ISTQB® Foundation Level Certified Model-Based Tester

Syllabus

Version 2015

International Software Testing Qualifications Board

Model-Based Tester Working Group: Stephan Christmann (chair), Anne Kramer, Bruno Legeard, Armin Metzger, Natasa Micuda, Thomas Mueller, Stephan Schulz; 2014-2015.

Revision History

Version	Date	Remarks
Beta 1.0	8 May 2015	First beta version – Update of the alpha version on the basis of findings from cross-functional review and technical editing
Version 2015 - Final	23rd October 2015	Final version after beta review.

Table of Contents

Acknowledgements

This document was produced by a team from the International Software Testing Qualifications Board Foundation Level Working Group.

The ISTQB® Foundation Level Certified Model-Based Tester (CTFL-MBT) team thanks the review team and all National Boards for their suggestions and input.

At the time the CTFL-MBT Syllabus was completed, the Model-Based Tester Working Group had the following membership: Stephan Christmann (MBT Working Group Chair), Anne Kramer, Bruno Legeard (MBT Author Group Co-Chair), Armin Metzger (MBT Author Group Co-Chair), Natasa Micuda (MBT Exam Question Group Chair), Thomas Mueller (ISTQB CTFL Working Group Chair), Stephan Schulz (MBT Review Group Chair).

Authors: Stephan Christmann, Lars Frantzen, Anne Kramer, Bruno Legeard, Armin Metzger, Thomas Mueller, Ina Schieferdecker, Stephan Weissleder.

Exam Question group: Bob Binder, Renzo Cerquozzi, Debra Friedenberg, Willibald Krenn, Karl Meinke, Natasa Micuda, Michael Mlynarski, Ana Paiva.

Exam Question reviewer group: Eddie Jaffuel, Ingvar Nordstrom, Adam Roman, Lucjan Stapp

Review group: Stephen Bird, Thomas Borchsenius, Mieke Gevers, Paul Jorgensen, Beata Karpinska, Vipul Kocher, Gary Mogyorodi, Ingvar Nordström, Hans Schaefer, Romain Schmechta, Stephan Schulz, Szilard Szell, Tsuyoshi Yumoto.

The team thanks also the following persons, from the National Boards and the model-based testing expert community, who participated in reviewing, commenting, and balloting of the Foundation Level Certified Model-Based Tester Syllabus: Patricia Alves, Clive Bates, Graham Bath, Rex Black, Armin Born, Bertrand Cornanguer, Carol Cornelius, Winfried Dulz, Elizabeta Fourneret, Debra Friedenberg, Kobi Halperin, Kimmo Hakala, Matthias Hamburg, Kari Kakkonen, Jurian van de Laar, Alon Linetzki, Judy McKay, Ramit Manohar, Rik Marselis, Natalia Meergus, Ninna Morin, Klaus Olsen, Tal Pe'er, Michael Pilaeten, Meile Posthuma, Ian Ross, Mark Utting and Ester Zabar.

This document was formally approved for release by the General Assembly of the ISTQB® on October 23, 2015.

0 Introduction to this Syllabus

0.1 Purpose of the Document

This syllabus forms the basis for the ISTQB® Certified Model-Based Tester, Foundation Level. It is a Specialist Extension to the ISTQB® Certified Tester Foundation Level (CTFL). The ISTQB® provides this syllabus as follows:

1. To National Boards, to translate into their local language and to accredit training providers. National Boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To Exam Boards, to derive examination questions in their local language based on the learning objectives.
3. To training providers, to produce training material and determine appropriate teaching methods.
4. To certification candidates, as one source to prepare for the exam.
5. To the international software and systems engineering community, to advance the profession of software and systems testing, and as a basis for books and articles.

The ISTQB® may allow other entities to use this syllabus for other purposes, provided they seek and obtain prior written permission.

0.2 Overview

The ISTQB® Foundation Level Certified Model-Based Tester syllabus is a specialist module at the Foundation Level. The ISTQB® Certified Tester Foundation Level (CTFL) certification is a required pre-requisite for the ISTQB® Foundation Level Certified Model-Based Tester (CTFL-MBT) exam.

A person with the CTFL-MBT Certificate has extended the broad understanding of testing acquired at the Foundation Level to enable capabilities required for the Model-Based Testing (MBT) process, methodology and techniques.

Expected Business Outcomes, career paths for testers and the intended audience for this syllabus are presented in the related Overview document [ISTQB_MBT_OVIEW].

0.3 Learning Objectives

The Learning Objectives support the Business Outcomes and are used to create examinations for achieving the CTFL-MBT certification.

In general, all parts of this syllabus are examinable at a K1 level, i.e., the candidate will recognize, remember and recall terms and concepts stated in the syllabus. The relevant Learning Objectives at K2 and K3 levels are indicated at the beginning of each chapter within this syllabus.

1 Introduction to Model-Based Testing – 90 mins.

Keywords
MBT model, model-based testing

Learning Objectives for Introduction to Model-Based Testing

1.1 Objectives and Motivations for MBT
FM-1.1.1 (K2) Describe expected benefits of MBT
FM-1.1.2 (K2) Describe misleading expectations and pitfalls of MBT

1.2 MBT Activities and Artifacts in the Fundamental Test Process
FM-1.2.1 (K2) Summarize the activities specific to MBT when deployed in a test process
FM-1.2.2 (K1) Recall the essential MBT artifacts (inputs and outputs)

1.3 Integrating MBT into the Software Development Lifecycle
FM-1.3.1 (K2) Explain how MBT integrates into software development lifecycle processes
FM-1.3.2 (K2) Explain how MBT supports requirements engineering

1.1 Objectives and Motivations for MBT

Model-Based Testing (MBT) is an advanced test approach using models for testing. It extends and supports classic test design techniques such as equivalence partitioning, boundary value analysis, decision table testing, state transition testing, and use case testing. The basic idea is to improve the quality and efficiency of the test design and test implementation activities by:

- Designing a comprehensive MBT model, typically using tools, based on project test objectives.
- Providing an MBT model as a test design specification. This model includes a high degree of formal and detailed information that is sufficient to automatically generate the test cases directly from the MBT model.

MBT and its artifacts are closely integrated with the processes of the organization as well as with the methods, technical environments, tools, and any specific lifecycle processes.

1.1.1 Main Motivations for MBT

There are two main aspects of testing activities which provide the fundamental motivation for MBT and describe how MBT supports the improvement of the quality of the testing:

- Effectiveness
  - Modeling is a process which fosters close communication with the stakeholders.
  - Improved communication helps to create a common perception and understanding of the requirements in the given domain and to detect potential misunderstandings.
  - In the case of graphical MBT models, project stakeholders (e.g., business analysts) can be involved more easily.
  - Modeling supports continuous competency improvement for testers in a given domain.

- The abstraction level of the MBT model makes it easier to identify parts of a system where tests will more likely identify problems.
- Generating and analyzing test cases is possible before any real system exists.

- Efficiency
  - Early modeling and model verification and validation help avoid defects in the early stages of the development process. MBT models can be shared with project stakeholders (e.g., business analysts, developers) to verify requirements and identify gaps within those requirements. In that way, MBT supports the "early testing" principle.
  - MBT artifacts from previous projects can be (re-)used for further development of tests.
  - MBT supports automation (e.g., to generate testware) and helps to reduce defects that can be introduced when testware is manually created and maintained.
  - Different test suites can be generated from the same model (e.g., with different test selection criteria) fostering efficient adaptation to changes in the test basis or the test objectives.
  - MBT can be used for various test objectives and to cover different test levels and test types during testing.
  - MBT helps to reduce maintenance costs when requirements change because the MBT model provides a single point of maintenance.

1.1.2 Misleading Expectations and Pitfalls of MBT

Test teams that are going to use MBT should have realistic expectations of the advantages and limitations of this test approach. Typical misleading expectations, misunderstandings and pitfalls include:

- MBT solves all problems
  MBT does not replace traditional test design techniques (see [ISTQB_FL_SYL]), but supports them in a way that allows testers to improve their understanding of the domain and to perform tests more effectively and efficiently. A test team that ignores and/or misuses those test design techniques will not solve this problem by introducing MBT.

- MBT is just a matter of tooling
  Appropriate tool support is essential for the success of MBT, but acquiring the tool should not be the first step. Instead, the decision to introduce MBT should be based on the definition of measurable objectives regarding test improvements. MBT impacts the whole test process (section 1.2). Therefore, introducing MBT requires strong support from management to lead the process and tool changes in the organization.

