2012 International Conference on Software and Computer Applications (ICSCA 2012)
IPCSIT vol. 41 (2012) © (2012) IACSIT Press, Singapore

Test Requirement Model: as a Bridge Connecting SUT and Test Design

Mei Tian, Ji Wu, RuiLong Gong [+]

Dept. of Computer Science and Technology
Beijing University of Aeronautics and Astronautics
Beijing, P.R.China

Abstract. Test requirement is the one that specifies the tasks a test system needs to do and the corresponding conditions or capabilities to meet or to provide when it implements the test. In this paper, we propose a general-purpose test requirement modeling language under the context of model driven engineering. Three components of test requirement are identified and carefully defined in the meta-model, i.e. test object, test purpose and test constraint. To facilitate the test requirement model construction, four views on SUT structure and behavior, test purpose and test constraint are defined based on UML. To validate the approach of the proposed test requirement model, a case study is carried and discussed in this paper.

Keywords: model driven engineering, test requirement, test object, test purpose, test constraint.

1. Introduction

In recent years, as software system increases in size and complexity, the task of testing such system becomes increasingly complex. First, the work is heavy, for not only each functional module of large-scale software system needs to be tested, but also non-functional testing is performed. Besides, it requires testers to analyze the specific specifications, models and documents of system under test (SUT) to deeply understand its functionality and relevant non-functional characteristics, which provides a basis for the design of test cases and test data. Finally, there usually needs a team to complete the tasks of test design and implementation with the consideration of test scenario, data characteristic and test environment for given test requirements. Test requirement plays important role in the test process, as similar as the software requirement does in the development process. For software requirement gathering and specification, there are a lot of languages, models, techniques and tools available. Nowadays the researches on test requirements tend to use different terms from different perspectives to define the content of test requirement, such as test scenarios [1,2], test sequence [2], test objective [3,4] etc, so there is not a unified definition of test requirement. Therefore, how to provide a general-purpose test requirement modeling language to define clearly the test tasks and the corresponding conditions becomes the motivation of this paper.

Test requirement not only benefits the design of test cases, but also contributes to the evolution of testing. Test is a continuously iterative process. The code of SUT may be modified for fixing bugs. Test cases may be obsolete, redundant or incomplete as test proceeds. To maintain or improve test suite, one needs the knowledge of SUT and the traceability between test cases and SUT. Therefore, it is necessary to build test requirement model as a bridge between test cases and SUT.

Test requirement specification is the result of the collaborative work of testers and developers, so we need a systematic and precise approach to specify it. Formal language is an accurate enough one to describe

[+] Mei Tian. Tel.: + 8601015210832823; fax: +(please specify).
E-mail address: gynh@sei.buaa.edu.cn.

requirements, but it is difficult to write and read. Therefore, we adopt Unified Modeling Language (UML) as the base to define test requirement language in this paper.

2. Language Definition for Test Requirement

Test requirement model is an intrinsic part of test system model that specify the structure and behavior of a test system. In a short, test system is a system that implements a test by invocating the specified SUT with the specified data under the specified environment. In general, test system is composed of test script, test data, test configuration file and the corresponding test executor tool. The main effort for developing a test system lies on specifying the things to do (i.e. test requirement), designing test cases and data, implementing test cases and configuring the test. Test system development is not trivial. The methodologies and guidelines invented for program system development definitely work for test system development. In this paper, we focus on applying the model driven approach to develop test requirement, and of course this approach can be extended to test design and test configuration.

2.1. What does Test Requirement Exactly Mean?

By referring the definition of software requirement [5], we have the following formal definition.

Definition 1: Test requirement is

(1) a condition or capability that test system need to possess to achieve a test objective, or

(2) a condition or capability that must be met or possessed to satisfy a contract, standard, specification or other documents, or

(3) a documented representation of the condition or capability as in (1) and (2).

Based on our thorough analysis of test and its engineering process, three components are identified to refine test requirement, i.e. test object, test purpose and test constraint.

Definition 2: Test Object (TO) is an object of SUT that need to be tested.

$$\text{TO} = < EUT, P, M, F > \quad (1)$$

(1) EUT(Entity Under Test) could be class, component, sub-system or system;

(2) P is the port of EUT via which test system interacts with EUT;

(3) M identifies the relative messages (defined by EUT) that could go through P;

(4) F is the feature or functionality or service provided by EUT.

Take web based email application testing as an example. The login page could be an EUT for testing the login service, i.e. the F. The port for this page is a secured http port (https port) and the corresponding messages include http post message and http get message.

