Tomasz Straszak, Michał Śmiałek
Institute of Theory of Electrical Eng., Measurement and Information Systems
Faculty of Electrical Engineering, Warsaw University of Technology
t.straszak@iem.pw.edu.pl, smialek@iem.pw.edu.pl

Acceptance test generation based on detailed use case models

Abstract

Tests performed in order to verify compliance of a software system with customer expectations cover different areas. Some of them verify the functionality, other – the business domain logic, the non-functional characteristics or the user interface. Usually they are done separately, but on the same functional areas. This paper presents the concept for the Requirements Driven Software Testing (ReDSeT) tool, which allows for automatic integrated test generation based on different types of requirements. Tests are expressed in newly introduced Test Specification Language (TSL). The basis for functional test generation are detailed use case models. Furthermore, by combining different types of requirements, relations between tests are created. The constructed tool acknowledges validity of the presented concept.

Keywords: acceptance tests, test generation, use cases, requirements, model based testing

Introduction

The main goal of a software development project is to deliver a software product that meets the expectations of the customer. Verification of compliance with the requirements of the stakeholders is possible by carrying out acceptance tests [1]. Acceptance testing is the process of comparing the system under development to its requirements and needs of its users. These tests are performed usually by the customer through comparing the system's operation to the original contract between the stakeholders and the developers. This contract should be understandable for the stakeholders and at the same time precise enough for the developers to produce efficient software.

To describe the expected functionality of the software system, use cases are commonly used [2]. Use cases describe interactions between external actors and the system, which lead to specific goals according to the given scenarios. Such requirements are supposed to be satisfactory to define the tests, that will be used during acceptance testing.

To improve the development of tests from use cases, a number of automatic test generation mechanisms were proposed. Examples of such approaches can be found in work by El-Attar and Miller[3], Gutiérrez et al. [4], and Nebut et al. [5]. Beside use cases, requirements specifications contain other types of requirements, that describe different aspects of the desired software. These requirements also should be verified by executing corresponding tests. Some work has been done on the generation of tests based on business rules (see Junior et al. [6]), GUI requirements (see Bertolini and Mota [7]), and even on non-functional requirements (see Dyrkom and Wathne [8]).

All these mechanisms use model transformation forming the area of *Model-based testing (MBT)*, which is an evolving technique for generating a suite of test cases from requirements [9]. Although different types of tests are generated from requirement models describing the same software system usually they are not related, because they verifies different aspects of the system.

This article describes the idea of automatic generation of different types of tests integrated in functional test cases and test scenarios executed during acceptance testing. These tests are generated on the basis of interrelated requirements describing many aspects of the developed software system, which makes this idea MBT compliant. The element that integrates different types of testing is the functional test case corresponding to the use case scenario as shown in Fig. X. 1.

