# Formalizing the ISO/IEC/IEEE 29119 Software Testing Standard

Shaukat Ali
Simula Research Laboratory
Oslo, Norway
shaukat@simula.no

Tao Yue
Simula Research Laboratory, University of Oslo
Oslo, Norway
tao@simula.no

***Abstract*—Model-based testing (MBT) provides a systematic and automated way to facilitate rigorous testing of software systems. MBT has been an intense area of research and a large number of MBT techniques have been developed in the literature and in the practice. However, all of the techniques have been developed using their own concepts and terminology of MBT, which are very often different than other techniques and at times have conflicting semantics. Moreover, while working on MBT projects with our industrial partners in the last several years, we were unable to find a unified way of defining MBT techniques based on standard terminology. To precisely define MBT concepts with the aim of providing common understanding of MBT terminology across techniques, we formalize a small subset of the recently released ISO/IEC/IEEE 29119 Software Testing Standard as a conceptual model (UML class diagrams) together with OCL constraints. The conceptual model captures all the necessary concepts based on the standard terminology that are mandatory or optional in the context of MBT techniques and can be used to define new MBT tools and techniques. To validate the conceptual model, we instantiated its concepts for various MBT techniques previously developed in the context of our industrial partners. Such instantiation automatically enforces the specified OCL constraints. This type of validation provided us feedback to further refine the conceptual model. Finally, we also provide our experiences and lessons learnt for such formalization and validation.**

***Index Terms*—Model-Based Testing, ISO/IEC/IEEE 29119, UML, Test Case Generation, Modeling Methodology.**

## I. INTRODUCTION

Model-based testing (MBT) [1-3] provides a systematic way of testing software systems in a cost-effective manner by utilizing models as backbone models to facilitate various testing activities including test strategy definition, test data generation, and automated oracle. MBT, more specifically test case generation from models in the context of this paper, has gained attention in both industry and academia based on a variety of models (e.g., UML and SysML [4]) for various types of testing such as functional testing and extra-functional testing (e.g., robustness) and a large number of MBT techniques and tools have been developed [2, 3, 5-12].

We have been working on several MBT projects at Certus Software Verification and Validation Center (http://certus-sfi.no/) with industrial partners including Cisco Systems, ABB Robotics, Tomra AS, and Western Geco since 2007. While defining MBT techniques for our industrial partners, we were unable to find standardized concepts and terminology to do so. The only standardization efforts for MBT we could find were UML Testing Profile (UTP) Version 1 from Object Management Group (OMG) [13] and ETSI's ES 202 951 standard (released 2011), which were not sufficient enough to support MBT in our industrial partners. Even thorough investigation of MBT techniques in both academic literature and practice revealed that most of the techniques use their own concepts and terminology that usually differ from the concepts and terminology in other techniques and sometimes have conflicting semantics. As a result, we have to define our own concepts and terminology to define MBT techniques for our industrial contexts.

With the advent of the ISO/IEC/IEEE 29119 Software Testing Standard [14] in September 2013, we decided to formalize it as a conceptual model consisting of testing concepts, their relationships, and constraints. The conceptual model covers all the concepts required for defining an MBT technique. The conceptual model is implemented as a UML Class diagram with constraints specified in the Object Constraint Language (OCL) using the IBM Rational Software Architect tool, which facilitates automated validation of constraints on the instances of the models (Object diagrams in our case). Notice that in this paper, we formalized only a small subset of the standard related to the test case design and more comprehensive formalization of the standard is our future work.

The formalization of the conceptual model offers several benefits including: 1) It provides a unified meaning of MBT concepts based on the standard terminology to researchers and practitioners, 2) Such formalization may be used as a reference model to develop MBT tools in the future, 3) The conceptual model serve as a starting point to define a new MBT technique, 4) The conceptual model with the tool support can be used to automatically determine conformance of an MBT technique in terms of all the mandatory concepts to the standard, 5) As a reference model, the conceptual model may facilitate communication among different standardization bodies (e.g., ETSI and OMG) that can potentially initiate a joint effort for the standardization of MBT.

To validate the conceptual model, we instantiated the concepts in the conceptual model for our existing MBT techniques. Such instantiation provided us feedback on the conceptual model, which was further used to refine the conceptual model. Finally, we also report our experiences and lessons learnt from the development and validation of the

conceptual model. The rest of the paper is organized as follows: We present the conceptual model of the standard in Section II followed by the methodology to use the conceptual model in Section III. Section IV presents the mapping of various MBT techniques to the concepts in the conceptual model and we present results and discussion based on the mapping in Section III. Section VI presents the related work, whereas we conclude the paper in Section VII.

## II. CONCEPTUAL MODEL OF THE ISO/IEC/IEEE 29119 SOFTWARE TESTING STANDARD

In this section, we provide the conceptual model that we developed based on the standard. Notice that for the sake of clarity we provide different views of the model based on the concepts and in the implementation all the concepts are linked. The conceptual model was implemented as a set of UML class diagrams together with OCL constraints in the IBM's Rational Software Architecture (RSA) tool. In this paper, we provide a few OCL constraints as examples. The RSA tool allows automated validation of OCL constraints when the conceptual model is instantiated as object diagrams. TABLE I shows the list of defintions of various concepts from the standard. Notice that these definitions are taken as it is from the standard. To avoid cluttering, we didn't provide each definition within double quotes.

