# Quality of MBT Test Models and Competences Needed for Creation of Good Models

## Or How Does ISO/IEC 25010 Help Us Understand What Good Test Models Are Like

Matti Vuori and Antti Jääskeläinen
Tampere University of Technology
Department of Pervasive Computing
Tampere, Finland
matti.p.vuori@tut.fi, antti.m.jaaskelainen@tut.fi

***Abstract*— Quality of test models is very critical for the tests generated in model-based testing and the management of the models during their whole life cycle. The quality characteristics of test models were analysed at abstract level using ISO/IEC 25010 quality model standard as a framework. Also, the necessary competences to produce the desired characteristics were assessed. The analysis demonstrates that ISO/IEC 25010 can be used as a quality framework for test models and that the quality characteristics of the models are very diverse and like those of application code, thus enabling the use of similar quality management principles and methods in assuring the quality of the test models. The results also provide guidance for building the competences of testers who perform model-based testing and modelling in particular, as a wide array of competences was noted that traditionally are not acknowledged. The results will aid in developing model-based testing technology that is better integrated to the activities and cultures of software engineering.**

***Keywords—model-based testing; modelling; quality models; ISO/IEC 9126; competence***

## I. INTRODUCTION

The test model is a key element in model-based testing (MBT) as it defines the expected behaviour of the system under test. For the purposes of this paper, we will consider the test oracle to be a part of the test model although in practice it might be a separate system. Most often, the qualities of the model such as completeness and coverage, as in [11], are considered, but as MBT is getting more widely used in the industry, more attention has been paid to issues that relate the whole lifecycle of the test model. Yet, thorough analyses of the quality factors are lacking, especially in a form of a complete quality framework or a quality model.

While little research has been done on developing quality models specifically for test models, there is some work regarding models used in model-driven engineering. Mohagheghi et al. developed such a quality model based on six different criteria [8]. Mohagheghi and Dehlen have also assessed the requirements of a quality framework in model-driven engineering [9]. Apart from these generic quality models, Lemaitre and Hainaut have developed a domain-specific quality framework for data models [7].

Some methods have been developed for testing models. Hamann and Gogolla proposed the development of unit tests for UML models along the lines of xUnit frameworks, and argue that their use can improve the quality of the models [3]. Andrews et al. [1] and Mayerhofer [10] have also presented techniques for testing UML models. However, the narrow focus of these methods limits their ability to improve the overall quality of models.

For this assessment we use the quality model of ISO/IEC 25010 [5] as a framework and analyse how its quality factors would help us understand the quality of the test models. We also understand that test modelling is a task that requires many skills besides just modelling skills, so in order to educate and train the future test modellers, we assess what kind of competences – approaches, knowledge and skills – would be required or beneficial in the creation and management of good test models.

Yet, at this point we wish to remind the reader that there is more to an MBT system than just the models and handling them. Understanding of the quality of a test system should by definition start at the level of the overall test automation system, in which the MBT system is just one part. Besides the model, an MBT system has many elements critical to good testing, including the test execution engine, system under test (SUT) adaptation system, test coverage and strategy systems. Each of those requires attention, but now we concentrate on the test model.

One might ask what kind of context this paper applies in. Generally, for the ideas to have relevance, the context should have many of the following characteristics:

- MBT has a critical role in the testing process.
- It is performed in professional manner, in most projects.
- There is general maturity in the actions of the organization.
- The overall environment is one where "general understanding" of software engineering applies, that is, there are no radical requirements or practices in place.

The paper is structured as follows: In Section II we take a look into the reference standard that was selected. After that in Section III, to orient the reader to the analysis, we make a preliminary comparison between writing test models and software. That will help us later in reflecting the detailed findings to the overall nature of the work. Section IV contains the main analysis where we assess how modelling relates to the main quality characteristics of ISO/IEC 25010. After that, we wrap the main findings together and make conclusions about the issues.

## II. THE ISO/IEC 25010 STANDARD

ISO/IEC 25010 "Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models" is the latest international software quality model standard, very similar to ISO/IEC 9126 [6], which is very widely used globally. The standard presents a structure of quality factors for a software system, which can be called a quality model. In detail, the structure consists of general software characteristics, which are divided into subcharacteristics. The hierarchy of characteristics and subcharacteristics is as follows:

- Functional suitability – characteristics about the set of functions and their specified properties that satisfy stated or implied needs. Subcharacteristics: Functional completeness, functional correctness, functional appropriateness.
- Performance efficiency – characteristics about the relationship between the level of performance of the software and the amount of resources used. Subcharacteristics: Time behaviour, resource utilization and capacity.
- 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. Subcharacteristics: Co-existence, interoperability.
- Usability – characteristics about the effort needed for use, and on the individual assessment of such use, by users. Subcharacteristics: Appropriateness recognizability, learnability, operability, user error protection, user interface aesthetics, accessibility.
- Reliability – characteristics about the capability of software to maintain its level of performance for a period of time. Subcharacteristics: Maturity, availability, fault tolerance, recoverability.
- Security – The degree of protection of information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them. Subcharacteristics: Confidentiality, integrity, non-repudiation, accountability, authenticity.
- Maintainability – characteristics about the effort needed to make specified modifications. Subcharacteristics: Modularity, reusability, analysability, modifiability, testability.
- Portability – characteristics about the ability of software to be transferred from one environment to another. Subcharacteristics: Adaptability, installability, replaceability.

The model presented in the standard is aimed to be a checklist for identifying concrete requirements and criteria for quality control. The old ISO/IEC 9126 standard has already become a well-known and relatively well-understood basis for many types requirement and quality assessments and thus could be used for many purposes with little tailoring. As the ISO/IEC 25010 brings the quality model standard more up-to date, the same applies to that too.

A note on the differences between ISO/IEC 25010 and ISO/IEC 9126. Compliance with standards or regulations in regard to the characteristics was included in ISO/IEC 9126-1. The compliance is now left outside the scope of the quality model as any issues of compliance can be identified as part of requirements for a system.

## III. HOW DOES MODELLING COMPARE WITH WRITING APPLICATION CODE?

Are test models software? Testware in general is understood to be very much like software and to demand similar handling. That clearly applies to test scripts and utilities, but test models are much the same – execution-ready structures just waiting for getting started by a test execution engine.

Some clear similarities between program code and test models include these:

- They are artefacts created by a programmer or test modeller using his/her personal tools.
- They are created using formal syntax and semantics, either textual or visual. Textual models are in fact often written using a programming language, either a special testing language or a general-purpose programming language such as Python or Java.
- Both must be crafted carefully. The smallest errors can cause them not to work properly.
- Code and test models can (and often should) be designed to have an architecture where more abstract functionality is placed on one layer and low-level details on another level, possibly aided by separate libraries.
- They have a specific system context and environment where they are used. Application code is executed for a purpose and similarly, test models are used to test something.
- They have a lifecycle of requirements specification, design, implementation and testing before being adopted into use.

- During that, their quality is managed in reviews and inspections.
- Both need to be altered and extended as their requirements evolve.
- Both application code and models can be crafted incrementally and that is usually advised.
- Both code and models are documented and stored to support their usage in the same project and in other projects.
- Both require an execution infrastructure.

Of course there are differences too. Those include these:

- Programs are usually written in languages that are at least mostly readable and comprehensible to any trained programmer. In contrast, models may be created using languages that are difficult to understand without more specialized training. Furthermore, many testers lack any kind of background in programming or formal languages.
- If program code is incomplete, that is a flaw which cannot be handled by other means, whereas test models can be incomplete and that is seen as a good characteristic, as it shows focusing on the issues under test and not modelling the whole functionality. "Less is more" often applies to test models.
- For production code, its ability to run implies that something productive can be done with it, but simple running is not sufficient for any test tool element. There must be the element of challenging the target system present. That implies a deeper focus from the maker to the artefact.
- Writing application code is clearly creating something new, but modelling is about making a representation of something existing.

## IV. Assessment of MBT Test Models Against ISO/IEC 9126 Criteria and the Associated Competences

As noted, the ISO/IEC 9126 has a hierarchy for the quality factors. This section is divided by the top level quality factors and the lower level factors are presented in *italics*, for example top level characteristic "Functional suitability" contains a lower level subcharacteristic "*functional appropriateness"*.

### A. Functional suitability

The model obviously needs to have the functionality required from a test model (*functional appropriateness*). It must be able to express those aspects of the SUT it is meant to test, whether they be the control flow of a task, the structure of legal input data, or the timing behaviour of a real-time system. In practice, this often means that the model must be able to express the states of the system and any transitions between states so as to be able to guide the testing of those. However, a good model will not express everything that the SUT does, only those elements of behaviour that are needed for the testing. Therefore, *functional completeness* is assessed against the goals of testing and against the planned purpose of the model.

The expression must also be correct (*functional correctness*) in a sense that the model reflects the system as it should be. That is a traditional design requirement. But testing needs to challenge the behaviour so the model also needs to contain the elements of undesired behaviour – things that should not happen but for which the system and the model must be ready to handle. That is the essence of testing. Of course the model should also have no errors regarding to its own design criteria. There may also be some official requirements for compliance with standards, laws and similar. This is essential especially in safety-critical domains where standards such as IEC 61508-3 [4] may require MBT to be carried out with certain strictness and to have certain aspects modelled.