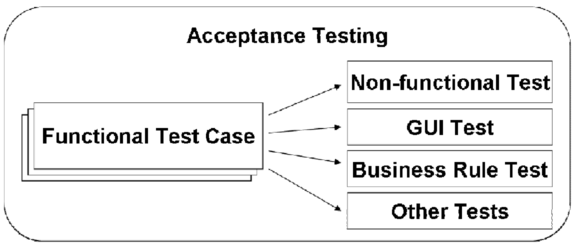


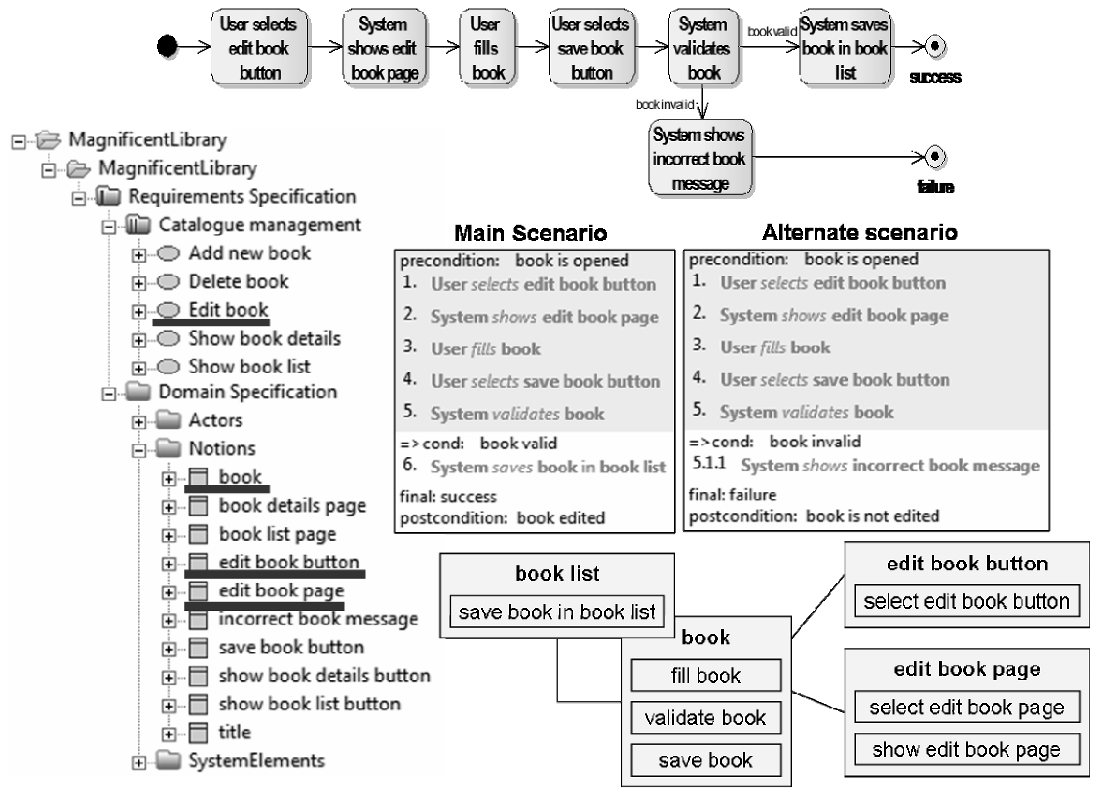
Fig. X. 1. Acceptance test suite based on functional test cases

This concept is based on the test metamodel defined as the Test Specification Language (TSL) and implemented within the ReDSeT tool (Requirements Driven Software Testing). The tests are generated automatically based on the requirements specification created with RSL (Requirements Specification Language) [10]. As RSL gives a notation for precise use case scenarios, generation of test cases verifying the system behaviour is significantly facilitated. Additional information contained in scenario sentences (notions from the domain vocabulary) and other related requirements allows for generation of tests of different types. All the tests generated on the basis of RSL-based requirements form a complete test suite for acceptance testing.

1.1. Detailed requirements expressed in RSL

As in other test generation solutions, the basis for automatic generation of tests is the precise specification of requirements. As mentioned above, the described solution is based on the requirements specification created with RSL. The main features of this language are: clear separation of descriptions of the system's behaviour and descriptions of the system's domain. Functional requirements can be presented in three equivalent forms: structured text with hyperlinks to domain elements, activity diagram and sequence diagram. It allows for precise specification of requirements, which is understandable even for ordinary people who do not have technical expertise. The language has a precise specification of its syntax and semantics [10] with methods of its use explained e.g. by Śmiałek et al. [11]. Fig. X. 2 shows an example requirements specification, created in RSL.

All the elements of a requirement specification are grouped in packages in a tree structure. Simple requirements described with the free text can be used to define business rules or non-functional aspects of the system. Use cases describing the functionality of the system are described with structured scenarios. Scenarios are consist of numbered sentences in a simple grammar SVO(O). These sentences are constructed with notions stored in the domain vocabulary. This is illustrated with two scenarios (main and alternative) of the *Edit book* use case. The same information is presented in the form of an activity diagram that is generated automatically from the scenarios.



Main Scenario

precondition: book is opened
1. User *selects* edit book button
2. System *shows* edit book page
3. User *fills* book
4. User *selects* save book button
5. System *validates* book

=> cond: book valid
6. System *saves* book in book list

final: success
postcondition: book edited

Alternate scenario

precondition: book is opened
1. User *selects* edit book button
2. System *shows* edit book page
3. User *fills* book
4. User *selects* save book button
5. System *validates* book

=> cond: book invalid
5.1.1 System *shows* incorrect book message

final: failure
postcondition: book is not edited

Fig. X. 2. Example of detailed use cases expressed RSL

The notions are referred to in scenario sentences through hyperlinks (*book, book list, edit book button, edit book page*) and are presented on a notion diagram, that is similar to a class diagram. The relationships between notions, and notion operations are defined automatically according to the scenario sentences where these notions appear or are defined manually by the requirements engineer. The notions and their operations used in use case scenarios describe the business logic and the user interface elements.

