4

Unit Level Testing

Unit level testing is used to verify the behavior of a single unit within a program. Herein, the unit which should be tested must be isolated from other units within the system in order to prevent interference of test results. In object-oriented programming, a single unit is typically a class or operation defined within a class. Typical programming errors which can be discovered during unit level tests include division by zero, wrong path setting, or incorrect pointer settings.

Black-box testing can be applied for all levels of testing. They only differ in the kind of *System Under Test* (SUT) they address. While component and integration testing is performed on several units of a system working together, and system testing refers to the whole system, unit level testing is performed on individual units within the system. In this chapter, we discuss the concerns of unit level testing, including how to use the UTP for unit level test specifications. The use of the profile is illustrated by means of the library example.

4.1 UTP and Unit Level Testing

Unit level tests can be derived from UML models. A unit in a UML model might be a single class or an operation. Black-box test is performed without the knowledge of the internal structure of the SUT. Thus, the prerequisite to unit testing the SUT using UML is that the tester must get access by calling public operations or by sending signals via an interface.

☞ Tip 1 Unit level test scope

For unit level testing, an SUT is an individual unit (e.g., a class or an operation) which is to be tested.

UTP Concept 1 SUT

SUT stands for system under test. For unit level testing, the SUT is an object or an operation which can be tested by calling operations or sending signals via public interfaces.

Before defining a test suite in UML, a new package for unit level testing needs to be created, and the system model must be imported to the test package in order to get access to the SUT. Figure 4.1 illustrates the package dependencies of the TTDTestingProfile package with predefined UTP concepts, the system model package LibrarySystem of the Library Example and its newly created test package LibraryUnitTest. The test model imports the system model in order to get access to the SUT during testing. To enable test specification, the test model needs import of the UTP concept package.

UTP Methodology Rule 1 Creating UTP test model

1. Create a new UML package for the unit level test.
2. Import the SUT system model package.
3. Import the package where UTP concepts are defined.

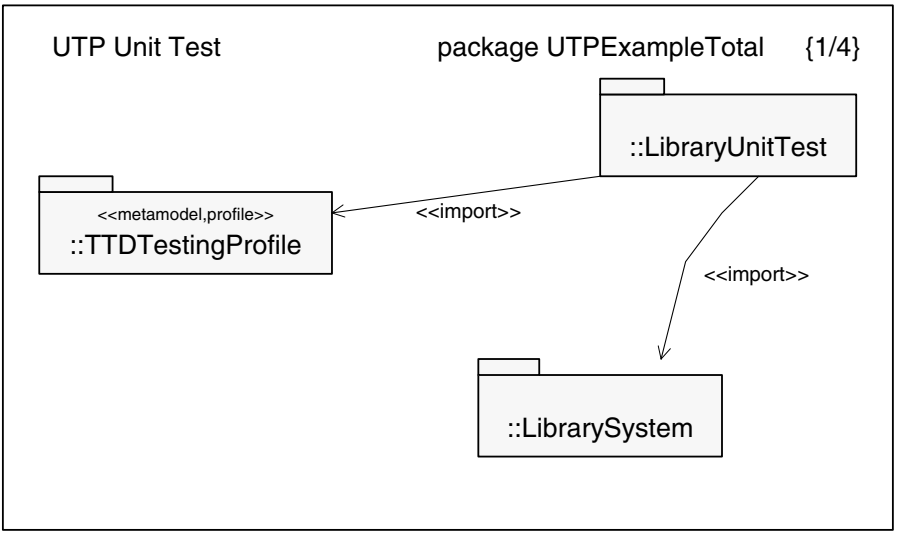

Fig. 4.1. Package imports

When using UTP, tests are performed within a given contex. This context is where test cases are defined, as well as the test configuration and related concepts. In a unit level UTP model, a test context is a stereotyped class which contains the unit level test cases as operations. In order to create a test context within the UTP model, a new UML class needs to be created and the UTP test context stereotype must be applied to the class.

UTP Concept 2 Test context

The test context is a stereotyped class that contains the test cases (as operations) and whose composite structure defines the test configuration. The classifier behavior of the test context provides the test control, which determines the order of execution of test cases.

