabulary on which this series of international standards are built and provides examples of their application in practice. Part 1 is informative providing definitions, a description of the concepts of software testing and ways to apply the software testing process defined in this part of ISO/IEC/IEEE 29119 and guidance for the other parts.

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. The use of different practices in test planning is demonstrated; as well as how automation can be used to support testing. The involvement of testing in defect management is also discussed. Annex A describes the role of testing within the larger scope of verification and validation. Annex B provides a brief introduction to metrics used to monitor and control testing. Annex C contains a set of examples showing how to apply the standard in different life cycle models. Annex D provides examples on detailed test sub-processes. Annex E provides additional information on the roles and responsibilities typically encountered in test groups and tester independence. Finally, the Bibliography is at the end of the document.

Note that Title Case is used throughout this part of ISO/IEC/IEEE 29119 to denote processes and documents that are specified in ISO/IEC/IEEE 29119-2 and ISO/IEC/IEEE 29119-3 (e.g. Test Planning Process, Test Plan), whereas lowercase letters are used for documents that form parts of other documents (e.g. the project test strategy is an element of the Project Test Plan).

The test process model that the ISO/IEC/IEEE 29119 series of software testing standards are based on is defined in detail in ISO/IEC/IEEE 29119-2 Test Processes. ISO/IEC/IEEE 29119-2 covers the software testing processes at the organizational level, test management level and for dynamic test levels. Testing is the primary approach to risk treatment in software development. This standard defines a risk-based approach to testing. Risk-based testing is a recommended approach to strategizing and managing testing that 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/IEEE 29119-3 Test Documentation. Software testing techniques that can be used during testing are defined in ISO/IEC/IEEE 29119-4 Test Techniques.

Together, this series of international standards aims to provide stakeholders with the ability to manage and perform software testing in any organization.

Software and systems engineering — Software testing —

Part 1: Concepts and definitions

1 Scope

This part of ISO/IEC/IEEE 29119 specifies definitions and concepts in software testing. It provides definitions of testing terms and discussion of concepts key to the understanding of the ISO/IEC/IEEE 29119 series of software testing international standards.

2 Conformance

ISO/IEC/IEEE 29119-1 is informative and no conformance with it is required.

The ISO/IEC/IEEE 29119 software testing series of standards contain three standards where conformance may be claimed:

— test processes;

— test documentation;

— test techniques.

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

3 Normative references

This document does not require the use of any normative references. Standards useful for the implementation and interpretation of this part of ISO/IEC/IEEE 29119 are listed in the Bibliography.

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC/IEEE 24765 and the following apply.

NOTE The following terms and definitions are provided to assist with the understanding and readability of parts 1, 2, 3 and 4 of the ISO/IEC/IEEE 29119 Software Testing standards so some terms defined in ISO/IEC/IEEE 29119-1 will not be used in ISO/IEC/IEEE 29119-1 and will only be used in another standard in the ISO/IEC/IEEE 29119 series. Only terms critical to the understanding of these standards 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
accessibility testing
type of usability testing used to measure the degree to which a test item can be operated by users with the widest possible range of characteristics and capabilities

4.2
actual results
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

EXAMPLE Outputs to hardware, changes to data, reports, and communication messages sent.

4.3
backup and recovery testing
type of reliability testing that measures the degree to which system state can be restored from backup within specified parameters of time, cost, completeness, and accuracy in the event of failure

4.4
black-box testing
see specification-based testing (4.39)

4.5
capacity testing
type of performance efficiency testing conducted to evaluate the level at which increasing load (of users, transactions, data storage, etc.) compromises a test item's ability to sustain required performance

4.6
compatibility testing
type of testing that measures the degree to which a test item can function satisfactorily alongside other independent products in a shared environment (co-existence), and where necessary, exchanges information with other systems or components (interoperability)

4.7
coverage item
see test coverage item (4.54)

4.8
decision
types of statements in which a choice between two or more possible outcomes controls which set of actions will result

Note 1 to entry: 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).

4.9
dynamic testing
testing that requires the execution of the test item

4.10
endurance testing
type of performance efficiency testing conducted to evaluate whether a test item can sustain a required load continuously for a specified period of time

4.11
equivalence partition
subset of the range of values of a variable, or set of variables, within a test item or at its interfaces such that all the values in the partition can reasonably be expected to be treated similarly by the test item (i.e. they may be considered "equivalent") by the test item

4.12
equivalence partition coverage
proportion of identified equivalence partitions of a test item that are covered by a test set

Note 1 to entry: 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 could be impossible.

4.13
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

4.14
error guessing
test design technique in which test cases are derived on the basis of the tester's knowledge of past failures, or general knowledge of failure modes

Note 1 to entry: The relevant knowledge could be gained from personal experience, or might be encapsulated in, for example, a defects database or a "bug taxonomy".

4.15
expected results
observable predicted behaviour of the test item under specified conditions based on its specification or another source

4.16
exploratory testing
experience-based testing in which the tester spontaneously designs and executes tests based on the tester's existing relevant knowledge, prior exploration of the test item (including the results of previous tests), and heuristic "rules of thumb" regarding common software behaviours and types of failure

Note 1 to entry: Exploratory testing hunts for hidden properties (including hidden behaviours) that, while quite possibly benign by themselves, could interfere with other properties of the software under test, and so constitute a risk that the software will fail.

4.17
feature set
collection of items which contain the test conditions of the test item to be tested which can be collected from risks, requirements, functions, models, etc.

Note 1 to entry: This could be the set of all features for the item (its full feature set), or a subset identified for a specific purpose (the functional feature set etc.).

4.18
Incident Report
documentation of the occurrence, nature, and status of an incident

4.19
installability testing
type of portability testing conducted to evaluate whether a test item or set of test items can be installed as required in all specified environments

4.20
load testing
type of performance efficiency testing conducted to evaluate the behaviour of a test item under anticipated conditions of varying load, usually between anticipated conditions of low, typical, and peak usage

4.21
maintainability testing
test type conducted to evaluate the degree of effectiveness and efficiency with which a test item may be modified

4.22
Organizational Test Policy
an executive-level document that describes the purpose, goals, and overall scope of the testing within an organization, and which expresses why testing is performed and what it is expected to achieve

Note 1 to entry: It is generally preferable to keep the Organizational Test Policy as short as possible in a given context.

4.23
Organizational Test Process
test process for developing and managing organizational test specifications

4.24
organizational test specification
document that provides information about testing for an organization, i.e. information that is not project-specific

EXAMPLE The most common examples of organizational test specifications are Organizational Test Policy and Organizational Test Strategy.

4.25
Organizational Test Strategy
document that expresses the generic requirements for the testing to be performed on all the projects run within the organization, providing detail on how the testing is to be performed

Note 1 to entry: The Organizational Test Strategy is aligned with the Organizational Test Policy.

Note 2 to entry: An organisation could have more than one Organizational Test Strategy to cover markedly different project contexts.

4.26
pass/fail criteria
decision rules used to determine whether a test item, or feature of a test item, has passed or failed after testing

4.27
performance testing
type of testing conducted to evaluate the degree to which a test item accomplishes its designated functions within given constraints of time and other resources

4.28
portability testing
type of testing conducted to evaluate the ease with which a test item can be transferred from one hardware or software environment to another, including the level of modification needed for it to be executed in various types of environment

4.29
procedure testing
type of functional suitability testing conducted to evaluate whether procedural instructions for interacting with a test item or using its outputs meet user requirements and support the purpose of their use

4.30
product risk
risk that a product may be defective in some specific aspect of its function, quality, or structure

4.31
project risk
risk related to the management of a project

EXAMPLE Lack of staffing, strict deadlines, changing requirements.

4.32
regression testing
testing following modifications to a test item or to its operational environment, to identify whether regression failures occur

Note 1 to entry: The sufficiency of a set of regression test cases depends on the item under test and on the modifications to that item or its operational environment.

4.33
reliability testing
type of testing conducted to evaluate the ability of a test item to perform its required functions, including evaluating the frequency with which failures occur, when it is used under stated conditions for a specified period of time

4.34
retesting
re-execution of test cases that previously returned a "fail" result, to evaluate the effectiveness of intervening corrective actions

Note 1 to entry: Also known as confirmation testing.

4.35
risk-based testing
testing in which the management, selection, prioritisation, and use of testing activities and resources are consciously based on corresponding types and levels of analyzed risk

4.36
scenario testing
class of test design technique in which tests are designed to execute individual scenarios

Note 1 to entry: A scenario can be a user story, use-case, operational concept, or sequence of events the software may encounter etc.

4.37
scripted testing
dynamic testing in which the tester's actions are prescribed by written instructions in a test case

Note 1 to entry: This term normally applies to manually executed testing, rather than the execution of an automated script.

4.38
security testing
type of testing conducted to evaluate the degree to which a test item, and associated data and information, are protected so that unauthorized persons or systems cannot use, read, or modify them, and authorized persons or systems are not denied access to them

4.39
specification-based testing
testing in which the principal test basis is the external inputs and outputs of the test item, commonly based on a specification, rather than its implementation in source code or executable software

Note 1 to entry: Synonyms for specification-based testing include black-box testing and closed box testing.

4.40
statement coverage
percentage of the set of all executable statements of a test item that are covered by a test set

4.41
statement testing
test design technique in which test cases are constructed to force execution of individual statements in a test item

4.42
static testing
testing in which a test item is examined against a set of quality or other criteria without code being executed

EXAMPLE Reviews, static analysis.

4.43
stress testing
type of performance efficiency testing conducted to evaluate a test item's behaviour under conditions of loading above anticipated or specified capacity requirements, or of resource availability below minimum specified requirements

4.44
structural testing
see structure-based testing (4.45)

4.45
structure-based testing
dynamic testing in which the tests are derived from an examination of the structure of the test item

Note 1 to entry: 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 2 to entry: Techniques include branch testing, decision testing, and statement testing.

Note 3 to entry: Synonyms for structure-based testing are structural testing, glass-box testing, and white box testing.

4.46
suspension criteria
criteria used to (temporarily) stop all or a portion of the testing activities

4.47
test basis
body of knowledge used as the basis for the design of tests and test cases

Note 1 to entry: 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.

4.48
test case
set of test case preconditions, inputs (including actions, where applicable), and expected results, developed to drive the execution of a test item to meet test objectives, including correct implementation, error identification, checking quality, and other valued information

Note 1 to entry: A test case is the lowest level of test input (i.e. test cases are not made up of test cases) for the test sub-process for which it is intended.

Note 2 to entry: Test case preconditions include test environment, existing data (e.g. databases), software under test, hardware, etc.

Note 3 to entry: Inputs are the data information used to drive test execution.

Note 4 to entry: Expected results include success criteria, failures to check for, etc.