Definition 3: Test Purpose (TP) specifies the objectives of testing the 'test objects'. Test purpose could be a general description, or a specific scenario (defined by the messages identified in test objects) to be covered.

Test purpose refers to protocol transaction scenario in ISO/IEC 9646-1, which focuses on protocol conformance testing. Here we extend it with more general description. In the future, we would calibrate the test requirement language to specific domains. Test purpose can serve as the guideline for designing test case and one test purpose needs to be satisfied with a set of test cases.

Definition 4: Test Constraint (TC) is the constraint on test object or test purpose for guiding test design.

$$\text{TC} = < Property, Predicate > \quad (2)$$

(1) Property is a set of properties of test object or test purpose that need to quantitatively constrained.

(2) Predicate is a Boolean statement on the specified properties. If the predicate is evaluated as true, the constraint is said satisfied. There are two types of test constraints, offered one and required one. Offered test constraint denotes the constraint on test system capability and required test constraint represents the constraint on SUT.

As a conclusion, test object defines the objects to be tested by the test system under development, and the test purpose specifies the objectives of test from the view of test management. Test purposes could be hierarchically managed as test groups. Test constraint is used to put constraints on test object (required constraint) and test purpose (offered constraint).

2.2. Test Requirement Meta-model

According to the conceptual analysis in the above section, we developed the meta-model shown in Figure 1. TestRequirement is defined with three meta-classes, TestObject, TestPurpose and TestConstraint. TestObject involves EUT, Port, Message and Feature. Messages are sent or received through Port, which belongs to EUT. Port is the interface from which test system interact with the EUT. TestPurposes are organized into TestGroup for hierarchical and categorized management. TestGroup can be refined into sub-groups. TestPurpose defines the objective of testing from the view of covering the features of SUT or a (partial) scenario defined by a sequence of messages. Message can be generalized as SendMessage, ReceiveMessage and RefuseMessage. SendMessage goes from test system to trigger the feature of scenario of SUT. ReceiveMessage is a message returned from SUT as a response to the SendMessage. RefuseMessage is an error message that is not expected to receive from SUT (test would be evaluated as fail once received). Test system will make the test evaluation based on SendMessage, ReceiveMessage and RefuseMessage.

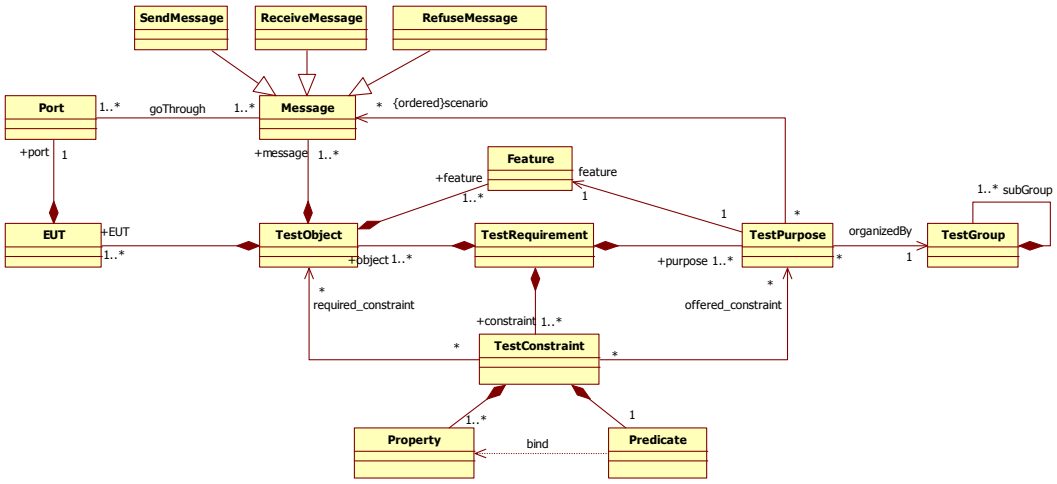

Fig. 1: the Meta-model of Test Requirement

3. Test Requirement Views

The meta-model defines the structure and the semantics of test requirement model. As a profile of UML, we need to develop the corresponding views to support to model test requirement. According to the meta-model, we propose four views to develop test requirements, SUT architecture view, SUT feature view, test purpose view and test characteristic view. Beyond the four views, we propose the labeling based modeling approach that works directly on the UML models of the software under test.

