1

INTRODUCTION TO MODEL-BASED TESTING

This chapter covers the learning objectives of syllabus Section 1.1 – "Objectives and Motivations for MBT."

1.1 WHY DO WE NEED NEW APPROACHES TO TESTING?

In the earliest days of software testing, developers just made sure the program was running correctly. Debugging and testing were the same. The notion of testing as a separate activity came up in 1957. Unlike debugging, testing assures that the software really solves the problem [5]. Since then, "Software Tester" became a profession along with special qualification schemes such as the ISTQB Certified Tester.

The significance of testing directly correlates with the complexity of software applications and, thus, constantly increases. Software systems become not only larger, but also more interoperable. They support extensive workflows on various platforms. Product variants also add to complexity. Ideal for marketing, because they allow you to target different market segments, they are a hell to test! As a result, we have a huge number of test cases to execute and, rapidly, lack the overview. We are no longer able to answer questions such as "What shall be tested?/What has been tested?" or "Which tests are important?/What may be left out?" "How do you want to check completeness of large document-based test specifications?" Besides, the documentation of the test idea is lost amidst the detailed test instructions. Thus, though being the authors, we later do not remember why we tested something the way we did.

Model-Based Testing Essentials–Guide to the ISTQB® Certified Model-Based Tester Foundation Level, First Edition. Anne Kramer and Bruno Legeard.

The industrial demands regarding time-to-market and cost reduction constantly increase and definitely affect testing. Testing less not being an option, we need faster and better tests. "Better" means improved objectivity, repeatability, and transparency. This is where model-based testing (MBT) comes in. The main idea is to formalize and automate as many activities related to test case specification as possible and, thus, to increase both the efficiency and effectiveness of testing. Instead of writing a test case specification with hundreds of pages, we draw a model (called an *MBT model*) and, then, use a tool to generate the tests. In the manager's dreams, it is even simpler. We reuse models from software design and obtain the corresponding test cases simply by pressing a button. However, as we discuss in this book, MBT is not just reusing development models and working with a tool, but an entire new approach to test design and test implementation.

There is another, psychological motivation for MBT. Let us face the truth: being a software tester is still not as prestigious as it should be. Testing is perceived as a destructive activity [6]. Testers try to find bugs, looking for flaws. Even worse, they really find them! In addition, they constantly complain about imprecise or inexistent requirements. Testers ask uncomfortable questions and expect an answer. They work so well that testing quite often becomes more expensive than expected. Testing is one of these cost factors (known as "cost of quality") managers try to reduce, and nobody really knows, how expensive it would have been without testing ("cost of poor quality"). It is very difficult to measure quantitatively what the tester's work is worth.

By writing MBT models, the tester applies a technique also used in requirements analysis and development to manage complexity by abstraction. As we discuss later, modeling is not so different from coding. If testers are able to do this, they are on equal terms with the developers and communication becomes easier. Ideally, the developer perceives the tester's work as helpful rather than as criticism. In addition, an MBT model visualizes the complexity of testing. For those who decide on budgets, the risk of leaving out specific scenarios becomes far more obvious than in a traditional, document-based approach.

1.2 WHAT IS MODEL-BASED TESTING?

The idea of MBT is ingeniously simple. If we had to summarize this book in one sentence, our recommendation would be the following: "Draw pictures whenever possible to explain what you are doing or going to do, and use it to create and maintain your testware." This is what modeling is all about, at least if we think about graphical models (i.e., diagrams). There is also a brief section on textual modeling languages in this book. Some MBT approaches combine graphical and textual representations, either to visualize the textual model in a diagram or to complete the graphical model with textual information.

Writing models is nothing magic. We do it instinctively. If a process becomes too complicated to explain, we draw a flowchart. If we want to describe how the different components of a larger system relate to each other, we draw a block diagram. Both the flowchart and the block diagram help us in structuring and visualizing complex issues.

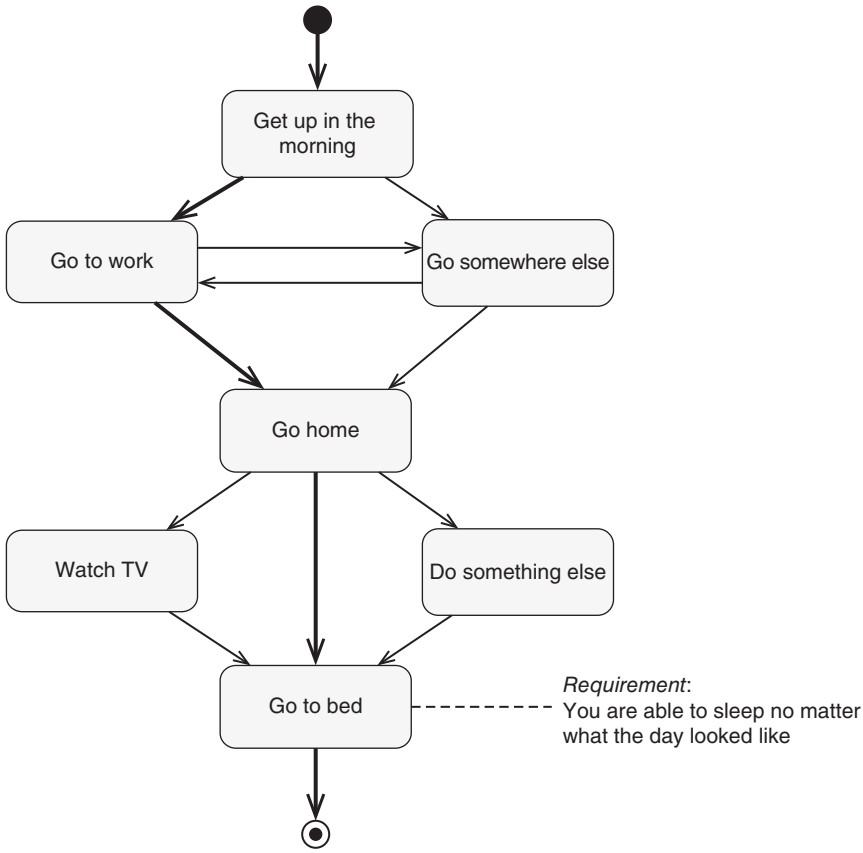

Figure 1.1 Test of a normal working day with some variations (activity diagram; Note: Throughout this book, we use the simplified modeling languages defined in the ISTQB MBT syllabus [13] (see Section 3.3). However, you should be aware that this is only one out of many possible notations. Depending on the selected modeling language, notation paradigm, and tool support, the same MBT model may look different).