The critical competences for this include: Modelling skills are obviously important. This is the ability to express the relevant aspects of the SUT in a model at appropriate abstraction level. This often requires reverse engineering skills as the systems are not usually documented properly in this regard so the tester must rely on "mental reverse engineering". Testing skills are the core skills of any tester. This includes the tester's mindset and approach of not just executing the system but challenging it. An important "meta-skill" is the ability to select tools and techniques that work well together. Especially in a safety-critical domain, the tester must know and understand any requirements for the models, such as the requirements in mandatory safety standards.

### B. Performance efficiency

Model-based testing and test automation in general must be fast, as the state space can be very large and the idea of MBT is to cover as much of it as is possible. That is one element of good *time behaviour*. A related issue is *resource utilization*, resources here meaning RAM memory and such. Again, the models and their execution need to use as little memory resources as possible so the computer capabilities will not become a bottleneck in testing. Ways to reach this include focusing the model on the essential things that are tested, to keep the state space and number of state transitions small enough. For good modelling, less is often more. This same design principle also results in better maintainability and usability of the models. The subcharacteristic *capacity* means that the maximum limits of the system must be high enough to meet requirements, and is more a characteristic of the whole MBT system. However, in model-based concurrency or load testing it must be ensured that the model is capable of simulating processes in sufficient numbers.

The critical competences for this include: Understanding how different features of the modelling formalism impact the complexity of the resulting models is important. The tester needs focusing skills – keeping the test goals in mind and modelling for those. General understanding and appreciation of real-life test execution systems are essential.

### C. Compatibility

*Co-existence* for models requires their independence from other models or other test code. That means for example good variable and state naming conventions so there will be no conflicts between models that are loaded at the same time to the test system. *Interoperability* between the model and other test system components is mostly achieved by using file and data formats between models and other systems that match the tools used. Also relevant to MBT is interoperability between different models; ideally models for different aspects of the SUT can be combined to test them all at once.

The critical competences for this include: Programming skills contain understanding of the name spaces and similar. Appreciation of and skills in designing good file formats and interfaces is a key software developer skill.

### D. Usability

The very first thing about usability is *appropriateness recognizability*. When there is a large library of models and other testware, a tester must be able to recognize which one of them is appropriate for the testing task. Most often this is based on various metadata: good file naming, good description of the model in its file and catalogues. Recognition should not depend on examining the models, just like reading application code should not be needed to understand its purpose. Rather, the necessary information should be available in documentation, or even inferable from file names.

*Learnability* is important especially for maintaining and reviewing the models. For a person to understand and learn how to use the models, they need to be clear and concise and they need to use good design and style guidelines. Proper documentation is often very important, as even clear and simple models can be hard to understand without written notes and instructions. Having a modular architecture that keeps the size of the models down is important here. The models should also concentrate on the test goals and not include everything – that would just clutter the models and make it hard to see what their essence is related to testing. Obviously, the models need to be clearly represented in diagrams. When the models are drawn by hand, they should have a clear layout and essential elements marked – for example where the execution starts in a use case.

*Operability* in the standard means whether users can operate and control the software. In modelling, it means that testers can use the models for test generation. Big part of that is how the model expresses how it is used. In practical situations it is also important for the tester to be able to change the model's runtime configuration so as to match it to some type or goal of testing.

*User error protection* is aided by paying attention for example to naming conventions and modular design. That way the common slips of referring to wrong elements are reduced during the development and maintenance of the models. Errors in test run specification are reduced with clear concepts that make it clear what any configuration elements mean.

*User interface aesthetics* is also something not to neglect. When the models are attractive, people like to work with them, like to learn them and thus do good testing with them. The details of model aesthetics depend on the used modelling languages.

*Accessibility* means that the models "can be used by people with the widest range of characteristics and capabilities", in particular by those with disabilities. This is mostly a feature of the modelling and test generation tools rather than the models themselves.

It should also be noted that there are two main modelling types: visual modelling and modelling by writing programming code (the result of which can be visualized). Both of those have special usability issues, into which we will not go here.

The critical competences for this include: The modeller needs to have understanding about factors that influence understandability of models and a realization that models are not created for him/herself, but for others. For good architecture, modular design skills are needed and for keeping the tests focused. The focusing is aided by good testing orientation, which results in every element in the model having a clear purpose. Of course, when we are talking about usability, the modeller should understand the basics of designing user-friendly systems.