All the requirements can be related. To depict relations between use cases, a special *invoke* relationship is used. It allows to determine under what conditions and in which step of a use case scenario another use case is to be called (see Śmiałek et al. [11] for more details).

RSL is based on a formal metamodel. This allows for automatic processing of information contained in the requirements specification. We will use this characteristics of RSL for generating test cases.

1.2. Automating test generation

To define acceptance test suite and to ensure accurate and automatic transition from RSL-based requirements to tests, Test Specification Language (TSL) was developed. This language is based on a metamodel defined in EMF (Eclipse Modeling Framework) [12] and is out of scope of this article.

The main idea of TSL is to provide the notation for reusable tests, that are understandable for non technical people and precise enough for detailed verification of the software system. All tests are grouped in a tree structure, named the Test Specification (see Fig. X. 3), that groups tests assigned to a specific release of software. Each test contained in the test specification represents a procedure for software verification for a single requirement. Such a verification is made by examining all the check points defined inside a test.

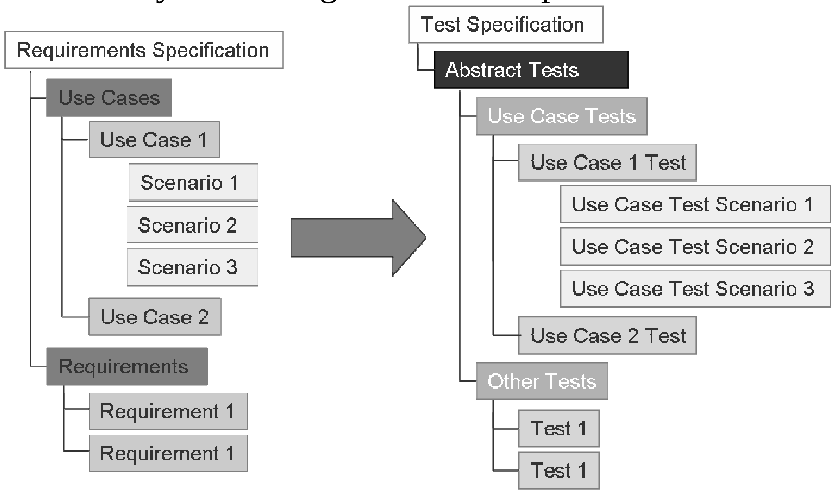


Fig. X. 3. Test generation based on the requirements specification

The basic structure of a TSL test specification consists of two packages: Abstract Tests and Concrete Tests. The first of these includes tests generated directly from the requirements specification: mostly Use Case Tests but also other related tests of other types. A use case test corresponds to a use case, and includes test scenarios, as shown in Fig. X. 3.

A use case test scenario includes the initial condition (a precondition sentence) that must be met before the execution of actions described in this scenario and the final condition (a postcondition sentence) that describes the desired state of the system after the scenario is executed.

Every use case test scenario, generated from an RSL use case scenario, is a sequence of actions forming a dialogue between the primary actor and the system. Every such action is expressed by a single sentence in a simple subject-verb-object (SVO) grammar (see Graham [13] for an original idea). These sentences describing single actions can have check points assigned. In addition to action sentences, two additional sentence types were introduced: condition and control sentences. They are used in a scenario to express the flow of control between alternative scenarios of the same use case as well as between scenarios of different use cases (see [14]).

An important feature of the requirements specified with RSL, is the possibility to create relationships between requirements. Due to generation of test specifications on the basis of these requirements, relationships between tests are also created. Relations binding use cases depicted as *invoke* are transferred to become relations between use case tests. This brings information from which step of the use case test scenario and under what conditions, a scenario of another use case test should be called. Other requirements' relationships are transferred to relationships binding other tests than use case tests.

1.3. Instantiating concrete tests

A scenario of a use case test determines the conditions, steps and check points that will be subject to verification for the use case implementation. Such algorithms will be used in acceptance testing after placing them in test scenarios and assigning specific test data values.

Test scenarios are grouped by the second package in the basic structure of the test specification (see Fig. X. 4). They are defined by a test engineer as a set of ordered instances of use case test scenarios, that are named functional test cases. Functional test case is composed of ordered steps in form of SVO sentences. Each step can contain check points with assigned test data values and can be related with test cases of other types. Test cases of other types are automatically created during instantiation. They are related with functional test case the same as other tests are related with use case test scenarios and particular scenario sentences.