Now, everybody who has tried knows that testing really is a complex issue. So why not use models to describe what we want to test and how we are going to do it?

Figure 1.1 shows an example of a very simple model. It describes a normal working day we all know well enough. We get up in the morning, go to work, back home, and straight into bed. In France, they have a nice expression for this kind of life, which is "Métro, boulot, dodo" (literally "subway, work, sleep"). The bold arrows in the figure indicate this typical working day. Fortunately, some variants exist. If we are not too tired, we watch TV or do something else before going to bed. Sometimes we do not directly go to work or we may go somewhere else for lunch and then get back to work. Depending on the path you take through the model there are small variations in this daily routine.

What has the model in Figure 1.1 to do with testing? Imagine that the person whose day we described in the model is the system under test. The requirement we have to check is that the system is able to sleep no matter what variation of the daily routine it went through. Thus, each path through the model corresponds to a sequence that we could test, that is, a potential test case. Unfortunately, if we do not put any limit to the back and forth between working place and somewhere else, we obtain an infinite number of possible paths, corresponding to an infinite number of test cases. Performing an infinite number of tests is neither possible nor does it make sense. In most cases, the worker mostly leaves his working place just once to get some lunch. We should definitely test this situation!

For the model in Figure 1.1, this means that we neglect repeated loops. This leads us to 18 different sequences, corresponding to 18 test cases.

Exercise 1 *Find the 18 possible sequences in Figure 1.1 under the assumption that repeated loops are not allowed.*

Now, imagine a slightly different situation. If the person works in sales, he or she will probably perform several trips per day to visit some customers. This is not covered by our 18 test cases. However, is it necessary to test two, three, four, five … trips per day? Probably not. Usually, it is sufficient to prove that the system works for the longest sequence. Maybe, five trips per day is a reasonable upper limit to test in a 19th test case.

There is something important to understand from this simple example. It is one thing to draw a model and a second thing to obtain test cases from it. This is very different from traditional test design where you only write down those test cases you want to execute (even if, in the end, there may be no time for them). We have to think about what we really want to test, and it strongly depends on the system under test and its context which sequences we should select. Of course, similar considerations influence traditional test design, but they are far more explicit in MBT. We use the term "test selection criteria." Those criteria guide us in the selection of tests we want to generate from our MBT model.

It is only a small step from Figure 1.1 to our first test case derived from an MBT model. All we have to do is to follow a path from start to end and to collect the information in between. The test case "Métro, boulot, dodo" (the bold path in Figure 1.1) looks like Table 1.1.

TABLE 1.1 Our First Test Case Generated from a Model

Test Case – Normal Working Day 01
Get up in the morning
Go to work
Go home
Go to bed
Requirement: You are able to sleep no matter what the day looked like
passed □ failed □

Figure 1.1 looks like a simple description of the system under test we could find in our requirements specification, but it is only a start. Testers can be very mean. It is their task to find errors. They try to push the system under test to its limits. A tester will probably try 100 trips back and forth to work. Even more, he will not stay with the model in Figure 1.1. Instead, he will add additional arrows and check, whether it is possible to watch TV at work or to sleep somewhere else. He will discuss the enriched model with developers and other stakeholders and discover that there is an undocumented requirement that the system does not snore. This is already the first step toward quality (and a restful night).

Even if this is not the most efficient way to do MBT, a simple model similar to the one in Figure 1.1 helps already a lot, especially if you have to test complex systems. It is a bit like writing a mind map. In the beginning, you do not know where to start, but during the modeling process, you start seeing the abstraction layers. In addition, you can discuss the test design with others and show them the picture. You draw the model to clarify, to document, and to discuss the test idea.

To perform MBT on a higher maturity level, we need additional information on how the tests stimulate the system and what they exactly check. Using a modeling tool, we are able to add a detailed description to each action in a way that they are contained in the model, but invisible in the picture. As a human, we can only see them in a separate window, for example, if we double-click on the action, but a test case generator will be able to collect them into a test case. One question, however, remains. Which paths through the model shall we test and which paths may we leave out? Tools can help by selecting paths according to given criteria, but even then, we have to decide about the criteria that we wish to apply. There will be a larger section dealing with test selection criteria later on.

1.3 BENEFITS OF MBT

MBT positively affects the efficiency and effectiveness of testing on several aspects. On one hand, MBT models help in managing complexity, improve communication between testers and other project stakeholders, and foster common understanding of requirements. On the other hand, they provide us with better control on generated test cases and with the option to automate test design and implementation activities. Finally, MBT models are a kind of knowledge management.

1.3.1 MBT Models Provide an Overview and Help in Managing Complexity

The biggest advantage of using models to describe tests is the fact that they help in keeping the overview on what is actually tested. The hackneyed saying in this context is “a picture is worth a thousand words.” The estimation on how many words fit on a page A4 vary between 250 and 500. Since test specifications are usually not so dense, let us be conservative and assume an average of 250 words. Thus, one picture of an MBT model replaces four pages of document-based test specification. Exaggerated? Not necessarily. It depends on the model.