### E. Reliability

Reliability of the test system is important. The model needs to run without disturbances sometime for long durations. One of the strengths of MBT is indeed the ability to test for long durations without exhausting the test cases. This means that the overall test system needs to be much more reliable than traditional test systems.

*Availability* refers to the degree that the system is operational and accessible when required for use. There are many viewpoints to that. Availability during design requires access to the modelling tools, sometimes concurrently. Good modularity enables different people to have simultaneous access to the model's components. Non-availability may be caused by delays in correcting the model to meet system changes or by repairs to the model's defects. The model's defects may cause problems during test runs that affect the availability of the model and the whole test system.

When a system runs reliably without problems and testers can trust it to do so, it can be said to have *maturity*. Software systems get mature by testing and using them and "ironing out" all the defects. The same applies to test models. They must be tested before production to have confidence to their reliability.

Especially in the long test runs the model needs to handle any deviations and problems in the test environment and adapter gracefully so that the runs are not interrupted (*fault tolerance*). *Recoverability* is needed when interruptions nonetheless occur, though this is more a characteristic of the overall test execution system.

The critical competences for this include: Understanding about robustness in design is important, especially when models are created using a programming language. The tester needs a mindset for aiming at good robustness and a working

ethic where the testing tools are tested properly before applying them in production.

### F. Security

Generally, for MBT, the generic security approaches of test automation are sufficient. As far as modelling is concerned, the requirements of *confidentiality* can be mostly satisfied by making sure that the models do not command the SUT to expose confidential information to outsiders. However, if the models are created with a language that has access to the system where it is executed, it must also be ensured that they do not expose any information present in that system. Most of the time, access to sensitive system resources can and should be avoided entirely. Furthermore, if the SUT itself is confidential, the models must either not describe any of its confidential parts or be considered confidential themselves.

In ISO/IEC 25010, *integrity* means that the system may not allow unauthorized access to or modification of programs or data. Again, this requires that the models may not command the SUT to do anything that would compromise system integrity, or misuse system resources to do so themselves.

*Non-repudiation* means "degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later". In the context of testing, this has no implications in the security sense, but it is essential for the documentation of the testing. For example, in safety-critical domains we need to keep track of what has been tested and with what results. So, this is mostly an issue of test logs and handled at the MBT system level, not by the models. Similarly, *accountability* is not relevant at the model level, nor is *authenticity*. Those are mostly issues in the system under test and may be considerations for tests.

The critical competences for this include: Understanding about security issues is important for any tester. The tester must understand architectures so that any confidential elements are hidden from the models.

### G. Maintainability

All testware should have a long lifespan and even during one project there will be a lot of maintenance done for the models, due to changes in the system under test. So, maintainability is something to take seriously. Too often it has been found that even good test arrangements are neglected at some point, because they are found to be too hard to maintain.

*Modularity* in the context of models suggests an architecture where models are split into smaller models, making it easier to encapsulate some behaviour to a smaller package, aiding in its understanding and reducing dependencies. Another issue here is the division of testing functionality between the model and the adapter. Decisions regarding that are very important.

*Analysability* relates to the possibility to find out programmatically or manually what the model does and to check that it is still in good shape after changes. Practices used in that include static analysis and any ways that the modeller or another person tries to find out what the model does. Ability to do simulation runs without executing the target system is often essential. This is very much related to the *testability* of the models but testability is important also for the very first times the model is used: all testware needs to be tested before taken into production use.

*Reusability* means the possibility of testing more than one system or multiple aspects of a single system with the same models. It can be improved with good modular design, where generally useful functionality is packaged into separate modules.

*Modifiability*, the ability to make modifications to the model, is something to reach for by good design: appropriate abstraction level, good architecture, simplicity and so on. After the modifications, the already mentioned analysability and testability play a big role, as any modifications need to be tested – just like modifications to application source code.

Of course, the maintainability issues are very critical in agile development, where the system under test is under constant change and thus the test models also need to be revised frequently.

The critical competences for this include: Design for maintainability skills, including the attitude towards supporting the whole lifespan of the models in any design decisions. Modularity requires skills in architecture design. Many competencies critical for usability also have an impact here, as the models need to be understandable and clear in order to be maintainable.

### H. Portability

For organizational efficiency, the models need to be adaptable to any product and project they are targeted to (*adaptability)*. Often the same testware needs to be used for many products of a product line or product family and that should be handled by an easy method of configuration. This is usually aided by a hardware-independent abstraction level that leaves most of the adaptation to a lower level adapter subsystem.

