Standards in Software Test Automation

This is the fourth booklet in a series from the

EUREKA ITEA 3
TESTOMAT Project
The Next Level of Test Automation

Please follow us on:

Web:https://www.testomatproject.eu/
Twitter: @TestomatProject
YouTube:https://www.youtube.com/testomatproject

Content

1. What is the purpose of the standards?

There are many types of organizations producing software and development methodologies e.g., information technology (IT), personal computers (PC), embedded, mobile, and scientific, Agile, waterfall, large and small production organizations. These factors influence software testing as well as if a standard should or would be used [see ref 2].

A standard is an agreed way of doing something, e.g. a product, managing a process, delivering a service.
Standards can cover a vast range of activities and specify the smallest detail. The software industry, as one of the fastest growing businesses over the

last two decades, is also facing the problem of global competition.

The selection of standard is a crucial decision affecting the future of the organization in particular, and industry in general. Many of them provide a good learning of process and best practice.

For example, Software Quality Management (SQM) is to manage the quality of software and its development process. Then (QMS) can help in every aspect, e.g. organizational structure, procedures, processes and resources needed to implement in order to achieve quality management, [see ref. 12].
In automated testing for quality standards, different standards are used

for different type of domains. E.g. the manufacturing industry must meet strict quality standards.

Often these mandatory standards boil down to ensuring that a certain quality property is present or absent, regardless of the system run.

In general standards, the usage of following a standard has become more of a marketing advantage on a volunteer basis, than a mandatory fulfillment.

Following a standard means a competitive advantage, and it is common that companies follow the main stream in their area of working. In some domains, it is very restricted by regulatory laws, e.g. to operate an

aircraft or drone and how their software systems work and are verified. This requires extensive testing.

For example governance systems often fall under specific national laws of what needs to be recorded from the system, and lately the GDPR[1] laws impact how data of systems needs to be taken care of, including access, and times of saving it, and much more.

Standards describe a current ‘body of knowledge’ that provides the basis for a professional discipline:

- Communication
- Common terminology
- Professional qualifications
- Certification/compliance schemes

[1] GDPR (The General Data Protection Regulation)

- Benchmark of ‘good industry practice’
- Contracts
- Interoperability and consistency.

These guideline documentations reflect agreements on products, practices, or operations by: national or international associations; recognized industrial, professional and trade associations or governmental bodies [see ref. 8]. As a result, these bodies are rarely based on solid scientific ground, and have sometimes also commercial intentions. The work in Hatcliff, [see ref. 3] remarks that industry’s capability to verify and validate critical systems has not kept up with the ever-increasing complexity of software- intensive systems. Specifically, it is claimed that mere compliance with existing standards,

techniques, and regulations cannot guarantee the safety properties of these systems.

2. Why is TESTOMAT Project interested in these standards?

Several techniques are becoming more automated in the last few years, such as testing, monitoring or customer feedback; however, the potential for automating quality properties analysis has not yet been sufficiently studied [see ref. 4].

In general, the Technology Readiness Levels (TRL) of the automated solutions in testing for quality standards vary depending on the domain, applicable standards, and quality attributes of interest. The goal for each tool targeted in TESTOMAT Project will

be to raise its TRL by at least one level. We estimate that current TRL levels for the knowledge providers are 3-4, thus it seems realistic that every tool could reach at least TRL levels 4-5. As a part of this project, we thought we would share our insights working with Standardizations.

3. Standardization Bodies

There are thousands of standards organizations around the world, and they can standardize pretty much anything to make life easier, safer, and more productive, or with the purpose of commercial intention to close alternatives out.

Figure 1. Standardization Bodies[2]

Often, these bodies have agreements to cooperate with each other. They may endorse each other's standards, build upon them, and/or purposely avoid duplicating efforts [see ref. 1]. A small selection of standardization bodies is shown in the Figure 1.

[2] Taken from *https://www.bcs.org/upload/pdf/sreid-120913.pdf*

4. General Testing Standards

The development of software quality and safety standards has given momentum to recent developments in software testing. The standard ISO 25010, derived from earlier standard ISO/IEC 9126, contains the characterizes quality hierarchically.