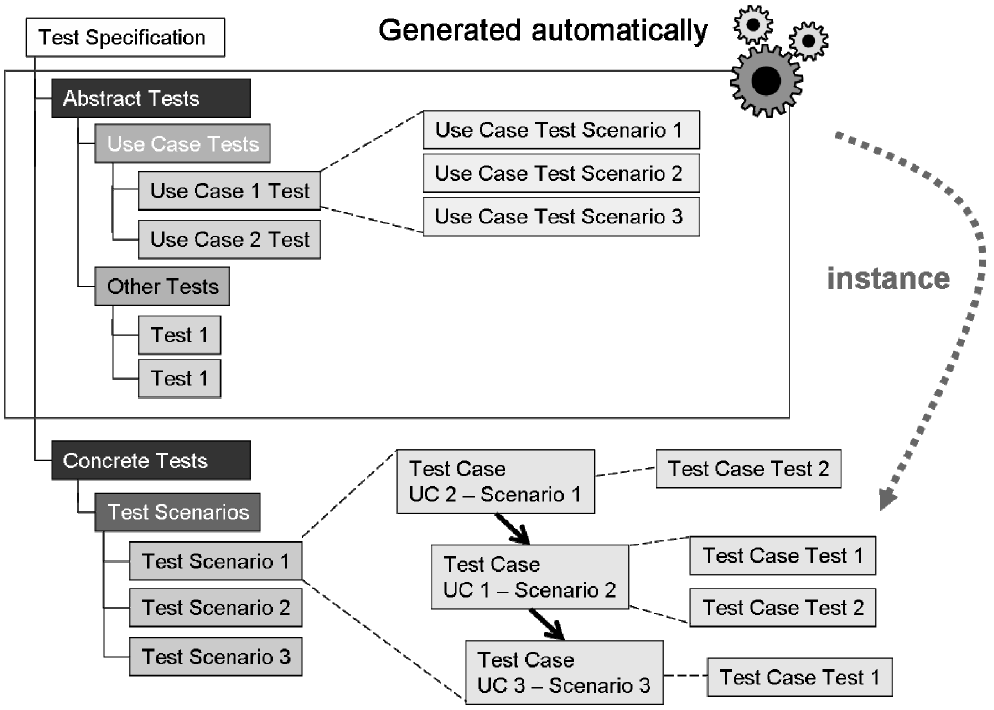


Fig. X. 4. Test scenarios composed of concrete test cases

A test scenario constructed with test cases builds also the context for the test data. The initial test data values are set by the test engineer as the precondition values of the test scenario. Test data values describe basic business objects as well as GUI elements. The test data in the scope of one test scenario are passed between test cases as its' precondition and postcondition values. Test data values are changing according to the functionality and the business logic that are under tests. Although the test cases cannot be formally related to each other, within the test scenarios they simulate business processes that are implemented in the system being tested.

Particular use case test scenarios, can be instantiated as test cases in the scope of other test scenarios with different test values. It makes the tests reusable.

The described above idea of creating test cases allows for verifying of system’s application logic. Use case test cases form the skeleton of a test specification. This can be supplemented with other types of testing, such as business logic, GUI or non-functional tests, depending on the content of the requirements specification. The extension of the TSL metamodel allows to attach to the use case test scenario steps other specialized tests. Such a test specification would be complete for acceptance testing.

1.4. Tool support

The tool supporting the described idea of automatic test generation based on requirements is called ReDSeT (Driver Software Testing Requirements). It is based on the Eclipse Rich Client Platform. This enables integration with ReDSeeDS tool (www.redseeds.eu [15]) which provides advanced editors for requirements (for use cases, notions and other requirements) described with RSL. The generated test specification can be included in the same Eclipse project as the requirements specification and code. This enables integration of activities at different stages of the software development project.

ReDSeT tool perspective is divided into several areas. The Test Specification Browser allows to manage the test specification, which is organised in a tree structure. The current test cases and the test scenarios are presented in the Test Editor area. The Detailed Test View is dedicated for viewing check points and editing test values contained in all the types of tests.

The tool does not allow to edit the tests generated automatically based on the requirements. It is expected that all the necessary information about the check points will be derived from the specification requirements. If there is a need to clarify the test, the requirements should be modified and then the test specification should be regenerated. This assures strict compliance of the tests with the requirements.