TABLE 1.2 Our First Test Case with Detailed Test Instructions

Test Case – Normal Working Day 01 (Extended Version)
• Switch off the alarm clock • Get out of the bed • Take a shower • Get some breakfast • Brush teeth • Dress for work • Leave the house
• Take the car • Drive to work • Search a parking lot • Memorize location • Go to office
• Leave the office • Search the car on the parking deck • Drive home • Park the car • Enter the house
• Brush teeth • Undress for the night • Check the alarm clock • Switch off the lights • Go to bed
Requirement: You are able to sleep no matter what the day looked like passed □ failed □

Take the example in Figure 1.1. We decided to derive 19 test cases from the model, all of them being at least as long as the one in Table 1.1, which is one-fifth of A4. Thus, this simple model already corresponds to roughly four pages of text, but the test case does not even clearly describe what to do. With more detailed test descriptions, the generated test case may look like the one in Table 1.2. The text nearly tripled in size, but the picture remains the same. In addition, the MBT model separates the conceptual test design from technical test implementation details. This is why models help in keeping the overview – and the best is – the use of models for testing purposes is not limited to a specific test objective or to a specific application

domain! We use models to test the business processes and rules of a large enterprise IT system, as well as models for load testing of a graphical user interface of a web application. Obviously, the various MBT models look extremely different, but the underlying concept is the same.

By using MBT models instead of prose, we introduce a higher level of abstraction in test design. Just compare Figure 1.1 with Table 1.2. The picture shows the context of our tests far better than the textual description. Possibly, our test strategy requires extended tests for critical parts. If you show the MBT model to an expert (e.g., a software developer), he will be able to tell you, in which part of the model (and, thus, of the system) the tests will probably detect more problems than in the other parts, because the part implements a new feature or because it is more complex than the rest.

1.3.2 MBT Models Help Communicating

For a moment, put yourself in the shoes of the developer: The project deadline approaches. You still have to implement some features and there is a bug you are unable to locate. Of course, you are stressed – and this is exactly the moment when the tester starts asking uncomfortable questions. "What happens to the data entered beforehand, if I press Cancel? Does 'Cancel' always take me back to the start screen? Are you sure?"

Next, try the tester's shoes. You still have some time, but from experience you know that you should start writing the tests, now. The requirements are not as bad as they could be. Of course, some points remain unclear, especially regarding the Cancel button. Unfortunately, all the system-matter experts are currently under stress. They do not have much time for you. How can you bring your understanding of the requirements and your questions to the point?

Draw a picture! If this becomes too complicated, draw several pictures. If you use a tool, you can construct so-called hierarchical models with different abstraction layers. Thus, you may discuss the general workflow with the product manager and the implementation details with the developer, showing each of them the part of the model that corresponds to his level of understanding. The model fosters a common perception and understanding of the requirements and helps identifying misunderstandings.

A common scenario from personal experience

From time to time, customers ask for an exemplary MBT model for a feature that is usually not too complex. We call this an MBT prototype. The scenario that follows is always the same. Based on the existing documentation we construct the model. If questions arise, we comment them in the model. When we present the model to the customer the discussion starts … between the customer representatives. Most often, the group does not have the same understanding of the feature and, sometimes, we really detected an inconsistency in the specification or an unspecified behavior.
(Anne Kramer)

1.3.3 MBT Models Validate Requirements

Writing good requirements is extremely difficult. You may spend a lot of time in formulating and reviewing them, but the ultimate check takes place when you start working with them. Only then, you will see how good they are in reality.

Obviously, the first to work with the requirements are the developers. Somehow, but they steer around most problems regarding unclear requirements. Either they are more inventive or they have shorter communication channels than testers. (The latter is definitely true, if the test is outsourced.) The next to work with the requirements are the testers. Good testers are rigorous. They despair of unclear or conflicting requirements because it is their task to prove that the software conforms to those requirements. Quite often, testers detect major problems in the requirements that the developer must fix. Thus, the earlier the testers start their work the better it is.

With MBT, it is not necessary to know all implementation details to start writing tests. In a top-down modeling approach, the first version of the MBT model has a high level of abstraction and does not yet contain any detailed test descriptions. It describes the main features to be tested, the checks to perform, and their order. The checks usually refer to the requirements. In Figure 1.1, this is the feature "You are able to sleep no matter what the day looked like." We still do not know anything about the detailed instructions in Table 1.2, but we may start thinking about possible ways to keep the system awake. (A real bad horror shocker on TV would do.)

Of course, you do not necessarily need to write a model for these preliminary thoughts. However, it is an excellent way to validate requirements early, especially if the requirements are in prose. The earlier you start with MBT modeling activities, the earlier you will be able to share your thoughts with business analysts and developers, to generate test cases and to perform them. This helps implementing the "early testing" principle. If you start the MBT modeling activity at the end of the requirements analysis phase, you may even have the chance to resolve unclear or conflicting requirements before they are implemented wrongly – and, as we all know by now, the earlier you find a defect, the faster, easier, and cheaper it is to fix.

1.3.4 MBT Models Visualize Tests

Classic document-based test specifications easily count several hundreds of pages. Reviewing them is a hard task. People's concentration has its limits. They are certainly able to check the consistency within one test case, but it is very hard to check, whether each test aspect has been fully covered.

Show them the model! It represents your idea of the tests. You can show them the paths you decided to take and those you left out, and explain your reasons. Even better: the model helps you to remember your own thoughts and decisions when asked in a few months.