4.49
Test Case Specification
documentation of a set of one or more test cases

4.50
Test Completion Process
Test Management Process for ensuring that useful test assets are made available for later use, test environments are left in a satisfactory condition, and the results of testing are recorded and communicated to relevant stakeholders

4.51
Test Completion Report
report that provides a summary of the testing that was performed

Note 1 to entry: Also known as a Test Summary Report.

4.52
test condition
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 1 to entry: Test conditions can be used to derive coverage items, or can themselves constitute coverage items.

4.53
test coverage
degree, expressed as a percentage, to which specified test coverage items have been exercised by a test case or test cases

4.54
test coverage item
attribute or combination of attributes that is derived from one or more test conditions by using a test design technique that enables the measurement of the thoroughness of the test execution

4.55
test data
data created or selected to satisfy the input requirements for executing one or more test cases, which may be defined in the Test Plan, test case or test procedure

Note 1 to entry: Test data could be stored within the product under test (e.g. in arrays, flat files, or a database), or could be available from or supplied by external sources, such as other systems, other system components, hardware devices, or human operators.

4.56
Test Data Readiness Report
document describing the status of each test data requirement

4.57
Test Design and Implementation Process
test process for deriving and specifying test cases and test procedures

4.58
Test Design Specification
document specifying the features to be tested and their corresponding test conditions

4.59
test design technique
activities, concepts, processes, and patterns used to construct a test model that is used to identify test conditions for a test item, derive corresponding test coverage items, and subsequently derive or select test cases

4.60
test environment
facilities, hardware, software, firmware, procedures, and documentation intended for or used to perform testing of software

Note 1 to entry: A test environment could contain multiple environments to accommodate specific test sub-processes (e.g. a unit test environment, a performance test environment etc.).

4.61
test environment readiness report
document that describes the fulfilment of each test environment requirement

4.62
Test Environment Requirements
description of the necessary properties of the test environment

Note 1 to entry: All or parts of the test environment requirements could reference where the information can be found, e.g. in the appropriate Organizational Test Strategy, Test Plan, and/or Test Specification.

4.63
Test Environment Set-up Process
dynamic test process for establishing and maintaining a required test environment

4.64
test execution
process of running a test on the test item, producing actual result(s)

4.65
Test Execution Log
document that records details of the execution of one or more test procedures

4.66
Test Execution Process
dynamic test process for executing test procedures created in the Test Design and Implementation Process in the prepared test environment, and recording the results

4.67
Test Incident Reporting Process
dynamic test process for reporting to the relevant stakeholders issues requiring further action that were identified during the test execution process

4.68
test item
work product that is an object of testing

EXAMPLE A system, a software item, a requirements document, a design specification, a user guide.

4.69
test level
specific instantiation of a test sub-process

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.

Note 1 to entry: Test levels are synonymous with test phases.

4.70
test management
planning, scheduling, estimating, monitoring, reporting, control and completion of test activities

4.71
Test Management Process
test process containing the sub-processes that are required for the management of a test project

Note 1 to entry: See Test Planning Process, Test Monitoring and Control Process, Test Completion Process.

4.72
Test Monitoring and Control Process
Test Management Process for ensuring that testing is performed in line with a Test Plan and with organizational test specifications

4.73
test object
see test item (4.68)

4.74
test phase
specific instantiation of test sub-process

Note 1 to entry: 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).

4.75
Test Plan
detailed description of test objectives to be achieved and the means and schedule for achieving them, organised to coordinate testing activities for some test item or set of test items

Note 1 to entry: A project can have more than one Test Plan, for example there could be a Project Test Plan (also known as a master test plan) that encompasses all testing activities on the project; further detail of particular test activities could be defined in one or more test sub-process plans (i.e. a system test plan or a performance test plan).

Note 2 to entry: Typically a Test Plan is a written document, though other plan formats could be possible as defined locally within an organization or project.

Note 3 to entry: Test Plans could also be written for non-project activities, for example a maintenance test plan.

4.76
Test Planning Process
Test Management Process used to complete test planning and develop Test Plans

4.77
test practice
conceptual framework that can be applied to the Organizational Test Process, the Test Management Process, and/or the Dynamic Test Process to facilitate testing

Note 1 to entry: Test Practices are sometimes referred to as test approaches.

4.78
test procedure
sequence of test cases in execution order, and any associated actions that may be required to set up the initial preconditions and any wrap up activities post execution

Note 1 to entry: 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.

4.79
Test Procedure Specification
document specifying one or more test procedures, which are collections of test cases to be executed for a particular objective

Note 1 to entry: The test cases in a test set are listed in their required order in the test procedure.

Note 2 to entry: Also known as a manual test script. A test procedure specification for an automated test run is usually called a test script.

4.80
test process
provides information on the quality of a software product, often comprised of a number of activities, grouped into one or more test sub-processes

EXAMPLE The Test Process for a particular project may well consist of multiple sub-processes, e.g. a system test sub-process, a Test Planning sub-process (a part of the larger Test Management Process) or a static testing sub-process.

4.81
test requirement
see test condition (4.52)

4.82
test result
indication of whether or not a specific test case has passed or failed, i.e. if the actual result observed as test item output corresponds to the expected result or if deviations were observed

4.83
test script
test procedure specification for manual or automated testing

4.84
test set
set of one or more test cases with a common constraint on their execution

EXAMPLE A specific test environment, specialized domain knowledge or specific purpose.

4.85
test specification
complete documentation of the test design, test cases and test procedures for a specific test item

Note 1 to entry: A test specification could be detailed in one document, in a set of documents, or in other ways, for example in a mixture of documents and database entries.

4.86
test status report
report that provides information about the status of the testing that is being performed in a specified reporting period

4.87
test strategy
part of the Test Plan that describes the approach to testing for a specific test project or test sub-process or sub-processes

Note 1 to entry: The test strategy is a distinct entity from the Organizational Test Strategy.

Note 2 to entry: The test strategy usually describes some or all of the following: the test practices used; the test sub-processes to be implemented; 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.

4.88
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 process for a test project

Note 1 to entry: A test sub-process could 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.

4.89
test technique
see test design technique (4.59)

4.90
test traceability matrix
document, spreadsheet, or other automated tool used to identify related items in documentation and software, such as requirements with associated tests

Note 1 to entry: Also known as: verification cross reference matrix, requirements test matrix, requirements verification table, and others.

Note 2 to entry: Different test traceability matrices could have different information, formats, and levels of detail.

4.91
test type
group of testing activities that are focused on specific quality characteristics

Note 1 to entry: A test type could be performed in a single test sub-process or could 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).

EXAMPLE Security testing, functional testing, usability testing, and performance testing.

4.92
testing
set of activities conducted to facilitate discovery and/or evaluation of properties of one or more test items

Note 1 to entry: Testing activities could include planning, preparation, execution, reporting, and management activities, insofar as they are directed towards testing.

4.93
testware
artefacts produced during the test process required to plan, design, and execute tests

Note 1 to entry: Testware can include such things as documentation, scripts, inputs, expected results, files, databases, environment, and any additional software or utilities used in the course of testing.

4.94
unscripted testing
dynamic testing in which the tester's actions are not prescribed by written instructions in a test case

4.95
volume testing
type of performance efficiency testing conducted to evaluate the capability of the test item to process specified volumes of data (usually at or near maximum specified capacity) in terms of throughput capacity, storage capacity, or both

4.96
white box testing
see structure-based testing (4.45)

5 Software Testing Concepts

5.1 Introduction to Software Testing

Software testing is necessary because:

— Information on the quality characteristics of the test item(s) is required by decision makers;

— The test item(s) being tested does not always do what it is expected to do;

— The test item(s) being tested needs to be verified;

— The test item(s) being tested needs to be validated; and/or

— Evaluation of the test item(s) needs to be conducted throughout the software and system development life cycle.

It is generally accepted that it is not possible to create perfect software. It is therefore necessary to test software before it is released to the users to reduce the risk of mistakes in software production having a negative impact when the software is used. It is equally necessary to ensure that testing is performed well.

Mistakes or introduced defects occur and 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 has no impact on the operation of the software if it is not encountered when the software is used. But if a defect is encountered under the right conditions when the product is put to use, it may cause the product to fail to meet the user's legitimate need.. Serious consequences may follow from a user experiencing a software failure; for example, a defect may compromise business reputation, public safety, business economic viability, business or user security, and/or the environment.

Dynamic testing is necessary, but not sufficient to provide reasonable assurance that software will perform as intended. Additional static testing activities, such as peer reviews and static analysis, should be performed in combination with effective dynamic testing activities.

The primary goals of testing are to: provide information about the quality of the test item and any residual risk in relation to how much the test item has been tested; to find defects in the test item prior to its release for use; and to mitigate the risks to the stakeholders of poor product quality.

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.

Product quality has many aspects, including conformance to specifications, absence of defects, and fitness in meeting the requirements of the product's users. ISO/IEC 25010, *System and software quality models* defines eight quality characteristics that can be measured or assessed by testing (see 5.5.3).

Software testing should focus on providing information about a software product and finding as many defects as possible, as early as possible in the development process, under given constraints of cost and schedule. Testing considerations include:

— Testing is 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 generic testing processes (see ISO/IEC/IEEE 29119-2 Test processes for more details).

— The organizational test process sets and maintains the test policies and test strategies that apply across the organization's projects and functions.

— Testing should be planned, monitored and controlled. ISO/IEC/IEEE 29119-2 Test processes includes a test management process, and can be applied to testing in all development life cycles and the management of exploratory testing.

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

— Testing entails examining a test item.

— 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. IEEE 1028) 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 performed

by other groups within a project or defined by other non-testing standards. This is because the static testing activities are considered highly important for complete life cycle 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.

— Static testing may also include the use of static analysis tools which find defects in the code or documents without the code executing (e.g. a compiler, a cyclomatic complexity analyser, or a security analyser for code).

— Dynamic testing consists of more than "just" running executable test items; it also includes both preparation activities and follow-up activities. The Dynamic Test Processes described in ISO/IEC/IEEE 29119-2 cover 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 in a given work item.

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

— Testing, whether static or dynamic, should aim to provide both types of confirmation, though with the expectation that confirmation will not be immediate because of the discovery of defects.

5.1.1 The Role of Testing in Verification and Validation

— This standard addresses only some of the verification and validation activities. Specifically it addresses software testing, which is a major activity of verification and validation. Other standards, e.g. ISO/IEC 12207 and IEEE 1012, address other verification or validation activities. This standard addresses testing; it does not address the other activities of validation and verification (e.g. V&V analysis, formal methods) except in passing. 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 hierarchy 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 item. 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 clause 5.4 Risk-Based testing.