There are 8 major headings: functional suitability, reliability, usability, performance efficiency, maintainability, and compatibility, with 26 sub-headings.

Since ISO 25010 includes issues such as security, environmental, and health and safety *risk*, it partly overlaps software safety certification standards. See later about these. In fact, ISO 25010 merely defines quality attributes

and 25023 describes some simple one-tier metrics.

While ISO 25020 does include a measurement reference model, market analysis has shown that less than 28% of companies use the standard models and 71% have developed their own variant [see ref. 13]. Therefore, it is generally considered necessary to build experimental research tools to integrate a quality model with a practical assessment toolkit. Such research has led to the quality dashboard concept; presenting data provided by quality measurement tools (e.q. QALab, Sonar, XRadar) with the goal of providing an overview of the quality data of a software system.
Unfortunately, current dashboards generally lack a connection between

metrics used and quality factors, thus impacts and rationales for the metrics are missing. On the other hand, a couple of landmark models do integrate measurements and quality factors, e.g. COQUAMO [see ref. 14], COQUALMO [see ref. 15] and the QUAMOCO project which is the benchmark for software quality Quamoco. Unfortunately, these models would be important but are not updated to modern agile and DevOps software development, and therefore partly obsolete and in need of new scientifically sound metrics.

Many organizations around the globe develop and implement different standards to improve the quality needs of their software. This following chapter briefly describes some of the widely

used standards related to Quality Assurance and Testing.

Software Testing ISO Standards:
ISO/IEC 9126 (that is now obsolete and replaced by the ISO/IEC 25000-series). This standard deals with the following aspects to determine the quality of a software application: Quality model, External metrics, Internal metrics and Quality in use metrics.

This standard presents some set of quality attributes for any software such as: Functionality, Reliability, Usability, Efficiency and Maintainability.

ISO/IEC 9241-11: This standard deals with the extent to which a product can be used by specified users to achieve specific goals with Effectiveness,

Efficiency and Satisfaction in a specified context of use. This standard proposed a framework that describes the usability components and the relationship between them. In this standard, the usability is considered in terms of user performance and satisfaction.

ISO/IEC 25000-series

ISO/IEC 25010 (2011) - Systems and Quality Models is commonly known as the standard providing guidelines for Software Quality Requirements and Evaluation, **SQuaRE**. It aimed to replace ISO/IEC 9126. This standard helps in organizing and enhancing the process related to software quality requirements and their evaluations.

SQuaRE is divided into sub-parts such as:

- ISO 2500n - Quality Management Division
- ISO 2501n - Quality Model Division
- ISO 2502n - Quality Measurement Division
- ISO 2503n - Quality Requirements Division
- ISO 2504n - Quality Evaluation Division

The main contents of **SQuaRE** are:

- Terms and definitions
- Reference Models
- General guide
- Individual division guide
- And the actual standard which is related to Requirement Engineering (i. e. specification,

planning, measurement and evaluation process)[3].

ISO/IEC 25020 - Measurement reference model and guide: This standard provides an explanation and a reference model that is common to quality measure elements. It provides measures of software product quality and quality in use and guidance to users for selecting or developing and applying measures.

ISO/IEC 25021 - Quality measure elements: Defines a set of recommended base and derived measures, which are intended to be used during the whole software development life cycle. The document

[3] Software Testing - ISO Standards

describes a set of measures that can be used as an input for the software product quality or software quality in use measurement.

ISO/IEC 25022 - Measurement of quality in use: Describes a set of measures and provides guidance for measuring quality in use through five areas: Effectiveness, Efficiency, Satisfaction, Freedom from Risk, and Context Coverage, with its 11 sub-characteristics.

ISO/IEC 25023 - Measurement of system and software product quality: Describes a set of measures and provides guidance for measuring system and software product quality. It contains definitions and suggested measurements of System/Software

Product Quality characteristics: Functional suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability, and Portability. Each of these characteristics is further divided into 31 sub- characteristics. See Figure 2.