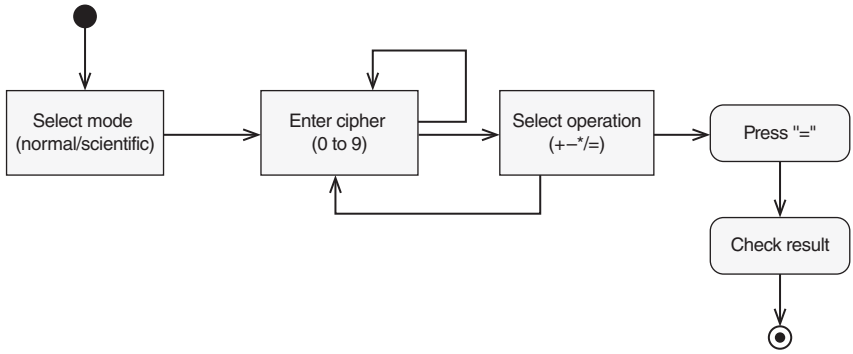

Figure 1.2 A well-known application (activity diagram).

Try yourself by doing Exercise 2. How well do you understand the model in Figure 1.2 without any explanation?

Exercise 2 *The model in Figure 1.2 describes the test of a commonly known application. Do you recognize the system under test? What exactly shall be tested?*

The figures are easier to understand than lengthy textual descriptions and to discuss with non-technical stakeholders, for example, business analysts. Again, this facilitates communication and helps in validating requirements at an early stage of the project.

1.3.5 Model-Based Test Generation Is Well Supported by Tools

All the above-mentioned advantages are due to the MBT model itself. They do not depend on any tool support apart from a model editor. However, only a test case generator will provide you with the full advantages of MBT, that is, the possibility of automation.

Various model-based test generators are available on the market. They differ a lot regarding the required input and the generated output. In one aspect, however, they are alike. Test generators are cheaper, faster, and more precise than humans are. They can do all the work that requires precision, but no particular intelligence. For example, they may traverse all paths through the model, gathering information on the way and create an output like Table 1.2.

Of course, someone has to draw the MBT model. The big advantage is that we only have to do it once and the generator replicates the information automatically. The generator may even create several test suites from the same model using different test selection criteria. In traditional test design, the tester spends only a small part of

his time on the initial test design, that is, the test idea. The rest is writing and/or implementing test cases. In MBT and with a test case generator, we reverse this ratio. The tester spends the majority of time on analyzing the behavior of the system under test and imagining possible test scenarios and much less on deriving test cases thanks to automated test generation. The improved efficiency is a major argument in favor of tool support, especially in the maintenance phase of the tests.

Once we have the model, we may generate different sets of test cases from it, depending on the test objectives. We already practiced this in Section 1.2 when we derived 18 test cases corresponding to the different variations of a normal working day in office, plus one additional test case for the continuously traveling sales representative. Again, appropriate tool support considerably facilitates the task of test case selection.

Apart from test case generation, tools also support other test-related activities, such as documenting traceability between test cases and requirements, documenting the test approach or generating test scripts and interfaces for automated test execution. In our first model describing the test procedure for a normal working day (Figure 1.1 in Section 1.2), we linked the requirement to the action that checked its fulfillment. The test case generator may automatically include this information to all generated test cases and/or generate a separate traceability matrix from the model, thus keeping both test specification and traceability matrix permanently synchronized. A picture of our model together with the applied test selection criteria documents our test approach in a test plan. Finally, test case generators usually provide interfaces to (or include features of/are part of) test management tools. Altogether, MBT supports all kinds of automation in the testing process.

1.3.6 MBT Models Document Knowledge and Helps in Sharing It

Imagine the following scenario: Project A is finished. The first version of your product is out in the market. Most developers and testers left the team to go for new opportunities. Then, Project B starts. The target is to develop an enhanced version of the same product with some new features and a low-cost variant of the hardware.

From a testing perspective, we have to ensure that the existing features still work as before. In addition, we will have to add new tests to verify the enhancements in version 2. In a traditional document-based approach, you would probably keep the existing test specification, possibly adapt it a bit and write completely new test cases for the new features. However, this is not the most efficient way to work. First, it will be difficult to test the new features in combination with the old ones without duplicating parts of the test procedure specification. Second, the number of test cases will constantly increase leading to longer and longer execution times. Third, you have no idea why the existing test cases are written the way they are.

As stated before, models document the test idea. For new testers, it is easier to understand both the system under test and the test approach by looking at the figure, rather than by reading long documents. They may then reuse the model, or parts of it,

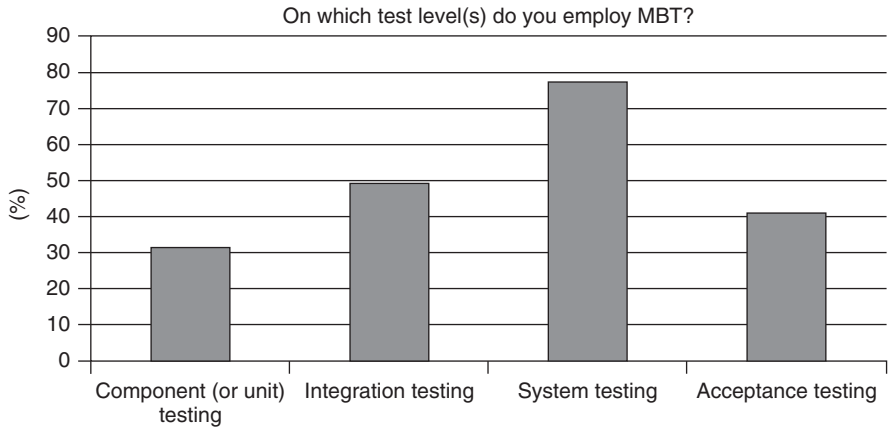

Figure 1.3 Test levels reported by MBT practitioners (from 2014 MBT User Survey).

and adapt it to their specific needs. If well done, it is possible to set up model libraries, which is a very efficient form of knowledge management all testers benefit from. Modeling helps the testers to understand what they describe and, thus, to acquire continuously new competencies and domain knowledge.

By the way, most developers and test automation engineers hate writing documents. To lesser extent, they even hate reading them. They accept the visual representation of the same content in a model far better than the textual representation.