UTP Methodology Rule 2 Define test context in a UTP test model.

To create a test context class for a unit level test, the following steps are needed:

1. *Define a test context class with stereotype <<Test Context>>.*
2. *List all the test cases as public operations in the test context class.*

Figure 4.2 shows the test context definition for the Library example. Inside Package LibraryUnitTest, a test context class called LibraryUnitTestContext is created and assigned with stereotype <<TestContext>>. This test context

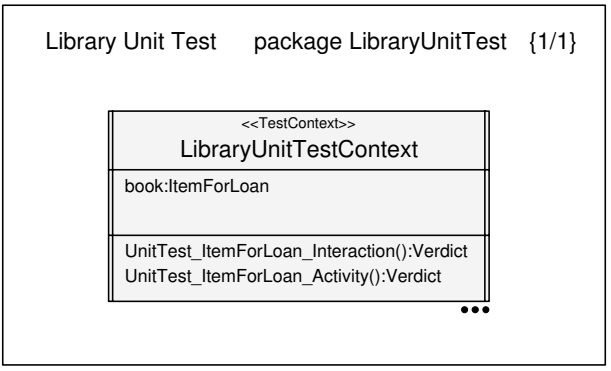

Fig. 4.2. Test context

class owns an attribute called book of class ItemForLoan and defines two test cases called UnitTest_ItemForLoan_Interaction() and UnitTest_ItemForLoan_Activity(). The concrete behavior specification of those two test cases are shown in other diagrams later in this chapter. Both test cases are public operations returning test verdicts as results.

As the names of the test cases already indicate, our unit level tests are performed on library items for loan. These operate on the concrete system specification related to the library items. These are part of the specification of the library system itself.

The concrete classes and signals that are part of the library system are available to the unit test package as they have been imported from the LibrarySystem package. Figure 4.3 shows the class and signal definitions of the library system. For our unit level testing example, the class ItemForLoan is the class to be tested. An Item in a library can be a Book, CD, DVD, or Video. It has several attributes, including one or more authors, a title, an ISBN number, and a publishing date. The ItemForLoan class represents an item that can be loaned to a library user. It adds the perspective that an item can be in either a good or a bad condition. If an item is in a bad condition, it needs to

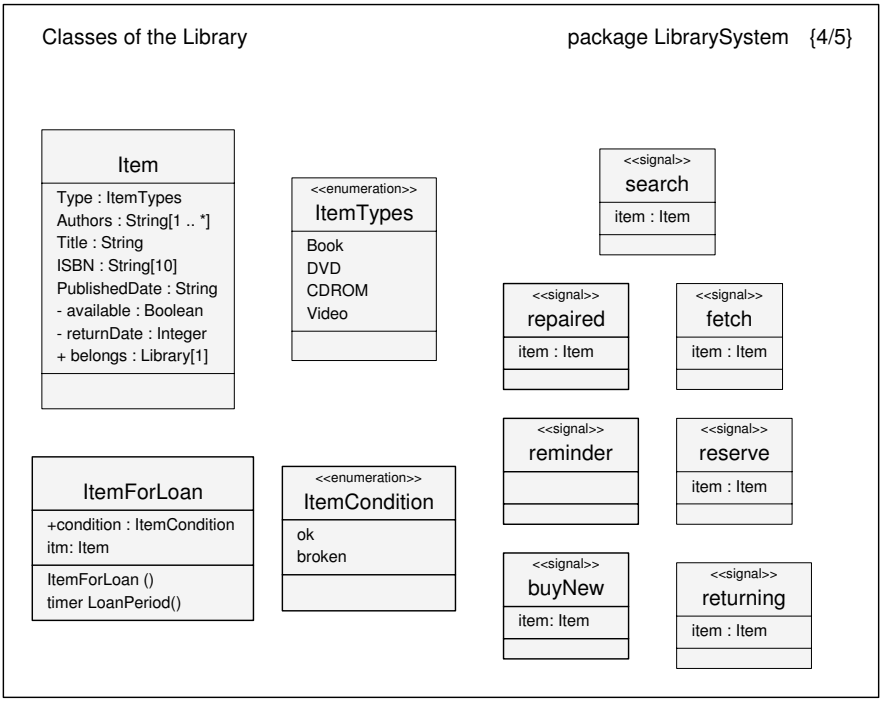

Fig. 4.3. Classes and signals in the Library System

be repaired before handed out again to another borrower. The signals shown in Figure 4.3 illustrate the ways that a tester can trigger various behaviors on the SUT.