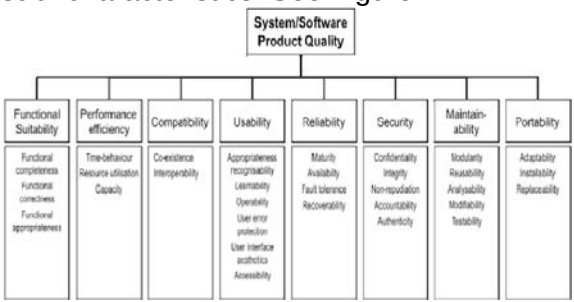


Figure 2. System/Software Product Quality

ISO/IEC 25024 - Measurement of data quality: Defines quality measures for quantitatively measuring data quality in

terms of characteristics defined in ISO/IEC 25012.

ISO/IEC 25030 - Quality Requirements: Provides requirements and guidance for the process used to develop quality requirements, as well as requirements and recommendations for quality requirements.

ISO/IEC 25040 - Evaluation reference model and guide: Contains general requirements for specification and evaluation of software quality. Provides a framework for evaluating the quality of software product and states the requirements for methods of software product measurement and evaluation.

ISO/IEC 25041 - Evaluation guide for developers, acquirers and independent evaluators: Provides requirements, recommendations and guidelines for developers, acquirers and independent evaluators of the system and software product.

ISO/IEC 25042 - Evaluation modules: Defines the structure and content of the documentation to be used to describe an evaluation module. These evaluation modules contain the specification of the quality model, the associated data and information about its application.

ISO/IEC 25045 - Evaluation module for recoverability: Provides the specification to evaluate the sub characteristic of recoverability defined

under the characteristic of reliability of the quality model (ISO 25000).

ISO/IEC 12119: This standard deals with software packages delivered to the client. It does not focus or deal with the clients' production process. The main contents are related to the following items:

- Set of requirements for software packages.
- Instructions for testing a delivered software package against the specified requirements (Software Testing - ISO Standards).

ISO/IEC/IEEE 29119 Software and Systems Engineering Software Testing: Is an attempt to unify several standards (the old IEEE Std 829, and BS7925-1 and 2).

- Part 1: Concepts and definitions

- Part 2: Test processes
- Part 3: Test documentation
- Part 4: Test techniques
- Part 5: Keyword-driven testing ISO/IEC/IEEE 29119-4:2015 [see ref. 8]

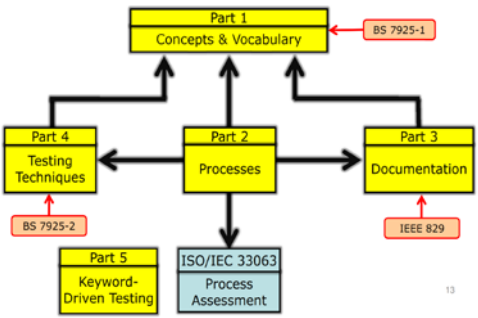


Figure 3. ISO/IEC 29119 –Structure[4]

Even if this standard is recent, it misses several scientific established and

[4] Taken from *https://www.bcs.org/upload/pdf/sreid-120913.pdf*

automatic approaches, like e.g. search-based testing, constraint modelling etc. It further lacks focus on e.g. test automation, concurrency (hardware) etc.

ISO/IEC/IEEE 29119-1 Concepts and definitions: Facilitates the use of the other parts of the standard by introducing the vocabulary on which the standard is built and provides examples of its application in practice. Part 1 provides definitions, a description of the concepts of software testing, and ways to apply these definitions and concepts to the other parts of the standard [see ref. 8].See figure 4.

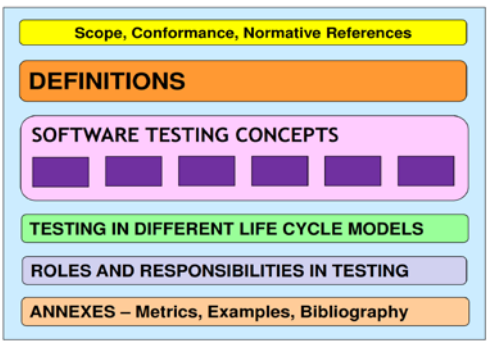


Figure 4. Concepts & Vocabulary[5]