1.3.7 MBT Covers Many Test Objectives, Test Levels, and Test Types

First, MBT more or less addresses all test levels. In the 2014 MBT User Survey, we asked MBT practitioners, on which test level(s) they employ MBT [4].[1] They reported all test levels (see Figure 1.3), even if we observe a tendency toward model-based integration testing (50%) and a clear trend toward model-based system testing (77%).

Similarly, MBT is not limited to specific application domains (see Figure 1.4). In this book, we focus on software testing, partly due to our personal background and partly, because software testing is so complex.

Regarding the type of testing, a large majority of MBT practitioners (97%) employ models for functional testing, but MBT also supports other types of testing such as performance, security, and usability testing (see Figure 1.5).

Finally, MBT supports all kind of test objectives depending on the selected modeling language, the subject and focus of the MBT model and various test selection criteria, as discussed later in this book.

[1] We limited the survey question to the four test levels defined in Ref. [6]. System integration testing (e.g., end-to-end testing) is located somewhere between the system and acceptance testing.

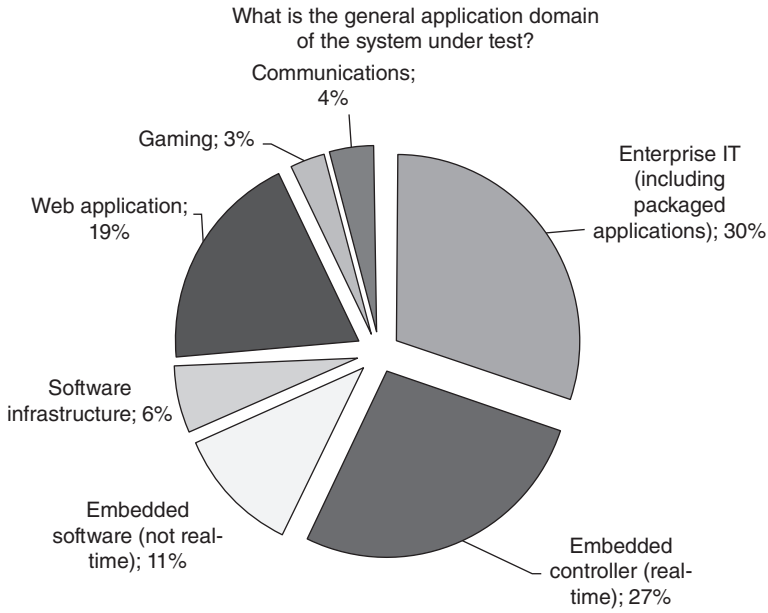

Figure 1.4 Application domains reported by MBT practitioners (from 2014 MBT User Survey).

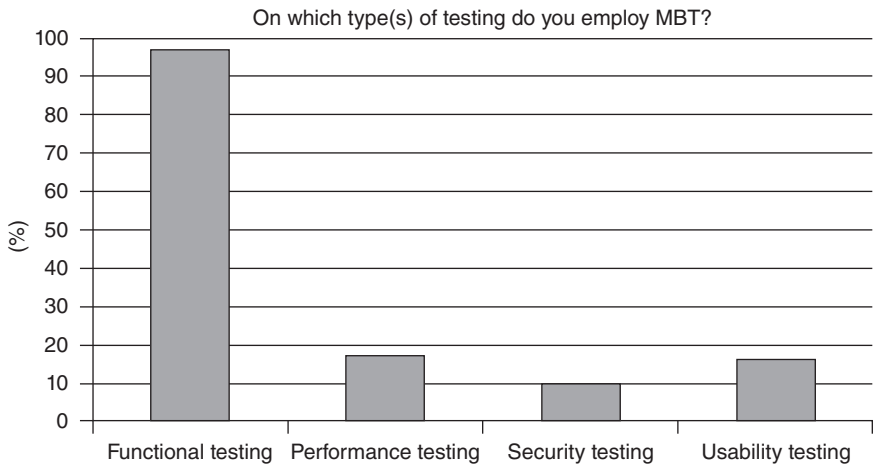

Figure 1.5 Testing types reported by MBT practitioners (from 2014 MBT User Survey).

1.4 PITFALLS OF MBT

MBT provides benefits but with a cost. In this section, we sum-up several drawbacks and pitfalls of MBT.

1.4.1 Introducing MBT Is a Change

MBT requires some "rethinking." Many people call it a paradigm shift. From a formal point of view, the expression "paradigm shift" does not exactly fit in this context (see box "Why MBT is not a paradigm shift?"). MBT does not overthrow traditional test design, but extends and supports classic test design techniques such as equivalence partitioning, boundary value analysis, decision table testing, state transition testing, and use case testing.

Nevertheless, MBT is a change. The need for explicit test selection criteria is one example. The MBT model represents a set of possible tests, but it is neither feasible nor reasonable to generate and/or execute all of them. The MBT model in Figure 1.1 represents an infinite number of possible test cases. Somehow, we have to select, because "more tests" is not a synonym for "better tests." This is definitely different from document-based test design. There, you would never write down a test scenario you do not intend to execute.

Why MBT is not a paradigm shift?

Thomas Kuhn developed the concept of "paradigm shift" in his book "The Structure of Scientific Revolutions" [7]. *A paradigm corresponds to the core concepts that we use to analyze and explain the world. It provides model problems and solutions to the scientists. Kuhn claimed that scientific progress is not evolutionary, but rather revolutionary. From time to time, a scientific discipline passes through a state of crisis because of significant anomalies that are observed, but may not be explained by the current paradigm. During such a crisis, new ideas come up and eventually end up in a new paradigm that fits better.*

Initially restricted to natural science, the expression "paradigm shift" has become very popular. Nowadays, we use it in various contexts. (If you are further interested in that topic, please refer to Robert Fulford's column on the misuse to the word "paradigm" [8].*)*

MBT is not a paradigm according to the definition of Kuhn. MBT models are not in contradiction with "traditional" test specifications described in natural language, even if MBT requires a different way of thinking and testers with higher qualification. Instead, we should consider MBT as an add-on providing a higher level of abstraction and the additional possibility of automated test case generation.