4.1.1 State Machines

In order to generate test cases for the SUT ItemForLoan, we need a complete behavioral specification of the SUT class. UML state machines are the standard way to describe the behavior of a class.

☞ Tip 2 State machine adequacy

In order to derive useful test cases, the class behavior must be described. State machines provide good means for class behavior description. Test cases can be generated from the state machine specification, and the more precise and complete the state machines are, the better the derived test cases.

In our library example, class ItemForLoan is to be tested. Figure 4.4 illustrates its behavior specification by a State Machine. An ItemForLoan can exist in various states. In its initial state, it is available to any borrower. After it has been borrowed by a certain library user, it may be returned back in a good or broken condition. In latter case, the item will be repaired (toBeRepaired). In case that the item is still borrowed, but requested by another user, the item will be reserved for that user. When a reserved item is returned, depending on the condition of the item, it will be repaired (repairForReserver or availableForReserver). If it is available for the reserver, the reserver gets the item as soon as it is returned, and the item moves into the borrowed state. In cases where a broken item cannot be repaired, it will be replaced by a new version of the item.

4.1.2 Interactions

In UTP, a test scenario is described informally by a test objective, which is simply the textual description of the purpose of a test case. Test cases are the concrete realization of a set of behaviors which achieve the test objective. When using state machines for behavioral specification, test cases can be generated by simulating the behavior of the state machine. This simulation seeks to achieve a given coverage criteria (e.g., transition coverage, state coverage) [2] by ensuring that test data values are used to trigger the SUT in a way that helps achieve the coverage criteria.

In the library example, one test objective may be to *"verify that a broken item returned to the library is sent for repair."* To achieve this objective, concrete test cases with message flow between the SUT and the test system are created. This can be done using UML interaction diagrams or activity