### A. Test Case and Test Set

The first and foremost concept in testing is *Test Case* whose definition is shown in Concept 1 in TABLE I and is modeled as an abstract concept *Test Case* in Fig. 1. Since the ISO standard is defined for software testing in general, we specialized the concept into two concepts to support MBT: *Abstract Test Case* and *Concrete Test Case* as shown in Fig. 2 and Fig. 3 respectively.

TABLE I RELEVANT DEFINITION FROM ISO/IEC/IEEE 29119 SOFTWARE TESTING STANDARD [14]

| # | Concept | Definition |
|---|---|---|
| 1 | Test Case | Set of test case preconditions, inputs (including actions, where applicable), and expected results, developed to drive the execution of a test item to meet test objectives, including correct implementation, error identification, checking quality, and other valued information |
| 2 | Test Set | Set of one or more test cases with a common constraint on their execution |
| 3 | Test Design Technique (Test Model) | Activities, concepts, processes, and patterns used to construct a test model that is used to identify test conditions for a test item, derive corresponding test coverage items, and subsequently derive or select test cases |
| 4 | Test Case Specification | Documentation of a set of one or more test cases |
| 5 | Test Basis | Body of knowledge used as the basis for the design of tests and test cases |
| 6 | Test Procedure | Sequence of test cases in execution order, and any associated actions that may be required to set up the initial preconditions and any wrap up activities post execution |
| 7 | Test Procedure Specification | Specification document specifying one or more test procedures, which are collections of test cases to be executed for a particular objective |
| 8 | Test Data | Data created or selected to satisfy the input requirements for executing one or more test cases, which may be defined in the Test Plan, test case or test procedure |
| 9 | Test Item | Work product that is an object of testing |
| 10 | Test Plan | Detailed description of test objectives to be achieved and the means and schedule for achieving them, organized to coordinate testing activities for some test item or set of test items |
| 11 | Feature Set | Collection of items which contain the test conditions of the test item to be tested which can be collected from risks, requirements, functions, models, etc. |
| 12 | Test Condition/Test Requirement | Testable aspect of a component or system, such as a function, transaction, feature, quality attribute, or structural element identified as a basis for testing |
| 13 | Test Strategy | Part of the Test Plan that describes the approach to testing for a specific test project or test sub-process or sub-processes |
| 14 | Test Coverage | Degree, expressed as a percentage, to which specified test coverage items have been exercised by a test case or test cases |
| 15 | Test Coverage Item | Attribute or combination of attributes that is derived from one or more test conditions by using a test design technique that enables the measurement of the thoroughness of the test execution |
| 16 | Test Result | Indication of whether or not a specific test case has passed or failed, i.e. if the actual result observed as test item output corresponds to the expected result or if deviations were observed |
| 17 | Actual Result | Set of behaviours or conditions of a test item, or set of conditions of associated data or the test environment, observed as a result of test execution |
| 18 | Expected Result | Observable predicted behaviour of the test item under specified conditions based on its specification or another source |
| 19 | Pass or Fail Criteria | Decision rules used to determine whether a test item, or feature of a test item, has passed or failed after testing |

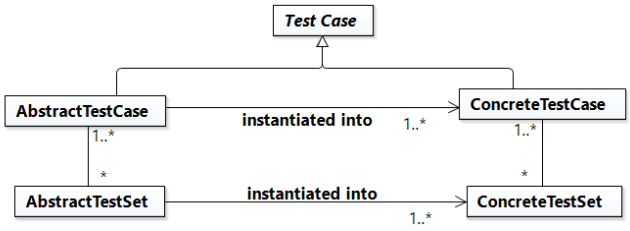


Fig. 1. Conceptual model for *Abstract/Concrete Test Case* and *Test Set*

One abstract test case can be instantiated into one or more concrete test cases based on test data. An *Abstract Test Set* is a set of one or more abstract test cases, whereas a *Concrete Test Set* consists of more than one concrete test case. Depending on how many times one abstract test case is instantiated, the size of concrete test set can be more than the size of abstract test set. This rule is represented as an OCL constraint below:

```
context AbstractTestCase inv:
self.concretetestcase->size() >1 implies
self.abstracttestset-
>collect(abstracttestset.abstracttestcase-
>size())->forAll(atsize:Integer|
self.concretetestcase-
>collect(concretetestcase.concretetestset.concr
etetestcase->size())-
>forAll(ctsize:Integer|ctsize>atsize))
```

On the other hand, if each abstract test case is instantiated exactly once, the size of abstract test set and concrete test set must be equal. This rule is represented as an OCL constraint below:

```
AbstractTestCase.allInstances()-
>forAll(atc:AbstractTestCase|atc.concretetestca
se->size()=1) implies
AbstractTestSet.allInstances()-
>collect(ats:AbstractTestSet|ats.abstracttestca
se->size())->forAll(atsize:Integer|
ConcreteTestSet.allInstances()-
>collect(cts:ConcreteTestSet|cts.concretetestca
se->size())-
>forAll(ctsize:Integer|ctsize=atsize))
```

An *Abstract Test Case* (Fig. 2) comprises of three items: 1) At least one *Input/Stimulus* to *Test Item* (Concept 9 in TABLE I, commonly known as System Under Test in software testing literature [15]). Example of the stimulus to a test item include access to test item via application programming interface (API): 2) A set of *Test Data Specification* related to *Input/Stimulus* (e.g., specification of ranges of values of input parameters). For example, a set of valid values of all parameters of an API: 3) Specification of *Expected Result* (Concept 18 in Table 1). For example, values of state variables of test item representing a correct state.