ISO/IEC/IEEE 29119-2 Test processes: Part 2 defines a generic test process model for software testing that is intended for use by organizations when performing software testing. See figure 5. It comprises test process descriptions that define the software

[5] Taken from *https://www.bcs.org/upload/pdf/sreid-120913.pdf*

testing processes at the organizational level, test management (project) level, and dynamic test process levels. The processes defined in this standard can be used in conjunction with different software development lifecycle models [see ref. 8]

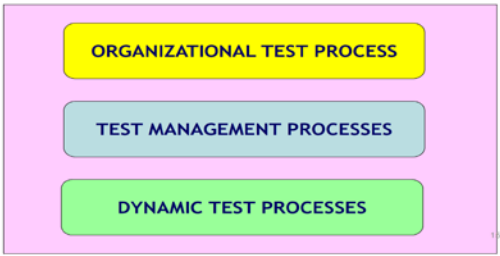


Figure 5. Testing Processes[6]

[6] Taken from *https://www.bcs.org/upload/pdf/sreid-120913.pdf*

ISO/IEC/IEEE 29119-3 Test documentation: Part 3 deals with software test documentation and includes templates and test documentation examples that are produced during the test process (based on IEEE Std 829).

The documents that are defined in ISO/IEC/IEEE 29119-3 are as follows:
Organizational Test Process Documentation:

- Test Policy
- Organizational Test Strategy

Test Management Process Documentation:

- Test Plan (incl. a Test Strategy)
- Test Status
- Test Completion

Dynamic Test Process Documentation:

- Test Design Specification
- Test Case Specification

- Test Procedure Specification
- Test Data Requirements
- Test Data Readiness Report
- Test Environment Requirements
- Test Environment Readiness Report
- Actual Results
- Test Result
- Test Execution Log
- Test Incident Report [see ref. 8]

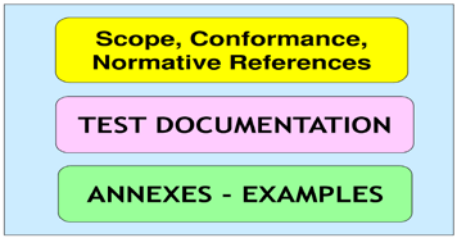


Figure 6. Test Documentation[7]

[7] Taken from *https://www.bcs.org/upload/pdf/sreid-120913.pdf*

ISO/IEC/IEEE 29119-4 Test Techniques: Part 4 provides standard definitions of software test design techniques and corresponding coverage measures that can be used during the test design and implementation processes.

The standard's test design techniques are categorized into three main categories: Specification, Structure and Experience-Based Test Design Techniques. Note that particularly this standard is not up to date on automatic and more recent test design techniques like mutation testing, search-based testing, property-based testing and similar approaches.

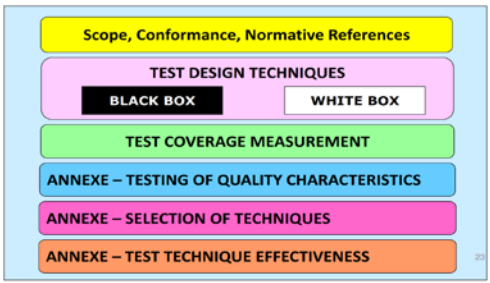


Figure 7. Test Techniques[8]

I**SO/IEC/IEEE 29119-5 Keyword-driven testing:** This standard covers so called keyword-driven testing, which is basically an acronym of using macro's, and as such simplifying a test procedure. This standard is intended for users "who want to create keyword-driven test specifications, create

[8] Taken from *https://www.bcs.org/upload/pdf/sreid-120913.pdf*

corresponding frameworks, or build test automation based on keywords [see ref. 8].

IEEE 829 Test documentation: IEEE 829-2008, also known as the 829 Standard for Software and System Test Documentation, was an IEEE standard that specified the form of a set of documents for use in eight defined stages of software testing and system testing, where each stage potentially producing its own separate type of document. This standard is included in ISO/IEC/IEEE 29119.