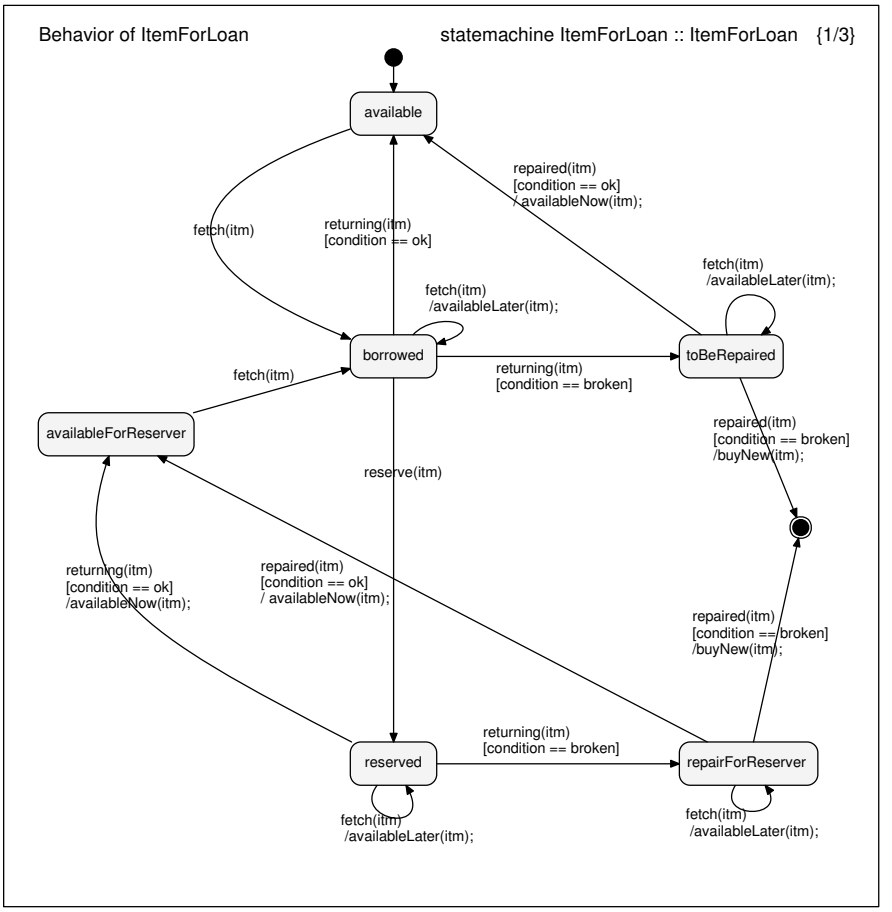

Fig. 4.4. State machine of item

UTP Concept 3 Test objective

A test objective describes the purpose of the test in an informal way.

diagrams to show communication between the test system and the SUT. For unit level test cases in a UTP model, the test context instance represents the test system.

Depending on predefined test strategies and coverage criteria (e.g., statement, branch, path, or other coverage criteria), the state machine of ItemForLoan can be simulated and the graph can be traversed in order to derive the test cases. In this example, we utilized *state coverage criteria*, which seeks to reach all of the states in the ItemForLoan state machine (Figure 4.4) at least

UTP Methodology Rule 3 Specifying test cases with interaction diagrams

For a unit-level test case, the test context instance can be used to trigger the SUT. Interaction diagrams provide good means for detailed test behavior specification. This is done using the following steps:

1. *Define a test case by creating a new interaction diagram.*
2. *Initiate the test case with an instance of the test context.*
3. *The test context instance creates the SUT instances and triggers it using operation calls or signals.*
4. *The order of these calls/signals is derived from the system model by simulation with an appropriate coverage criterion.*
5. *At the end of the test case, set the unit level testing verdict by returning a verdict value. For system derived test cases, this is typically* pass.

once. A single test case is enough to achieve this coverage criterion. Figure 4.5 shows the derived unit level test case called UnitTest_ItemForLoan_Interaction in an interaction diagram.

UTP Concept 4 Test case

A UTP test case concretizes a test objective by triggering inputs and observing outputs of the system. A test case always returns a test verdict.

The diagram shows in the beginning the test context instance TestLibrary-TestContext, which creates a new instance called book of class ItemForLoan. The item is available and will be fetched by the user. In its borrowed state, it can be reserved by a further user and when the first borrower returns the item in a broken condition, it gets repaired for and borrowed to the reserver. When the reserver returns the item in an irreparable condition, the item is replaced by a new exemplar. If this test case terminates successfully, all states of the ItemForLoan object will have been covered.

A test must deliver its result in a certain form to the external world. In UTP, they are in the form of test verdicts. Predefined verdicts in UTP include *pass, fail, inconclusive, and* error. The *pass verdict indicates that the SUT behaves correctly for the specified test behavior. The* fail verdict describes that the SUT does not behave according to the behavior. The *inconclusive verdict is used when the test neither passes nor fails, but is still valid according to the specification of the SUT. Finally, the* error verdict indicates an issue within the test system itself.

Test cases generated automatically from the system specification typically represent the expected correct behavior of the system. Thus, the test results

