SYSTEM TEST CASES FROM USE CASES

Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, Jesús Torres
University of Sevilla, Avd. Reina Mercedes sn. Sevilla, Spain
{javierj, escalona, risoto, jtorres}@lsi.us.es

Keywords: System testing, use cases, generation of test cases.

Abstract: Use cases have become a widely used technique to define the functionality of a software system. This paper describes a new, formal and systematic approach for generating system test cases from use cases. This process has been designed specially for testing the system from the point of view of the actors, through it graphical user interfaces.

1 INTRODUCTION

System testing is a black-box technique which verifies the satisfaction of the requirements of the system under test (SUT) (Burnstein, 2003). Early testing is the generation of test cases in early development phases. This is not a new idea. Two surveys (Denger, 2003) and (Gutiérrez, 2004) (22 different approaches in total) expose that there are many lacks in the exiting approaches. One lack is the absence of a formal process and the absence of free available tools. Another lack is that approaches are not complete; this means that they describe how to generate partial test cases, mainly test actions, without describing other important elements such as test data, expected result, executable test scripts, or test coverage.

In a previous work it was described how to generate test cases from use cases for web application using existing approaches (Gutiérrez, 2005). This paper tries to resolve both lacks offering a formal approach for obtain executable test scripts from use cases. It has been specially designed to be used in early development phases. It also uses UML and UML Testing Profile (OMG, 2002) (called UMLTP from now). Related works may be found in (Denger, 2003) and (Gutiérrez, 2004).

2 A PROCESS TO GENERATE TEST CASES FROM USE CASES

This test process is focused on testing use cases whose principal actor is human. A test case is composed of three elements: test action, test values and expected results. Test actions are the actions developed by the test case over the system under test (SUT). Test values are the information needed by the test case. Expected results are the responses of the system that allows evaluating whether the test is satisfied or failed. The results for this process are: test objectives, a set of test cases to verify each objective and test scripts. Test cases are expressed using models and graphical notation defined in UMLTP when possible.

2.1 Testing Models

The models used to store the information about test cases are: test objective model, test data model, interface model and event model.

1. Test objective models.
A test objective is an element named according to the description of what should be tested. The UMLTP does not define any notation to represent test objectives. Thus, we use activity diagrams.

A test objective is a path through the activity diagram. Test objectives might be automatically extracted from the activity diagram applying a coverage criterion, like all-edges and all-transitions. Every test objective will have at least one test case to verify it. An example is shown in table 4.

2. Test data model.
Test data model describes the structure and values of the test data. The first task is to identify operational variables (or simply variables) of a use case. An operational variable is an explicit input or output, an environmental condition or a representation of the

SUT (Binder, 1999). The domain of every variable is divided into data partitions. UMLTP uses class diagrams and stereotypes to describe the hierarchy of data partitions. After that, test values are generated for every data partition. Case study shows an example of data structures, partitions and test values in figure 4.

3. Interface model.
Our aim is to test the functionality throughout a graphical interface, not to test the graphical interface itself. The objective of this model is to describe the interface used for the test case to interact with the system. Since this process has been designed to be applied in early development phases, this model represents a high abstract description of the GUI. UML Testing Profile, and UML in general, does not include any specific notation for GUI, so it will be used class and object diagrams to represent the components and states of a GUI.

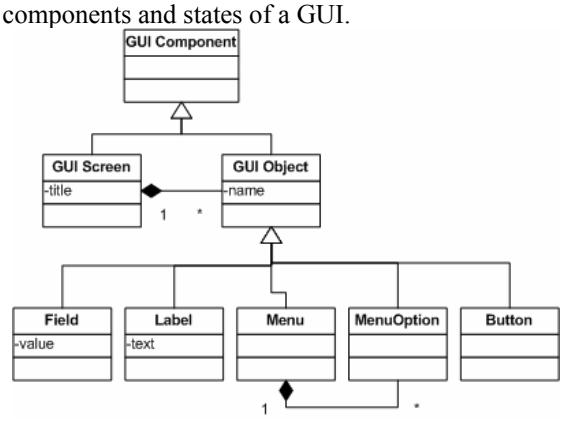

Figure 1: Example of components for interface models.

Figure 1 shows a class diagram with some elements from an interface model.

4. Event model.
Frequently, actor activities from a test objective are too abstract to be directly translated into a test script. It is proposed to build up an event model to address to this complexity. A set of events describes how to perform actor activities identified in the test objective model. If an activity needs to supply information to the system, this information should have been defined in the test data model. This paper introduces a simple set of messages to express events. These messages are listed in table 1. Due to their simplicity, the messages might be easily extended. Event model also includes an assert message (table 1). This message is sent by the test case to itself to verify an attribute of the GUI. This message allows codifying the expected results into a test script, an example is shown in case study.