The main change is it encompasses details of what should be contained in a test plan, and that the master test plan guiding sub-test plans, to create a planning hierarchy. Of course, this is

obsolete as planning, but important, as it defines all stages (to be documented) for traceability in the test process.

IEEE 610 - Standard glossary of software engineering terminology: IEEE Std 610.12 is a revision and re-designation of IEEE Std 729. This standard contains definitions for more than 1000 terms, establishing the basic vocabulary of software engineering. Building on a foundation of American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) terms.

Unfortunately, very few of the IEEE standard vocabularies are aligned, and these vocabularies have not kept up with the area of software testing, or automation. Instead IEEE launched SWEBOOK to resolve the diversity.

1044-IEEE standard classification for software anomalies (2009): This standard provides for the core set of attributes for classification of failures and defects. It is recognized that there are other attributes of failures or defects that are of unique value to specific applications or business requirements.

The aim here is that this standard is applicable to any software (including operating systems, database management systems, applications, testware, firmware, and embedded software) and to any phase of the project, product, or system life cycle by which the software is developed, operated, and sustained. Classification attributes are unaffected by the choice of life cycle model, and that choice is outside the scope of this standard. Some tailoring of classification attribute

values based on the chosen life cycle is expected and consistent with the intent of this standard.

IEEE 106: A methodology for establishing quality requirements, identifying, implementing, analyzing, and validating the process, and product of software quality metrics.

IEEE 730: A standard for software quality assurance plans.

IEEE 830: A guide for developing system requirements specifications. This is a beginner guide, but lacks automation approaches, e.g. modelling of requirements.

IEEE 1008: A standard for unit testing. This standard has some old, but basic approaches for unit testing, addressing mostly procedural languages. And

missing most of the advancements of Static Analysis and more modern test technologies, e.g. mutation testing or search-based testing existing today, and as such basing itself in a relative manual approach.

IEEE 1012: A standard for Software Verification and Validation. This standard is on a very high level, and therefore easy to follow.

IEEE 1028: A standard for software inspections. There is a simple (time consuming) manual view of this process, originating from Fagan Inspections. It does not adhere to best practice.

IEEE 1059: Guide for Software Verification and Validation Plans.

IEEE 1061: A standard for software quality metrics and methodology. This standard has also a mark of being too old.

IEEE 12207: A standard for software life cycle processes and life cycle data.

BS 7925-1: Yet another vocabulary of terms and definitions used in software testing.

BS 7925-2: A standard for software component testing now in (Software Testing - ISO Standards). Basic metrics for testing simple coverage.

Both of the BS standards has been included in the new ISO/IEC/IEEE 29119 standard.

5. Model Based Testing Standard - UML

5.1 OMG UML Testing Profile

The OMG UML Testing Profile standards (i.e., version 1.x - the latest published version was 1.3 - and version 2.0) are standardized test modeling language based on UML for designing, visualizing, specifying, analyzing, constructing, and documenting relevant artifacts of various testing approaches. Since UML lacks dedicated testing concepts, UTP closes this conceptual gap of UML on one hand by introducing dedicated concepts on abstract syntax level. On the other hand, it supports a visual and (semi-)formal approach of building test specifications.

UTP is independent of any particular methodology, thus, enabling its

application in various domains and project contexts. Furthermore, UTP reuses UML's graphical syntax (diagrams) in order to leverage the well-established notation of UML for testing purposes. Clear notations help to overcome communication barriers among stakeholders. Since both system architects/engineers and test engineers leverage the very same underlying modeling language, a better understanding of the outcome of either side is fostered by UTP.

5.2 UML Testing Profile 1.x (UTP 1)

As far back as 2001, a dedicated working group at the OMG applied itself to collecting industry accepted testing practices and concepts in order to make them available for MBT approaches.

These efforts resulted in the adoption of the UML Testing Profile (UTP) 1.0 by the OMG in 2005 [2]. UTP 1.0 was the first UML profile that was standardized for the renewed UML 2.0; it was even released before UML 2.0.