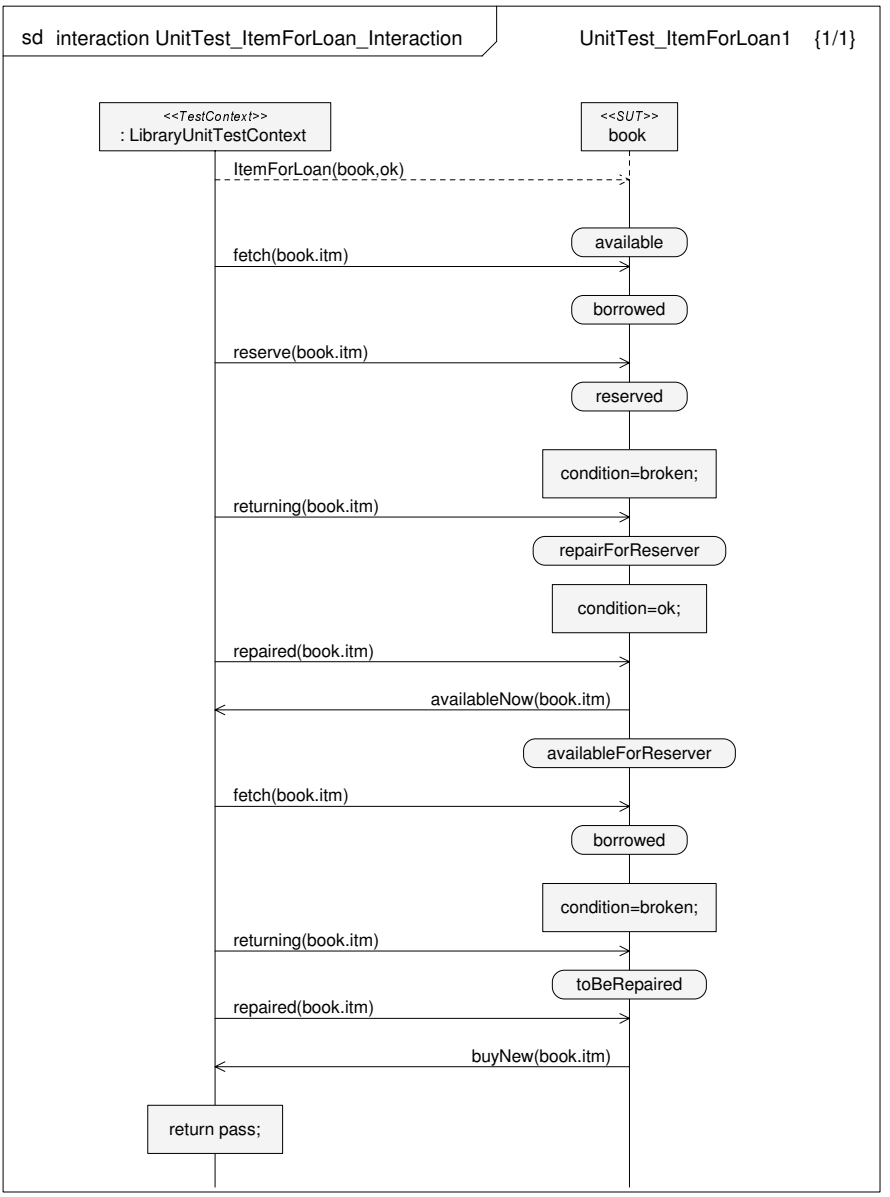

Fig. 4.5. Test case—interaction diagram

are usually set to *pass* at the end of these test cases. Accordingly, the verdict of test case UnitTest_ItemForLoan_Interaction in our library example is assigned to pass.

UTP Concept 5 Test verdicts

In UTP, each test case returns a verdict. Predefined verdicts are pass, fail, inconclusive*, and* error.

4.1.3 Activity Diagrams

Test cases can also be illustrated by UML activity diagrams. Activity diagrams provide a slightly different view of the system behavior than interaction diagrams. While the latter concentrates on the communication between the objects, an activity diagrams lays its focus more on the local behavior of an object.

Figure 4.6 shows a test case for the Library Example in an activity diagram, also derived from the state machine introduced in Figure 4.4. Here, the operation calls and message exchanges between the objects are not in the focus. Instead, the internal activities of each of the objects are illustrated.

UTP Methodology Rule 4 Specifying test cases with activity diagrams

For a unit-level test case, the test context instance can be used in order to trigger the SUT. Activity diagrams provide good means for detailed test behavior specification. To do so, the following steps are needed:

1. *Define a test case by creating a new activity diagram.*
2. *Initiate the test case with a test context instance by creating a partition for the test context in the activity diagram.*
3. *Parallely to test context, a partition for the SUT instance is needed.*
4. *Derive test behavior from the system model by simulation and applying coverage criteria.*
5. *At the end of the test case, set the unit level testing verdict by returning a verdict value. For test cases derived from the system specification, this is usually the* pass *value.*

Two partitions called LibraryUnitTestContext and ItemForLoan and their activity flows are shown in Figure 4.6. The transitions and state information in the state machine of ItemForLoan are transformed to activities in the diagram. Within the ItemForLoan partition, the availability of the library item is checked. Depending on its status, two different activity flows are traversed. In