- Models are always correct
  As in manual test design, the tester may introduce defects when creating the MBT model. MBT models require thorough verification and validation by applying reviews, model static analysis, model simulation and so forth. Any change in the MBT model propagates to all generated testware related to the modified model element, so each change must be carefully reviewed prior to implementation.

- Test case explosion will occur
  The application of MBT improves the methods and coverage in the test design. In the case of pure combinatorial test case generation, this can lead to test case explosion. This challenge can be addressed by adapting the test case generation strategies and algorithms, and by applying intelligent filter mechanisms.

1.2 MBT Activities and Artifacts in the Fundamental Test Process

1.2.1 MBT Specific Activities

When deploying model-based testing, the test process includes activities specific to MBT that are not usually used in classic test analysis and design. These include:

- MBT modeling activities (activities related to the administration of the MBT model, development and integration of the modeling approach, definition of modeling guidelines, development of the structure of the MBT model, development of model elements, e.g., specific diagrams of the MBT model, tooling related activities,)
- Generation of testware based on the MBT approach and selection criteria, e.g., test case generation

At a minimum, model-based testing impacts test analysis and test design activities in the fundamental test process [ISTQB_FL_SYL]. Depending on the test objectives, MBT can also be used with a broader scope relating to all parts of the fundamental test process. The following MBT specific activities should be considered:

- Test planning and control can include implementation of the MBT specific activities of the fundamental test process (MBT tooling, guidelines, specific metrics, MBT artifacts as part of the baseline for project planning, etc.).
- Test analysis and design can include MBT modeling activities, test selection criteria and advanced test coverage metrics.
- Test implementation and execution can include MBT test generation and MBT test adaptation.
- Evaluation of exit criteria and reporting can include advanced coverage metrics (based on structural and/or explicit model information) and MBT-based impact analysis.
- Test closure activities can include establishing model libraries for re-use in future projects.

A focus point of the impact of MBT on the test process is process automation and artifact generation. MBT promotes a shift of test design into earlier project phases compared with classic test design. It can serve as an early requirements verification method and can promote improved communication especially by using graphical models. Model-based tests often augment traditional tests. This provides an opportunity for the test team to verify consistency between the automatically generated MBT test cases and the manually created traditional test cases.

Although the activities of the fundamental test process using MBT are similar to those performed in the traditional test process, the implementation may change significantly. As described in section 1.1, MBT

- has impact on quality, effort, communication and stakeholders involved in the test process
- shifts the test design activities to earlier stages in the test process.

To ensure the adoption of MBT, it is important to establish acceptance for the method and the related changes with the teams working with MBT.

1.2.2 Essential MBT Artifacts (Inputs and Outputs)

MBT models may be either obtained by developing dedicated MBT models or by re-using models developed for system design (see section 2.3.5). MBT models support different abstraction levels and specific test information according to the modeling concept applied. The artifacts obtained from the MBT models will reflect these different abstraction levels.

According to the information and abstraction level, MBT is integrated into the test process with different input and output artifacts.

Input artifacts:

- Test strategy
- The test basis including requirements and other test targets, test conditions, oral information, and existing design or models
- Incident and defect reports, test logs and test execution logs from previous test execution activities
- Method and process guidelines, tool documents

Output artifacts include different kinds of testware, such as:

- MBT models
- Parts of the test plan (features to be tested, test environment, ...), test schedule, test metrics
- Test scenarios, test suites, test execution schedules, test design specifications
- Test cases, test procedure specifications, test data, test scripts, test adaptation layer (specifications and code)
- Bidirectional traceability matrix between generated tests and the test basis, especially requirements, and defect reports

1.3 Integrating MBT into the Software Development Lifecycles

1.3.1 MBT in Sequential and Iterative Software Development Lifecycles

Two main categories of software development lifecycle models are considered in this section: sequential, such as the V-model, and iterative-incremental, such as Agile development. In both cases, the MBT approach should primarily be adapted to the project test objectives so that the benefits of MBT can be obtained corresponding to the test objectives.

Organizations implement software development lifecycles in many different ways. The test process using MBT should be adapted accordingly. For example, MBT may focus on acceptance testing in one project, and on automated system testing in another, depending on the project test objectives.

The following are common to both sequential and iterative software development lifecycles:

- Test Levels:
  - MBT is mostly used in higher test levels (integration, system and acceptance), due to its capability to abstract the complexity of functional requirements and expected behavior.
  - MBT is less commonly used in component (unit) testing, but some approaches are based on code annotation techniques.
- Test Types:
  - MBT is primarily used for functional testing with MBT models representing the expected behavior of the system and/or of its environment.
  - Enriched or dedicated MBT models may be used for non-functional testing (security, load/stress, reliability).

Testers normally own the MBT models but may choose to share them. For example, the models (or portions thereof) may be shared with non-technical stakeholders for validation and to verify requirements. Developers may be interested in the models for test automation purposes, particularly when automated scripts generated from the MBT model will be used for continuous integration to provide early and frequent feedback.

Specific activities for MBT in a sequential software development lifecycle include:

- Integrating model-based testing as part of the test strategy for the project, with respect to the risk assessment, project context and testing objectives, including requirements traceability to MBT model elements
- Starting MBT modeling as soon as possible in the project:
  - To stimulate communication between stakeholders
  - To enable early detection of unclear, incomplete or inconsistent requirements
- Adapting test planning activities and roles to include:
  - New testware elements (e.g., MBT models, test selection criteria)
  - Progress reporting on MBT activities

The major adaptations of MBT in an Agile lifecycle [ISTQB_AT_FL_SYL] are the following:

- MBT models are developed in an iterative and incremental way and tests are generated according to the content of the iteration.
- User stories are part of the test basis. Each covered user story is linked within the model to automatically manage bi-directional traceability between user stories and tests.
- MBT can be used to implement acceptance test-driven development (ATDD).
- With respect to the whole-team Agile good practice, testers using model-based testing are part of the Agile team together with developers and business representatives – see [ISTQB_AT_FL_SYL] section 1.1.

1.3.2 Supporting Requirements Engineering

MBT helps requirements engineering in the following ways:

- It facilitates the communication between business, development, and testing around the expected behavior of the software to be developed by providing graphical MBT models such as business process models and/or state diagrams. These models help all stakeholders discuss and validate the details of the expected behavior.
- It clarifies and improves the quality of requirements and/or user stories by sharing full or partial MBT models with business representatives and/or developers to produce a common understanding of expected software behavior. Tests generated from MBT models comprise possible user scenarios that can be reviewed by business representatives and/or developers.
- It supports early validation of the requirements even in stages where they are still subject to change.

Requirements engineering and MBT work together. The requirements and related risks are inputs for MBT modeling and test generation activities, and the MBT activities can automate the creation and maintenance of bi-directional traceability links between requirements and tests.

2 MBT Modeling – 250 mins.

Keywords
test model

Learning Objectives for MBT modeling

2.1 MBT Modeling

FM-2.1.1 (K3) Develop a simple MBT model for a test object and predefined test objectives using a workflow-based modeling language (refer to section 9.1 – "simple" means less than 15 modeling elements)
FM-2.1.2 (K3) Develop a simple MBT model for a test object and predefined test objectives using a state transition-based modeling language (refer to section 9.2 – "simple" means less than 15 modeling elements)
FM-2.1.3 (K2) Classify an MBT model with respect to the subject and to the focus
FM-2.1.4 (K2) Give examples of how an MBT model depends on the test objectives

2.2 Languages for MBT Models

FM-2.2.1 (K1) Recall examples of modeling language categories commonly used for MBT
FM-2.2.2 (K1) Recall typical representatives of modeling language categories relevant for different systems and project objectives

2.3 Good Practices for MBT Modeling Activities

FM-2.3.1 (K1) Recall quality characteristics for MBT models
FM-2.3.2 (K2) Describe classic mistakes and pitfalls during modeling activities for MBT
FM-2.3.3 (K2) Explain the advantages of linking requirements and process related information to the MBT model
FM-2.3.4 (K2) Explain the necessity of guidelines for MBT modeling
FM-2.3.5 (K2) Provide examples where reuse of existing models (from requirements phase or development phase) is or is not appropriate
FM-2.3.6 (K1) Recall tool types supporting specific MBT modeling activities
FM-2.3.7 (K2) Summarize iterative MBT model development, review and validation

2.1 MBT Modeling

In the context of testing, MBT models are excellent for expressing what should be tested, to communicate among stakeholders, and to summarize all relevant information for the test design. The models also have some applicability for test management activities.