The same design features aid in *replaceability* – a clean architecture makes it easier to replace a model with another, perhaps to use another model that better supports another test goal or a model made by another tool, for example to increase diversity in test design – models created in different ways will find different defects. Both of these require *installability*, which results not only of design of the model, but adhering to any standards or conventions that the test system is designed by. In the best case, a well-designed model can just be chosen for a test run, but in the worst case, manual tailoring can be needed.

The critical competences for this include: Architectural design skills and modular programming skills. Interface design skills.

## V. Evaluation and Conclusions

The ISO/IEE 25010 turned out to be a good tool for systematically extracting and making visible many quality factors or characteristics that are normally not discussed in the context of modelling.

This study shows that test models indeed have similar quality factors as software systems. Most of the quality factors listed in the standard are directly applicable to models, and most of those that are not still apply to MBT tools and processes in general. One implication of that is that quality assurance practices for software development, such as reviews, really are feasible for test models.

The findings, when elaborated further, enable discussion about the characteristics and quality management for the models in systematic manner.

For practical purposes, the long list of characteristics should in each context be condensed into a shorter list that is applicable as a checklist in reviews and such. But for MBT tool developers, the full analysis is invaluable. Therefore, this assessment should help tool developers in the creation of more mature, industrial-quality tools that can more easily be integrated into the software engineering activities in companies.

Of course, having any particular framework means that there may still be some issues that the framework does not include or show with a correct prioritization. For that we need to use alternative quality frameworks and that work is currently undergoing at TUT. We need to remember that in the standard itself it is recommended that the identification and analysis of quality attributes should be started from scratch. Yet, the analysis with ISO/IEC 25010 gives us a solid baseline to start working on that.

Another implication is that test modelling is a broader task than just “modelling” and requires many traditional software engineering competences and considerations, of which there currently is no clear view. Such a view is needed for example in training, in which there always are time pressures and a serious pressure to prioritize and make choices for topics.

## VI. Future Research Needs

The whole area of quality models for test automation is something that requires further actions. For example, while it is good for models to be independent, they are often very much tied to the test adapters and execution engines by design and the core MBT system is part of the overall test automation system. In the future, we rely more and more on test automation so we need to assess its quality properly and ensure that all the people who are working with the systems have the broad competences required. Some general work on tester competences is currently undergoing at TUT and will continue to shed more light into the competences related to model-based testing.

## References


[1] Andrews, Anneliese; France, Robert; Ghosh, Sudipto and Craig, Gerald. 2003. Test Adequacy Criteria for UML Design Models. Journal of Software Testing, Verification and Reliability, Vol 13, p. 95-127.

[2] Franch, Xavier and Carvallo, Juan Pablo. 2003. Using Quality Models in Software Package Selection. IEEE Software, January/February 2003, pp. 33-41.

[3] Hamann, L.; Gogolla, M. 2010. Improving Model Quality by Validating Constraints with Model Unit Tests. Model-Driven Engineering, Verification, and Validation (MoDeVVa), 2010 Workshop on , vol., no., pp.49,54, 3-3 Oct. 2010.

[4] IEC 61508-3. 2nd edition. 2011. Functional safety of electrical/electronic/programmable electronic safety-related systems. Part 3: Software requirements. 234 p.

[5] ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. 34 p.

[6] ISO/IEC 9126-1:2001. Software engineering – Product quality – Part 1: Quality model. 33 p.

[7] Lemaitre, Jonathan and Hainaut, Jean-Luc. 2008. A Combined Global-Analytical Quality Framework for Data Models, in Proceedings of the 3rd International Workshop on Quality in Modeling (QiM'08 @ MODELS'08), pp. 46-58.

[8] Mohagheghi, Parastoo; Dehlen, Vegard and Neple, Tor. 2009. Towards a Tool-Supported Quality Model for Model-Driven Engineering. Proceedings of the 3rd Workshop on Quality in Modeling, p. 74-88. Gøteborg: IT University of Göteborg.

[9] Mohagheghi, Parastoo and Dehlen, Vegard. 2010. An Overview of Quality Frameworks in Model-Driven Engineering and Observations on Transformation Quality. In Proceedings of QUASOSS '10 2nd International Workshop on the Quality of Service-Oriented Software Systems, pp 3 - 17, Oslo, Norway, October 05 - 05, 2010, ACM.

[10] Mayerhofer, Tanja. 2012. Testing and debugging UML models based on fUML. In Proceedings of the 2012 International Conference on Software Engineering, ICSE 2012, p. 1579-1582. IEEE Press.

[11] Utting, Mark and Legeard, Bruno. 2007. Practical Model-Based Testing. 433 p.