A *Concrete Test Case* (Fig. 3) is similar to *Abstract Test Case*; however the only difference is that *Test Data Specification* has been realized into concrete values called as *Test Data* (Concept 8, Fig. 4). In other words, the exact values for the parameters of stimulus or other parameters of a test case (e.g., configuration parameters) have been selected by applying any test data generation strategy, such as equivalence partitioning and boundary value analysis [15, 16]. Each concrete test case has unique set of test data. This rule is formalized as an OCL constraint below:

```
context ConcreteTestCase inv:
self.stimuli->isUnique(stimuli.testdata) or
self->isUnique(testdata)
```

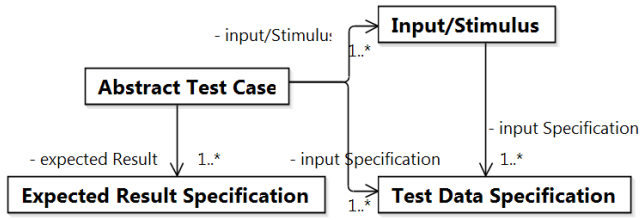


Fig. 2. Conceptual model for *Abstract Test Case*

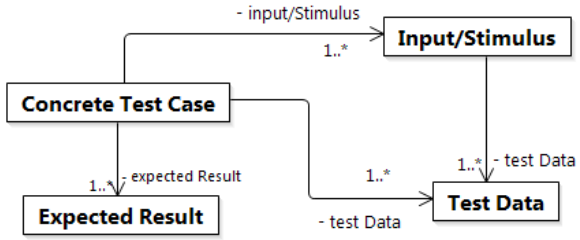


Fig. 3. Conceptual model for *Concrete Test Case*

A *Test Set* (Concept 2) is specialized into two types: *Abstract Test Set* that contains *Abstract Test Cases*, whereas *Concrete Test Set* contains *Concrete Test Cases*. The difference between the *Abstract Test Set* and *Concrete Test Set* is the same as *Abstract Test Case* and *Concrete Test Case*, i.e., a *Concrete Test Case* has exact data values and *Concrete Test Set* is composed of such concrete test cases.

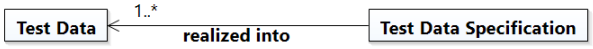


Fig. 4. Conceptual model for *Test Data*

Notice that an MBT technique doesn't necessarily need to have these two levels and only *Concrete Test Case* and *Concrete Test Set* are mandatory. An OCL constraint implementing such rule is shown below:

```
AbstractTestCase.allInstances()->size() = 0
implies AbstractTestSet.allInstances()->size()
= 0
```

### B. Test Model

The *Test Model* concept (Concept 3) is modeled in Fig. 5 showing its relationships to other testing concepts and is the model used to derive abstract/concrete test cases. An example of *Test Model* in UML is a UML State Machine, from which a set of abstract/concrete test cases (e.g., a sequence of states and transitions) can be derived. A *Test Model* is developed using a *Test Design Technique* (Concept 3) based on *Test Case Specification* (Concept 4) that documents one or more test cases usually in textual format. *Test Case Specification* is usually a subset of *Test Basis* (Concept 5), which is a "Body of knowledge used as the basis for the design of test cases" [14].

*Test basis* is the first set of documents that are used for deriving test case specifications followed by developing a test model. Examples of test basis include a set of requirements specifications and all other information about a test item independent of testing considerations. In contrast, test case specifications are documents containing all the testing requirements for a test item. To define an MBT technique, it is not always necessary to have documents available for test basis and test case specifications, as we realized while working with our industrial partners and most of the test case specifications were rather available as tacit knowledge from test engineers and test managers.

### C. Test Procedure

*Test Procedure* (Concept 6) is defined by ISO as: "Sequence of test cases in execution order, and any associated actions that may be required to set up the initial pre-conditions and any wrap up activities post execution" [14] and its relationships to other related concepts are shown in Fig. 6. As for *Test Case*, we specialized *Test Procedure* into *Abstract Test Procedure* and *Concrete Test Procedure* and the only difference between the both is that *Abstract Test Procedure* contains *Test Data Specification*, whereas *Concrete Test Procedure* contains *Test Data*. A MBT technique doesn't necessarily need to have separation between abstract and concrete test procedure, as is the case with test case and test set. Moreover, a test procedure may not even exist in a particular context and the corresponding OCL constraint is shown below:

```
TestProcedure.allInstances()->size()>=0
```

### D. Test Plan

*Test Plan* (Concept 10) defines the *Test Objectives* and *Testing Activities* to achieve them. The most important part of *Test Plan* is *Test Strategy* (Concept 13) in the context of MBT that actually describes the approach for testing *Test Item* (Concept 9) based on *Test Objective* (e.g., functional testing or extra-functional testing) as shown in Fig. 6. Examples of testing activities include test generation, test minimization, and test scheduling. In the context of the current paper, we are only focusing on test generation as a testing activity.

