Toward a Consistent and Strictly Model-Based Interpretation of the ISO/IEC/IEEE 29119 for Early Testing Activities

Reinhard Pröll and Bernhard Bauer
Institute for Software & Systems Engineering, University of Augsburg, Germany

Keywords: Model-Based Testing, Test Management, Mutation Testing, ISO/IEC/IEEE 29119, Software Testing Lifecycle.

Abstract: Effective and sufficient testing has always been a challenging task in software development. The ongoing increase of software complexity forces developers and testers to make extensive use of the concept of abstraction, thereby leading to model-based approaches. Further, standardization organizations aim for harmonized process templates to assure a certain quality level of the processes behind. In order to combine the process-level advice as well as the concept-level advice, we aim for a consistent and strict application of model-based methodologies throughout the testing processes, introduced by the ISO/IEC/IEEE 29119 standard for software testing. After a brief introduction to the standards content and a critical view on it, we focus on our model-based interpretation of the postulated processes. Thereby, we extend the original idea of model-based testing, incorporating the separation of concerns on the model-level, to form a broad information basis. Subsequent activities are aligned with these concepts, in order to make sure a purely model-based testing life cycle, with respect to consistency and quality of development artifacts. Following the related work of impacted research areas, we end up with a conclusive statement on the intended combination of approaches.

1 INTRODUCTION

Over the last decades, the level of software complexity steadily increased. Ongoing research in the fields of software development and testing tries to tackle the complexity by advanced approaches. Beside the continuous improvement of applied working techniques, standardization organizations tend to aggregate and generalize knowledge about field-proven approaches, in order to give users guidance and support certification of products. Especially in the field of software testing, many standards were published, subsumed by the ISO/IEC/IEEE 29119. In order to align the testing activities with the increased requirements, imposed by the higher complexity level, active standards force the application of advanced techniques, such as Model-Based or Risk-Based Testing.

Problem Statement

Nevertheless, we identified three major challenges for test-related activities and processes in the context of highly software-intensive systems.

First, we see a strong need for the consistent automation of testing activities. For nearly every dedicated part of the software testing life cycle, a huge variety of valuable approaches ready for automation has been investigated by Felderer and Schieferdecker (2014), Elbaum et al. (2002), Utting et al. (2012), and Jia and Harman (2011). Combining a subset of these approaches, to fully instantiate a software testing life cycle, still manual steps bridging the conceptual gaps between the dedicated parts are necessary. In order to overcome this weakness, we propose a consistent conceptual and knowledge basis throughout all test disciplines.

Second, we are convinced that future software testing has to take place in early stages of development. Due to the fact, that costs for errors, detected in late phases, are extremely high, the overall costs for software testing in early development stages are expected to be much lower. This effect raises in scenarios, where the overall level of complexity is much higher. Therefore, we envision a complete instantiation of the software testing life cycle applicable to the early development artifacts, namely design-time artifacts, which are meant to evolve over time. Further, drawbacks about the expressiveness of testing activities in such early stages are not expected to have a great impact on the

Pröll, R. and Bauer, B.
Toward a Consistent and Strictly Model-Based Interpretation of the ISO/IEC/IEEE 29119 for Early Testing Activities.
DOI: 10.5220/0006749606990706
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 699-706
ISBN: 978-989-758-283-7

feedback, which is generated based on test results.

Last, we are convinced that future testing needs to make extensive use of the concept of abstraction, to overcome complexity. For example, Test Case Explosion in our opinion is mostly reasoned in a too detailed level of test activities. In many cases, this ends up in thousands of tests challenging mostly low-level concepts, that could also be assured by adequate code/model quality techniques during development. Consequently, this imposes higher requirements on the quality of involved development artifacts, representing the abstract interpretation of the test specification. The resulting high-quality set of models again supports the early testing activities.

In order to bring together existing approaches for dedicated parts of the software testing life cycle, as well as targeting the challenges previously identified, we propose an orchestration of model-based techniques to ease early testing efforts.

Outline

Throughout section 2 an introduction to testing processes and the orchestration of activities incorporated by the ISO/IEC/IEEE Standard 29119 is presented. Based on the included process templates and a critical view of the standard itself, section 3 describes the consistent and strictly model-based interpretation of processes and their activities. Section 4 presents related research, which impacts our contribution. Finally, section 5 concludes the benefits and shortcomings of a strictly model-based instantiation of test management and dynamic testing processes.