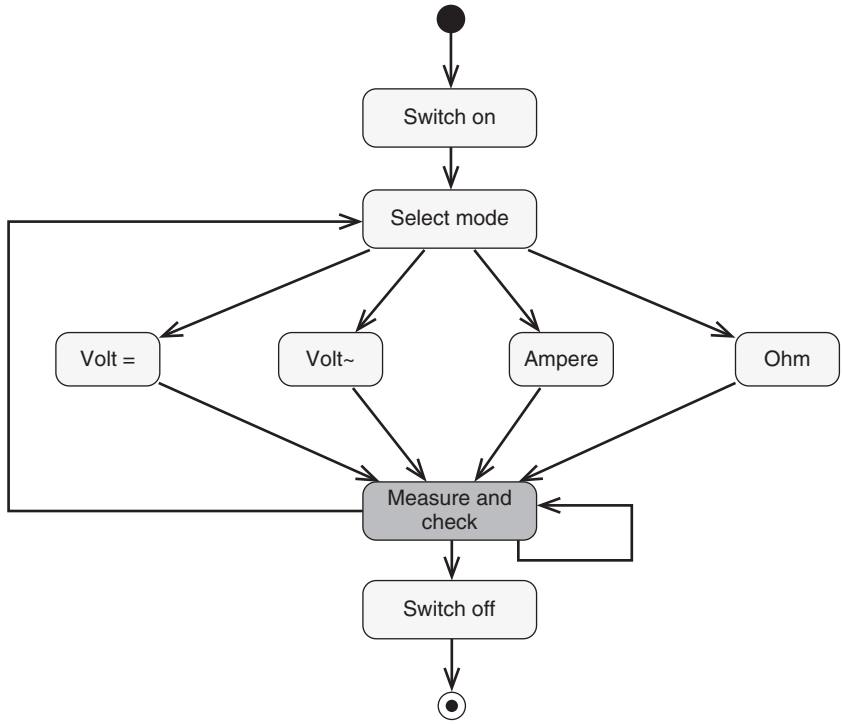

Figure 1.6 MBT model describing the measurement modes of a multimeter (activity diagram; Note: In this figure, we highlight the action where the verification takes place (rounded gray rectangle)).

Even the test cases themselves look different. Figure 1.6 shows the different measurement modes of a multimeter (direct and alternating voltage, electric current, and resistance). Theoretically, we may obtain an infinite number of test cases from this model. In practice, we will limit ourselves to a set of test cases following some logic we have in mind, which corresponds to the test objectives in this particular project. For example, we may derive one test case per measurement mode, each of them checking repeated measurements in the selected mode, plus one test case checking all four modes one after the other in a single sequence. The logic in mind is called "full coverage" of the measurement modes, but with limited effort. We made sure that each model element is reached at least once and add one test case to cover possible switches between measurement modes. The selection of the sequences depends on the algorithm the MBT tool uses for test generation. In the extreme, we may end up with one very long sequence checking everything in a row.

Psychologically, this is not always easy to accept. In fact, you delegate part of the work to the tool and, as is always the case with delegation, you should accept the other person's way of working, as long as the results meet the goals (in this case the test objectives).

Sometimes, the tool will even generate test cases you never thought of. From the multimeter example in Figure 1.6, you may derive test cases with many mode switches and many consecutive measurements in the same mode. The tool will propose an order of actions, which is probably not the one you would have obtained in manual test design. Of course, the fact that the tool uncovers unusual scenarios is rather an advantage than a disadvantage, but it definitely needs getting used to (see box "Play the game").

Last, but not least, introducing MBT to an existing test process is comparable to any other organizational change project. It concerns everybody involved in development and quality assurance, introduces new tools, forces testers to learn new and rather formal modeling languages, and so on. Section 10.1 addresses the question how to overcome such potential obstacles. In short, changes take time and should be well prepared. Never underestimate human resistance against change.

Play the game

Once upon a time, I had to review an MBT project to analyze the effort savings. It was a large IT project developed by an outsourced team in a context of legacy code migration. The testing team was skilled and mastered the tool chain well, including the newly introduced MBT tool.

Then, I discovered that the MBT models were developed to produce test cases based on detailed test case specifications, meaning already designed test cases. This seems wonderful and easy to do: from the test case specifications, you draw your MBT model, and then generate the corresponding test cases including the test implementation. However, it is not as easy as it seems to be. Since they wanted to reproduce precisely the test design they had already, they struggled with the MBT tool (which could be any tool of this kind) and finally did not obtain the expected effort gains.

Why did they stick to the existing test design? They could have changed and written their MBT model from scratch, for example, starting from the requirements to avoid fixing themselves on an existing test design. The answer is simple.

Managing changes in an organization is a process. They initially thought that MBT is just like using a new test editor. But it is not. MBT impacts your test process, and you have to play the game to get the benefits.

(Bruno Legeard)

1.4.2 MBT Is Not for Free

Do you speak model? Probably not. Unlike a natural language, we do not learn modeling languages at school, at least not to the same extent. Even with a degree in computer science, you are not immediately able to construct a model exactly the way it should be for MBT, because that way strongly depends on the MBT tooling. You speak model, but with an accent, the tool cannot understand.

Changes need time. The most pessimistic prediction was given by Max Planck: "A new scientific truth does not triumph by convincing its opponents (…), but rather

because its opponents eventually die, and a new generation grows up that is familiar with it" [9]. You will need some time and possibly even some training to learn the handling of the modeling tool, the modeling language and the specific aspects the test case generator brings in. Moreover, you have to buy these tools. The initial costs represent a potentially high initial threshold. To convince your sponsor you should have an idea regarding expected costs and savings.