A *Test Item* (Fig. 6) has a set of related test conditions or test requirements (*Test Condition*/*Test Requirement* Concept number 12) that describe the testable aspects of the *Test Item*, for example, a function, transaction, or a feature that are related to *Feature Set* (Concept 11, for example, features, requirements, and functions) of the *Test Item*.

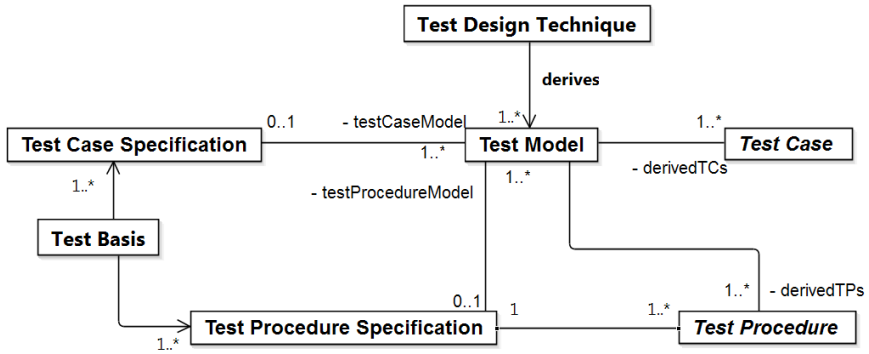


Fig. 5. Conceptual model for *Test Model*

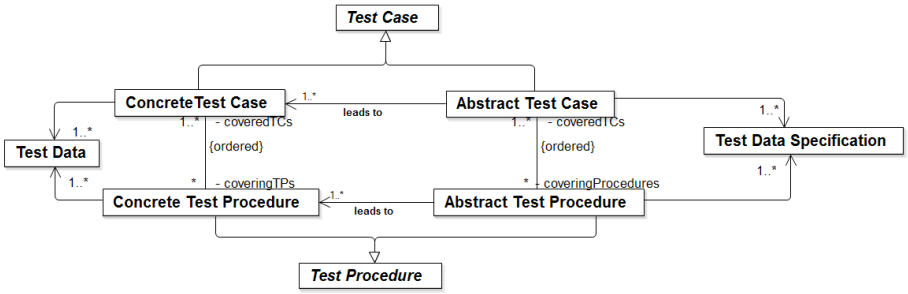


Fig. 6. Conceptual model for *Test Procedure*

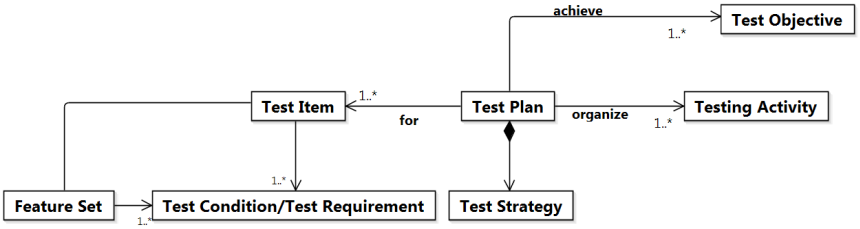


Fig. 7. Conceptual model for *Test Plan*

### E. Test Coverage

*Test Coverage* (Concept 14) is calculated when a test driver (*Test Driver*) executes a set of concrete test cases on *Test Item* and as a result, a set of coverage items (*Coverage Item*/*Test Coverage Item* concept 15) defined by test conditions (*Test Conditions*) are covered as shown in Fig. 7. Typical examples of test coverage of test item include code coverage, for example, statement coverage and branch coverage.

Notice that *Test Coverage* in the standard specifically refers to the coverage of coverage items on *Test Item* and the standard doesn't define an explicit concept for the coverage of *Test Model*. However, the coverage of *Test Model* is implicitly covered in the *Test Strategy* inside *Test Plan*. For example, considering example of a UML state machine as a test model, typical test coverage includes *All State Coverage* and *All Transition Coverage*.

### F. Test Result

A test driver executes a concrete test case on a test item and uses *Test Result* to compare expected result specified in the test case with the actual result obtained from the test item using a *Pass/Fail Criteria* (Concept 19) to determine if the test case is passed or failed (Fig. 8 and Fig. 9). In practice, in addition to pass or fail, the result of a test case can also be inconclusive or may also be an error. However, these concepts are not represented in the standard.

## III. Methodology

In this section, first we provide two key activities in automated MBT. First, in Section III.A, we discuss about how to transit from test basis to the derivation of one or more test models. In Section III.B, we discuss two key types of test generators to support generation of executable test cases. Finally in Section III.C, we provide the two-step procedure that we used to validate the conceptual model.