2 THE ISO/IEC/IEEE 29119 STANDARD FOR SOFTWARE TESTING

In order to define a widely agreed assistance in the field of software testing, the *ISO/IEC/IEEE 29119 standard* has been developed. It is meant to replace a set of pre-existing standards in this field, such as IEEE 829 (2008) for test documentation, the ANSI/IEEE 1008 (1987) for unit testing, the ANSI/IEEE 1012 (2012) for system and software verification and validation, the BS 7925-1 (1998) representing the vocabulary of terms in software testing and finally the BS 7925-2 (1998) incorporating the software component testing topics. Targeting a holistic presentation of best practices as well as state of the art processes, the standard is split up into five parts covering the most important aspects.

ISO 29119-1 (2013) introduces basic concepts and definitions, to set up a common understanding across professionals having in mind the earlier standards.

In contrast to the rework of predominant knowledge, ISO 29119-2 (2013) details the process artifacts contributing to the standards generic definition of a potential software testing reference process. Additionally, it focuses on the cross relations between process artifacts of the distinct areas investigated in section 2.1, section 2.2 and section 2.3 of this paper. Altogether, this part can be seen as the central artifact integrating the topics formerly distributed and isolated in multiple standards and books.

ISO 29119-3 (2013) again focuses on integrating work already presented within the IEEE 829 standard, last updated in 2008. Its main purpose is to define the essential test documentation artifacts and their purpose within the presented test related processes. Thereby the general structure and central aspects of the technical documentation are defined.

Furthermore, ISO 29119-4 (2015) investigates on a set of test design techniques applied by a conventional testing process. Apart from the set of techniques described within the *BS 7925-2* standard, more state of the art test concepts are incorporated. Additionally, the respective set of test coverage metrics is presented in the second part of this document.

ISO 29119-5 (2016) introduces an efficient and consistent reference approach for keyword-driven testing, representing a field-proven and intuitive technique. Further, requirements on related tooling are specified enabling testers to use intermediate work products across a heterogeneous tool landscape. Advanced topics covered by this part of the standard address the concept of hierarchical keywords and templates for technical keywords.

Beside the partitioning of concepts across the five distinct documents, the content is further structured into three test process levels with mutual dependencies, shown in fig. 1.

2.1 Organizational Test Process

The standard defines an organizational test process to specify knowledge, relevant for the whole organization and not dedicated to a certain project, in a controlled way. This process template may flexibly be applied to diverse scenarios, e.g. the process for development and management of the organizations test policy as well as the

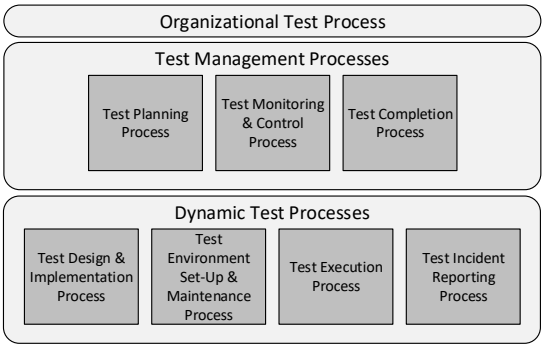

Figure 1: Multi-layer model of test processes (as per ISO 29119-2 (2013)).

therefrom-derived test strategy. The template consists of three activities, namely *Development of Organizational Test Specification (OT1)*, *Monitor And Control Use Of Organizational Test Specification (OT2)*, and *Update Organizational Test Specification (OT3)*. OT1 represents the first and singular task of creating an Organizational Test Specification (OTS), since OT2 and OT3 constitute an iterative feedback loop for updating and maintaining the specified artifacts.

2.2 Test Management Processes

Apart from the organization-wide test processes, project-specific processes are further refined. In contrast to the presented process template, test management processes follow a high-level division into three sub processes.

Test Planning Process - This process determines a sequence of nine activities (TP1 - TP9) representing the essential steps toward a valid test plan artifact. Starting with activities correctly interpreting context information and allocating workload (TP1 - TP2), later steps focus the identification, mitigation, and deduction of a strategy for a risk-driven plan (TP3 - TP6). At this point, the standard emphasizes, that these sequential steps may be carried out iteratively, in order to raise the overall quality and focussedness of the final test plan. The closing activities aim for the serialization and communication of the agreed test plan (TP7 - TP9).