The SUT architecture view and SUT feature view model components to specify test object based on the models of software architecture and feature. Test object is expressed as UML class with the stereotype << test object>>. The EUT and Port can be labeled directly in the component diagram of SUT. The input and output message types are presented as the attributes of Port. SUT feature is labeled on the use case diagram of SUT, which describes how users interact with software system. All the test purposes labeled in SUT diagrams will be collected to form a test purpose view in which the test purposes are hierarchically organized, using the Work Breakdown Structure (WBS). Test group can be refined to the specific test purpose. The detailed description of a test purpose is demonstrated in a sequence diagram. The property of test constraint is defined as test characteristic in our prototype, which refers to the service quality characteristic [6] defined by Object Management Group (OMG). We use the simple class diagram to build the test characteristic view. Test characteristic is defined as a class, whose attributes contain different metrics, named dimension. Test characteristic can be divided into two classes: general characteristic and specified characteristic. The general

characteristic usually refers to the test characteristic applicable in different types of SUTs, while the specified one refers to the one meaningful for some domain or application. An attribute of the general characteristic to be specialized is expressed with the stereotype << bind>>.

4. Case Study

This section introduces a case study of applying the visualized approach to specify test requirement model based on Graphical Modeling Framework. The case study chosen is the conformance testing of Session Initiation Protocol (SIP), which is to test whether the implementation of SIP stack meets some specified standard (RFC 3261) [8]. The test requirements in plain text are as follow. The test plan contains both functional and performance testing on the implementation of SIP stack, i.e. the SUT. The main functionalities to be covered in the testing are the initiation and termination of SIP session, which is performed by the communication between the user agent client and server. The client can send SIP requests and then the server receives and returns a SIP response. One kind of termination of SIP session is started by the remote user agent. The scenario is that the remote one sends the message BYE to the local one and expects to receive the response message '200 OK'. The time for replying must be less than 500 milliseconds, otherwise the test is considered as fail. In performance testing, there need at least 100 parallel virtual users to trigger the simultaneous communications among user agents and servers.

At first we can build the test purpose diagram to provide the holistic view of the tasks involved in this testing, as shown in Figure 2. There are two sub-groups of test purposes, functional testing (ID of FunTest) and performance testing (ID of PerTest). The FunTest can be further decomposed into two sub-groups of Start test purpose (covering the initiation of SIP session) and Release test purpose (covering the termination of SIP session). Furthermore, there are three specific test purpose indexed as TP-1, TP-2 and TP-3 to satisfy the upper level of test purpose groups. The PerTest can be satisfied easily with the TP-4 and the corresponding constraint of "concurrency > 100". That means in the performance testing, we do not care the scenario or flow to stimulate the communications, but the number of parallel virtual users. The test purpose TP-2 is shown in Figure 3, where the TC (Test Component) represents the remote user agent, and the SUT is the local one. We focus on the partial flow from the BYE message to the 200_for_BYE message, and will listen at the TimeOut when waiting for the response message. The constraint on the response time is naturally identified and labeled between the BYE and 200_for_BYE message.

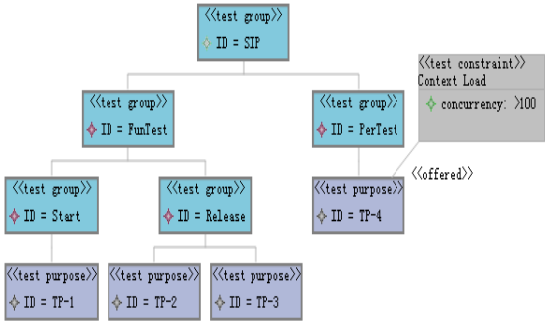

Fig. 2: the Organization of Test Purposes

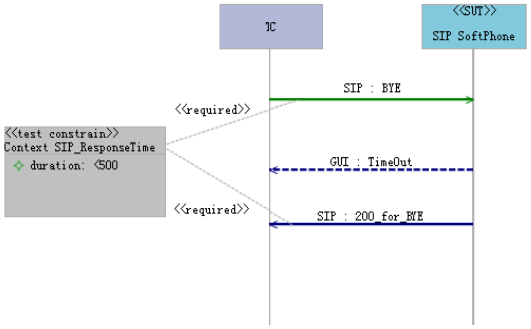

Fig. 3: the Specific Scenario of Test Purpose

In the process of building the test purpose view, the test objects need to be identified, such as the user agent, its port, SIP messages and functionality of the initiation and termination of session. Then we can directly label the test objects in the model diagrams of SUT, as in Figure 4 and Figure 5. In the SUT architecture view (Figure 4), user agent and the implementation of SIP stack are labeled as the EUTs. The SIP messages referred in Figure 3 are identified as test object attached with the SIP port which defines the interface of the SIP_Protocol. Besides, the user operation and corresponding prompt messages via the Graphical User Interface (GUI) port are labeled as port because we need to listen at the timeout message from the GUI port. In the SUT feature view (Figure 5), the SIP sessions between the remote and local user agent are represented as the use cases of CallAsCaller, RemoteHandUp and UserHandUp respectively. These three use cases are directly identified as test objects and covered by the TP-1, TP-2 and TP-3 respectively.