An MBT model is developed with the goal of deriving (or identifying) test cases. The MBT model should include the necessary information for later test case generation, e.g., to enable the selection of reasonable test cases according to project test objectives, to allow the generation of the test oracle, and to support the bi-directional traceability between requirements and generated test cases.

Designing the MBT model is an essential and demanding activity in MBT. The quality of the MBT model has a strong impact on the quality of the outcome of the MBT-based test process. MBT models are often interpreted by tools and must follow a strict syntax. MBT models may also need to support other quality characteristics such as those defined in modeling guidelines.

2.1.1 MBT Modeling Activities

Every model is expressed in a specific modeling language. The modeling language defines the artifacts that comprise the model elements and the rules one should follow to build models in that language. There are modeling languages to express structure, behavior, or domain-specific artifacts among others.

The following important MBT modeling questions should be asked:

- What quality characteristics of the test object will be modeled?
  Most commonly the focus is on functionality, but performance factors and other non-functional aspects such as security can also be part of the MBT model. To keep models manageable and the MBT-based test process efficient, it is recommended to model only the aspects of the system or its environment that are relevant for testing and for the test objectives.

- Which modeling languages are suitable?
  Every model is expressed in a specific modeling language. The modeling language defines the artifacts that comprise the model and the rules one should follow to build models in that language. There is a large variety of modeling languages to represent the structure, the behavior or other aspects (e.g., data, workflows, communications protocols) of the test object or its environment.

- What is the appropriate level of abstraction?
  Abstraction is useful to master complexity and to focus the MBT model on the test objectives. However, the more abstract the model is, the more demanding is the test adaptation for test execution. Furthermore, the abstraction level often determines the audience/stakeholders who are able to discuss the test design at the given abstraction level.

The answers to these questions and the chosen MBT toolset will influence the MBT activities.

Note: Two simplified graphical modeling languages are provided in Appendix A. Training and exam preparation should cover the use of both languages.

2.1.2 Subject and Focus of MBT Models

Three types of model subjects are considered in this syllabus:

- System model
  A system model describes the system as it is intended to be. Test cases generated from this model check if the system conforms to this model. Examples of system models are class diagrams for object-oriented systems or state diagrams describing the states and state transitions of reactive systems.

- Environment model / Usage model
  An environment model describes the environment of the system. Examples of environment models include Markov chain models describing the expected usage of the system.

- Test model

A test model is a model of (one or several) test cases. This typically includes the expected behavior of the test object and its evaluation. Examples of test models are (abstract) test case descriptions or a graphical representation of a test procedure.

An MBT model generally combines several or all of these subjects. For example, it may represent the test object including data elements and also the usage of the test object in a given context.

The focus of an MBT model can be structural, behavioral, or a combination of both:

- Structural models describe static structure. Examples of structural models are class diagrams and interface specifications or classification trees for data modeling.
- Behavioral models describe dynamic interactions. Examples of behavioral models are activity diagrams or business process models describing activities and workflows, and state diagrams describing inputs and outputs of a system.

An MBT model generally combines structural aspects (e.g., description of the interfaces of the test object) and behavioral aspects (e.g., the expected behavior of the test object).

2.1.3 MBT Models Depend on Test Objectives

When developing an MBT model it is important to consider the test objective since this determines the subject and focus of the model. Examples of test objectives together with suitable model characteristics for that objective are shown in the table below.

Test Objective	Example Model	Subject	Focus
Verify that the business workflow is implemented correctly	A business process model describing the workflow	System	Behavior
Verify that the system provides correct responses to requests when in a specific state	A UML (Unified Modeling Language) state machine	System	Behavior
Verify the availability of an interface	A description of the structure of the interface	System	Structure
Gain confidence that the test object will be suitable for the expected usage by users	A usage model describing the behavior of the user	Environment	Behavior
Verify the configurations of a system	A data model using a classification tree	Data	Structure

2.2 Languages for MBT Models

2.2.1 Main Categories of Modeling Languages for MBT

Models are denoted by the use of modeling languages. Modeling languages are defined in the following ways:

- By their concepts (also known as abstract syntax and often textually described, but sometimes specified in meta models)
- By their syntax (also known as concrete syntax and often defined by grammar rules)
- By their semantics (often defined by static and dynamic semantic rules)

Diagrams are representations of models in a graphical modeling language, such as class diagrams, sequence diagrams or state machine diagrams in UML.

There are many modeling languages that can be used for MBT. In order to select a good "fit for purpose" language, the tester needs to know the main characteristics that differentiate modeling languages for MBT:

- Modeling concepts
  Modeling languages differ in the set of concepts they support. Depending on the test objective, the models can represent structural aspects (e.g., architectures, components, interfaces of the software), data aspects (e.g., formats and semantics of passive objects of the software) or behavioral aspects (e.g., scenarios, interactions, execution of the active objects of the software). See also section 2.1.2.

- Formalism
  The degree of formalism of a modeling language ranges from limited formalization to full formalization. While the latter is the most rigid and enables full-fledged analysis of the models, the former is often more practical. At a minimum, the language should have a formal syntax (e.g., including structured text or structured tables). It may also have a formally defined semantics.

- Presentation formats
  Modeling languages can use different presentation formats ranging from textual to graphical formats as well as combinations of these. Graphical formats are often more user-friendly and easier to read, while textual formats are often more tool-friendly and efficient to both write and maintain.

Categories of modeling languages include:

- Languages for structural models
  Such languages support the specification of structural elements of software such as interfaces, components, and hierarchies. An example is the UML component diagram.

- Languages for data models
  Such languages support the specification of the data types and values. Examples include the UML class diagrams and value specifications.

- Languages for behavioral models
  Such languages support the specification of events, actions, reactions and/or interactions of software. Examples include UML activity or interaction diagrams, state machines or Business Process Modeling Notation (BPMN).

- Integrated languages
  Typically, a modeling language is not limited to one of the aspects, but rather provides concepts for a number of aspects. An example is UML itself, in which the different diagrams can be used in combination to denote the different aspects of software.

2.2.2 Language Categories Relevant for Different Systems and Project Objectives

The selection of the modeling language for MBT is closely related to the original objectives of the project for which software is developed and needs to be tested. It also relates to the properties of the system for which the software is built. This results in a set of modeling language criteria. It may be necessary to

evaluate these criteria, accepting some, and eliminating others, based on relevance to system-specific attributes.

Examples include the use of:

- State diagrams to represent the expected behavior of a control/command test object
- Activity diagrams or BPMN models to represent the workflow for the end-to-end testing of an information system
- Decision tables and cause-effect graphs to represent business rules for system testing
- Timed models to represent responsiveness of a test object for timed test cases
- Feature models to represent the variants in the context of the software product line

Further examples for the relation of modeling languages and project objectives include:

- Non-functional requirements regarding certification of safety-critical or security-critical software where the software certification requires the linkage of software requirements to code and test cases
- Non-functional requirements regarding documentation of processes for auditing or certification purposes where the models are equipped with annotations for documentation

2.3 Good Practices for MBT Modeling Activities

2.3.1 Quality Characteristics for MBT Models

Model quality directly affects the generated output. Quality characteristics for models are:

- Syntactic quality (correctness)
  The MBT model is consistent with the rules regarding its formal description (modeling language, modeling guidelines) so that test case / test data generation can produce artifacts without running into problems due to incorrect syntax.

- Semantic quality (validity)
  The model content is correct with respect to what it shall describe; derived artifacts are “usable” (test scripts can be executed, manual test cases can be executed by the tester without producing failures due to incorrect test cases).

- Pragmatic quality (suitability)
  The MBT model is suitable for the given test objective and for the given test generator so that derived artifacts fulfill expectations.

Tools may check the syntax and, at least partly, the semantics of a model. Reviews check semantic and pragmatic quality. Model simulators complete those static techniques by providing dynamic checks of model quality.

It is recommended to incrementally develop the MBT model and to generate (and execute) the derived tests repeatedly, thus checking the model, the derived artifacts and the test object early and on a regular basis.

2.3.2 Typical Mistakes and Pitfalls in MBT Model Design

Newcomers to MBT tend to commit a number of common mistakes in model design including:

- Putting too many or too few details in the MBT model (wrong abstraction level for given test objectives)
- Trying to write one model that covers everything

MBT models should focus on those aspects that are relevant to meet the test objectives. It is better to write two models that emphasize particular aspects (for example, a state diagram to verify the system implementation and an activity diagram to validate the workflow) instead of one model covering all aspects.