The design of UTP 1.x was influenced by the concepts of TTCN-3. As a result, a mapping from UTP 1.x models to executable TTCN-3 test cases was a part of the standard.
UTP 1.x consists of five main parts: Test Architecture, Test Behavior, Timer, Test Data and Test Planning, herby described:

- **Test Architecture:** Offers concepts to model the interfaces to the test item (or system under test, SUT), the test environment, in particular by means of test components that run the test

cases, and the test configuration as part of the test context.

- **Test Behavior:** Offers concepts to define test cases, control statements and test logs.
- **Timer:** Offers concepts to model timers and the imperative use of them during test case execution.
- **Test Data:** Offers concepts to define test data and data pools. The Test Data package is rather concise, since concrete data values are already given by UML Value Specifications.
- **Test Planning:** Offers concepts to specify test objectives and the ability to trace test cases to test objectives and vice versa.

5.3 UML Testing Profile 2 (UTP 2)

In 2013, the renewal of UTP 2 started at OMG, initiated with a Request for Information (RFI) to solicit industrial

perception of UTP 1.x and needs for UTP 2.0. Similar to UTP 1.x, UTP 2.0 primarily supports the activities of a dynamic test process, in particular test analysis & design, test evaluation and the management of these activities. Test execution is not targeted by UTP in the first place, because UTP 2 is not intended to be precise, yet executable language. However, the generation of executable test cases Is one of the most important capabilities of UTP 2.
UTP 2 can be applied to:

- Specify the design and the configuration of a test system
- Build the model-based test plans on top of already existing system models.
- Model test cases & Model test data
- Model test environments
- Model deployment specifications of test-specific artifacts

- Provide necessary information pertinent to test scheduling optimization
- Document test case execution results
- Document traceability to requirements and other UML model artifacts.

UTP 2 offers a similar concept space as UTP 1.x, although it offers much more extensions. The main differences between UTP 1.x and UTP 2 are:

- Simplification of test context and test architectures.
- Provision of a dedicated test action language.
- Explicit support for test design.
- Explicit integration with SysML
- Flexible arbitration also for non-functional testing.
- Model-based test logging.

6. ETSI Standards

ETSI standards include world-leading communication protocols in the Telecom world (such as LTE, UMTS etc.) as well as standardized test suites for these protocols to simplify conformance testing for vendors. These standardized test suites are specified with two key technical testing standards:

- TTCN-3, and
- TDL

6.1 Testing and Test Control Notation v3 (TTCN-3)

The standardized Testing and Test Control Notation (TTCN-3) defines both a strongly typed, component-based test programming language and a test automation architecture that enables the implementation and automated

execution of test suites. TTCN-3 is standardized by ETSI as a successor of the TTCN (Tree and Tabular Combined Notation). Its main application area at ETSI was in particular for conformance, interoperability and performance tests of communication-based systems including protocols, services, and interfaces.

TTCN-3 is not a modeling language in the sense of SysML. The language is grammar-based and specified using extended Backus-Naur form. It comes along with an operational semantics standard, thus, TTCN-3 is an executable testing language. It defines its own type system, independent of any target platform or technology. Its language is platform independent.

This allows for implementing TTCN-3-based test automation solutions with varying programming languages retaining the ability to exchange TTCN-3 test scripts (called modules) among those test automation solutions.

The platform independency of TTCN-3 was one of the most important characteristics for ETSI to develop standardized conformance test suites in a platform-/technology -independent format. The TTCN-3 test automation architecture is highly modularized. TTCN-3 is not intended to be integrated with a model-driven software engineering approach, even if such implementations exist. It is a platform-independent specification of executable test suites.

TTCN-3 contains automated test execution of these test suites utilizing dedicated system adaptor (SA) implementations. System adaptors are responsible to transform the platform-independent types and actions into technical requests against the system under test and vice versa.

6.2 ETSI Test Definition Language (TDL)

The Test Description Language (TDL) is a language for the description of test models with emphasis on test data and test behavior including concepts of time. TDL defines a standalone metamodel, which resembles a subset of UML. The specification includes a comprehensive graphical syntax, which is largely based on UML, and a dedicated exchange format. Further,

there is an extension for structured test objective specifications.

Very recently, the TDL standards family has been enriched with a mapping to executable TTCN-3, and with a UML Profile for TDL.
TDL resembles UTP 1.x with respect to its goals.