5.1.3 Testing as a Heuristic

In engineering (and software engineering) a heuristic is an experience-based (trial and error) 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 sometimes they may not solve, or only partially solve, a problem. Much of systems and software testing is based on heuristics. For instance, they are useful when creating 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 strategy by employing multiple test strategies.

5.2 Software Testing in an Organizational and Project Context

Businesses involved in the development or acquisition of software products have an interest in developing and using effective, efficient and repeatable processes. To accomplish this they generally develop a robust set of software life cycle processes that are applied to the development projects that they perform. This standard is intended to be useful both for adoption across an organization and for use on specific projects. An organization would adopt the standard and supplement it with additional procedures, practices, tools and policies, as necessary. A specific software or system development project performed within an 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 across an organization. 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 organizational test strategies are usually only seen in more mature organizations. Testing can be, and is, performed without formal test policies and organizational test strategies in organizations of lower maturity, but this practice gives less coherence to the testing within the organization and typically makes the testing performed in projects less effective and efficient.

Software testing is performed as a context-managed process. 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 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 situations. Hence organizations and projects should tailor and refine the details of testing with reference to standards such as this.

The overall project plan 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 these organizational guidelines. It should also account for the constraints given in the overall project plan. A Project Test Plan includes a project test strategy and the project-specific decisions (including assumptions) used to derive this strategy. A major element of test planning is the weighing of various test needs and the balancing of resources across the various tests. The Test Plan records the outcome of this analysis. In this standard risk is the primary method of determining test needs (see clause 5.4), although the test practices described in clause 5.6 may be incorporated in the strategy.

The testing for a project will often be performed across a number of test sub-processes; each test sub-process may 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 situation that an organization is in, if any. This context is made up of laws, regulations and industry standards. Inside this regulatory situation the organization develops the necessary policies and procedures it requires in order to be successful. The Organizational Test Policy operates at this level. Within the organization each project is instantiated to meet some need or opportunity the organization has identified. The project context helps determine which life cycle model is chosen. At this level, based on the context of the project and life cycle model, a strategy for testing is determined. The project plan and strategy for testing form the basis for the Project Test Plan.

The Project Test Plan describes the overall strategy for testing and the test processes to be used. It establishes the context of testing for the project by determining the objectives, practices, 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 design techniques (static or dynamic) to use in order to complete the testing required by the particular sub-process plan. For more on test design techniques see clause 5.5.6 of this document and ISO/IEC/IEEE 29119-4.

Each sub-process test plan may address more than one test level (e.g. the security test plan may address several test levels) 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 strategy for test execution (e.g. scripted, unscripted, or a mixture of both).

Test plans include a test strategy (see clause 6). Test strategies are tuned to the specific test project context. Test strategies should address the specific items listed in Figure 1, which are defined in the other parts of this

standard. Each test plan (and strategy) will be unique, selecting different: test sub-processes; levels of automation; mixes of test design techniques; completion criteria; and their scheduling and resourcing. The planning and selection of these starts early in a project and will continue through the test life cycle as factors such as risk change. Many parts of the test plan and strategy should be expected to change, though changes will likely be restricted by project, organization, and /or regulatory constraints.

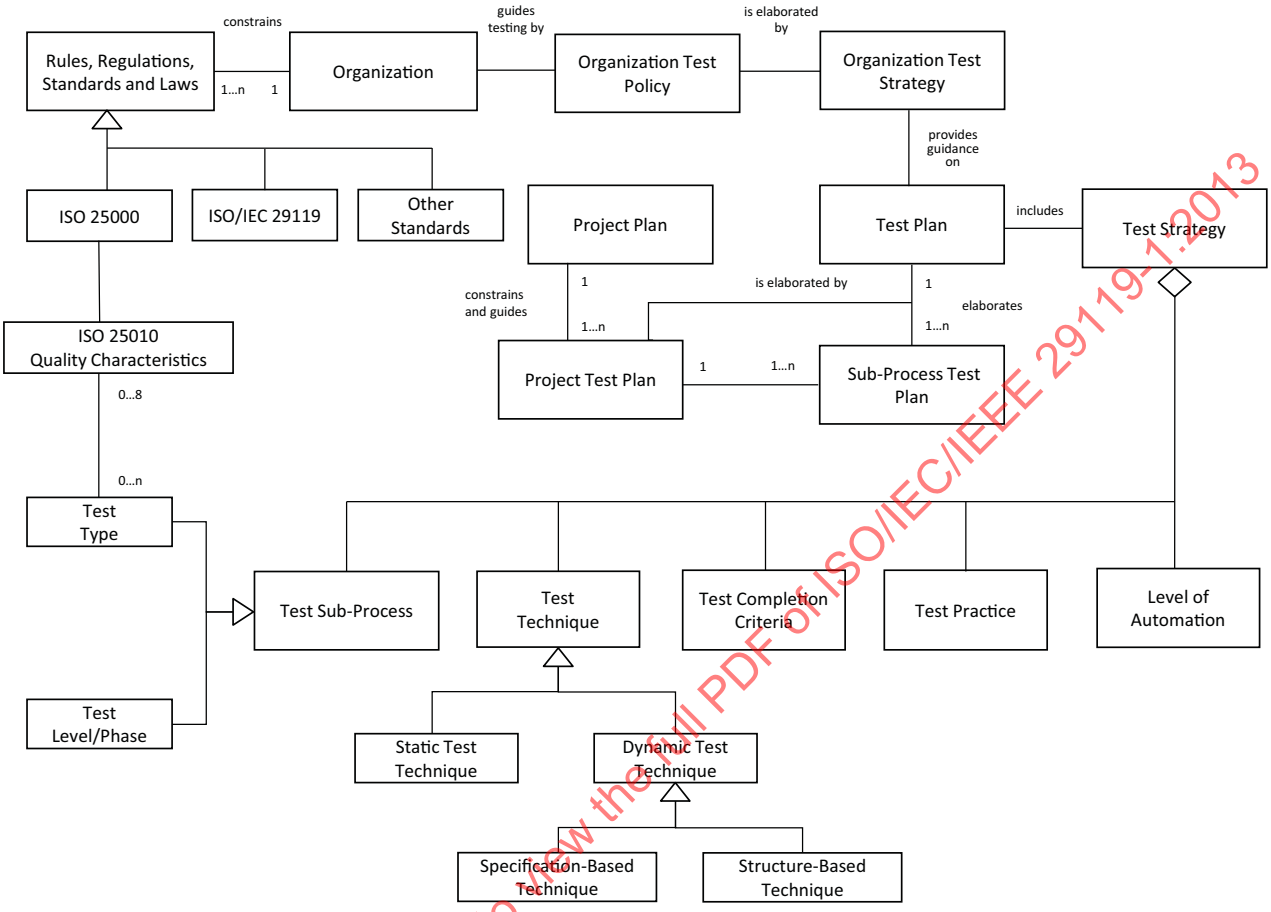

Figure 1 — Multi-layered test context diagram

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

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

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

— A test sub-process associated with a test level may contain more than one test type sub-process (e.g. functional and performance testing as part of system testing); and

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

The diagram also makes explicit the relationship between test types and quality characteristics (as defined in ISO/IEC 25010 System and software quality models). 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.

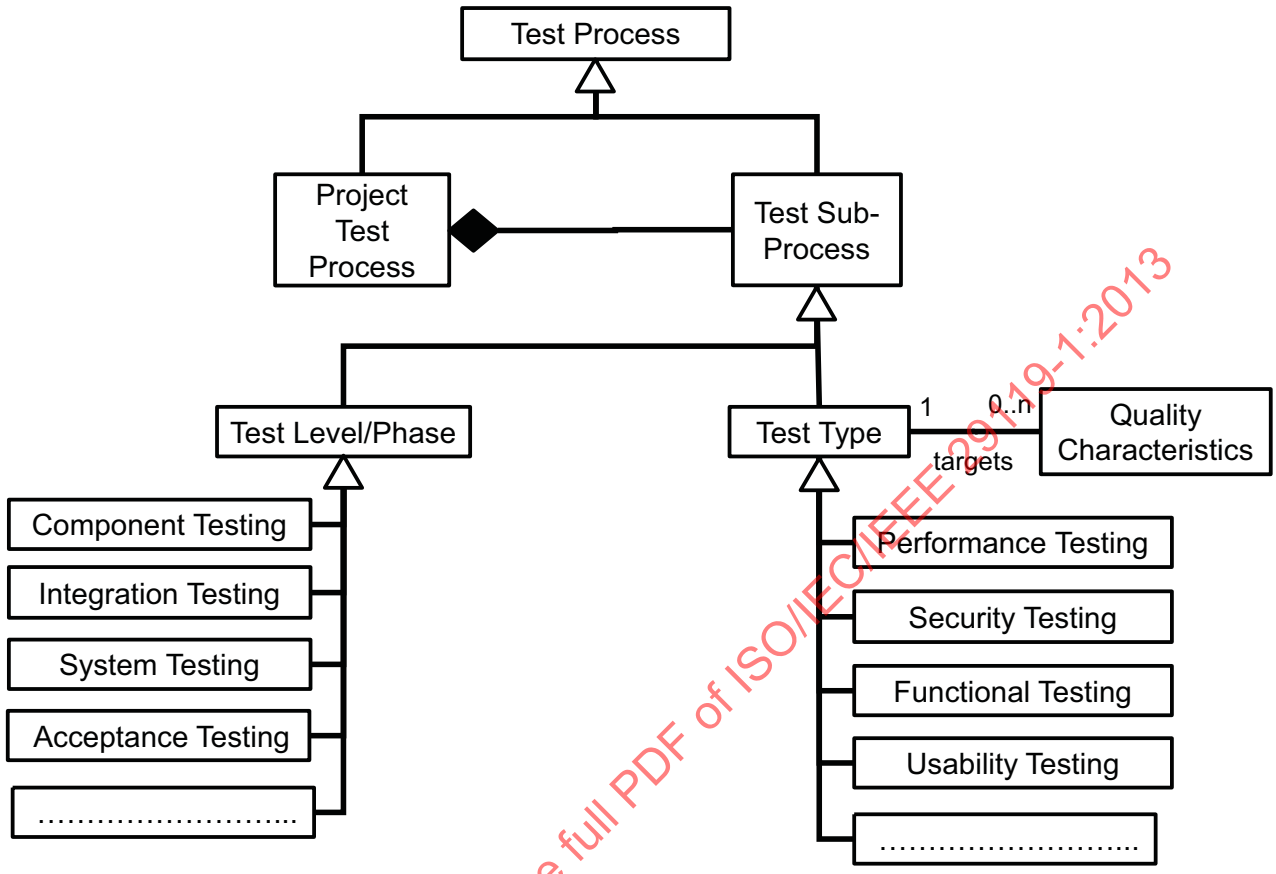

Figure 2 — The relationship between the generic test sub-process, test levels and test types

5.2.1 The Test Process

This standard uses a three layer process model, which is described in detail in ISO/IEC/IEEE 29119-2, 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 the 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 test processes used for dynamic testing.