Test Monitoring & Control Process - Addressing subsequent work of the test management processes, the test monitoring and control process manages the gap between the planned testing activities and the actually executed set of tests. Following the set-up of monitoring structures (TMC1), previously identified test measures are monitored (TMC2) and in consequence, trigger control structures (TMC3). This potentially iteratively carried out monitor-and-control loop steadily collects information about the dynamic testing processes captured within detailed reports (TMC4). Further, based on this set of monitoring and control information, the decision whether testing is completed or not is made.

Test Completion Process - Initiated by the test-monitoring infrastructure, active testing terminates triggering a final test completion process. The template completion process splits up into four different activities, again either executed in a sequential or iterative way. Starting off with archiving of test assets (TC1) and cleaning up the test environment (TC2), the major contributions of this process are reflective and documenting tasks (TC3 and TC4), to continuously improve the overall testing process landscape.

2.3 Dynamic Test Processes

Tightly coupled to the test management processes, the dynamic testing processes dealing with topics around the creation and execution of concrete tests are further detailed. Fig. 2 introduces the process-level integration of the concepts, no matter which test phase or test type defines the context.

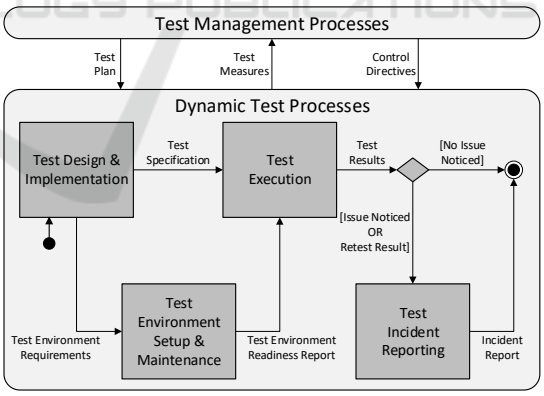

Figure 2: Process-level integration of Dynamic Test Processes (as per ISO 29119-2 (2013)).

Test Design & Implementation Process - This process includes the work to be done before the real testing may start. Impacted by the previously introduced test plan, representing the strategic and policy-driven view on the activities, the test design and implementation is developed (TD1 - TD6). Mainly, three different process artifacts are generated. The *Test Design Specification*, which is derived from

a certain set of features and related test conditions. The *Test Case Specification* additionally defines test coverage items, connected to monitoring processes to check meta-information of test cases. Based on these two artifacts the *Test Procedure Specification* is specified by assembling test sets and implementing related test procedures.

Test Environment Set-up & Maintenance Process - Beside the initial set-up of the test environment, included activities assure the seamless integration and executability of tests inside the test environment by continuous maintenance (ES1 and ES2).

Test Execution Process - Based on a working test environment, processes assuring the test execution take place. The proposed test execution process scheme consists of three activities. Following the execution of previously specified test procedures (TE1) the outcomes are further evaluated (TE2) and documented (TE3), thereby triggering multiple feedback loops to improve the overall testing process.

Test Incident Reporting Process - In case of unforeseen incidents, a special type of process is integrated to the standard proceeding. In a two step sub-process, the observed incidents are analyzed (IR1) and further documented by a report (IR2), specific to the stakeholders responsible on that feature.

As we have seen, the standard tries to template a solution for the orchestration of heterogeneous test-related processes. Nevertheless, the targeted domain, where adaptability and context-awareness of applied approaches is essential, marks a problematic case for standardization efforts.

2.4 Standardization of Software Testing

Standardization means to incorporate predominant approaches of a domain and extract the agreed consensus, to ensure quality across application scenarios. In the testing domain, which lives from the creativity of testers and their abilities to do structured problem-solving, standardization is hindering. For example, setting up a standard-conform test process, may restrict the intended approach to a degree, where its meaningfulness is cut down to a significantly lower level. Here standard-conformance is preferred over appropriacy of testing. In case of inverted preferences, the missing standard-conformance may further affect the customers' acceptance of the developed product.