### A. Designing a Test Model

Fig. 10 shows an activity diagram for defining a test model that is used as a key component for deriving test cases and test procedures. The activity diagram is divided into three activities, where the first activity is related to identifying the test basis that determines body of knowledge available to derive test case specification, test procedure specification, and test requirements of test item shown as three parallel activities in Fig. 10. Notice that in a certain application context, the test basis, test case specifications, and test procedure specifications may not be explicitly available in the form of documents and may be available as domain knowledge from the domain experts. The third activity involves defining test models and there are several possibilities for defining test models: 1) One may need to create one test model for test case specification and one test model for test procedure specification: 2) A test model only for test case specification and there is no test procedure specification available: 3) More than one test models for test case specification and test procedure specification.

One way of MBT using UML State machines involves two test models: 1) A class diagram capturing state variables and stimulus as operations/signals representing the static structure of test item: 2) A state machine modeling the behavior of test item.

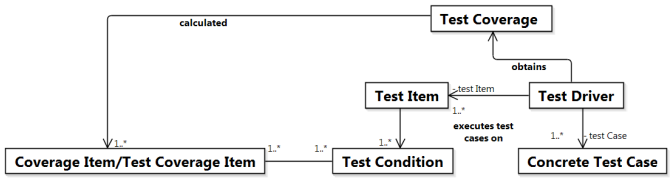


Fig. 8. Conceptual model for *Test Coverage*

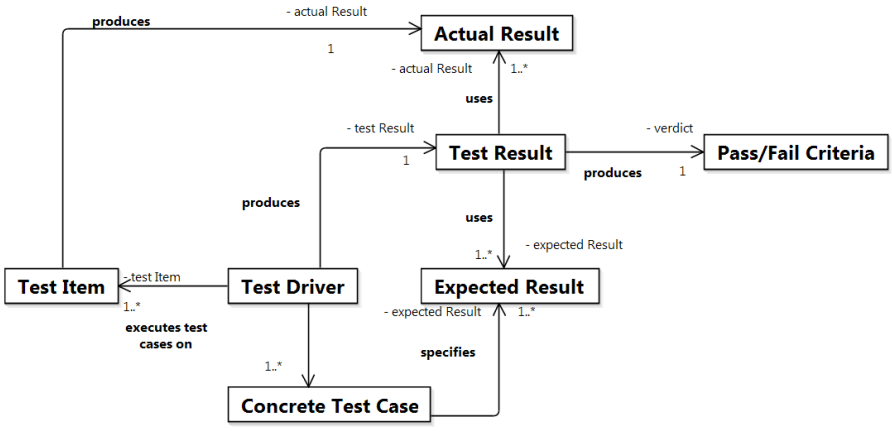


Fig. 9. Conceptual model for *Test Result*

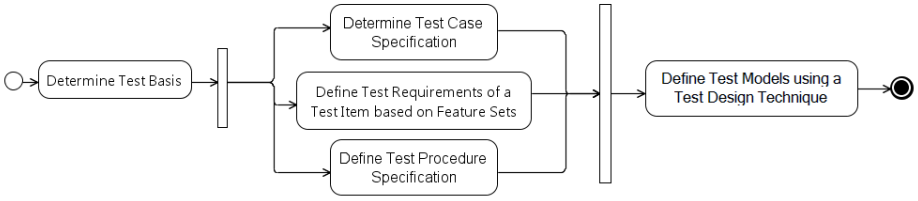


Fig. 10. An activity diagram for deriving test models

### B. Generation of Test Cases and/or Test Procedures

In this section, we discuss the activities for transforming test models into test cases and test procedures as shown in Fig. 11. As a first step, an *Abstract Test Generator* uses test models as input and generates *Abstract Test Set* applying one or more test strategy, e.g., covering certain structural features of a the test model (Fig. 11). In the second step, the abstract test set is transformed into concrete test set using *Concrete Test Generator*. Notice that these two levels of test generator are dependent on the application context and only a test generator that generates concrete test cases is mandatory. The choice of having an abstract test generator facilitates moving some of the complexity of platform specific test generator into concrete test generator. Such a separation allows defining various platform specific concrete test generators that permits generating executable test cases in various test scripting languages such as Java and Python. Moreover, an abstract test generator separates the details of test strategies from the concrete test generator, e.g., coverage criteria and facilitates incorporation of new test strategies in abstract test generator without affecting the implementation of concrete test generator. In case of having only concrete test generator, all the complexity of test strategies and platform specific details are inside the generator and maintenance requires much more effort than using the two types of generators.

### C. Validation Procedure

The conceptual model was validated in two phases. During the first phase, the conceptual model was developed incrementally based on the standard's documents. The authors validated the conceptual model using the inspection methods. In the second phase, the conceptual model was validated by instantiating its concepts based on the MBT techniques from our existing industrial applications. The two techniques were based on UML and its extensions, whereas the third one was based on a domain specific language. Based on these instantiations, we obtained the final version of the conceptual model, which is presented in the paper. Typical problems that we encountered and eventually addressed during the revisions of conceptual model were mostly related to incorrect cardinalities on associations between two concepts and incorrect associations.