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

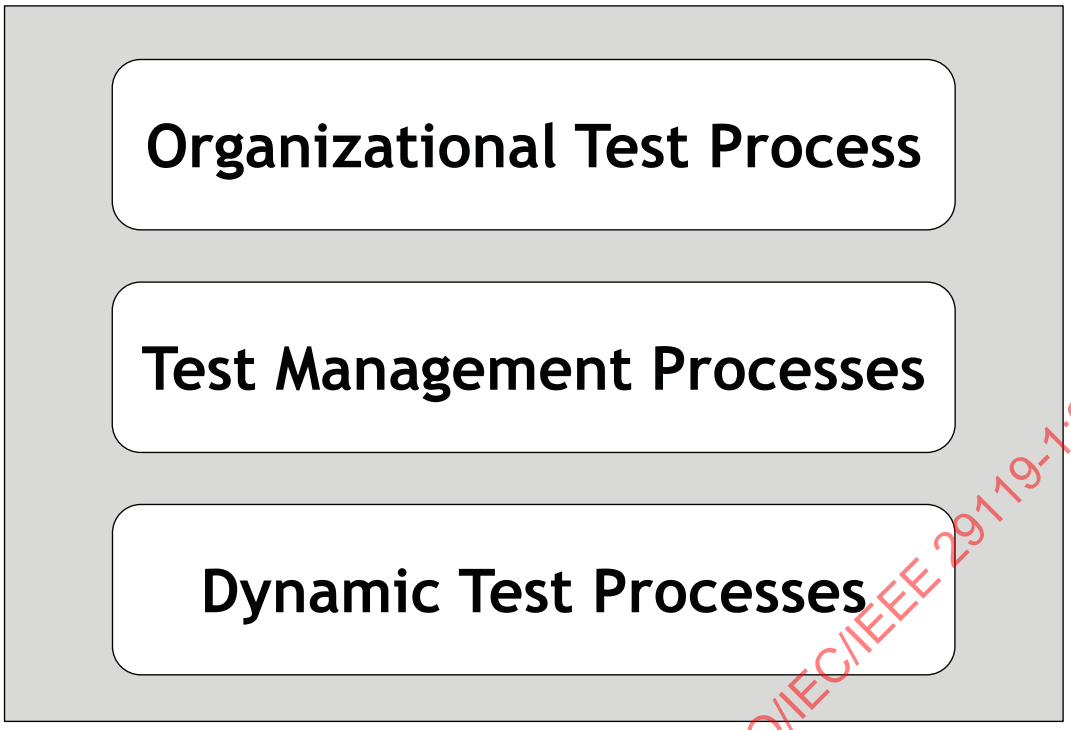

Figure 3 — The multi-layer relationship between test processes

The 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 Test Policy also guides the preferred direction and behaviour regarding the formulation and execution of the Organizational Test Strategy and the organization's test processes. The creation, implementation and maintenance of the Organizational Test Policy are described by the Organizational Test Process.

The Organizational Test Strategy expresses requirements and constraints on the test management processes and the dynamic test processes to be performed on all the projects run within the organization, unless they are too dissimilar in character, in which case multiple Organizational Test Strategies may be formulated. It is aligned with the Organizational Test Policy, and describes 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 project constraints, 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, balancing the given constraints (resources and time) with the defined scope and quality of the test work to be done. This is documented in a Project Test Plan. During testing, monitoring activities are performed to ensure that testing is progressing as planned and to ensure that risks are being treated appropriately, and if any changes to the testing activities are required then control directives will be issued to the relevant test process or sub-process. During this period of monitoring and control, Test Status Reports may regularly be produced to inform stakeholders of test progress. 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.

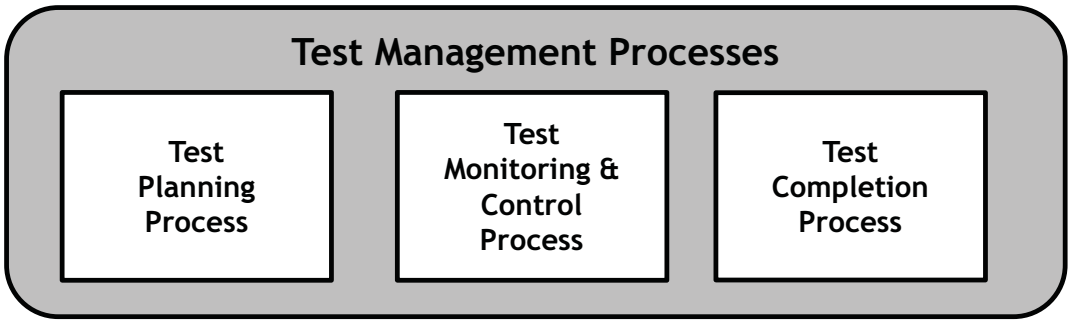

Figure 4 — The test management processes

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-processes. 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 processes are outlined in Figure 5 and fully described in ISO/IEC/IEEE 29119-2. Processes for static testing are described in other published standards (e.g. IEEE 1028).

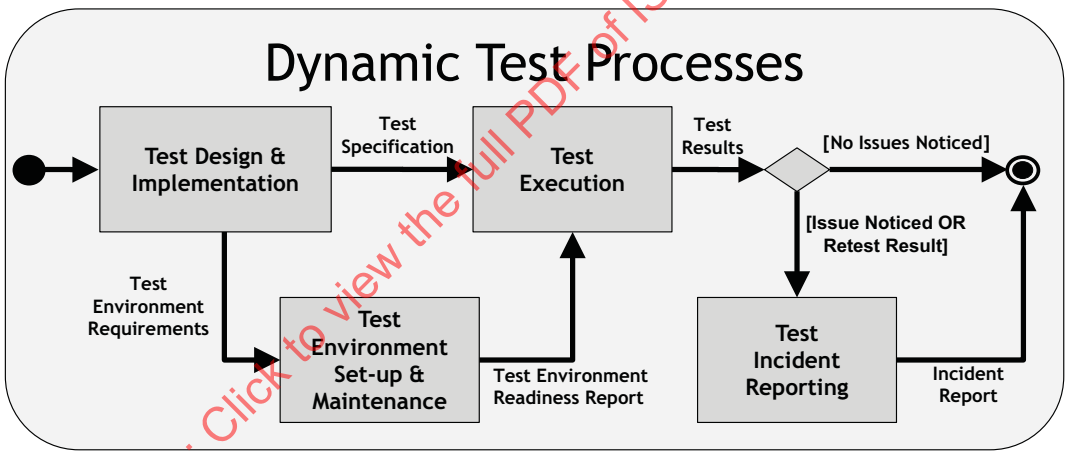

Figure 5 — Dynamic test processes

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 ISO/IEC/IEEE 29119-2.

5.3 Generic Testing Processes in the Software Life cycle

Software has an expected life cycle from its initial conception to its eventual retirement. Software testing takes place within the wider context of software development and maintenance. This clause outlines an example software development lifecycle and some of the relationships between its sub-processes and test processes. ISO/IEC 12207:2008 outlines software life cycles in detail. ISO/IEC 15288 outlines system life cycle process in detail. An example system life cycle is illustrated in Figure 6.

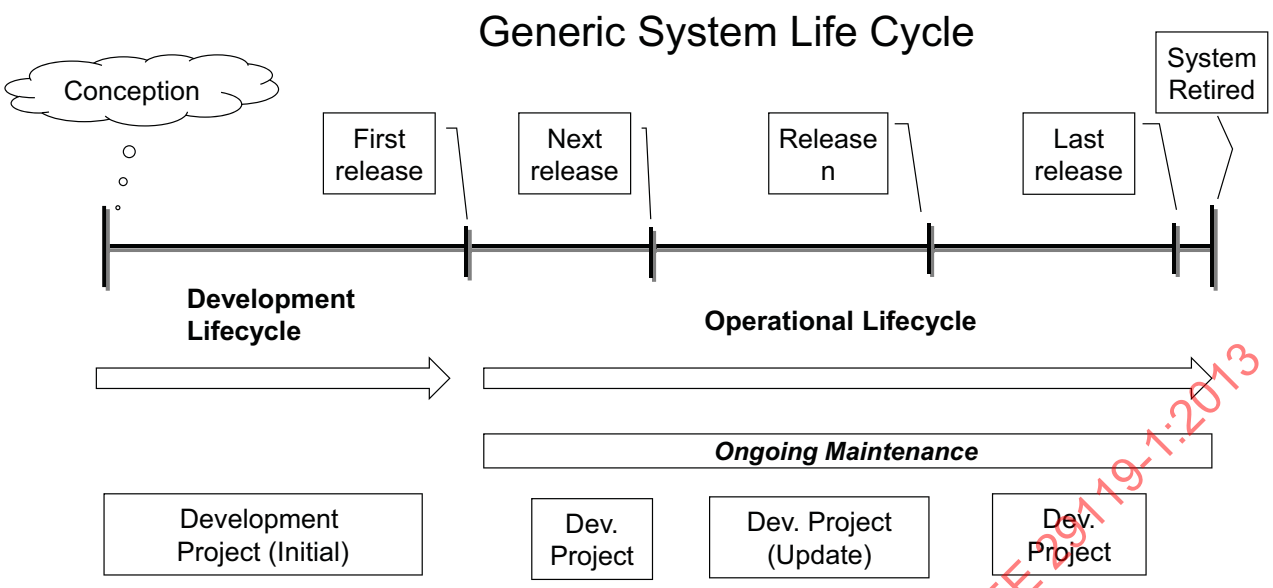

Figure 6 — An example Software Life cycle

A software life cycle generally comprises multiple sub-life cycles. Figure 6 shows that a software life cycle is often made up of one or more development life cycles and one or more operational life cycles.

The period of time from conception to the initial release is known as the development life cycle, which is a subset of the software 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.

Testing is also performed in order to evaluate software that has been purchased to meet business requirements. A framework for evaluating and testing Commercial Off-The-Shelf (COTS) software purchases can be found in ISO/IEC 25051.

5.3.1 Development Project Sub-processes and their Results

Software and system development typically consists of a few common building blocks. In the software industry these building blocks are typically referred to as ‘phases’, ‘stages’, ‘steps’, ‘levels’, or more generically as ‘development sub-processes’.

These can include:

— Requirements engineering;

— Architectural design;

— Detailed design; and

— Coding.

The particular approach to system development adopted by an organization or project will determine how these development sub-processes 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, every few weeks.

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.

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.