Table 1: Messages for event model.

Message	Meaning
ClickOn(component)	Perform a one-click event over the indicated GUI component.
Screen(screen)	Search for the indicate GUI screen and set the focus over it.
SetField(field, value)	Set the indicated value into the field object.
Assert(component. attribute, value)	Verify that the attribute of the component indicated matches with the value.

2.2 Steps to Generate Test Cases

We suggest a process of six steps to generate test models and to obtain executable test scripts. These steps are shown in the activity diagram in figure 2 and described in the following paragraphs.

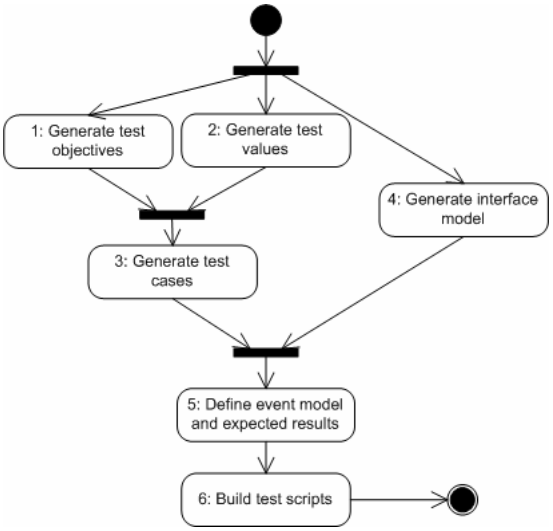

Figure 2: Activities to generate test cases.

The first step is to build up a test objective model from a use case, as described in point 2.1. The second one is to build up the test data model as described in point 2.1. In the third step, test cases are generated combining test objectives with test values. The number of test cases is determined by the test objectives and the different partitions for the variables involved in that test objective. In the fourth step, interface model is generated, as described in point 2.1. In the fifth step event model is generated. Finally, event messages, test values and assertions are translated into test scripts, completed with test harness (Binder, 1999) and executed over the real SUT.

3 CASE STUDY

This section applies the process described in section 2 over a real system to generate test cases. The system under test is an implementation of a classic notepad. The use case selected to generate test cases is Open File (table 2).

Table 2: Template for use case "Open File".

Description	Load document from file
Precondition	No
Main scenario	1 User select "Open file" option. 2 System asks for the file to open. 3 User selects a file. 4 System loads the file and shows the document.
Alternative / errors	3 User may cancel the loading operation at any time. 4 If file does not exist or there is an error, system shows an error message.
Post condition	No.

A full coverage for the use case is selected. This means that at least one test case for every identified test objective will be generated.

3.1 Generation of Test Objectives

First of all, the test objective model is built (figure 3). Activities 01 and 03 are developed by the user and activities 02, 04, 04.1 and 04.2 are performed by the system. Step 4 (table 4) has been divided into activities 04, 04.1 and 04.2, and due to their results they may be different if there is an error when opening the file.

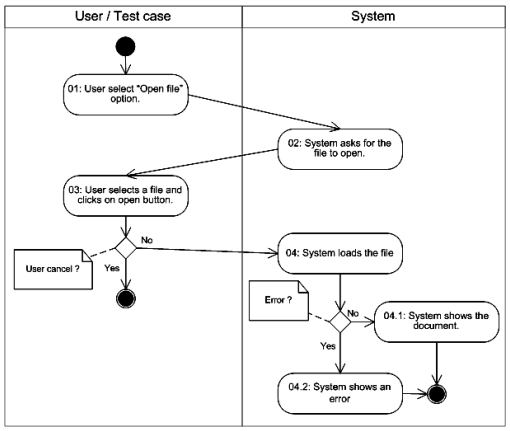

Figure 3: Test objective model.

Table 4 shows test objectives obtained by traversing figure 3.

3.2 Generation of Test Values

Firstly the variables involved in the use case are identified. Test objective model in figure 3 reveals that there are, at least, two variables (the same number as decision nodes). Variables and domains are resumed in table 3.

Table 3: Variables and domains.

Variable	Domain
User-Option	Options available for the user: load file or cancel.
File	File to open

User-Option is a variable of an enumerated type. However, File is a variable of a complex type. For the testing purpose there must be known, at least, the name of the file, its content and its attributes.

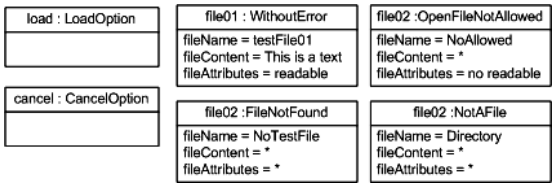

Figure 4: Test data.