## IV. Mapping to MBT Techniques

In this section, we present mapping of various concepts from the standard to various MBT techniques that we developed with our industrial partners in the last several years. In Section IV.A, we present the mapping of the concepts of various MBT techniques based on UML state machines that we applied to several industrial case studies and Section IV.B presents the mapping to a test case specification language that we developed in the context of one industrial partner by defining a domain specific language to support automated testing.

### A. UML State Machine-Based Testing

In this section, we will discuss MBT using UML state machines for supporting functional testing and robustness testing.

*1) Functional Testing :* While performing functional testing using UML state machines, we worked with the following four case studies.

*Video Conferencing System (VCS).* The first case study is about black-box system-level testing of the Videoconferencing Systems (VSCs) developed by Cisco Systems, Norway by means of systematic test automation [17]. We targeted one particular VCS called C90 and we modeled all the 20 subsystems using UML state machines and UML class diagrams.

*Safety Monitoring Component (SMC).* The second case study is about MBT of Safety Monitoring Component (SMC) developed by ABB robotics. SMC exhibits state-based behavior and we used UML state machines to test SCM. The implementation of SMC is in C++ [6].

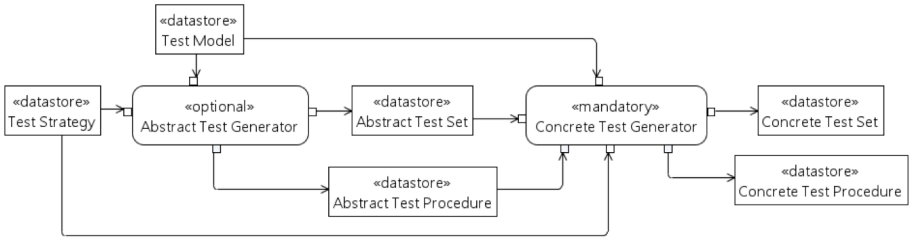


Fig. 11. Activity diagram for test generators

TABLE II MAPPING OF CONCEPTS FROM VARIOUS MBT TECHNIQUES TO THE CONCEPTS IN THE STANDARD

| # | Concept | State Machine (SM) | Aspect State Machine (ASM) | RTCM |
|---|---|---|---|---|
| 1 | Abstract Test Case | Sequence of (State->Transition->State) | Same as SM | Sequence of statements |
| 2 | Abstract Test Set | A set of abstract test cases | Same as SM | A set of test case specification |
| 3 | Concrete Test Case | A sequence of statements setting configuration parameters, checking state variables, calling APIs with data (In Python, Java) | Same as SM | Same as SM |
| 4 | Concrete Test Set | A set of concrete test cases | Same as SM | Same as SM |
| 5 | Stimulus | Trigger of type Call/Signal Event | Change Event | API |
| 6 | Test Design Technique (Test Model) | UML State Machine and Class diagrams are test models | Aspect State Machine (ASM), Aspect Class Diagram, UML Class diagrams and state machines | Test Case Specification in RTCM |
| 7 | Test Case Specification | Specifications about VCSs, Safety Monitoring Component, Bottle Recycling System, control system for marine seismic acquisition | Specifications about VCSs | Specifications about VCSs |
| 8 | Test Basis | Specification, Requirements Documents, User Manuals | Same as SM | Same as SM |
| 9 | Test Procedure | - | - | - |
| 10 | Test Procedure Specification | - | - | - |
| 11 | Test Data | Parameters of APIs of VCS modeled on Triggers as Signal, Call Event, Time Event | Values of states of Test Item and environment | Parameters of API |
| 12 | Test Data Specification | Guards, Time Event | Change Event | Branches in test case specifications |
| 13 | Test Item | VCS, Safety Monitoring Component, Bottling Recycling System, Control System of marine seismic acquisition | VCS | VCS |
| 14 | Test Plan | - | - | - |
| 15 | Feature Set | Features of Test Items | VCS Features | VCS Features |
| 16 | Test Condition/Test Requirement | Covering each feature of test item | Covering each VCS feature | Covering each VCS feature |
| 17 | Test Strategy | - | - | - |
| 18 | Test Coverage/Test Model Coverage | API Coverage, State Coverage, Configuration Coverage | Robustness Properties coverage | Same as SM |
| 19 | Test Coverage Item | API, Status, Configuration | Property | Same as SM |
| 20 | Test Result | - | - | - |
| 21 | Actual Result | State values | State values | State values |
| 22 | Expected Result Specification | OCL Constraints as State Invariants | Same as SM | Conditions written in restricted natural language |
| 23 | Pass or Fail Criteria | Evaluation of OCL Constraints | Same as SM | String matching |

*Control System for Marine Seismic Acquisition.* The third case study is from WesternGeco, who is a market leader in the field of seismic systems. The case study [10] is about a large and complex control system for marine seismic acquisition. The system controls tens of thousands of sensors and actuators in its environment. The timing deadlines on the environment are in the order of hundreds of milliseconds. The system was developed using the Java language.