Consequently, we see the ISO/IEC/IEEE 29119 standard as a reference for basic test-related building blocks and as a collection of template processes in the area of testing. Therefore, we aim for an interpretation (and not an instantiation) of the presented standard satisfying the needs of our focused application context, namely design-time testing activities.

3 A MODEL-BASED SOFTWARE TESTING LIFE CYCLE

Targeting a strictly model-based interpretation of the previously presented processes, we hereafter introduce the set of methodologies covering most of the different aspects of the ISO/IEC/IEEE 29119 standard. In order to overcome the identified problems, we further try to incorporate and combine multiple field-proven techniques in model-based testing and testing in general. Beyond, we aim for the adaption of concepts, applied in different contexts, and the use of their beneficial effects. All over, we aim for a combination of testing methodologies consequently tackling the origins of test complexity, further applicable throughout all stages of software development, starting with the design phase.

Fig. 3 provides an overview of the essential steps incorporated into this work.

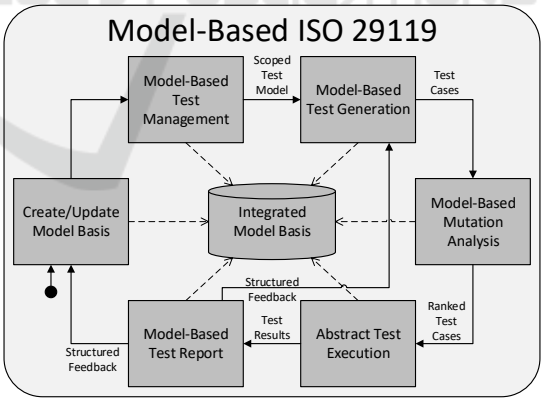

Figure 3: Model-based and standard-oriented software testing life cycle.

In contrast to the mainly sequential process descriptions of the addressed standard, we clearly state the need for an iteratively carried out testing process, reflected by its cyclic nature. Further, we like to underline that the presented process is not only limited to the early testing activities stated in the introduction. It may also be carried out in later testing phases.

3.1 Integrated Model Basis

Starting with the most essential part of the model-based interpretation of testing activities, we introduce a model repository called *Integrated Model Basis*. This model repository includes various domain-specific models, connected by a so-called *integration model*. Whenever we use the term *domain*, we talk about purpose-specific views on the system under development resulting in separated model artifacts, e.g. the test model artifact specified within the UML Testing Profile or the reliability model artifact specified via fault trees, in case of a safety critical system. The integration model artifact further acts as a storage for data reflecting organizational or management aspects, as far as these are quantifiable. With the integrated model basis sometimes called the Omni Model - the domain-specific knowledge, its cross-domain relations, as well as organizational/management aspects - is expanded to its greatest. As introduced by Rumpold et al. (2017), the *Architecture and Analysis Framework* (A3F) represents a prototypical implementation of the Omni Model approach.

Beside the pure integration of knowledge, the framework enables the user to flexibly define analyses on top of this model data. Analyses may for example be used for code generation tasks or the general task of producing purpose-specific data, e.g. dynamic cross-domain documentation artifacts or change impact analysis results.

Investigating on the relation between the ISO/IEC/IEEE 29119 standard and our intended model-based interpretation of activities, the Omni Model approach tackles nearly every process artifact of the standard. Due to the incorporation of multiple viewpoints into a joint model basis, all layers of the standard are affected. The *Organizational Test Process*, the *Test Management Processes*, and the *Dynamic Test Processes* are based on the modeled information, also reflected by fig. 3.

3.2 Model-Based Test Management

Beyond, the *Test Management Processes* presented by the standard either serve a documentation or reporting purpose for the organizational part, or an planning purpose impacting on the *Dynamic Testing Processes*. The latter is focused by our Model-Based Test Management approach, extending traditional risk management or risk-based testing efforts to the multi-domain application scenario.

Aiming for the reduction, prioritization, and selection of a set of test cases, our test management process targets a reduced and strongly focused sub-model of the original test model, i.e. a scoped test model. In order to end up with such a scoped test model, the test purpose needs to be narrowed by constraining so-called aspects, defined within the integration model. *Aspects* may either represent a certain characteristic of a connected domain-specific model (intrinsic aspect), or a quantifiable risk value imposed by some management decisions (synthetic aspect). The aspect concept together with its constraint mechanism enables users to plan their testing activities within a multi risk-aware methodology. Beside the use case of test planning, a beneficial use in regression testing scenarios is thinkable.