- Specification of easy-to-understand test descriptions that can be presented in different representation formats suitable for different stakeholders (graphical, textual, user-specific)
- Specification of test objectives and development of tests by testers lacking programming skills
- Independence from execution languages and platforms and hiding of implementation details
- Iterative test development along all product development phases, from requirements disambiguation, via

design, system and acceptance testing
- Support of manually developed and automatically generated tests by a common platform.

Test cases described by TDL are restricted to sequence diagrams solely. Integration with SysML models is achieved using the UML Profile for TDL. A mapping to TTCN-3, which facilitates test execution, is also part of the standard. There is a lack of dedicated concepts for test design activities, test logging and test evaluation. Similar to UTP 1.x, the verdict calculation mechanism is static and cannot be adapted to additional needs.

7. Domain Specific Standards

7.1 Safety Critical Systems

The development of safety-critical software systems requires the introduction of mature development process into the organization and the use of acknowledged standards. Certification of such a system relies on the ability of the organization to demonstrate and document proof of the correct application of the processes and standards. Example of safety critical standards are ISO 26262 for automotive, IEC 62279 for rail and RTCA/DO-178C for avionics. Safety standards are strongly recommended or mandate specific testing and/or static analysis approaches or coverage models. These technologies are used for specific Safety Integrity Levels (SILs), i.e. levels that identify software

criticality. Furthermore, the safety standards place strong requirements on test processes and tools, before a safety case can be fully certified. Most Safety Critical Systems are regulated both nationally in regulatory bodies, and in organizations that are international.

For safety-critical systems, the certification process starts with the initial requirements analysis. A certification liaison person acts as the main communicator between the manufacturer of the system and the certifying authority. For the specific standards are mandatory, the liaison will make sure that these are documented from the very beginning of the project. A verification plan should be produced for approval by the

authority. During the project phases, appropriate documentation and supporting data should be produced and submitted to the authority. A series of reviews are conducted and if all terms of the verification plan are met, the certificate or license will be issue [see ref. 11].

In principle the main caveat in safety critical testing is keeping well described documentation and traceability to all artifacts. The test process and all testing activities are then duly performed and recorded.

In RTCA/ DO-178C The Test Process of assurance can be seen in Figure 8.

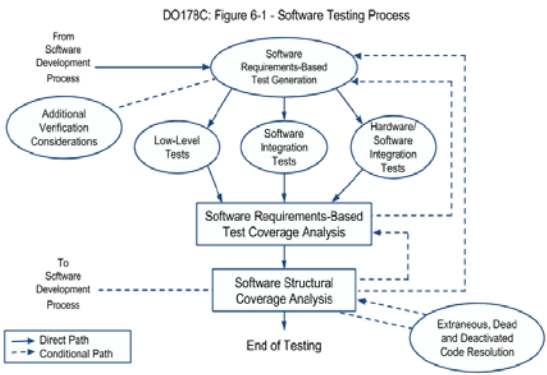


Figure 8. RTCA/DO-178C Test Process

In the TESTOMAT Project our Safety Critical parties, e.g. SAAB, Bombardier, Ponsse, FFT, are all aiming for the testing approach where we improved on the existing standard. Our partners

check that this way of testing fulfills the criteria for the regulatory authorities. In fact, in the TESTOMAT Project we have implemented new ways to test that find more bugs on top of these mandatory safety-critical standards. We introduced a new test design technology (i.e. mutation testing) that really pushes the limit of existing coverage requirements. Further we found set of automation approaches complementing non-functional testing, by using extensive simulations – not only saving costs, but also increasing the reliability. As a result, we have shown that these new approaches are feasible to use in commercial context, in real safety critical systems. By using our result, we produced safer systems at reasonable cost.

7. Conclusion and Discussion on Standards

In this booklet, we have pointed out that several techniques such as testing, monitoring or customer feedback, have been introduced in the last few years; however, the potential for automating quality properties analysis has not yet been sufficiently studied and is lacking in most of the existing standards (except a few). Standards describe a current 'body of knowledge' that provides the basis for a professional discipline (e.g., for communication, certification schemes, compliance schemes, interoperability and consistency).