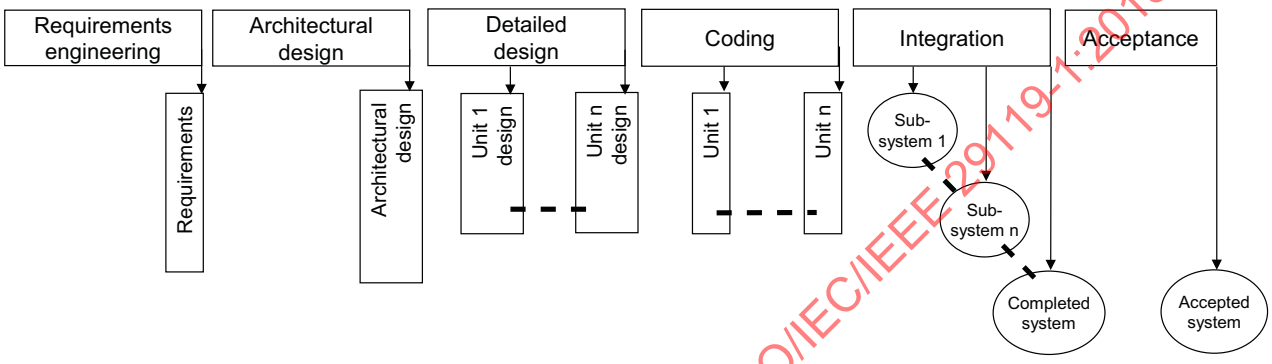

Figure 7 — Example of development sub-processes

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 software 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 process is generally concerned with keeping an acceptable or agreed level of reliability and availability of the system. This involves production of new versions of the system with corrections of highest priority defects found in operation and possibly the introduction of high priority 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, 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.

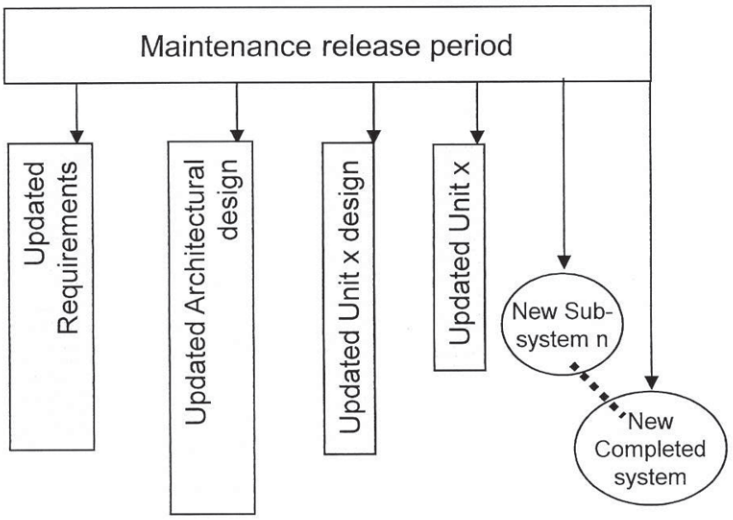

Figure 8 — Example Maintenance Release Period

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 software development life cycle. Some of these are:

— Quality assurance;

— Project management;

— Configuration management; and

— 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 will fulfil established technical or quality requirements. This is achieved by the imposition of methods, standards, tools, and skills that are recognised as the appropriate practice for the context. The Quality Assurance process uses the test results and other information to investigate, rank and report any problem (including any risk) in the design, planning or execution of the software engineering processes.

Measures should be collected during testing, as they can provide information about the quality of test processes and/or test items 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 including the management of the test project within the overall project. Regardless of who has responsibility

for the individual processes, project management and test management processes are closely related, as shown in Figure 9.

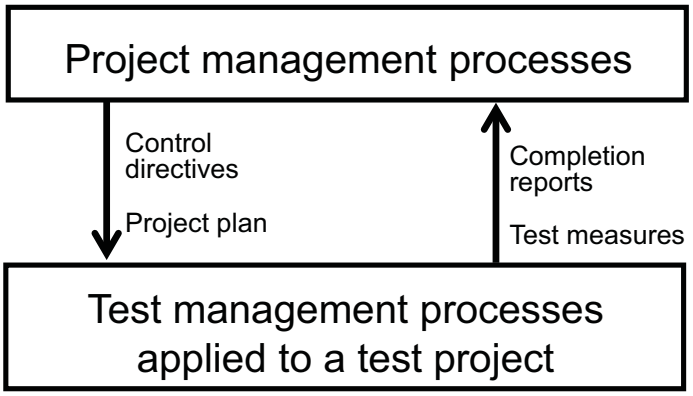

Figure 9 — Relationship between the overall project and the test project

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, measurements collected from detailed test activities are analysed 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, which will need to be issued to the test project to help ensure that the test project is kept under control.

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. The purpose of configuration management is to establish and maintain the integrity of work products. It is good practice to test an organization's or project's configuration management system before its operational use, to find out whether it meets the organizational or project requirements.

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 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, Organizational Test Strategy);

— Test plans;

— Test specifications; and/or

— Test environment configuration items such as: test 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 a 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 that is under 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 that is under configuration management and use that as the basis for a test specification which is subsequently placed under configuration management.

For the purpose of being able to recreate a problem so it can be analysed further, it is good practice (where possible) to make the configuration management system comprehensive and robust enough that at any point in the future a test can be repeated under the exact same conditions as before. It is acceptable practice to exclude certain types of testing - for example, unit testing - from the repeatability requirement as long as the exception is announced in the Organizational Test Strategy or the Project Plan.

The configuration management reports for a test process should provide detailed measures which may be needed for analysing the progress and status of incidents.

5.3.3.4 Process Improvement and Testing

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

The test processes 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); and

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 measurements 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. Improvements identified may only apply to a given project, or they may be rolled out across all projects in the organization. These new test processes should be monitored to assess whether they produce the expected return on investment.

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 (e.g. practices, techniques and types) exist to aid in choosing an appropriate sample to test and these are discussed and outlined in this standard. A key premise of this standard is the idea of performing the optimal testing within the given constraints and context using a risk-based approach.

This is achieved by identifying the relative value of different strategies for testing, in terms of the risks they mitigate for the stakeholders of the completed system, and for the stakeholders developing the system. Carrying out risk-based testing ensures that the risks with the highest priority are paid the highest attention during testing. Other standards can help with the determination of the risk of the systems in operation, e.g. ISO/IEC 16085 Risk Management.

Risks can be categorized in a number of ways. For instance, risks can be identified in terms of not satisfying regulatory and/or legal requirements, failing to meet contractual obligations, related to unsuccessful progress

and completion of the project (project risks) and a work product not achieving its expected behaviour (product risks).

When performing risk-based testing, risk analysis is used to identify and score risks, so that the perceived risks in the delivered system (and to the development of this system) can be scored, prioritized and categorized and subsequently mitigated.

5.4.1 Using Risk-Based Testing in the Organizational Test Process

The Organizational Test Policy sets the context for the testing in the organization. In doing so it should identify organizational risks that may impact the test process. For example, an organization producing medical software may have to comply with strict quality standards; the Organizational Test Policy for this organization could include requirements to address safety-related standards and thereby attempts to mitigate the impact of failing to do this on the business.

The Organizational Test Strategy should address the approach to managing risk in the test process. In the above example of an organization producing medical software, the Organizational Test Strategy could provide guidelines on how the testing should be performed in the organization to contribute towards managing the risk of failure to comply with the regulatory standards. In addition to this the Organizational Test Strategy could mandate a specific approach to applying risk management that will be used in the test management processes.

5.4.2 Using Risk-Based Testing in the Test Management processes

The categorised risk profile (the identified set of risks and their scores) is used to determine what testing should be performed on the project – this is described in the test strategy, which forms part of the test plan. For instance, the risk scores can be used to determine the rigour of testing (e.g. test sub-processes, test design techniques, test completion criteria). The prioritization of risks may be used to determine the test schedule (testing associated with higher priority risks is typically addressed earlier). The type of risk can be used to decide the most appropriate forms of testing to perform. For instance, if there is a perceived risk in the interfaces between components, then integration testing may be deemed necessary, and if there is a perceived risk that users may find the application difficult to use, then usability testing may be deemed necessary.

It is good practice to involve as wide a range of stakeholders as possible in these activities to ensure that as many risks as possible are identified and their respective mitigating actions (or not – we may decide to not address some risks) are agreed.

A benefit of using this approach is that at any point the risk of delivery can be simply communicated to stakeholders as the residual risks.

The results of performing testing against perceived risks and also the evolving business situation will naturally cause the risk profile to change over time thus requiring risk-based testing to be considered as an on-going activity.

5.4.3 Using Risk-Based Testing in the Dynamic Testing processes

The risk profile is used to provide guidance to all the dynamic testing processes. In the test design process the product risks are used to inform the tester which test design techniques are most appropriate to apply. It can also be used to guide how much test analysis occurs during the test design process, with higher risk areas receiving more effort than low risk areas. Risk can also be used to prioritise the feature sets and, once they are derived, the test conditions and the test cases.

Dynamic test execution is a risk mitigation activity. Running test cases mitigates risk because if the test passes then the likelihood of the risk occurring is normally lower – and so the risk score is reduced. Similarly, if the test fails, the risk may increase. The process of test incident reporting would then be used to determine if the failed test case had resulted in an issue or if it required further investigation.

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 software 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 processes to it. A test sub-process starts with test planning. The test monitoring and control activity continues for 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 planned for 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 regression testing. Retesting and regression testing can be performed to determine that changes made to a work product in order to correct defects have corrected the problems and not caused further defects (for more information see clause 5.5.5). An example test sub-process is illustrated in Figure 10.

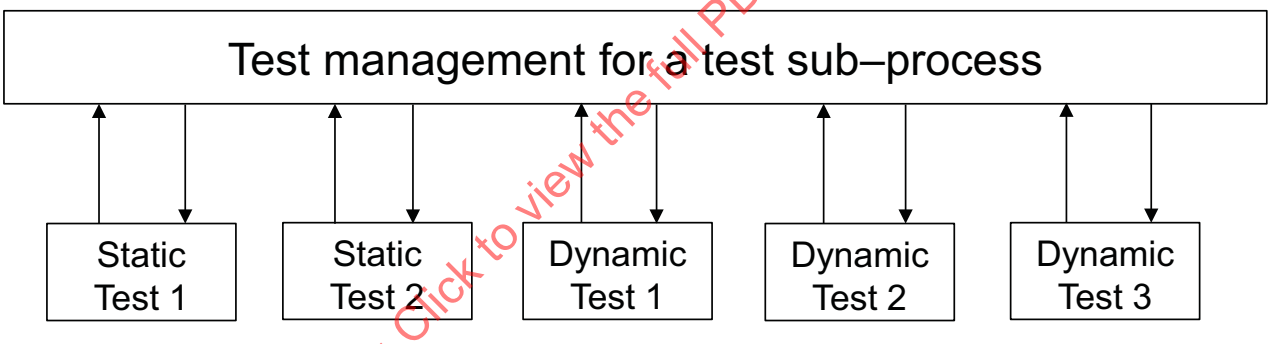

Figure 10 — Generic example test sub-process

The number of test sub-processes in a test project depends on the test strategy and the life cycle phases defined for the overall project. It does not, however, depend on the development life cycle (i.e. a sequential or evolutionary development cycle does not determine the number of test sub-processes required in itself).