A prototypical implementation of the *Model-Based Test Management* approach within the *A3F* reveals promising results. Technically, we realized the scoping mechanism within a chain of analyses finally producing a scoped model artifact. Another positive property of this kind of test management is its guidance for subsequent *Dynamic Test Processes*, by specifying the model basis for the test case generation steps and thereby closing the automation gap between these two process groups. Further, there is no more room for misinterpretation of the test plan, due to the direct mapping of information across domains.

3.3 Model-Based Test Generation

Switching over from the *Test Management Process* related activities to the *Dynamic Test Processes*, we further introduce our approach for test generation based on the before mentioned Omni Model approach. Especially the *Scoped Test Model*, representing a reduced version of the original test model with respect to management aspects in conjunction with the test plan, is the focus of this processing step. Based on the *Model Analysis Framework (MAF)* Saad and Bauer (2013), which works on graph representations, structural features and annotated data are evaluated regarding certain metrics to derive test cases. The chosen set of metrics may additionally be varied by structured feedback of previous runs of test suites. In our case a generic control flow graph representation of the test model is processed by MAF in order to generate test cases. The resulting set of test cases is represented by a set of paths through the input graph structure. This format of test cases gives us the flexibility to adapt to a certain target representation or leave it untouched to stay on the model level, which is preferable in early development stages.

Based on the highly-scalable Model Analysis Framework, the presented concept for the *Test Implementation Process* and related activities, continues the model-based and automatable chain of methodologies.

3.4 Model-Based Mutations and Abstract Execution

With respect to the challenge of increasing model quality and generated test cases identified in section 1, we introduce an intermediate quality assurance step. This kind of activity is not foreseen in the ISO/IEC/IEEE 29119 standard, although we strongly recommend some comparable activities in every testing process. Beyond the pure conformity with a certain coverage metric, which just awards the attribute of a completeness to the generated set of test cases, mutation testing challenges each test case for its ability to detect defects, which was first introduced by DeMillo et al. (1978). So far, mutation testing approaches are widely focused on injecting faults into a target code basis. In contrast to this, we focus on mutations of an abstract graph representation of behavioral models of the system under development. This enables to assess test cases, before any fragment of code needs to be written or generated.

For the assessment of test cases, mutation testing needs to execute the test cases against a mutated system under test. In order to make comparable functionality on the model level, we developed a prototypical abstract execution framework, which executes two graph representations in sync. Each of these graphs represents a certain interpretation of the specified behavior (control flow) of the system, e.g. from a testers and a developers viewpoint. The graph representations are generated out of the respective domain-specific models. Processed by a model-to-model transformation, each of them leads to a uniform graph structure including attribution data. The ongoing abstract execution, synchronized via the integration model, on the one hand runs the graph model representing the mutated system under development and on the other hand the graph model of the considered test case. The incorporation of the connected Omni Model information enables our graph interpreter to simulate whether the defect is detected by the test case or not. With this methodology, we aim for competeable results about the mutation analysis process as well as the expressiveness of abstract test execution runs.

Traditional Mutation Testing approaches are known to be very time consuming due to the extremely high number of test cases to be run for several times. The model-level adaption of mutation testing again reduces the efforts to be made and time to be spent for running test cases. The reason for that is the uniform execution environment, which abstracts from detailed timing, performance, or concurrency considerations of the target platform. The overall cost reduction hopefully forces users to apply this quality assurance technique in practical scenarios, to cut the amount of test cases to be executed on the product, to a minimal extent of high-quality test cases.

3.5 Model-Based Test Report

Targeting the consistent automation of all included processing steps and completing our model-based interpretation of the ISO/IEC/IEEE 29119 standard, we take a deeper look at the generated test results and how they may be used for structured feedback. The results produced by the abstract execution of test cases are collected and aggregated to a model-based test report. This report includes specific information about model elements affected by failed test runs, enabling the tester to use the *Architecture And Analysis Framework* of Rumpold et al. (2017) to generate purpose-specific data, e.g. human-interpretable test reports or machine-interpretable test reports for advisory tooling. Due to the fact, that the model-level description of information has never been obfuscated during the processing chain, we can clearly separate the affected model elements and their connected information in the Omni model repository. Based on the set of affected model elements valuable guidance may be generated automatically, forcing the test engineer to either investigate on the test model or any other connected domain-specific model of the system under development. As previously mentioned, the machine-interpretable advice represents feedback for the test generation step, by improving and adapting the test metrics applied.