2.3.3 Linking Requirements and Process Related Information to the MBT Model

Establishing traceability between requirements, MBT models, and generated tests is a good practice in model-based testing. By linking model elements with requirements, the following benefits can be achieved:

- The review of the MBT model is facilitated.
- Tests generated from the MBT model can be automatically linked to requirements.
- It becomes possible to generate tests based on requirements selection, and to decide which tests to execute first based on the priority of the selected requirements.
- This give the opportunity to measure requirements coverage by MBT generated tests.
- It enables all stakeholders (e.g., testers and test managers) to analyze the impact of requirements changes and to determine the required scope of regression testing.
- Test case generators may automatically generate traceability documentation (traceability matrix).

There are several implementation choices for linking requirements to model elements. For instance, a specific graphical symbol or textual keyword can represent the requirement in the model.

The tester may add other additional information either as new model elements with specific meaning or as attributes to existing model elements. Examples for additional information in MBT models include:

- Detailed content for test procedure specification (sequence of actions for test execution)
- Test script segments (function calls or keywords)
- Risks / hazards (related to functional and non-functional quality attributes or to project management)
- Priorities (of functionality or tests)
- Duration (estimated execution time)
- Required test equipment or rules for testing product configurations

The additional information helps to integrate model-based testing into the whole test process and supports test management in several ways:

- Risk or priority information can be linked to the MBT model, and linked to generated test cases, which may be used (e.g., to prioritize test execution).
- Any other project constraints and objectives can be reflected into the MBT model and thus can help to adapt test planning.
- When the interfaces (e.g., graphical user interface) are not yet stabilized in the specification, the MBT model may encompass the functional requirements that are defined. If so, the functional keyword-driven scripts may be generated earlier, and the automation phases can start earlier. Once the interfaces are specified, automated test scripts are ready and only the test adaptation layer remains to be implemented.

2.3.4 Modeling Guidelines for MBT

Modeling guidelines are documented instructions for how to design, write, and read MBT models. These guidelines:

- Support understanding and review of MBT models by all stakeholders involved
- Support conformance to syntactical requirements originating from the process, the domain, the organization or the specific tool
- Limit / extend the scope of the selected notation (e.g., defining a subset of UML or different meanings of elements)
- Foster a similar syntax and semantics of MBT models from various authors
- Teach good practices
- Support maintainability and reusability of MBT models

MBT projects without modeling guidelines take additional risks compared to MBT projects with modeling guidelines, as testers can make the classic mistakes of modeling more easily.

2.3.5 Reuse of Existing System Design or Requirements Models

As well as the development of MBT models from “classical” test basis documents, it is possible to reuse existing system design or requirements models (e.g., business process models or activity diagrams) directly as part of an MBT model, but this approach has its limits. The degree of reusability depends on the given model and the test objectives. For example, a state machine diagram from system design may serve to test a command and control system, but class diagrams are inadequate for workflow-based system testing.

Reusing existing system design models creates a single-source problem. Any error in the original system design model propagates to the MBT model. Testing is not as independent from system development and implementation as it needs to be. Therefore, it is recommended to write independent models by different authors: one for system design and one for MBT purposes. This fulfills separation of responsibilities and promotes independence.

If reused without any modifications, the tests derived from system design models are verification tests only (as opposed to validation). They check whether the system implementation corresponds to the specification described by the system design model.

For validation purposes, the tester may start from an existing model (e.g., from requirements elicitation) and enrich it with testing aspects. Once transformed into an MBT model, the initial system design model tends to grow. The tester will add model elements (for example, unforeseen transitions) to test unusual scenarios or error situations. At the end, the resulting MBT model reflects the tester’s mindset rather than the initial system design.

Consider the case in which the author of the existing model was not a tester. As a result, models from previous development phases do not necessarily respect MBT-related requirements and guidelines. A reused model does not necessarily follow the (tool-dependent) recommended practices and may lead to test case explosion.

2.3.6 Tool Support for Modeling Activities

Model editors are used to write models. This can be a specific modeling tool or any other flowchart editor (for graphical models), or a text / scripting editor (for textual models). In principle, it is possible to write MBT models with pencil and paper. While simple drawing tools may be sufficient for documentation purposes, they do not support further processing of the MBT model.

An MBT model editor may provide predefined model elements and possibly some syntactic and semantic checks. Tools that execute different paths through the model, thus validating the MBT model, are called model simulators.

To obtain test cases and/or test data from the MBT model automatically, a test case (and/or test data) generator is required.

2.3.7 Iterative Model Development, Review and Validation

The modeling of more complex behavior across multiple diagrams may easily lead to situations where a review of the MBT model alone may no longer suffice. A tester needs to be sure that tests generated from the MBT model will fulfill the expectations. Iterative development, model reviews, as well as frequent reviews of generated test artifacts, first by the tester but also with peers and other stakeholders involved in the software development lifecycle, are ways to manage this situation.

Applying these best practices enables:

- Testers to validate their point of view on the aspects to be tested with other stakeholders
- Testers to catch and fix errors as well as detect parts missing from the model early in MBT model development (before test execution)
- Testers to identify and communicate incomplete or inconsistent requirements
- Test managers to manage risks associated with the project
- The team to reduce the overall time needed to complete the MBT modeling activity in the test process

Iterative MBT model development is also related to iterative test generation. This means that each time some aspects are added to the MBT model, test generation is also updated. In essence test generation constitutes a simulation of the MBT model, which provides feedback on behavioral aspects to be tested.

3 Selection Criteria for Test Case Generation – 205 mins.

Keywords
coverage item, model coverage, test case explosion, test selection criteria

Learning Objectives for MBT Test Selection Criteria

3.1 Classification of MBT Test Selection Criteria
FM-3.1.1 (K2) Classify the various families of test selection criteria used for test generation from models
FM-3.1.2 (K3) Generate test cases from an MBT model to achieve given test objectives in a given context
FM-3.1.3 (K2) Provide examples of model coverage, data-related, pattern- and scenario-based and project-based test selection criteria
FM-3.1.4 (K2) Recognize how MBT test selection criteria relate to ISTQB Foundation Level test design techniques

3.2 Applying Test Selection Criteria
FM-3.2.1 (K1) Recall degrees of test artifact generation automation
FM-3.2.2 (K3) Apply given test selection criteria to a given MBT model
FM-3.2.3 (K2) Describe good practices of MBT test selection criteria

3.1 Classification of MBT Test Selection Criteria

From the same MBT model, various test suites may be generated. Using test selection criteria helps the tester to select a meaningful subset of test cases that best fits the targeted test objectives.

Test case generators play an essential role in achieving efficiency improvements with MBT.

Test selection criteria for MBT have been widely studied in the literature (see [Utt07], [Wei09], [Zan11]). This syllabus presents six families of test selection criteria. Some of them focus on coverage of specific items; others support project management aspects, testing of specific scenarios, or random aspects.

3.1.1 Test Selection Criteria

Coverage-based test selection criteria relate test generation with coverage items from the MBT model. The coverage items may be:

- Requirements linked to the MBT model
  This criterion requires that MBT model elements be linked to selected requirements. Full requirements coverage corresponds to a set of test cases that completely covers a selected set of requirements (each requirement is covered by at least one test case).

- MBT model elements

Model coverage is based on the internal structure of the MBT model. The tester, test manager or any other role involved defines coverage items and selects the set of test cases that covers a desired quantity of those items. Possible coverage items may be model elements such as:

- States, transitions and decisions in state diagrams (see “state transition testing” in [ISTQB_FL_SYL])
- Activities and gateways in business process models
- Conditions and actions in decision tables
- Statements and conditions in textual models

- Data-related test selection criteria
  These criteria relate to test design techniques (see [ISQB_FL_SYL]) such as:
  - Equivalence partitioning
  - Boundary value analysis

  They also include heuristics such as pairwise or, more generally, n-wise combinatorial test case generation (for more information, see [ISTQB_ATA_SYL]).

Other test selection criteria include:

- Random
  This selection criterion is comparable to a (more or less random) walkthrough of the model. In random test case selection, the different alternatives are all equally probable. Stochastic test case selection takes into account that different alternatives may have different probabilities.

- Scenario-based and Pattern-based
  The test case selection is based on predefined scenarios or patterns. A scenario may be an explicitly defined use case (see also “use case testing” in [ISTQB_FL_SYL]) or an explicitly defined user story (see [ISTQB_AT_FL_SYL]). A pattern is a partially defined scenario, which may be applied to the MBT model to produce several or multiple tests.