*Bottle Recycling Machine.* The fourth case study is an automated bottle-recycling machine developed by Tomra AS [10]. The system under test (SUT) was an embedded device 'Sorter', which was responsible to sort the bottles into their

appropriate destinations. The system communicated with a number of components to guide re-cycled items through the recycling machine to their appropriate destinations. It is possible to cascade multiple sorters with one another, which results in a complex recycling machine. The SUT was developed using the C language.

*Summary of Results*. For the first two case studies, we implemented UML state machines based testing techniques in a tool called Transformation-based Tool for UML-based Testing (TRUST) [6], for the rest of the two case studies separate tools were developed. The third column in TABLE II provides results for MBT solutions for these four case studies. For all these case studies, our tools transform test models into abstract test cases and then the abstract test cases were transformed into executable test cases.

*2) Robustness Testing:* This case study is an extension of case study 1 and is about supporting automated, model-based robustness testing of the C90 VCS. The VCS should be robust enough to handle the possible abnormal situations that can occur in its operating environment and invalid inputs. For example, C90 should be robust against hostile environment conditions (regarding the network and other communicating VCSs), such as high percentage of packet loss and high percentage of corrupt packets. Such behavior is very important for a commercial VCS and must be tested systematically and automatically to be scalable. More details on the robustness behavior of C90 and its functional modeling can be found in [17]. An extended version of the TRUST tool was used for MBT in this case. The results for this case are provided in the fourth column. The TRUST worked in two stages: a) In the first stage, it trans-forms test models, i.e., UML class diagrams and state machines and Aspect class diagram and Aspect state machines into abstract test cases: b) In the second stage, the abstract test cases were converted into concrete test cases.

### B. RTCM

In this section, first we provide a brief overview of our test case specification language followed by summary of key results.

*1) Approach:* Restricted Test Case Modeling (RTCM) [18] is a test case specification language, which is an extension of a language called Restricted Use Case Modeling (RUCM) [19]. The specialty of RTCM lies in providing a precise way of specifying natural language test case specifications with restriction rules, easy to use template, and a set of keywords. All the features of RTCM are captured formally in TCMeta extending UCMeta that is a metamodel behind RUCM. The tool support with RTCM (aToucan4Test) supports automated generation of executable test cases from RTCM specifications.

*2) Summary of Results:* We used the case study 1 (Section IV.A) with RTCM. More details of modeling the case study can be found in [18]. The fifth column in TABLE II shows the results related to RTCM. In case of this approach, test generator, i.e., aToucan4Test doesn't generate abstract test case and transformation is directly to executable test cases.

## V. Overall Results, Experience and Lessons Learnt

In this section, we provide discussion on our experience of mapping the concepts of standard to our MBT techniques.

### A. Unified Meaning of MBT Concepts

The results presented in TABLE II show the mapping of the concepts from MBT techniques based on UML state machines, Aspect state machines (extension of UML state machines), and RTCM (a domain specific language) [18]. As discussed in Section IV, such mapping was performed by instantiating the concepts of conceptual model in the IBM RSA tool. We noticed that even though the MBT techniques are different, the concepts can still be mapped to the standard software testing concepts in the standard even though the standard is still exclusively lacking model-based test design.

### B. Tailoring of the Standard

Based on our experience of working with the standard, we feel that the standard is defined at a very high level of abstraction. Such level of abstraction, on the one hand introduces ambiguity in understanding the concepts, whereas on the other hand, is applicable to different types of testing techniques such as model-based testing and code-based testing. To deal with the ambiguity, we developed the conceptual model, whereas with respect to its applicability to different types of testing techniques, we demonstrated its application to three types of testing techniques, i.e., based on standard UML state machines, aspect state machines (extensions of UML), and RTCM—a domain specific testing language.

### C. Guidelines to Develop New MBT Technique

The proposed conceptual model can be used as a starting point for the development of new MBT techniques. The designer of an MBT technique needs to consider at least all the mandatory concepts to support automated testing and ensure that all the constraints on the conceptual model are validated to be true. Notice that our conceptual model is generic and independent of any modelling language. This means that it can be used to define either totally new language for MBT or as extensions of existing languages. We demonstrated this by mapping three types of MBT techniques to the conceptual model in the paper.

### D. Conformance of an MBT Technique to the Standard

The implementation of the conceptual model as a metamodel together with the OCL constraints can help to assess how much an existing MBT technique or tool conforms to the standard. This can particularly help industry to select the tools and methodologies that conform to the standard since it is well-known fact that standard-based tools and techniques are preferred in the industry.

### E. Enabling Communication Among Different Standardization Bodies

The conceptual model may serve as a reference model and may be used to enable common understanding and unified communication among other standardization bodies focusing on MBT such as ETSI and Object Management Group. This can further initiate discussion among these standardization bodies to start a joint effort on the standardization of MBT techniques.