In the literature, you will find only few quantitative data evaluating MBT in industrial contexts and comparing the use of MBT with other testing approaches (see, e.g., Refs. [10, 11] in some embedded system domains). Many companies perform a pilot project to obtain those quantitative figures. Unfortunately, they are often reluctant to publish the results to avoid disseminating internal data. In addition, the figures strongly relate to a specific context and are only of limited use for other companies.

Therefore, we recommend that you perform your own pilot project. Define and record your own metrics to measure the efficiency of MBT in your context and compare them with data recorded in similar projects in your company. Of course, you need to ensure this similarity. Person-hours, number of requirements, types and number of features, and number of product variants are a good basis for comparison.

When comparing the costs, be careful about nonrecurrent cost (such as training cost) and manage other threats to validity of the comparison. If done thoroughly, you can succeed in obtaining reliable data for your own organization. They will help you to estimate MBT return on investment and to take a decision regarding MBT deployment. Chapter 10 in this book provides you with clear criteria for measuring return-on-invest, managing and evaluating the success of MBT introduction.

1.4.3 Models also Have Bugs

With appropriate MBT tooling, it is quite easy to react on changes such as late requirements or new information. You just change some part in the model and press a button. The test case generator will then propagate all changes to the test cases without creating additional effort. Unfortunately, the test case generator also propagates all errors within the model to the generated test cases.

You will realize that writing and maintaining a complex model is difficult, too. Once you introduce conditions that govern the generation of paths, writing an MBT model gets close to programming. Whenever you program something, you introduce bugs. Thus, we must literally "debug" our model.

Even without logical conditions, the model is not necessarily correct, just because it is a graphical representation as you can see in Figure 1.7. The model describes three possible states of a CD player: STOP, PLAY, and PAUSE. For simplicity, we neglected the states ON, OFF, and STANDBY. You can switch from STOP to PLAY, from PLAY to PAUSE – and vice versa. You can also switch directly from PAUSE to STOP, but the transition from STOP to PAUSE is wrong. The model describes a situation that does not correspond to reality. Based on the model, the test case generator produces erroneous test cases, which will definitely fail during execution.

Therefore, it is extremely important to review the model thoroughly. Most modeling tools provide features to verify the formal correctness of a model. Model

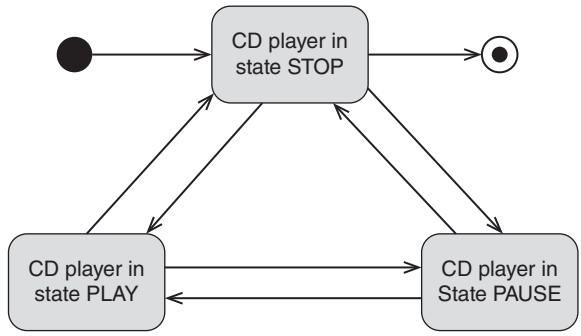

Figure 1.7 Example of an MBT model with bug (state diagram of a CD player; Note: In this figure, we use a different modeling language. Rounded rectangles represent states instead of actions (see Section 3.3)).

simulators support dynamic checks to validate the model. We deal with the question of MBT model quality in detail in Section 6.1.

1.4.4 MBT Can Lead to Test Case Explosion

A popular counter-argument against MBT is test case explosion. To understand this effect, look at Figure 1.8. It shows the data transfer with a messaging app. Four friends communicate in a chat room. There is no rule regarding the order of the chats received

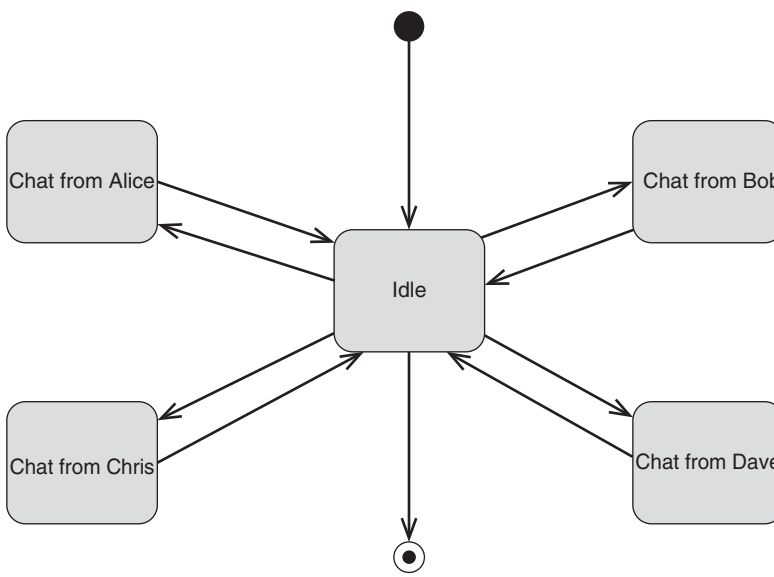

Figure 1.8 MBT model for testing the data transfer in a chat room (state diagram).

TABLE 1.3 Number of Possible Variations of the Messages in the Chat Room

Total Number of Chats	Number of Variations
0	$4^0 = 1$
1	$4^1 = 4$
2	$4^2 = 16$
3	$4^3 = 64$
4	$4^4 = 256$

Maximum Number of Chats Per Person	Number of Variations
0	1
1	65
2	7,365
3	Out of memory

by the system. Whoever wants to write may do it, even if it is several times the person in a row.

The model itself looks harmless, but the number of possible variations (meaning possible combinations of chats) increases exponentially with the total number of chats (see left hand-side of Table 1.3).