- Project-driven
  Project-driven test case selection is based on additional project-related information, which has been added to the model either to support test management and/or to achieve specific test objectives for the project. Project-related information includes risks, priorities, required test equipment or any other aspect that matters in this specific project. Selecting all test cases that require specific test equipment corresponds to applying a project-driven selection criterion.

MBT tools support at least one, but generally not all of the test selection criteria mentioned above.

3.1.2 Test Case Selection in Practice

In practice, the tester should consider combining several test selection criteria to obtain a subset of test cases that best fits the targeted test objectives. Examples for those combinations are requirements coverage and model coverage or scenario/pattern and data-related test selection criteria.

Apart from the targeted objectives, the application of the test selection criteria also depends on the following:

- The mechanisms provided by the MBT tool
- The design of the model
- The tester’s experience with the criteria

It is possible that the MBT model is formally correct, but produces test case explosion due to tool-specific generation algorithms.

3.1.3 Examples of Test Selection Criteria

Some examples of test selection criteria in relation with some modeling languages are (see [Utt07] – chapter 4):

- On activity diagrams or business process models:
  - Activity coverage (100% = coverage of each activity in the diagram)
  - Decision / gateway coverage (100% = coverage of each decision point in the diagram)
  - Path coverage (100% = coverage of all business scenarios in the diagram), with or without loops
- On state diagrams:
  - State and transition coverage (100% = coverage of each state or transition respectively)
  - Transition pair coverage (100% = coverage of each consecutive pair of transitions in the diagram)
  - Path coverage (100% = coverage of all paths, typically without loops, in the diagram)
- On decision tables:
  - Every condition
  - Every action
  - Every rule
- On data domains defined in the structural part of the model:
  - Equivalence partition coverage (e.g., with one representative per class)
  - Boundary value coverage (e.g., considering boundaries on ranges of numerical data)
  - Pairwise testing on defined domains (ensuring that all possible discrete combinations of each pair of input parameters are produced)
- On a textual model:
  - Statement coverage (100% = each executable statement is covered)
  - Decision coverage (100% = each decision is covered)
  - Decision condition coverage (100% = all condition outcomes and all decision outcomes have been exercised by the test suite).
  - Multiple condition coverage (all combinations of conditions in each decision are covered – leads to a high number of generated test cases).

3.1.4 Relation to Foundation Level Test Design Techniques

MBT supports standard ISTQB Foundation Level test design techniques such as state transition testing, equivalence partitioning, boundary value analysis, decision table testing and use case testing. The MBT model itself may contain coverage items of those test design techniques (e.g., equivalence partitions or boundary values described in the MBT model). Alternatively, the MBT model may be combined with other graphical or textual representations containing those items (e.g., decision tables or additional diagrams).

3.2 Applying Test Selection Criteria

3.2.1 Degree of Automation in Test Generation

MBT is usually associated with automated test case generation, but the test case generation process is not necessarily tool-based. The following methods can be used for test case generation:

- Manual test generation – Deriving test cases manually from the MBT model, following the paths and writing the corresponding test cases “manually”. However, this MBT approach has a low maturity level and does not provide the benefits regarding efficiency and effectiveness of automated test case generation mentioned in section 1.1.
- Automated test generation – Output artifacts such as test cases are generated automatically by an MBT tool and may be used as is without further post-processing.
- Semi-automated test generation – Intermediate solutions also exist where a tool is used, but manual steps are required in-between or at the end, for example to select specific test cases.

3.2.2 Pros and Cons of Specific Test Selection Criteria

Each test selection criterion has advantages and disadvantages. The choice depends on the testing objectives. These are examples of pros (pluses) and cons (minuses) for various test selection criteria:

- Requirements coverage
  (+) Often mandatory due to regulatory requirements
  (-) Precise definition is needed (for example, if a requirement is linked with several behaviors in the model, requirements coverage is achieved when covering all behaviors or just one)
- Model coverage
  (+) Helpful to understand where exactly a model could not be covered and to get a sense for completeness of test coverage in terms of the model
  (-) Some structural test selection criteria may produce test case explosion during test generation (for example: all path coverage in an activity diagram)
- Data-related test selection criteria
  (+) Often mandatory to cover equivalence partitions of the domains
  (-) May be heavily combinatorial
- Random
  (+) Helpful to select unexpected test cases
  (-) Random algorithms may generate long tests that have no business meaning
- Scenario-based / Pattern-based
  (+) Supports selection of use cases (e.g., for regression testing)
  (-) Requires extra effort to define and maintain scenarios and patterns
- Project-driven
  (+) Useful for test management
  (-) Requires extra effort to link specific information to the model

3.2.3 Good Practices of MBT Test Selection

Quite often, one criterion alone is not sufficient to cover all testing aspects required to reach the targeted test objective. In that case, the tester has to combine several criteria.

There are two approaches for combining criteria:

- Composition of criteria – The generated test suite contains only those test cases that fulfill all criteria that have been applied (intersection)
- Addition of criteria – The generated test suite contains all test cases that fulfill at least one criterion (union)

The selection criteria may influence the MBT modeling concept. In addition, it must be possible to reproduce the selection, that is, to obtain the same set of test cases from the same model when applying the same criteria. Therefore, it is essential to plan test selection activities and to document both the choices and the underlying reasons.

For some model coverage criteria, several sets of paths exist that fulfill the required coverage. Small changes in the MBT model may lead to a completely different selection, a fact that requires a new way of thinking. The model is the master; the generated test cases are derived artifacts.

Test selection criteria are also a way in MBT to master test case explosion; by using a single criterion or a combination of criteria, the tester may precisely tune test generation in order to satisfy test objectives and avoid test case explosion.

4 MBT Test Implementation and Execution – 120 mins.

Keywords
offline MBT, online MBT, test adaptation layer

Learning Objectives for MBT Test Implementation and Execution

4.1 Specifics of MBT Test Implementation and Execution

FM-4.1.1 (K2) Explain the difference between abstract and concrete test cases in the MBT context
FM-4.1.2 (K2) Explain the different kinds of test execution in the MBT context
FM-4.1.3 (K3) Perform updates of an MBT model and test generation caused by changes in requirements, test objects or test objectives

4.2 Activities of Test Adaptation in MBT

FM-4.2.1 (K2) Explain which kind of test adaptation may be necessary for test execution in MBT

4.1 Specifics of MBT Test Implementation and Execution

Once the test suite has been generated, the next step is to run the test cases. Test execution may be manual or may be automated by using a test execution environment that provides facilities to automatically execute the tests and record test results. In both cases, there must be a connection between the MBT model and the test execution artifacts.

4.1.1 Abstract and Concrete Test Cases in the MBT Context

MBT test generation may produce abstract test cases (high-level test cases) or concrete test cases (low-level test cases):

- Abstract test cases are test cases without concrete (implementation level) values for input data and expected results. They also do not specify test steps in fine detail.
- Concrete test cases are test cases with concrete (implementation level) values for input data and expected results, and with very detailed descriptions of test steps.

MBT models can contain different levels of abstraction addressing different output artifacts to be generated and also addressing different stakeholders. For example:

- Abstract test cases may be targeted for review by business analysts
- Concrete test cases may be directly executed by testers

MBT generated abstract test cases provide information regarding test conditions, input data and expected results without concrete values. The test information is expressed at an abstract level, for instance:

- It may define the test procedures with sequences of high level test actions instead of the detailed and completely defined test actions.
- It may provide equivalence partitions instead of concrete representatives of the partitions.

Moving from abstract test cases to concrete test cases ready for execution (either manually or automatically) requires complementing the abstract test cases with the following:

- Completely defined test actions
- Concrete and complete input data values
- Concrete values for expected results

This complementing information may be defined into the MBT model (for example, as documentation of the model artifacts such as actions/verification and data) or outside the MBT model (for example, using a data table to map abstract and concrete data values).

4.1.2 Different Kinds of Test Execution

MBT generated test cases may be executed manually or automatically.

For manual test execution, testers execute test cases that have been generated by the MBT tool. These test cases must be generated in a format usable for manual test execution. It may also be beneficial to be able to export the cases into a test management tool. Test execution and defect management are then performed in accordance with the ISTQB fundamental test process.

For automated test execution, the test cases must be generated in a form that is executable. There are three main approaches that are used to manage automated execution from abstract test cases (see [Utt07] – chapter 8):

- The adaptation approach – With this approach, test adaptation layer code is written to bridge the abstraction gap. This code is essentially a wrapper around the test object (similar to the keyword-driven testing approach)
- The transformation approach – With this approach, the MBT generated test cases are directly converted into automated test scripts (no test adaptation layer is required)
- The mixed approach – This is a combination of the two previously mentioned approaches in which the abstract test cases are transformed into concrete test cases, which are wrapped with adaptation layer code around the test object.