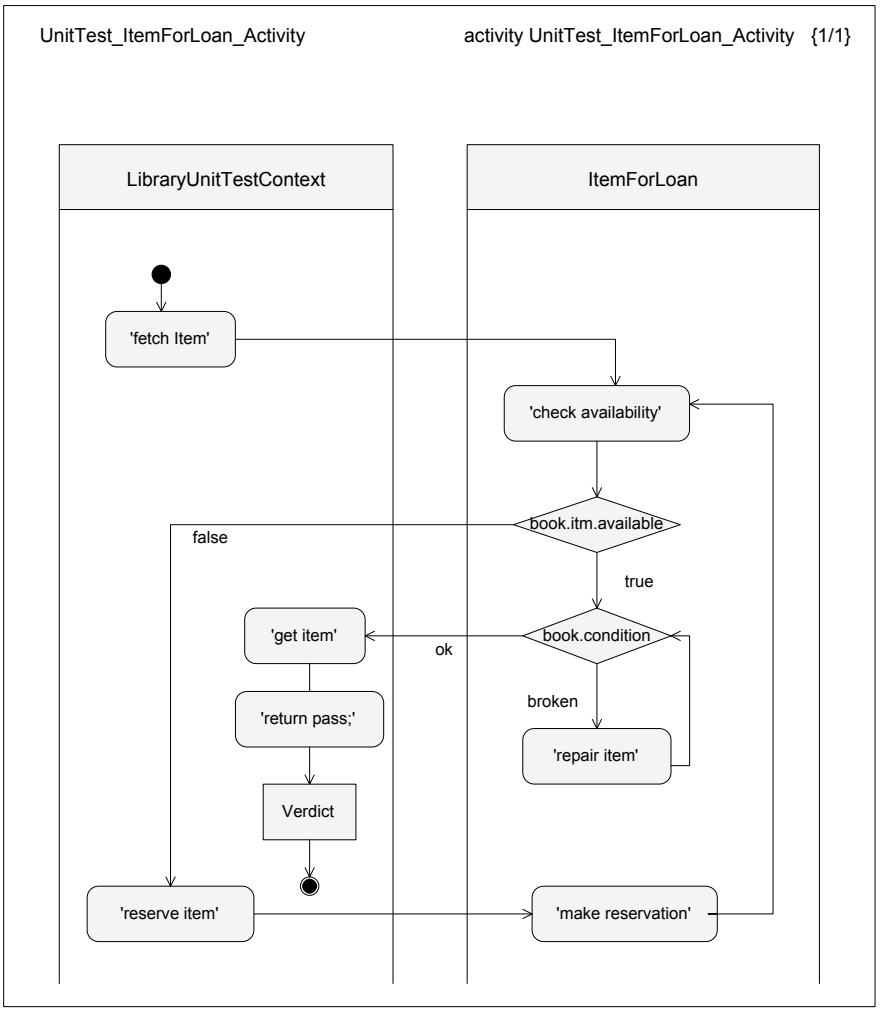

Fig. 4.6. Test case—activity diagram

this test case, the end of the activity diagram can only be reached if the book is in a good status and is available to another borrower. In this case, the test result will be set to pass.

4.2 Chapter Summary

In this chapter, we introduced several key UTP concepts, including SUT, test context, test cases, and predefined UTP test verdicts. These concepts can be used to model unit level tests (as well as other types of tests). We use state

machines as a basis for modeling the behavior of the SUT. We then illustrate how state machines can be simulated to allow the derivation of test cases. These are generated by traversing the graph using user-defined test strategies and coverage criteria. The test cases are modeled using behavior diagrams such as interaction diagrams (for focusing on communication between objects) or activity diagrams (for focusing on the local activity of the objects in the test case).

http://www.springer.com/978-3-540-72562-6

Model-Driven Testing
Using the UML Testing Profile
Baker, P.; Dai, Z.R.; Grabowski, J.; Schieferdecker, I.;
Williams, C.
2008, XIV, 184 p. 94 illus., Hardcover
ISBN: 978-3-540-72562-6