## VI. Related Work

There are a few efforts in standardization of model-based testing that currently exists. One well-known effort in this regard is UML Test Profile Version 1 [13] from Object Management Group (OMG). The profile provides lightweight extension to UML for various testing activities such as test design, specification, and visualization. The main purpose of the profile was to keep it general enough such that it can be used in various domains. The profile focused on modelling the interactions of SUT with its environment and test execution system. Another recent standard on model-based testing is the ETSI ES 202 951 standard published in 2011 with participation from various MBT tool vendors, users, and research institutes. The standard defines a language called Test Description Language (TDL) that supports modelling test description, test configuration, and test data. There are several software testing standards in the literature such as IEEE 829-2008 Standard for Software and System Test Documentation [20] and the recent ISO/IEC/IEEE 29119 Standard [14] that combines efforts from IEEE/ISO, and IEC. The work presented in this paper is a first attempt towards this direction, which formalizes the testing concepts of the ISO/IEC/IEEE 29119 standard to enable model-based test design. One of the aims of such formalization is to provide a common ground among the different standardization bodies such as ETSI and OMG to develop joint MBT standards in the future.

## VII. Conclusion

Model-based Testing (MBT) has proven to provide a systematic and automated way of testing complex systems and has been area of much focus in industry and academic in the last decade. A large number of MBT techniques and tools have emerged targeting various types of testing. Most of the techniques, however, lack a unified understanding of concepts of MBT and as result decreases the interoperability among the techniques and tools.

To provide a unified understanding of MBT concepts based on the standard testing terminology and relationship among the concepts, we formalizes the concepts from the recent software testing standard: the ISO/IEC/IEEE 29119 Standard as a conceptual model implemented as UML class diagram. A set of constraints in OCL is also defined to automatically enforce various constraints on the conceptual model. The conceptual model serves as a reference model that can be used as a starting point for developing new MBT techniques. The current version of the conceptual model formalizes only a small subset of the standard related to test case design.

To assess validate the conceptual model, we mapped the concepts of various MBT techniques we developed over the last several years with several industrial partner's to the conceptual model. Based on such validations, an improved version of the conceptual model is presented in the paper. In the future, we plan to extend our conceptual model to more comprehensively formalize the standard.

## References


[1] Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A.: Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science). Springer-Verlag New York, Inc. (2005)

[2] Neto, A.C.D., Subramanyan, R., Vieira, M., Travassos, G.H.: A survey on model-based testing approaches: a systematic review. Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007, pp. 31-36. ACM, Atlanta, Georgia (2007)

[3] Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann (2007)

[4] Weilkiens, T.: Systems Engineering with SysML/UML: Modeling, Analysis, Design. Tim Weilkiens (2008)

[5] Ali, S., Hemmati, H.: Model-based Testing of Video Conferencing Systems: Challenges, Lessons Learnt, and Results. In: IEEE International Conference on Software Testing, Verification, and Validation (ICST). (Year)

[6] Ali, S., Hemmati, H., Holt, N.E., Arisholm, E., Briand, L.C.: Model Transformations as a Strategy to Automate Model-Based Testing - A Tool and Industrial Case Studies. Simula Research Laboratory, Technical Report (2010-01) (2010)

[7] Ali, S., Iqbal, M.Z., Arcuri, A., Briand, L.: Generating Test Data from OCL Constraints with Search Techniques. Simula Research Laboratory (2012)

[8] Hemmati, H., Arcuri, A., Briand, L.: Reducing the Cost of Model-Based Testing through Test Case Diversity. 22nd IFIP International Conference on Testing Software and Systems (ICTSS), (2010)

[9] Hemmati, H., Arcuri, A., Briand, L.: Achieving scalable model-based testing through test case diversity. ACM Trans. Softw. Eng. Methodol. 22, 1-42 (2013)

[10] Iqbal, M.Z., Ali, S., Yue, T., Briand, L.: Experiences of Applying UML/MARTE on Three Industrial Projects. In: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 642-658. Springer Berlin Heidelberg, (Year)

[11] Sarma, M., Murthy, P.V.R., Jell, S., Ulrich, A.: Model-based testing in industry: a case study with two MBT tools. Proceedings of the 5th Workshop on Automation of Software Test, pp. 87-90. ACM, Cape Town, South Africa (2010)

[12] Shafique, M., Labiche, Y.: A systematic review of state-based test tools. Int J Softw Tools Technol Transfer 1-18 (2013)

[13] UML Testing Profile (UTP) Version 1, 2014, http://utp.omg.org/

[14] ISO/IEC/IEEE 29119 Software Testing, 2014, http://www.softwaretestingstandard.org/

[15] Binder, R.V.: Testing object-oriented systems: models, patterns, and tools. Addison-Wesley Longman Publishing Co., Inc. (1999)

[16] Ali, S., Briand, L.C., Hemmati, H., Panesar-Walawege, R.K.: A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation. IEEE Transactions on Software Engineering 99, (2009)

[17] Ali, S., Briand, L.C., Hemmati, H.: Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems. Software and Systems Modeling 11, (2012)

[18] Zhang, M., Yue, T., Ali, S.: A Keyword and Restricted Natural Language Based Test Case Specification Language for Automated Testing Simula Research Laboratory (TR 2014-01) (2014)

[19] Yue, T., Briand, L., Labiche, Y.: Facilitating the Transition from Use Case Models to Analysis Models: Approach and Experiments. Transactions on Software Engineering and Methodology (TOSEM) 22, (2013)

[20] IEEE Standard for Software and System Test Documentation, 2014, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=457827