Once generated, the automated test scripts are executed by a test execution tool. MBT tools may convert generated test cases into the test automation scripting language of the test execution tool. The MBT tool may also publish the generated automated test scripts into a test management tool.

There are two methods for automated test execution in the MBT context:

- Offline MBT – The automated test scripts are generated first (including expected results) and executed afterwards.
- Online MBT (also called “on-the-fly”) – Test generation and execution are realized simultaneously. Therefore, each test step is generated after executing the previous test step during the testing. The execution result may influence the path taken through the model.

4.1.3 The Impact of Changes on the MBT Artifacts

Changes are inevitable in a software project. The following changes can occur and should be expected:

- Change in requirements, in the test object or its environment that may impact:
  - The MBT model
    - Action
    - Conditions/expected result
    - Data
  - The MBT test selection criteria
  - The adaptation layer specification if it exists (and code if it exists)

- Change in the interface of the test object, but not in the functional requirements (e.g., a small change in the graphical user interface without impact on the functional behavior), that impacts:
  - The adaptation layer specification only (and code if it exists)
- Change in the test objective or test conditions that may impact:
  - The MBT model
  - The MBT test selection criteria
  - The adaptation layer specification if it exists (and code if it exists)

Change management on MBT artifacts should be based on a process that includes impact analysis, exploration of change options, application of changes on artifacts, and review activities.

4.2 Activities of Test Adaptation in MBT

In the case of manual test execution, test adaptation relates to the documentation of generated tests to fill the gap between the abstractions made in the MBT model and the concrete interfaces and test data of the system under test. For example, concrete data values may be provided as representatives of the specific boundary values. This adaptation process provides manual test scripts that are complete and sufficiently documented so that they can be directly used for manual testing.

In the case of automated test execution, test adaptation is the process of integrating the artifacts generated from an MBT model into the test execution framework on the basis of the adaptation layer specification. This process supports the practices of keyword-driven testing and/or data-driven testing.

Regarding keyword-driven testing, the keywords are defined in the MBT model and used in the generated test cases. In order to get fully automated test scripts, the following steps and activities are required:

1) Export the test cases as scripts in the language of the test execution tool. This can be done manually or automatically with an exporter (may be provided by the MBT tool)
2) Implement the keywords utilizing the adaptation layer specification in the language of the test execution tool. Either the tester in charge of test automation or the developer of the test object may do this.

Regarding data-driven testing, the MBT model describes abstract input data and expected results (e.g., based on equivalence partitioning). The generated test cases or scripts refer to these abstract input data and expected results.

In order to get fully automated test scripts, the following steps and activities are required:

1) Provide the concrete input data and expected results required in the adaptation layer specification. This data may be stored in a table or in a spreadsheet.
2) Link the test data formalized in the MBT model to the concrete test data in the test execution tool or in the test harness.

Each test script assumes specific initial preconditions. In order to be able to chain the execution of the automated test scripts, it is necessary to ensure that the preconditions are set correctly before each test script execution. This can be done in the following ways:

- By providing post-conditions inside the test scripts (which is not always possible) to be used as the preconditions for the next one
- By setting up the precondition at the beginning of each test script

Test adaptation should be prepared during MBT modeling activities, for example by developing the test adaptation specification in the same time than model elements are developed.

5 Evaluating and Deploying an MBT Approach – 60 mins.

Keywords
none

Learning Objectives for Evaluating and Deploying an MBT Approach

5.1 Evaluate an MBT Deployment

FM-5.1.1 (K2) Describe ROI factors for MBT introduction
FM-5.1.2 (K2) Explain how the objectives of the project are related to the characteristics of the MBT approach
FM-5.1.3 (K1) Recall selected metrics and key performance indicators to measure the progress and results of MBT activities

5.2 Manage and Monitor the Deployment of an MBT Approach

FM-5.2.1 (K1) Recall good practices for test management, change management and collaborative work when deploying MBT
FM-5.2.2 (K1) Recall cost factors of MBT
FM-5.2.3 (K2) Give examples of the integration of the MBT tool with configuration management, requirements management, test management and test automation tools

5.1 Evaluate an MBT Deployment

The deployment of an MBT approach in an organization often follows a classic process of product adoption:

1. Awareness - Establishing improvement objectives in the test process and identifying MBT as a possible technology to address all or some of these objectives. This identification of possible improvements is an important factor for the motivation to change the way of working in the test teams.
2. Interest - Learning more about MBT.
3. Evaluation - Analyzing main MBT principles and existing MBT approaches with respect to their applicability in the given project context.
4. Trial use – Defining key performance indicators (KPIs) and setting up a pilot project to measure improvements.
5. Adoption – Leading and reinforcing change with skill improvements and behavioral change in the organization.

Therefore, the estimation of ROI (Return on Investment) and the capability to evaluate the impact of MBT on testing project performance (on the basis of KPIs) is part of the adoption cycle.

5.1.1 ROI Factors for MBT Introduction

MBT generates costs both during the introduction and the operating phase. For a successful MBT introduction, costs have to be counterbalanced by benefits regarding:

- Net financial balance taking into account the complete test process on a long term scale

- Test design quality and positive impact on the complete development process

Cost, savings and improvements in the test process and the test quality have to be taken into account.

Costs (see section 5.2.2):
- MBT introduction
- Cost running MBT in the test process

Savings:

- Early requirements validation, thus avoiding costly bug fixes in later development phases: The top-down approach featured by MBT allows the MBT design in project stages, when the requirements are not yet fixed or only defined roughly – thus also enabling requirements validation on the current requirements' maturity.

- Reduced effort in test design by re-use of test design artifacts and by avoiding redundancies: In an MBT model, all possible paths are integrated, redundancy free, in one artifact compared to the redundancies in suites of test sequences implemented as manual test or automated test scripts.

- Reduced effort in test implementation by automated test artifact generation from the MBT models: Once the MBT model is developed, test cases are generated automatically, reducing the effort for maintenance activities.

- Early defect or fault identification and improved time to market:
  This is fostered by the capability of MBT to start early in test design and the advanced efficiency supplied by MBT.

- Test management support:
  By using the different MBT test generation strategies and filtering mechanisms, the test manager can select the "ideal" set of test cases that best fit the test goal.

- Potential reduction in the number of test cases defined using MBT (compared to test case design techniques without MBT) due optimized algorithms, thereby reducing the number of test cases that have to be executed. Test cases can be selected and reduced in number systematically based on uniform and systematic generation strategies.

Benefits for test quality:
- Improved test design methods and consistency of the test design
- Improved knowledge about test coverage and thus about the quality of the test design
- Test management support using MBT outputs including prioritization and quality assurance of the test design
- Improvements in traceability by improved content and organization of the test design

5.1.2 Organizational Objectives and their Relationship to the Characteristics of the MBT Approach

MBT can improve the test process regarding quality of the test, effort reduction and communication. The organization has to prioritize required improvements. The scope of improvements and their qualitative and quantitative impact on the organization highly depends on the characteristics of the MBT method. Thus

the MBT approach should be defined based on the scope of prioritized improvements. The following table shows some examples of organizational objectives and how MBT could be used to meet them.

Organizational Objectives	MBT Improvement Focus	Achieved By
Improved quality of testing	• Methods of the test design for higher test coverage • Traceability (enabling coverage metrics and better capability for impact analysis) • Process automation to avoid human errors	• Separate models for development and MBT activities (enabling the tester's mindset and encouraging independence) • Well-defined abstraction level based on the test plan • High degree of process automation including the generation of test artifacts and the execution of tests to reduce human errors • Well-defined test selection criteria adaptable for specific test objectives
Effort reduction	• Process automation • Traceability (supporting process automation)	• Shared MBT models (re-use of MBT models) • High degree of process automation and test artifacts generated automatically • Well-defined test selection criteria to generate the most efficient set of test cases to be executed
Improved communication	• Adequate, understandable and usable MBT approach reflecting the abstraction level of the thinking of stakeholders involved	• Suitable abstraction level for all stakeholders involved (e.g., high level and business-oriented for business analysts or detailed and test-oriented for testers)

A combination of organizational objectives can lead to contradicting requirements for the MBT approach. This can be resolved by developing different MBT models for different test objectives (see section 2.1.3).