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

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 design techniques to use. Examples of test objectives, test items, test basis, and test design 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 product;

— Assessment of whether the product has met stakeholder expectations;

— Assessment of whether 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 requirements (i.e. regulatory, design, contractual, etc.).

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 test item type. Deciding the relevant test objectives for a test item 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 acceptance test sub-process is applied in this context 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 software component;
  - A sub-system;
  - A complete system.
- Document-related test items:
  - A plan, e.g. project plan, Test Plan, or configuration management plan;
  - A requirements specification;
  - Architectural design;
  - Detailed design;
  - Source code;
  - A manual, e.g. user manual or installation manual;
  - Test specifications and test procedures.

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

5.5.3 Testing of Quality Characteristics

ISO/IEC 25010 System and software quality models outlines a model for software quality. This model describes eight quality characteristics, which define the quality attributes of a test item. Testing is an activity that measures the important quality characteristics for a given test item. The eight 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; and

— 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 (security testing) 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, 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 characteristic 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;

— 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 testing of an item can be planned, designed, inferred, implemented, and managed. This can include selecting test behaviour, pass-fail criteria, inputs, environment, practices, and techniques. 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;

— Experience of the tester or other Subject Matter Experts with functions needed by users or with the history of the product;

— 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; and/or

— 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 later 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/IEC 25010 System and software quality models; and

— Non-functional requirements – specifying how well the functionality should present itself and behave, aligning to the other quality characteristics outlined in ISO/IEC 25010 System and software quality models. 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 has remedied the original issue. Retesting is often performed by re-running the test case that produced the unexpected result that led to the identification of the original issue. However in order to retest effectively, new test conditions may be identified, analysed and new test cases written.

Regression testing is selective testing of a system or component that has previously been tested to verify that modifications have not caused unintended side-effects and that the system or component still complies with its original requirements. The intent of regression testing is to ascertain that modifications have not caused 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 Design Techniques

Test design techniques exist for static testing and for dynamic testing. The purpose of a test design technique is to assist the testers of test items in finding defects as effectively and efficiently as possible. Static testing typically achieves this by identifying apparent defects ("issues") in documentary test items or anomalies in source code. Dynamic testing achieves this by forcing failures of executable test items.

5.5.6.1 Static Test Design Techniques

Static testing can include a variety of activities, for example static code analysis, cross document traceability analysis and reviews to find issues and provide other information about software products.

Static testing is a form of testing and will often be included as part of a test strategy, and, as such, is in scope for this standard. Other international standards exist where techniques that can be used to accomplish static testing are described. This clause introduces the concept of static testing and shows the relationship with these other standards.

Review techniques and processes useful in software development are defined in the IEEE 1028:2008. Verification and validation activities are defined in IEEE 1012 (see Annex A). The main purpose of any of the static test design 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. Inspections may also serve the purpose of preventing future defects. The type(s) of static test design technique(s) to choose in any particular situation does not depend so much on the type of the test item, as on the risks involved (the higher the risk, the more formal static test design technique is recommended) and the secondary purpose of the static test design technique.

While static testing will often be within the scope of an organization's or project's software testing activities, it is only dealt with in the ISO/IEC/IEEE 29119 Software testing standards in detail in ISO/IEC/IEEE 29119-1 and by the above references. Techniques and processes for static testing are not addressed in the other ISO/IEC/IEEE 29119 Software testing standards.

5.5.6.2 Dynamic Test Design Techniques

Test design techniques for dynamic testing are techniques for identifying test conditions, test coverage items and subsequently test cases to be executed on a test item. The dynamic test design 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. ISO/IEC/IEEE 29119-4 describes each of the dynamic test design techniques in detail.

Specification-based test design 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 design technique(s) to use in any particular situation depends on the nature of the test basis and/or the test item and on the risks involved. Examples of specification-based test design techniques covered in ISO/IEC/IEEE 29119-4 are Boundary Value Analysis, State Transition Testing and Decision Table Testing.

Structure-based test design techniques are used to derive test cases from a structural feature, for example the structure of the source code or a menu structure. If these techniques are applied to application source code, then the expected results for test cases are derived from the test basis. The choice of which structure-based test design technique(s) to use in any particular case depends on the nature of the test basis and on the risks involved. These techniques are defined and described in detail in ISO/IEC/IEEE 29119-4 Test Techniques. Examples of structure-based test design techniques covered in ISO/IEC/IEEE 29119-4 are Branch Testing, Condition Testing and Data Flow Testing.

5.6 Test Practices

5.6.1 Introduction

The risk-based approach to testing, as described in clause 5.4, has been widely adopted and is the fundamental approach for the ISO/IEC/IEEE 29119 set of standards. There are a number of different practices for planning and implementing testing for a project. The traditional practice has been to base tests on requirements, derive test cases manually ahead of test execution, and use a mixture of manual and automated test execution. The use of risk-based testing does not mean that these practices cannot be used as a part of a test planning by those wishing to claim conformance to ISO/IEC/IEEE 29119-2. The choice of test strategy is determined by a variety of risks, such as to the organisation, the project and the test item. As an example, an organisation may face the risk of breaching a contract if it does not ensure that every

requirement is tested. Therefore using a requirements-based practice would be a way to manage this organisational risk. This clause introduces a number of test practices, each of which can be used as part of a test strategy created in conformance to ISO/IEC/IEEE 29119-2. Generally it is unlikely that any of these test practices would be used in isolation, but would instead be used as part of a larger test strategy.

This clause outlines some of the various test practices in use today to demonstrate some of the options available during test planning.

5.6.2 Requirements-Based Testing

The main purpose of requirements-based testing is to ensure that the requirements of the test item have been addressed (i.e. "covered") during testing, to determine whether the test item meets end-user requirements. In addition, it is used to collect and provide other valuable information to stakeholders during the life cycle, such as defects identified in the test item. In using this practice requirements are used to inform decision making around test planning, test design and implementation and execution processes. Note that Requirements-based testing can be used, in part, to accomplish the requirements verification outcomes outlined in ISO/IEC 12207.

Requirements-based testing predominantly uses scripted testing. Test cases are written prior to test execution, during the Test Design and Implementation process. The test cases are then executed during the Test Execution process according to the schedule defined in the Test Procedure. Analysis of the results of test execution are then used to further refine the testing, possibly leading to further test design and test case scripting. The test cases created during this further test design are documented and executed later in the test execution life cycle.

Requirements-based testing can be supported by other test practices with the intention that they will help to demonstrate that the requirements have been adequately tested. In addition to using requirements-based testing to ensure all requirements have been met, other practices (i.e. experienced-based testing) may be used to address other risks. For example, it is unusual for there to be a requirement that there are no regression defects left in a delivered product. However exploratory testing may be used as a way to address this regression risk.

The practicality of using requirements-based testing is highly-dependent on the context. If the requirements are incomplete or not consistently specified then the resultant testing may suffer in a similar manner. Even where requirements are well-specified, there is the danger that budgetary and time constraints may mean that it is not possible to test all requirements. When requirements are supplemented with information on their relative priorities then this can be used as a means of prioritizing the testing (in which case a risk-based approach can be used to prioritise the higher priority requirements). In practice, testers using requirements-based testing will often supplement the basic requirements in this manner so that the most important (highest risk) requirements are tested more thoroughly and earlier.

5.6.3 Model-Based Testing

All testing uses the concept of a model representing the test item's expected behaviour being available as the test basis. This model may be represented in the form of natural language requirements, mental images, mathematical equations, graphical notations (e.g. state transition diagrams, UML diagrams) or matrices (e.g. decision tables) – or, more typically, a mix of these. Traditionally, the tester uses the model to manually derive test inputs and expected results when performing specification-based testing, and the expected results when performing structure-based testing (here the test inputs are derived by considering the structure of the test item). Subsequently the tests are either executed manually or using testing tools.

Model-based testing uses a fundamentally different practice, but still based on a model of the expected behaviour. The difference is that with model-based testing the model has to be formal enough and/or detailed enough so that useful test information can be derived from the model. This model information can include test plans, design, procedures, inputs, risks, and/or oracles. Advantages of using model-based testing include the generation of test information, improved levels of automation in the test lifecycle (planning to documentation), and early identification of some kinds of errors. A consequence of this level of automation is that potentially millions of test cases can be automatically generated, executed and checked in a relatively small amount of time.

A typical context for using model-based testing would be for a critical application where a failure could lead to a large loss and where the application's required behaviour is amenable to modelling in a form that can be used by a model-based testing tool (and skilled staff are available to create and update the model). The UML graphical notations are often used as the basis for these models, although other notations are also used. Another consideration supporting its use would be that future maintenance of the application is expected on a regular basis and that modifying the model will be easier than updating automated test scripts (each time the model is updated the model-based testing tool could subsequently generate and run many tests at relatively low cost).

Using a model can provide benefits in other areas. For example automatic source code generation based on the model can reduce the instance of certain types of defects (i.e. mistyped code and defects which can be detected by model-checking) created during development. This helps the test process by reducing the number of defects in test items introduced into testing from the development process.

5.6.4 Mathematical-Based Testing

Mathematical-based test practices can be used to plan, design, select data and set up input conditions when the test item input or output space can be described in sufficient detail. Mathematical practices use branches of mathematics to drive the approach. These practices can help to reduce the human bias problem of test case selection and prioritization.

Mathematical based practices use a number of techniques, but in particular it uses of the following test design techniques (which are described in ISO/IEC/IEEE 29119-4):

— Combinatorial testing;

— Random test case selection;

In addition, statistical modelling can be used to determine test coverage (not addressed in ISO/IEC/IEEE 29119-4), for example:

— Stratified sampling;

— Fuzz testing;

— Cluster sampling; and

— Expert judgment sampling.

Mathematical-based test practices can also be used to objectively sample the test case space using mathematically- based tools and techniques.

Due to the large numbers of inputs that can typically be generated using mathematical practices, automated tools are usually needed to employ this practice.

5.6.5 Experience-Based Testing

Experience-based testing is based on:

— Previous testing experience;

— Knowledge of particular software and systems;

— Domain knowledge; and

— Metrics from previous projects (within the organization and from industry).