The ReDSeT tool is designed as a set of integrated plug-ins, which are responsible for: automatic generation of tests, test management, use case tests viewer, test scenario editor for composing sequence of functional test cases and editors for test data values. Test viewers and test data value editors for tests of other types may be attached to the tool as the additional plug-ins.
The repository of the test specification is based on the EMF technology. The TSL metamodel can be easily extended to handle other types of tests which are adapted to different types of requirements associated with use cases. As the repository of the test specification XML file is used. It gives the technical capabilities for easy extraction of test scripts for acceptance tests execution.

Conclusions

The proposed idea and the ReDSeT tool bring a complete solution for creating acceptance testing for the systems that are focused on user-system interaction. The basis for creation of a set of test scenarios are detailed use cases. Requirements defined in RSL significantly facilitate automatic test generation, and TSL allows for expressing interrelated tests of different types in a way that is comprehensible to the audience responsible for acceptance testing.

The automatically generated tests can be re-used in various test scenarios. The work on the preparation of the test specification can begin during the requirements formulation stage, and regeneration of tests allows to reach test complexity corresponding to detail level of final requirements.

It can be noted that the proposed method is based on black box testing and is independent of the implementation technology of the system under test. Since RSL and TSL are based on the metamodels, the whole idea is close to Model Based Testing. Traces from requirements to test cases are planned to be used for generating requirements with tests coverage reports. These traces will be subject to further research on regression test selection.

Relying TSL on the EMF-compliant metamodel and constructing the ReDSeT tool as an Eclipse plug-ins allows for easy extension of the described solution. To support another types of tests, the metamodel and appropriate editor plug-ins should be developed. In the future it is planned to extend the solution with detailed tests for the business logic and graphical user interface.

It is also planned to extend the tool with the mechanism for extracting of test scripts as input for tools that automates test execution (e.g. IBM Rational Functional Tester, Selenium). It would bring a complete solution for detailed use case based testing.

Bibliography

1. Myers G. J., Sandler C. and Badgett T.: *The Art of Software Testing*. Wiley Publishing, 3rd edition, 2011
2. Cockburn A.:. *Writing Effective Use Cases*. Addison-Wesley, 2000.

3. El-Attar M. and Miller J.: *Developing comprehensive acceptance tests from use cases and robustness diagrams*, Requirements Engineering, 15(3):285–306, September 2010
4. Gutiérrez J. et al.: *An approach to generate test cases from use cases*, ACM, Proc. ICWE ’06, p. 113–114, New York, NY, USA, 2006
5. Nebut C. et al.: *Automatic test generation: A use case driven approach*, IEEE Transactions on Software Eng., 32:140–155, 2006
6. Mendes Bizerra Junior E. et al.: *A method for generation of tests instances of models from business rules expressed in ocl*, IEEE Latin America Transactions, 10(5):2105–2111, 2012
7. Bertolini C. and Mota A.: *A framework for gui testing based on use case design*, IEEE Computer Society, Proc. ICSTW ’10, pages 252–259, Washington, DC, USA, 2010
8. Dyrkorn K. and Wathne F.: *Automated testing of non-functional requirements*, ACM, Companion to the 23rd ACM SIGPLAN conf. OOPSLA Companion ’08, pp. 719–720, New York, NY, USA, 2008
9. Dalal S. R. et al.: *Model-based testing in practice*, ACM, Proc. ICSE ’99, pages 285–294, New York, NY, USA, 1999
10. Kaindl H. et al.: *Requirements specification language definition*, Project Deliverable D2.4.2, ReDSeeDS Project, www.redseeds.eu, 2009
11. Śmiałek M. et al.: *Introducing a unified requirements specification language*, Proc. CEE-SET’2007, Software Engineering in Progress, pages 172–183. Nakom, 2007
12. Steinberg D., Budinsky F., Paternostro M. and Merks E:. *EMF: Eclipse Modeling Framework 2.0*, Addison-Wesley Professional, 2nd edition, 2009
13. Graham I. M.: *Task Scripts, Use Cases and Scenarios in Object-Oriented Analysis*, Object-Oriented Systems, 3(3):123–142, 1996
14. Śmiałek M. et al.:: *Complementary use case scenario representations based on domain vocabularies*, Lecture Notes in Computer Science, MODELS’07 4735:544–558, 2007
15. Śmiałek M. and Straszak T.: *Facilitating transition from requirements to code with the ReDSeeDS tool*, IEEE, Requirements Engineering Conference (RE), 2012 20th IEEE International, pp. 321–322., 2012