The situation becomes even worse if we decide to limit the number of chats per person rather than the total number of chats. In other words, each person may write 0 to n messages, n being the maximum number of messages we fixed. You can see the increasing number of variations at the right-hand side of Table 1.3. Between zero chats (system just idle) to a maximum of two chats per person, there are already three orders of magnitude. This is why we speak of "test case explosion." The table also shows the effect of the rapidly increasing number of test cases. Tools run out of memory and test manager despair.[2]

Technically speaking, limiting the total number of chats is equivalent to limiting the path length, whereas limiting the maximum number of chats per person corresponds to limiting the number of loops in the model.

Somehow, we have to limit the number of test cases and select a subset, which fits our test objectives. For example, with stress testing in mind, we may want to consider both extremely short and extremely long communications. Here, appropriate tool support is crucial. Still, do not expect the tool to do the thinking for you. The tool will provide you with some more or less intelligent algorithms to select the paths, but it cannot decide which paths are more important to test than others are.

Test case explosion definitely is an issue in MBT. The good news is that there are ways to avoid or at least reduce this effect. The test selection criteria presented in Section 8.1 are a powerful weapon against test case explosion, but it is also possible to adapt the MBT models, as discussed in Section 8.2. Figure 1.9 illustrates a third solution. Here, the MBT model itself is considerably simpler, but completed by external test data.

[2]For mathematical cracks who want to calculate the number of variations with maximum three chats, please contact us. We will send you the formulas.

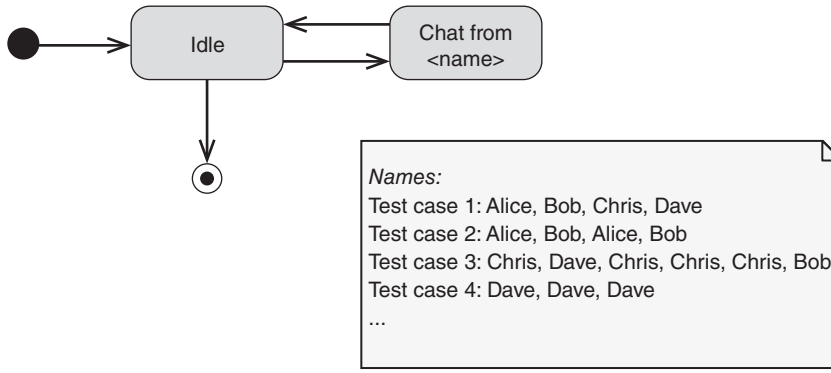

Figure 1.9 Limiting test case explosion by combining the model with external test data (state diagram).

1.5 WHAT CAN YOU REALISTICALLY EXPECT?

MBT divides the testing community. On one hand, we have a more or less well-founded reserve against a change in our established test approach. On the other hand, the expectation regarding increased efficiency and effectiveness of testing are high, and in some cases even disproportionate. The aim of this book is to give a realistic idea on what we can achieve with MBT, how we may do it, and where the limitations are.

We started using models in early 2000s and we are still so convinced of their advantages that we write a book on this topic. However, we must also admit that we went through the trough of disillusionment that is typical of technological hypes [12] (see Figure 1.10).

MBT is not a miraculous solution of all problems. Using models for testing purposes has many advantages, but it does not replace classic test design techniques. You have to know the traditional test design techniques such as boundary value analysis to be able to put them into an MBT model. Therefore, if your company struggles with those basic techniques, introducing MBT will not solve them. Still, modeling helps testers to "improve their understanding of the domain to perform tests more effectively and efficiently" [13].

Similarly, MBT is much more than buying a good tool. Of course, tooling plays a very important role in obtaining the full degree of efficiency MBT promises, but as the old saying goes: "A fool with a tool is still a fool." We discuss in Chapter 3 how MBT impacts the entire test process.

Last, but not least, MBT is not equivalent to complete test automation. If you currently execute your tests manually, it is not a good idea to switch from classic paper-based testing to MBT and from manual to automated tests in one step. As we discuss in Chapter 9, test automation is a challenge on its own.

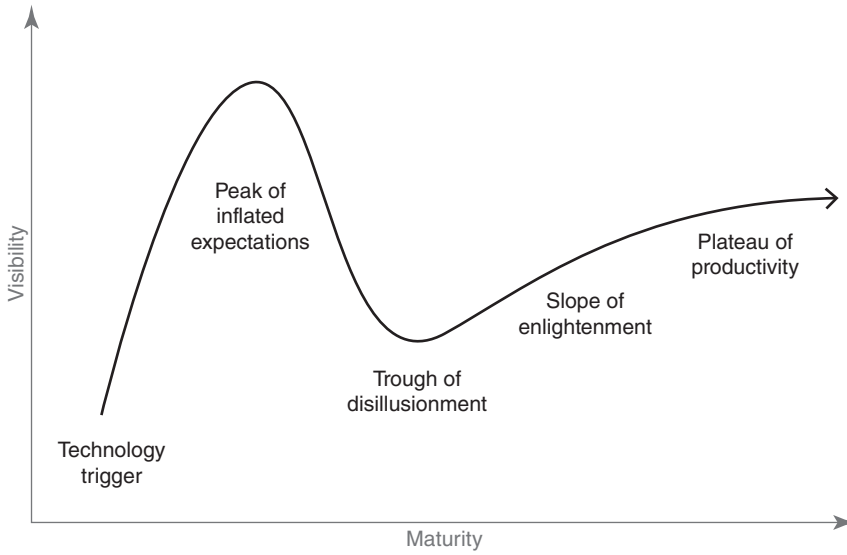

Figure 1.10 The Gartner "Hype Cycle" for technological innovations [12].

Thus, the decision to introduce MBT should be well thought out, based on clear goals and measurable objectives and, finally, receive full management support. Section 6.6 provides an overview on possible tool support for modeling activities, whereas Chapter 10 explains how to introduce MBT in your company. Introducing changes step-by-step with measurable (and monitored) intermediate goals is clearly recommended for any test process improvement.