ISO/IEC/IEEE 29119-4 describes the experience-based test design technique of Error Guessing. Other experience-based test practices include (but are not limited to) Exploratory Testing, Software Attacks and Ad Hoc Testing. Exploratory testing and software attacks are not included in ISO/IEC/IEEE 29119-4 as test design techniques as they are practices where a number of test design techniques can be applied.

Exploratory testing compresses the activities described in the ISO/IEC/IEEE 29119-2 'Test Design and Implementation' and the 'Test Execution' processes so that they are performed dynamically to accomplish the test objectives. When using exploratory testing, test cases are not normally designed and documented in advance, but the tester uses intuition, curiosity and the results of previous tests to decide what and how to test next.

Exploratory Testing, Error Guessing and Ad Hoc Testing are test practices that do not rely on large amounts of documentation (e.g. test procedures) to execute testing. On a continuum from scripted to unscripted testing, these experience-based test practices are primarily unscripted. Using such practices may mean that only tailored conformance to ISO/IEC/IEEE 29119-2 can be achieved.

5.6.6 Scripted and Unscripted Testing

The following table describes the advantages and disadvantages of Scripted and Unscripted Testing. It should be noted however that the two practices are not mutually exclusive on a project; they are most often combined to create a hybrid practice based on the risk level of the test item.

When deciding whether to use scripted testing, unscripted testing or a hybrid of both, the primary consideration is the risk profile of the test item. For example, a hybrid practice might use scripted testing to test high risk test items and unscripted testing to test low risk test items on the same project.

Table 1 — Contrast of scripted and unscripted practices in test execution

	Advantages	Disadvantages
Scripted Testing	Testing is repeatable; the test cases can simply be run again, thus providing good auditability for verification and validation activities. Scripted test cases can be explicitly traced back to requirements allowing test coverage to be documented in a traceability matrix. Tests cases can be retained as reusable artefacts for the current and future projects, possibly reducing the time required for future Test Design and Implementation.	Typically more time-consuming and costly than unscripted test execution; however if scripted test cases are reusable this may lead to savings over time. Test cases defined prior to test execution are less able to adapt to the system as it presents itself. Can be less stimulating for the test executors as most of the analysis work has already been completed. This can lead to testers losing focus and missing details during test execution.

	Advantages	Disadvantages
Unscripted Testing	Testers are not constrained by a script and can follow ideas generated by test execution in real time. Testers can tailor 'Test Design and Implementation' and 'Test Execution' to the behaviour of the system in real time. The test item quickly can be explored quickly.	Tests are not generally repeatable. The tester must be able to apply a wide variety of test design techniques as required, so more experienced testers are generally more capable of finding defects than less experienced testers. Unscripted tests provide little or no record of what test execution was completed. It can thus be difficult to measure the dynamic test execution process, unless tools are used to capture test execution.

5.7 Automation in Testing

It is possible to automate many of the tasks and activities described in the Test Management and Dynamic Testing processes outlined in ISO/IEC/IEEE 29119-2, as well as aspects of testing techniques in ISO/IEC/IEEE 29119-4. Test automation can be used to support the test practices outlined in clause 5.6 (e.g. Model-Based Testing is nearly always based on the use of automated test execution tools). The automation of testing requires the use of software usually referred to as test tools. Automated testing is often considered to be mainly concerned with the test execution of scripted tests rather than have testers execute tests manually, however many additional test tasks and activities can be supported by software-based tools. The following list provides examples of some of the areas covered by test tools:

— Test case management;

— Test monitoring and control;

— Test data generation;

— Static analysis;

— Test case generation;

— Test case execution;

— Test environment implementation and maintenance; and

— Session-based testing.

There are a wide variety of test automation tools available. They can be developed in-house, obtained commercially, or obtained from the open source community.

5.8 Defect Management

Project Test Managers are often responsible for defect management (also known as Incident Management) on a project. Defect management is covered in ISO/IEC/IEEE 29119-2 to support testers in the investigation and documentation of test incident reports and the retesting of defects when required. All other elements of defect management are not directly covered by the ISO/IEC/IEEE 29119 set of standards. Defect management concepts and processes can be found in the following standards: ISO/IEC 12207; ISO/IEC 20000; IEEE 1044.

Annex A
(informative)

The Role of Testing in Verification and Validation

Figure 11 defines a classification of verification and validation (V&V) activities. V&V can be done on system, hardware and software products. These activities are defined and refined in IEEE 1012 and ISO/IEC 12207. Much of V&V is accomplished by testing. The ISO/IEC/IEEE 29119 set of standards address the Dynamic and Static software testing (directly or via reference), thus covering parts of this verification and validation model. The ISO/IEC/IEEE 29119 set of standards are not intended to address all the elements of the V&V model, but it is important for a tester to understand where testing fits within this model.

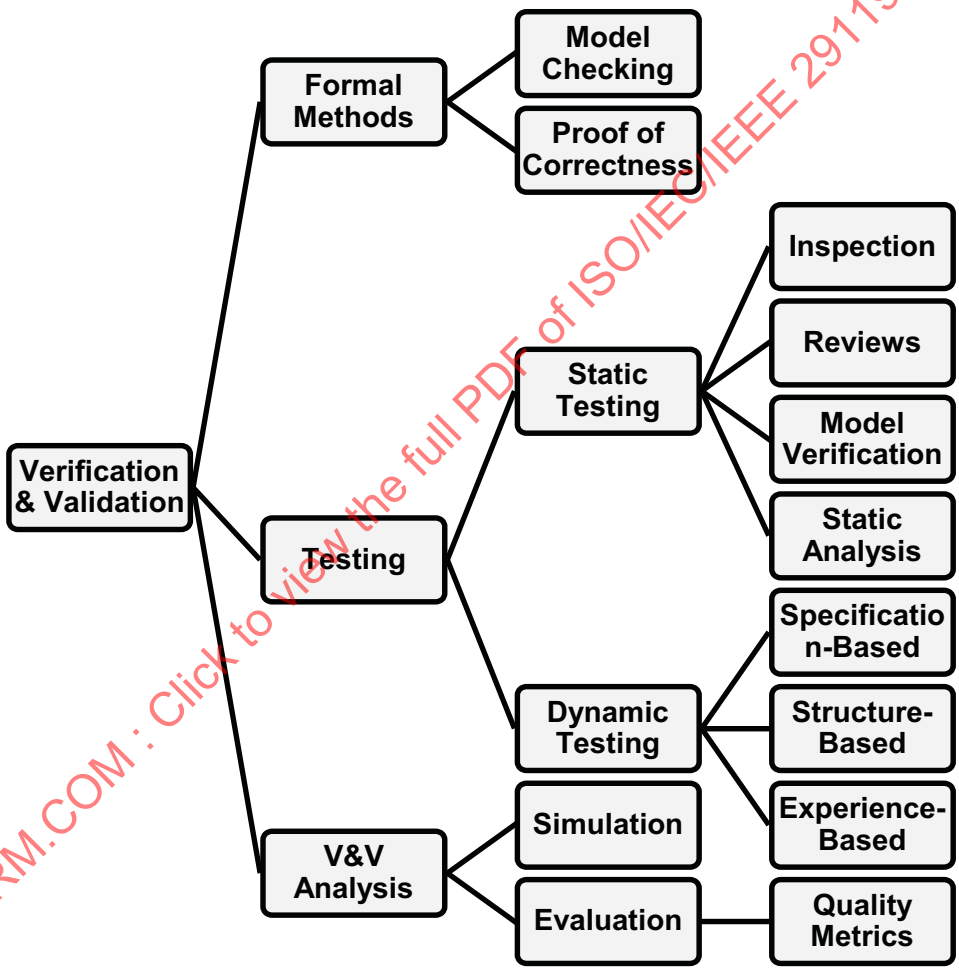

Figure A.1 — Hierarchy of Verification and Validation activities

Annex B
(informative)

Metrics and Measures

B.1 Metrics and Measures

The primary aim of testing is to provide information to help manage risks. In order to monitor and control testing and provide timely information to stakeholders it is necessary to effectively measure the test process. To measure the test process it is necessary to define what information to provide and how to obtain and present it. Thus all test efforts need to define and use metrics and provide measures in respect of both products and processes.

Some examples of metrics that can be used in testing are:

— Residual risk; number of risks mitigated/number of risks identified;

— Cumulative defects open and closed; number of defects opened each day compared to number of defects closed each day;

— Test case progress: number of test cases executed/number of test cases planned for execution; and

— Defect detection percentage: number of defects found in testing/number of defects found (overall).

Annex C
(informative)

Testing in Different Life Cycle Models

C.1 Overview

This annex gives examples of how testing might fit into different software development life cycle 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 life cycle models are classified here, in alphabetical order (and in the reverse order of their coming into existence), although this is not a comprehensive list.

— Agile

— Evolutionary

— Sequential (i.e. the waterfall model)

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

NOTE It is not the intention of this standard to standardise different lifecycle models; rather the intention is to establish the context for testing to aid those implementing these life cycles.

C.2 Agile Development and Testing

C.2.1 Agile Development Principles

Agile development typically follows these basic principles:

— Incremental development – each cycle delivers 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 (e.g. 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/). There is insufficient time and space to provide examples of this standard being implemented with each different Agile

method and framework 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 normally resulting in new functionality that can be delivered to the users (see Figure C.1). 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.