3.6 Shortcomings of the Model-Based STLC

Beside the introduction to the concepts behind the model-based software testing life cycle and the beneficial effects of the contributing methodologies, we now discuss potential shortcomings. It is out of the question, that the presented approach, especially its Omni Model basis poses strong requirements to the underlying infrastructure and general methodology of developing software. The strict application of separation of concerns within the Omni Model concept is positive for the subsequent application of

model-based approaches, but forces developers to so specify and support the cross-domain connections needed by nearly every step of the life cycle, e.g. the abstract synchronized test execution.

Beside the positive aspects of a fully automated testing approach, the danger of an evolution toward a black-box approach, not satisfying the testers needs, is omnipresent. Therefore, we strongly recommend a transparent way of automating the overall processing chain, to guarantee a comprehensible action-to-effect mapping through all phases of our testing life cycle. The documenting and reporting process steps of the addressed standard aim for such transparent processes, not explicitly incorporated by this contribution.

Furthermore, we think the quality of the included domain-specific models decides on the success of the combined methodologies. These days, model-based development artifacts are predominantly used for structured documentation, therefore mostly focused on manual (human) processing. The largely missing focus on *design-for-automation* and *design-for-testability*, which is reflected by current development artifacts, needs to be overcome to make such approaches applicable in practical scenarios.

4 RELATED WORK

The work on the model-based testing life cycle is affected by concepts distributed across multiple areas of software testing. Comparable in its basic approach, but missing the model quality assurance mechanisms, Daoudagh et al. (2015) introduced a toolchain for model-based design and testing. Another shortcoming compared to our contribution is its highly focussedness on so-called access control systems, which curtails potential application scenarios.

Further, the key concept of the presented life cycle, its integrated model basis, relates to earlier research in the area of domain-specific (meta-)modeling. de Lara et al. (2015) and Zschaler et al. (2009) both worked on approaches dealing with the orchestration of modeling languages and cross-relations on a meta-level. Beside this essential and versatile related concepts, our contribution is impacted by research of three distinct fields.

Model-Based Testing

Apfelbaum and Doyle (1997) original approach, which generates test cases from model-based specifications of the intended system, inspired our first versions of the consistent model-based testing life cycle. The even more restrictive definition by Utting et al. (2012), strictly separating the domain-specific data together with its positive effects, forced us to adapt the general idea behind to our *Integrated Model Basis*.
The authors furthermore introduced a taxonomy for model-based testing approaches, among others focusing on criteria like the *Paradigm* of the *Model Specification*, and the *Test Selection Criteria* applied for *Test Generation*, which again affected our work.

Test Management

The selection, prioritization, and reduction of test cases are driven by factors negatively affecting the targeted project result. Felderer and Schieferdecker (2014) evaluated multiple approaches dealing with these risk factors. Further, the set of risk-based testing approaches was ranked according to their potential for automation, which marks an important point of the previously presented life cycle. Instead of focusing on how to manage risks, Elbaum et al. (2002) presented work on how to rank test cases. The overlap of both contributions represents the major inspiration for our presented work.

Fault-Based/Mutation Testing

Beside, purely coverage-oriented test selection criteria extensively used in code-based testing approaches, we additionally focused on the expressiveness and effectiveness of tests, DeMillo et al. (1978) formerly investigated on with his contribution on mutation. Originated from this work, many modified versions of mutation testing evolved, widely surveyed by Jia and Harman (2011). Further, the challenge of generating meaningful test cases from UML-based models with respect to robustness, Krenn et al. (2015) orchestrated a set of tools called *MoMuT*. Concepts applied by these tools again inspired parts of our *Model-based STLC*, whereas their purpose is a slightly different one in our context.