5.1.3 Metrics and Key Performance Indicators

Introducing MBT in an organization should be based on clear objectives, metrics, and KPIs that can be used to measure the progress and results of the MBT activities.

Possible metrics and KPIs to be monitored include:

- The number of requirements managed and traced into the MBT model, and requirements coverage (percentage) by generated test cases
- The size and complexity of the MBT model
- The number of generated test cases / scripts, and the number of generated test cases / scripts per person-day
- The number of defects found in the requirements during MBT modeling activities
- The reusability level of MBT model elements from one project to another
- The level of usage of MBT models by the project stakeholders (business / development / testing)
- The percentage of efficiency gain with respect to previous approaches for test design in terms of productivity (cheaper tests)
- The percentage of efficiency gain with respect to previous approaches for test design in terms of defect detection (better tests)

Defining and monitoring metrics and KPIs is part of project management best practices when deploying an MBT approach.

5.2 Manage and Monitor the Deployment of an MBT Approach

5.2.1 Good Practices when Deploying MBT

The MBT-based fundamental test process together with its artifacts and tools should be closely integrated into the existing development process, test process and tool chain. This is also important for integration in the application management lifecycle tool chain, for example the integration of MBT with requirements engineering processes and tooling.

A seamless integration is a key factor for the success of the MBT introduction. Good practices here include:

- Configuration management of all MBT artifacts, including:
  - Test basis
  - MBT models and test selection criteria
  - Test cases and test scripts
  - Adaptation layer specification and code

  In a development process, artifacts such as release or deployment artifacts are versioned. This is mandatory to establish a working development and deployment process. To integrate and relate MBT artifacts in such a process, configuration management of MBT artifacts is also mandatory.

- Integration of the MBT test generation process with continuous integration
  Making a build self-testing is one of the key values of continuous integration. Once the code is built, the continuous integration server calls testing tools to check the new content. Not only unit testing tools, but also MBT tools should be integrated here, especially when MBT is used for continuous regression testing.

- Integration with requirements engineering and backlog management practices
  Requirements and backlog items (planned product features) must be tested before they are done (according to the definition of done [ISTQB_AT_FL_SYL]). For example, in an Agile process, the test process should be reflected in the backlog management process. If MBT is used for the testing of backlog items, the corresponding MBT artifacts should be traceable in the backlog management tool, i.e., “which test cases from which versioned model with which test objective were used to test this item”.

5.2.2 Cost Factors of MBT

The following tables relate initial costs and running costs of MBT to the testing activities of the ISTQB fundamental test process.

Initial costs (for the organization and for the project):

Initial Costs for the Organization	Initial Costs for the Project
• Checking existing resources and knowledge for MBT introduction • Evaluating MBT approaches and tools • Defining and implementing MBT methods and	• Creating project-specific MBT modeling and process guidelines • Creating the initial MBT model • Transforming assets (e.g., from textual test cases

processes • Integrating with requirement management, test management, continuous integration • Automating and integrating the MBT reporting • Establishing means for archiving MBT artifacts • Creating general MBT modeling and process guidelines • MBT coaching and training • Licensing of MBT tooling	to MBT models) • Migrating MBT models

Running costs:

Testing Activity	Running Costs
• General	• Licensing of tools (depending on the license model) and maintenance costs • Coaching and training of new team members
• Planning and Control	• Analyzing test basis with respect to MBT testability • Planning the development / enrichment / derivation of MBT models • Continuously checking MBT model quality
• Analysis and Design	• MBT Modeling • Refactoring MBT models • Validating and verifying models
• Implementation and Execution	• Choosing suitable test selection criteria • Generating executable test cases • Developing the test adaptation layer (in case of automated test execution) • Executing test cases (manually or automatically)
• Evaluating Exit Criteria and Reporting	• Ensuring traceability of defects • Documenting test completion criteria • Combining MBT evaluation and other test evaluations in a common report
• Closure	• Archiving MBT artifacts • Documenting gained knowledge with respect to MBT • Transitioning MBT artifacts and processes to maintenance

5.2.3Integration of the MBT Tool

Introducing an MBT tool into an organization follows the same principles that apply to introducing any other testing tool, see [ISTQB_FL_SYL], section 6.3. Since MBT typically is not as prevalent as traditional test approaches, the tool evaluation and introduction must be particularly solid and must not be underestimated. An important aspect of this evaluation is related to the integration of the MBT tool with configuration management, requirements management, test management and test automation tools.

A typical test tool chain embedding MBT is depicted in the diagram below.

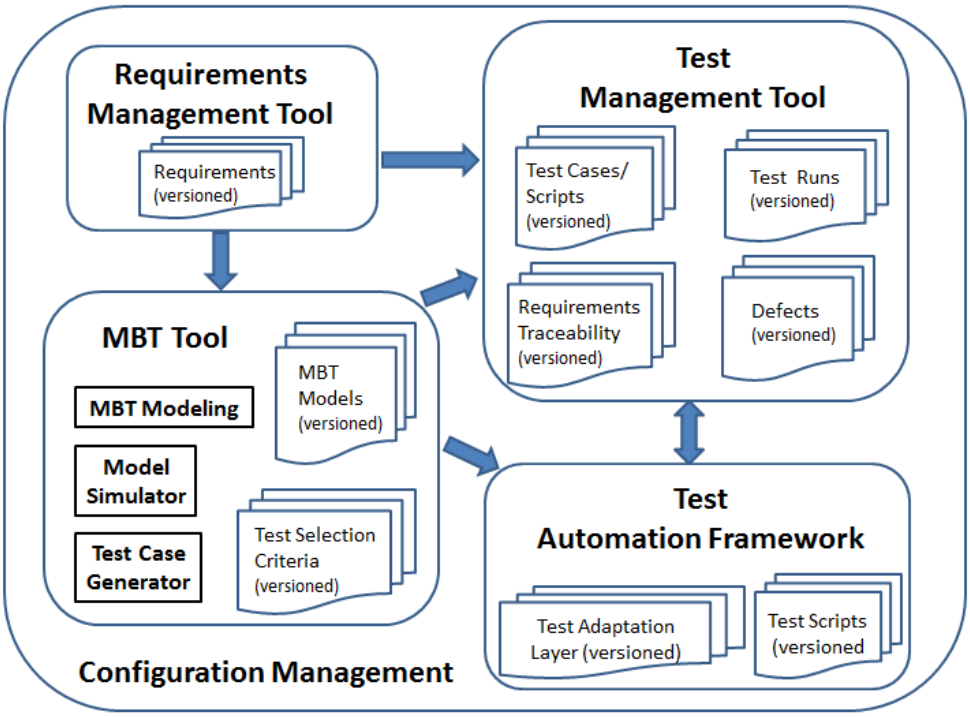

Figure 1 – A typical tool chain embedding MBT

This typical tool chain supports the main activities of the MBT-based fundamental test process, including:

- Iterative MBT model development, review and validation using the modeling tool, the model simulator and the traceability with the requirements
- Test generation by applying test selection criteria to the MBT model supported by the test case generator
- Generated test cases and test scripts as well as traceability links that can be exported to the test management tool and the test automation framework (in case of test execution automation)
- Configuration management for the MBT testware such as the MBT model and the adaptation layer

6 Abbreviations

Abbreviation	Meaning
BPMN	Business Process Modeling Notation
CTFL	Certified Tester Foundation Level
ISTQB	International Software Testing Qualifications Board
KPI	Key Performance Indicator
MBT	Model-Based Testing
ROI	Return On Investment
UML	Unified Modeling Language

7 Registered Trademarks

Trademark	Owner
UML®	Object Management Group, Inc.

8 References

Standards

[ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality 4 - Requirements and Evaluation (SQuaRE)
[ETSI_MBT] ETSI ES 202 951, Methods for Testing and Specification (MTS) - Model-Based Testing (MBT) - Requirements for Modeling Notations, Version 1.1.1 (2011-07)

ISTQB Documents

[ISTQB_AT_FL_SYL] ISTQB Agile Tester - Foundation Level Syllabus, Version 2014
[ISTQB_ATA_SYL] ISTQB Advanced Level Syllabus Test Analyst, Version 2012
[ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011
[ISTQB_GLOSSARY] Standard Glossary of Terms used in Software Testing, Version 3.0, 2015
[ISTQB_MBT_OVIEW] ISTQB Certified Model-Based Tester Overview, Version 2015

Referenced in this Syllabus

[Utt07] Mark Utting and Bruno Legeard, "Practical Model-Based Testing – A tools approach," Morgan&Kauffmann, 2007.
[Wei09] Stephan Weißleder. "Test Models and Coverage Criteria for Automatic Model-Based Test Generation with UML State Machines," PhD Thesis, Humboldt-Universität zu Berlin, 12/2009.
[Zan11a] Justyna Zander, Ina Schieferdecker and Pieter J. Mosterman, "A Taxonomy of Model-Based Testing for Embedded Systems from Multiple Industry Domains," In Model-based testing for embedded systems. CRC Press.