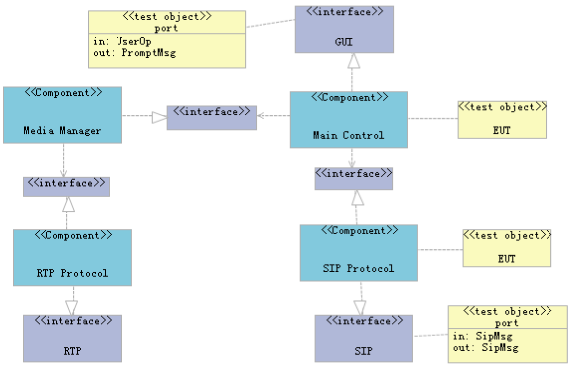

Fig. 4: the SUT architecture view of SIP

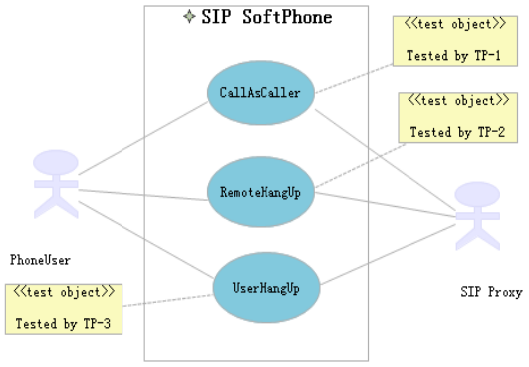

Fig. 5: the SUT feature view of SIP

Based on the developed test objects and test purposes, we could easily review and refine the test requirement model and put the necessary test constraints on the diagrams to clarify the semantics of test requirement, such as the response time constraint on TP-2 and load one and TP-4. These properties of test object or test purpose that need to constrained are defined as test characteristics. The dimension of test characteristic defines the metric, for example, ResponseTime test characteristic is measured by the duration time in Figure 6. It also shows the relationship between the general and specified characteristic.

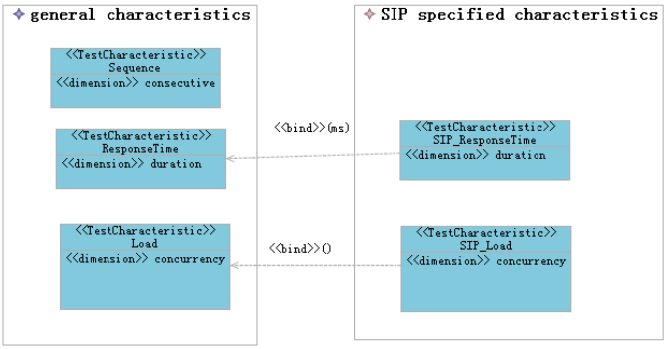

Fig. 6: Test Characteristic View of SIP

5. Acknowledgements

This research is supported by the funding from MOST under the National High-Tech Program project 2009AA01Z145.

6. References

[1] W. Linzhang, Y. Jiesong, Y. Xiaofeng. Generating test cases from UML activity diagrams based on gray-box method[C], Proceedings of the 11th Asia-Pacific Software Engineering Conference, IEEE, 2004, pp.284–291.

[2] Chang-ai Sun, Baobao Zhang, Jin Li. TSGen: A UML Activity Diagram-based Test Scenario Generation Tool[C], Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, 2009, v 2, pp. 853-858.

[3] A. Andrews, R. France, S. Ghosh. Test adequacy criteria for UML design models[J], Software Testing Verification and Reliability 13 (2003) 97–127.

[4] S. Ghosh, R. France, C. Braganza. Test adequacy assessment for UML design model testing[C]. Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE 03), IEEE Computer Society, 2003, pp. 332–343.

[5] IEEE Standard 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology[S], Dec, 1990.

[6] OMG: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification. Version 1.1. formal/2008-04-05. http://www.omg.org/spec/QFTP/1.1/PDF.

[7] Extended Backus–Naur Form. http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf.

[8] J. Rosenberg, H. Schulzrinne, G. Camarillo. “SIP: session initiation protocol”, RFC 3261, June 2002.