5 CONCLUSIONS

We have presented our concept of a consistent and strictly model-based software-testing life cycle, which is in line with the ISO/IEC/IEEE 29119 standard for software testing. As postulated in the problem statement, there are three major challenges we overcome with this approach. First, due to the Omni Model approach, which serves as a central

model connecting dedicated activities of the testing life cycle via model artifacts, the basis for consistent automation throughout all steps of the testing process is given. Second, the model-based nature of all activities gives testers the possibility to perform tests on early development artifacts and thus gain information about the presence of defects as soon as possible. Additionally, the model-based way of specifying information, in turn, enables testers to focus on the conceptual tasks by abstracting from details. The abstraction from details, which is carried out consistently, prevents effects like test case explosion to take place during testing, due to steadily encapsulating details and concentrating on the essential. Further, our approach is not only focused on the *Dynamic Test Processes* of the ISO/IEC/IEEE 29119 standard. Organizational and Test Management aspects are also aligned to the model-based way of specifying and using information. The testers view and the management view on the project are thereby harmonized, to produce high-quality products. All over, we introduced a new model-based approach, aiming for the detection of defects, introduced in early stages of software development, which is applicable to design-time artifacts as well as code.

Nevertheless, we previously discussed the shortcomings, potentially hindering the practical application as well as the acceptance by the testing community.

REFERENCES

ANSI/IEEE 1008 (1987). IEEE Standard for Software Unit Testing. Standard, IEEE Computer Society Press, Washington, USA.

ANSI/IEEE 1012 (2012). IEEE Standard for System and Software Verification and Validation. Standard, IEEE Computer Society Press, Washington, USA.

Apfelbaum, L. and Doyle, J. (1997). Model based testing. In *Software Quality Week Conference*, pages 296–300.

BS 7925-1 (1998). Software testing. Vocabulary. Standard, BSI British Standards, London, UK.

BS 7925-2 (1998). Software testing. Software Component Testing. Standard, BSI British Standards, London, UK.

Daoudagh, S., Kateb, D. E., Lonetti, F., Marchetti, E., and Mouelhi, T. (2015). A toolchain for model-based design and testing of access control systems. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 411–418.

de Lara, J., Guerra, E., and Cuadrado, J. S. (2015). Model-driven engineering with domain-specific meta-modelling languages. *Software & Systems Modeling*, 14(1):429–459.

DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41.

Elbaum, S., Malishevsky, A. G., and Rothermel, G. (2002). Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, 28(2):159–182.

Felderer, M. and Schieferdecker, I. (2014). A taxonomy of risk-based testing. *International Journal on Software Tools for Technology Transfer (STTT)*, 16(5):559–568.

IEEE 829 (2008). IEEE Standard for Software and System Test Documentation. Standard, IEEE Computer Society Press, Washington, USA.

ISO 29119-1 (2013). Software and systems engineering – Software testing – Part 1: Concepts and definitions. Standard, International Organization for Standardization, Geneva, CH.

ISO 29119-2 (2013). Software and systems engineering – Software testing – Part 2: Test Processes. Standard, International Organization for Standardization, Geneva, CH.

ISO 29119-3 (2013). Software and systems engineering – Software testing – Part 3: Test Documentation. Standard, International Organization for Standardization, Geneva, CH.

ISO 29119-4 (2015). Software and systems engineering – Software testing – Part 4: Test Techniques. Standard, International Organization for Standardization, Geneva, CH.

ISO 29119-5 (2016). Software and systems engineering – Software testing – Part 5: Keyword-Driven Testing. Standard, International Organization for Standardization, Geneva, CH.

Jia, Y. and Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, 37(5):649–678.

Krenn, W., Schlick, R., Tiran, S., Aichernig, B., Jobstl, E., and Brandl, H. (2015). Momut::uml model-based mutation testing for uml. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–8.

Rumpold, A., Pröll, R., and Bauer, B. (2017). A domain-aware framework for integrated model-based system analysis and design. In *MODELSWARD17*, pages 157–168.

Saad, C. and Bauer, B. (2013). Data-flow based model analysis and its applications. In *International Conference on Model Driven Engineering Languages and Systems*, pages 707–723. Springer.

Utting, M., Pretschner, A., and Legeard, B. (2012). A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312.

Zschaler, S., Kolovos, D. S., Drivalos, N., Paige, R. F., and Rashid, A. (2009). Domain-specific metamodelling languages for software language engineering. *SLE*, 9:334–353.