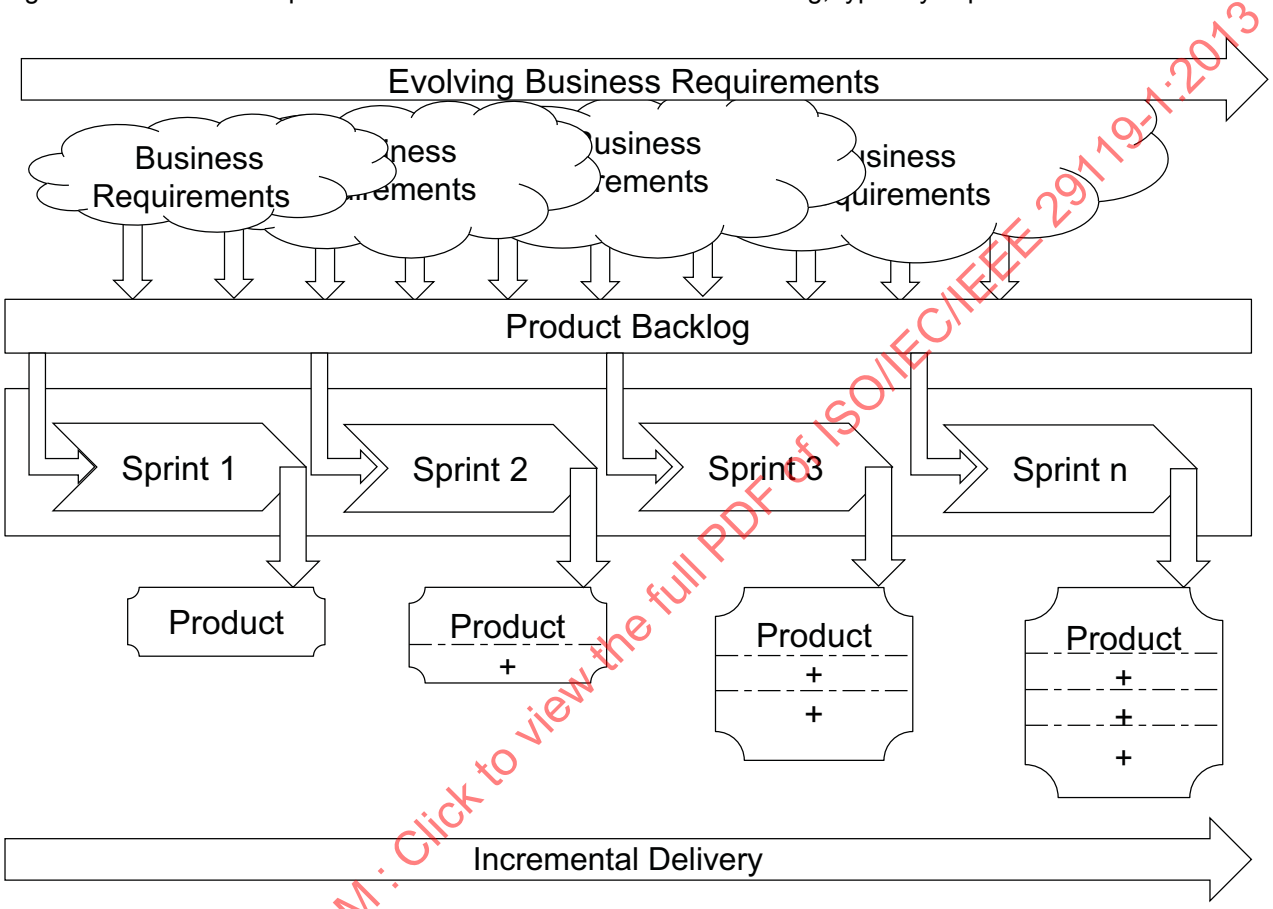

Figure C.1 — Example Scrum (Agile) project life cycle

The testing process model defined in this standard can be applied to the testing that is carried out in projects following agile development models.

C.2.2 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 primarily on the risk profile for the projects and products it covers (though the type of development model used may create additional types of risk that the Test Strategy must also address).

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) agree 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 C.2 below:

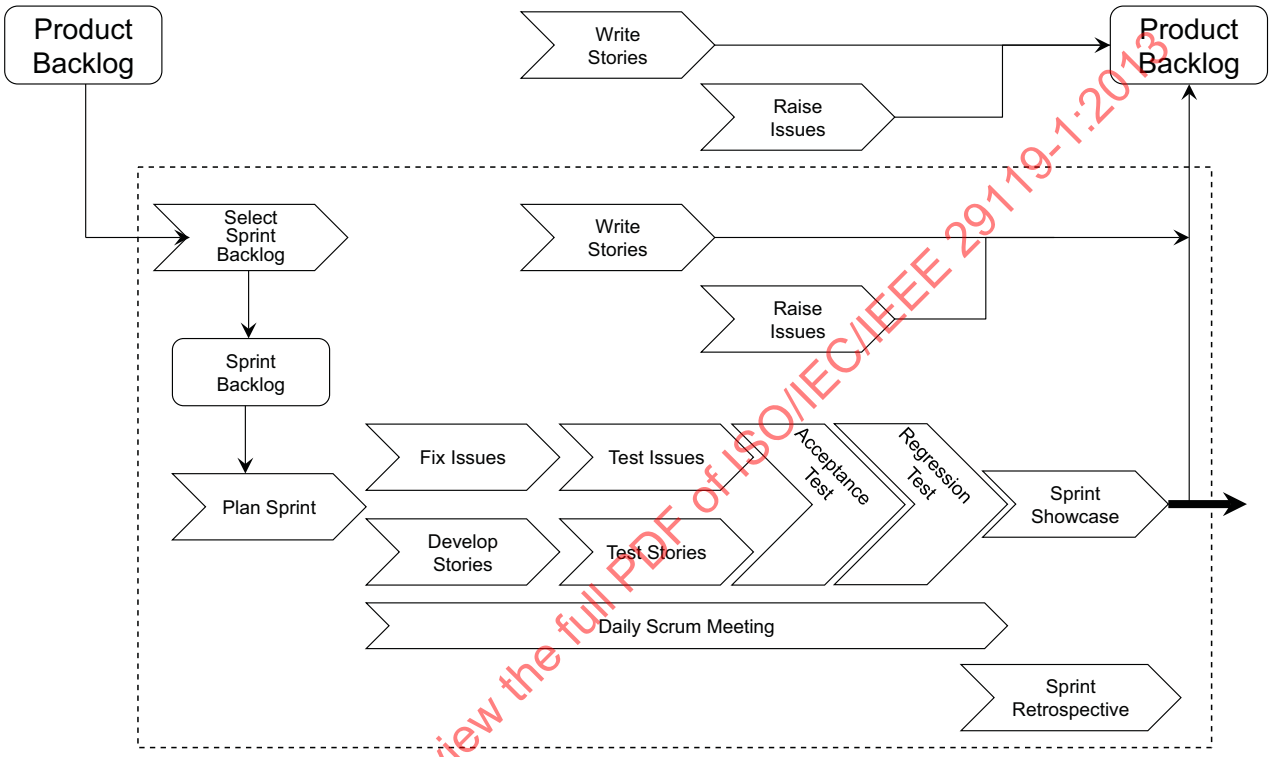

Figure C.2 — Example Agile Sprint life cycle

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 has been done; what is being done that day; and any issues the team faces. 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 having a lack of detailed requirements.

C.2.3 Test Sub-processes in Agile Development

The test activity is an integrated part of an agile development project, as shown in Figure C.2, 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 user 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.

— Testing of 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 user 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 determine 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 C.2). 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.3 Sequential Development and Testing

C.3.1 Sequential Development Principles

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 that 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.

C.3.2 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 development model(s) used. 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 out 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 warranted for the test sub-processes specified to be performed in the Project Test Plan. Plans are usually formally documented in sequential development.

C.3.3 Test Sub-processes in Sequential Development

The test sub-processes described for evolutionary development in Annex C.3 are also relevant for testing in a sequential project, although they are only performed once (there is only one pass through a sequential development model).

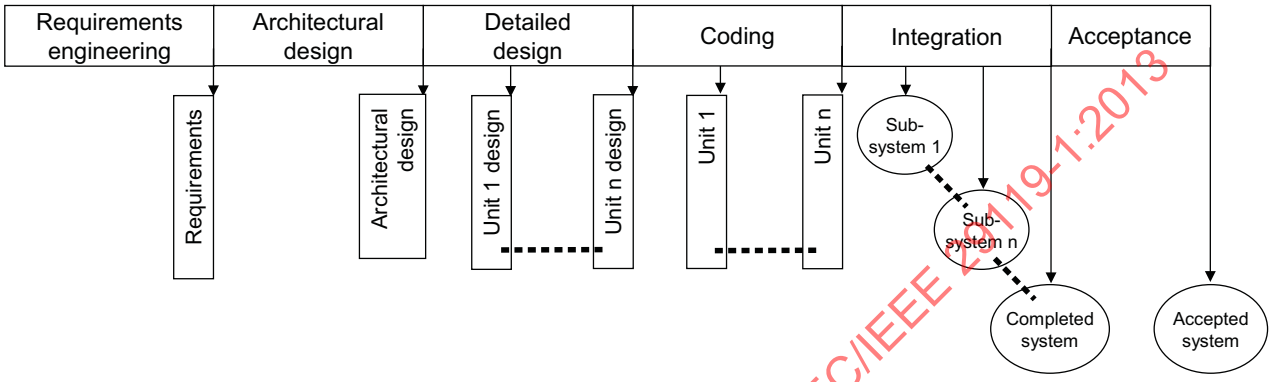

Figure C.3 — Test sub-process examples in Sequential Development life cycle

Note that Figure C.3 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 testing, Detailed Design testing and Source Code testing are defined. For each of these the corresponding development phase may be signed off on the basis of the results of the completed testing, i.e. when all the detailed developed items have been tested individually.

For the coding phase there are two different test sub-processes. The first is a source code test sub-process, constituted of static testing of the source code, the second is the component testing, constituted of dynamic testing of the executable or interpretable code. These test sub-processes 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, presuming they have already 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.

C.4 Evolutionary Development and Testing

C.4.1 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 higher 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.

Figure C.3 in C.3.3 shows a sequential life cycle. An evolutionary development life cycle can be characterised as a number of discrete sequential life cycles that each adds further capability to the system being developed.

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

C.4.2 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 development model(s) used. 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 primarily on the risk profile for the projects and products it covers (though the type of development model used may create additional types of risk that the Test Strategy must also address).

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 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 are taken on an on-going basis and are communicated to the stakeholders in updated Test Plan(s).

C.4.3 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 executable work products and the system being produced.

The first development phase in this example is requirements engineering, where the business, functional/non-functional, and system requirements are being specified. Figure C.3 only shows the test sub-process related to the first phase (requirements 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 testing. 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 design 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 C.3 this might include:

— Architectural design testing;

— Detailed design testing;

— Source code testing.

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 C.3. The planning and preparation activities in the acceptance test sub-process can start as soon as the risk of significant change in the requirements for this iteration 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 C.3 this might include the following sub-processes, which are similar to the acceptance sub-process:

— Component testing;

— Integration testing;

— System testing.

In the example in Figure C.3 a performance testing 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-processes may stretch over a number of development phases, and hence may have a variety of test items and associated test bases, test responsibilities, techniques, and environments, and test objectives, completion criteria, and plans, 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. 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.

Annex D
(informative)

Detailed Test Sub-process Examples

D.1 Overview

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 testing;

— Detailed design testing;

— Integration testing;

— Performance testing;

— Regression testing;

— Retesting;

— Story testing;

— System testing;

— Component testing.

Note that a regression testing and retesting 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.

For each static or dynamic test process planned for the test sub-process, the description includes:

— Test objective;

— Test item(s);

— Test basis;

— Applicable test processes;

— Suggested test design 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.

D.2 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 demonstrate to the customer the acceptability of the final system in terms of their specified requirements
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, user manuals, business process documentation
Dynamic test processes:	Test design and implementation; test environment set-up and maintenance; test execution; and test incident reporting.
Test design 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 design 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'.

D.3 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).