We can conclude that these are often not updated with the latest research and scientifically proven technologies. It usually takes years for these scientific

results to be sufficiently cost efficient, implemented and adapted to reach common usage - and as such, make their way into the standards. We still feel that we should push further on our findings of safety-critical software, as this could potentially save lives. We have taken the liberty to comment on the actual status of some of these standards, where there is a know-how and experience of using them in the TESTOMAT Project. For beginners in the field, we think these standards provide a good starting point for knowledge.

In general, the Technological Readiness Level (TRL) of the automated solutions in testing for quality standards vary depending on the domain, applicable standards, and quality attributes of interest.

The implementation of standards into practice can be facilitated or hindered depending on the factors in the organization. The following list shows a few of these factors:

- Understanding the standards used;
- Educating users about the benefits and hazards of the standards;
- Using the right standards related to the application domain;
- Using the standards only if they provide a benefit; otherwise, they will not be useful;

On the other hand, using too many standards may adversely affect software quality [see ref. 7]

References:

1. Bartleson, K. (2013). “10 Standards Organizations That Affect You (Whether You Know It Or Not)”.

https://www.electronicdesign.com/communications/10-standards-organizations-affect-you-whether-you-know-it-or-not
2. Hagar, J. (2014). "The Art of Software Testing Standards". https://www.softwaretestpro.com/the-art-of-software-testing-standards/
3. Hatcliff, J., Wassyng, A., Kelly, T., Comar, C. and Jones, P. (2014). "Certifiably safe software-dependent systems: Challenges and directions". In Proceedings of the Future of Software Engineering, FOSE 2014, pages 182--200, New York, NY, USA.
4. Harman, M., Jia, Y. and Zhang, Y. (2015) "Achievements, open problems and challenges for search based software testing". IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pages 1-12, April 2015.
5. ISO 25000: https://iso25000.com/index.php/en/iso-25000-standards/54-iso-iec-2504n
6. ISO/IEC/IEEE 29119-4:2015. "Software and systems engineering — Software testing — Part 4: Test techniques)" https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29119:-4:ed-1:v1:en

7. Laporte, C.Y. and April, A. “Software Quality Assurance”. Wiley-IEEE Computer Society, Inc., 2018.
8. Reid, S. (2013). “ISO/IEC/IEEE 29119, The New International Software Testing Standards”. Testing Solutions Group, London. https://www.bcs.org/upload/pdf/sreid-120913.pdf
9. Software Testing - ISO Standards, http://www.tutorialspoint.com/software_testing/software_testing_iso_standards.htm
10. The benchmark for software quality Quamoco: www.quamoco.de
11. Traussing, R. (2004). “Safety-Critical Systems: Processes, Standards and Certification”. Analysis, Design and Implementation of Reliable Software” Seminar, Paderborn, Germany.
12. Yadav, M., Kumar, S. and Kumar, K. (2014). “Quality standards for a software industry –A review”. IOSR Journal of Computer Engineering (IOSR-JCE), V. 16, pp 87-94.
13. Wagner, S. (2013). “Software product quality control”. Springer-Verlag Berlin Heidelberg 2013, Springer, Berlin, Heidelberg
14. Roche, J. M. (1994). “Software metrics and measurement principles”, ACM SIGSOFT Software Engineering, Vol. 19, pp. 77-85.
15. Chulani, S. Boehm. B.; Modeling software defect introduction and removal: COQUALMO (COnstructive QUALity MOdel), by USC Center for Software Engineering, 1999.

Acknowledgements:
This booklet is produced by
EUREKA ITEA3 TESTOMAT PROJECT
The Next Level of Test Automation

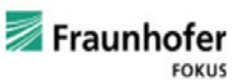
Find out about us on the

web:https://www.testomatproject.eu/

Follow us on Twitter @Testomatproject

This booklet is produced by a research collaboration between the following partners:

ERICSSON



ALERION

Version 1.1.1 - 2019-09-27
Feedback and comments are welcome

The TESTOMAT Project is Sponsored by:

Printed with support of ERICSSON