Now, we divide the domains into data partitions. Finally, at least one test value is generated for each partition. Object diagram in figure 4 shows tests values.

3.3 Build Test Cases

A test case is a test objective with a concrete value for its variables. Variables and their partitions are added to the test objectives, as shown in table 4.

Table 4: Test objectives with variables and partitions.

	Test objectives
1	01, 02, 03(f01: Without-Errors, op01:LoadOption), 04, 04.1
2	01, 02, 03(f02: With-Errors, op02:LoadOption), 04, 04.1
3	01, 02, 03(f03: *, op03:CancelOption), 04, 04.1

3.4 Generate Interface Model

It is assumed that the system under test is not built yet. So, it is generated an abstract description of the user interface with the minimum set of components to perform the use case.

Studying the use case, we realize that there are three screens involved: the main screen, where user actor clicks on open option, the file selection screen,

where user actor selects the file to open, and the error screen where system shows the error message, if any.

The interface model is shown in the object diagram in figure 5.

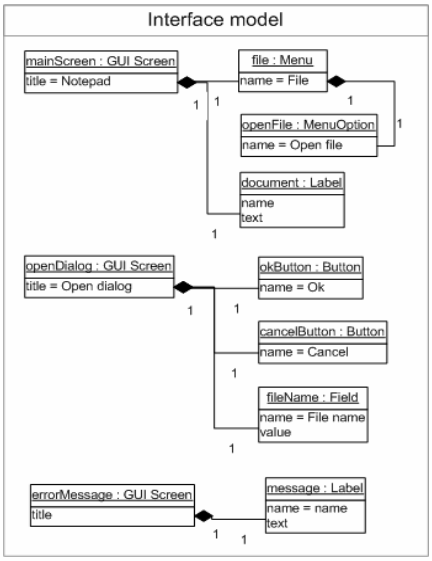

Figure 5: Interface model.

3.5 Generate Event Models and Expected Results

First, each user activity is refined using messages listed in table 1 and the user interface, defined in point 3.4. The event model to verify the main scenario is described as UML sequence diagram in figure 6.

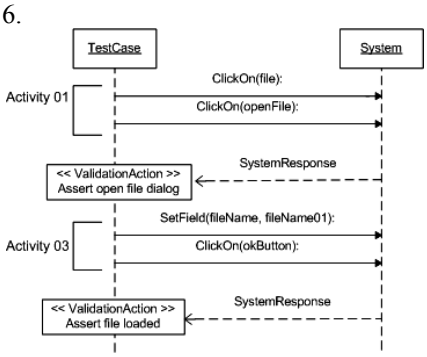

Figure 6: Event model.

Validation actions are implemented using the assert proposition shown in table 1 and activities diagrams as proposed in the UMLTP. Due their simplicity, the have been omitted.

The process has ended. There have been generated test actions (shown in the event model), test values (shown in the test value model) and expected results (shown in the event model too) that commits our test objectives (shown in the test objective model). Up to now, we have not needed the design or the code of the system.

3.6 Building Test Scripts

The information obtained in the points before, might be automatically translated into executable test scripts. Details of the implementation and test tool are needed to perform this step. It is used a real implementation of the notepad, called Stylepad, to generate test scripts. The Stylepad is distributed in the Java Developer Kit. It has been used an open source tool called Abbot to codify executable scripts.

4 CONCLUSIONS

This paper has shown a process for the early-testing of use cases. Although this process has been designed to test use cases from the perspective of human actors, it can be also used to test other actors. This process can be applied in early development stages. In fact, in case study described in section 3, all test cases have been generated before choosing a real implementation to test.

REFERENCES

Binder R.V. 1999. T*esting Object-Oriented Systems*. Addison-Wesley. USA.

Burnstein, I. 2003. *Practical software Testing*. Springer Professional Computing. USA.

Denger, C. Medina M. 2003. *Test Case Derived from Requirement Specifications*. Fraunhofer IESE Report.

Escalona M.J. 2004. *Models and Techniques for the Specification and Analysis of Navigation in Software Systems*. Ph. European Thesis. Department of Computer Language and Systems. University of Seville. Seville, Spain.

Gutiérrez, J.J., Escalona M.J., Mejías M., Torres, J. 2004. Comparative Analysis of Methodological Proposes to Systematic Generation of System Test Cases. *3º Workshop on System Testing and Validation*. Paris. France.

Gutiérrez J.J., Escalona M.J., Mejías M., Torres J. 2005. A practical approach of Web System Testing. *Advances in Information Systems Development*. Ed. Springer Verlag Karlstad, Sweeden.

Object Management Group. 2002. *The UML 2.0 Testing Profile*. www.omg.org