Other References

[Bak08] Paul Baker, Zhen Ru Dai, Jens Grabowski, Øystein Haugen, Ina Schieferdecker and Clay Williams, "Model-Driven Testing," Springer, 2008.
[Jac07] Jonathan Jacky, Margus Veanes, Colin Campbell and Wolfram Schulte, “Model-Based Software Testing and Analysis with C#," Cambridge University Press, 2007.
[Sch12] Ina Schieferdecker: Model-Based Testing. IEEE Software 29(1): 14-18, 2012.
[Utt12] Mark Utting, Alexander Pretschner and Bruno Legeard, "A Taxonomy of Model-Based Testing Approaches," Softw. Test. Verif. Reliab. 22 (5), 297–312, 2012.
[Zan11b] Justyna Zander (Editor), Ina Schieferdecker (Editor) and Pieter J. Mosterman (Editor), “Model-Based Testing for Embedded Systems," CRC Press, 2011.

9. Appendix A – Simple Modeling Language

For the purpose of the K3 level learning objectives, two simple graphical modeling languages are discussed here:

- The first one is a subpart of UML activity diagrams
- The second one is a subpart of UML state machines

In the next two sections, these subparts of UML, which can be used to practice MBT modeling, are defined by example.

9.1 A Simple Graphical Modeling Language for Workflows

This modeling language may be used to build MBT models representing workflows or activity diagrams. The idea is to model a flow of activities that are controlled by intermediate decisions. MBT models developed with this language can be used to generate test cases on the basis of test selection criteria (see chapter 3).

This graphical modeling language is a subpart of the UML activity diagram. It is composed of the following elements:

- A black circle representing the start (initial state) of the workflow and an encircled black circle representing the end (final state)
- Rounded rectangles representing actions
- Diamonds representing decision and merge nodes with possible labels (text)
- Arrows representing flows, possibly with expressions such as text or logical expressions (including arithmetic and Boolean operators)
- Bars representing the start (split) or end (join) of concurrent parallel activities – the sequence is continued after all parallel activities arrive at the join
- Requirements identifiers linking to activities using dashed lines
- Sub diagrams denoted by rectangles.

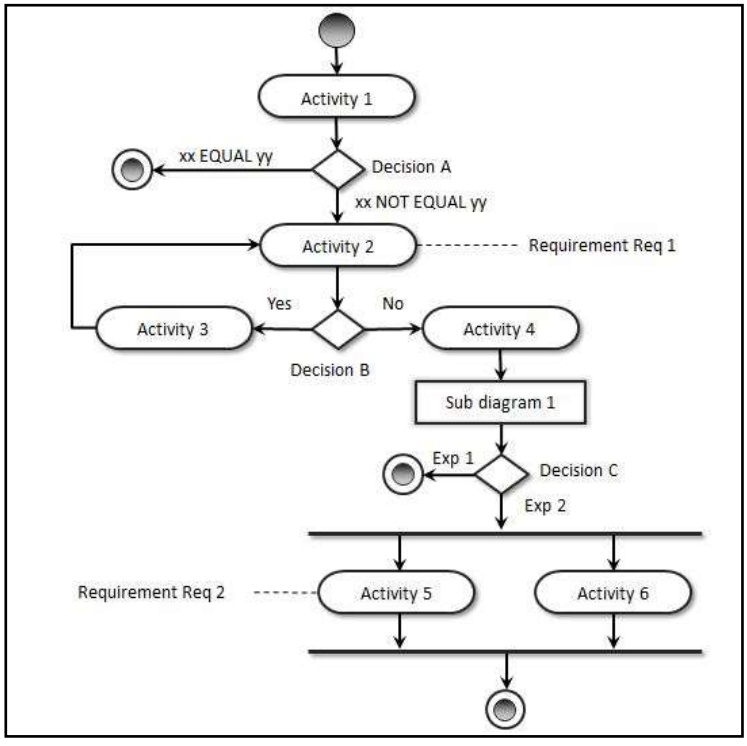

Figure 2 – Example of activity diagram

Such a description of a modeling language is rather informal and simple, but contains enough information to be used to create MBT models in the context of this syllabus.

Figure 2 represents an abstract example of an MBT model developed with this modeling language. It shows behavior that always starts with Activity 1.

The process terminates if “xx EQUAL yy”. Else, Activity 2 is executed.
Activity 3 and Activity 2 are repeatedly executed until the result of Decision B is “No”.
In this case, Activity 4 and the behavior of sub diagram 1 is executed.
The behavior stops if the result of Decision C is equal to Exp 1.
Else, Activity 5 and Activity 6 are executed in parallel, then the behavior stops.
The notation shows that Activity 2 is related to requirement Req 1 and Activity 5 is related to requirement Req 2.

9.2 A Simple Graphical Modeling Language for State Transition Diagrams

This graphical modeling language is a subpart of UML state machines. It is composed of the following elements:

- A black circle representing the start (initial state) of the workflow and an encircled black circle representing the end (final state)
- Rounded rectangles re-presenting states
- Arrows representing transitions with textual annotation “event [condition] / action” (the trigger referring to any event, the guard containing arithmetic and Boolean operators that form conditions, the effect calling any action)
- Diamonds representing decision and merge nodes
- Requirements identifiers linking to any kind of element using dashed lines
- Sub diagrams are denoted by rectangles

Such a description of a modeling language is rather informal and simple, but contains enough information to be used to create MBT models in the context of this syllabus.

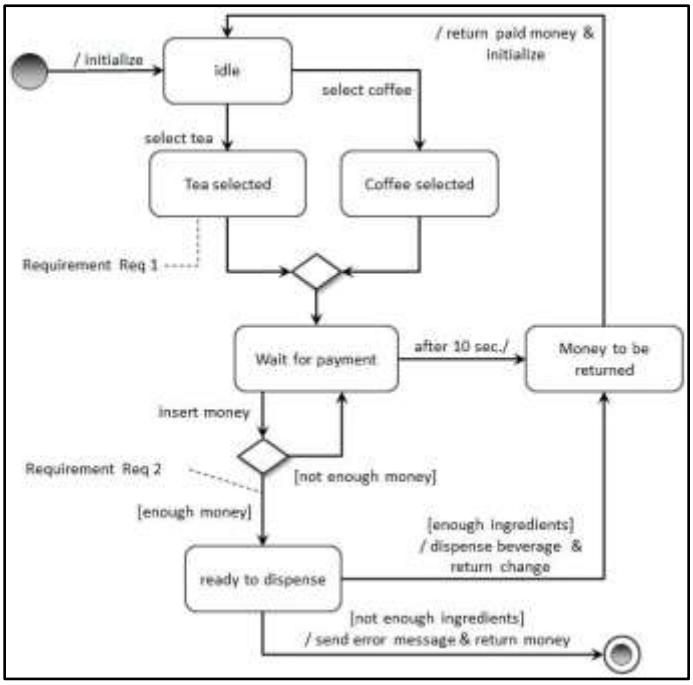

Figure 3 – Example of state transition diagram

Figure 3 represents an example of an MBT model developed with this modeling language. It shows the behavior of a beverage dispenser containing tea and coffee.

If the automaton is activated, the deposited amount is initialized and it is waiting for user inputs.
These user inputs may be “select tea” or “select coffee” indicating the desired beverage.
As a reaction to the selection, the corresponding preparations are made (e.g., setting different prices for tea for coffee). Afterwards, the user has to pay for the beverage.
The model contains a loop that makes sure that the automaton enters the state “ready to dispense” only after enough money has been inserted.
If the user stops inserting coins before the necessary amount was inserted, the automaton waits for 10 seconds before returning the inserted money and starting all over again.
If enough money has been inserted, the beverage is dispensed.
As the only exception, the automaton shuts down and returns the inserted money if not enough ingredients (water, tea, coffee) are available.

One can also see that the requirement Req 1 is linked to state “Tea selected”, indicating that a test that reaches this state also covers this requirement. Requirement Req 2 is linked to the transition to “ready to dispense” to state that the payment of the beverage is an important aspect.

Index