The Journal of Systems and Software 109 (2015) 214–228

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A Model-Driven approach for functional test case generation

J.J. Gutiérrez, M.J. Escalona*, M. Mejías

Department of Computer Languages and Systems, ETS Ingeniería Informática, IWT2 Research Group, University of Seville, Av. Reina Mercedes S/N, Seville, Spain

ARTICLE INFO

Article history:
Received 19 November 2014
Revised 19 July 2015
Accepted 4 August 2015
Available online 12 August 2015

Keywords:
Software quality assurance
Model-Driven testing
Early testing

ABSTRACT

Test phase is one of the most critical phases in software engineering life cycle to assure the final system quality. In this context, functional system test cases verify that the system under test fulfills its functional specification. Thus, these test cases are frequently designed from the different scenarios and alternatives depicted in functional requirements. The objective of this paper is to introduce a systematic process based on the Model-Driven paradigm to automate the generation of functional test cases from functional requirements. For this aim, a set of metamodels and transformations and also a specific language domain to use them is presented. The paper finishes stating learned lessons from the trenches as well as relevant future work and conclusions that draw new research lines in the test cases generation context.

1. Introduction

Software quality assurance is one of the most critical steps in software development. Validation and verification techniques were proposed in the software research context to ensure quality and most of them are mainly organized into test phase. Test phase frequently allows us to group each activity oriented toward validating each aspect of our software results, from a small piece of code (with unit tests) to a total piece of the final system (with acceptance user tests) (Binder, 2000).

However, despite its relevance, test phase is frequently planned with very few resources, without an expert group of tests and a reduced group of techniques or tools that help support it (Shah et al., 2014; Huda et al., 2015). Besides, test phase is frequently a good candidate to keep up a delay, with resources or time reductions, when a software project is delayed (Li et al., 2010; Felderer and Ramler, 2014). Consequently, both the software test research community and the enterprise environment are working in developing techniques, mechanisms and tools that enable reducing test phase cost ensuring final quality results (Nazir and Khan, 2012).

Model-Driven Engineering (MDE) could be a solution to get this goal. This new software paradigm is based on the design and use of models to obtain software artifacts. Its application in test context is known as Model-Driven Testing (MDT), which is being successfully utilized in different Software Testing contexts (Völter et al., 2013). This paper presents an approach that, applied to the enterprise context, endorses this sentence.

This work also introduces a MDT approach that mainly focuses on a very concrete type of test: functional testing. This kind of testing tends to guarantee that initial requirements defined by users are correctly supported by the final system (Bertolino et al., 2005). This approach takes advantage of MDT power to define a set of models, transformations and processes that allows defining, in a systematic way, functional tests from functional requirements, reducing time and assuring the right traceability between initial requirements and final tests. The approach is authorized by the current real validation that we are getting in the enterprise environment.

This MDT approach comprises five metamodels and four transformations. Two metamodels are used to model the required information from functional requirements. The first transformation allows improving the requirement metamodel with a simple graph describing the functionality expressed in the requirement. Such graph is redundant as it does not describe any new information, but it helps simplifying other transformations. Path analysis and Category-Partition Methods are two of the main techniques used to derive test cases from functional requirements. Therefore, two metamodels and transformations have been designed to perform these techniques. Finally this MDT approach adds a fifth metamodel and a transformation to offer a consolidate view of the paths and categories.

From the authors' point of view, one of the main challenges for functional test case generation is the lack of formalism in functional test cases. With regard to advantages, functional requirements provide flexibility in the techniques, for example, by means of use cases or user stories. However, this flexibility is a gap to be filled in when trying to automate operations from use cases. The formalism of functional requirements constitutes a hard task in terms of time, knowledge and money. However, as this paper illustrates in the practical cases, once achieved, it provides a valuable return of investment.

* Corresponding author: Tel.: +34 954552852; fax: +34 954556844.
E-mail addresses: javierj@us.es (J.J. Gutiérrez), mjescalona@us.es, mjescalona@iwt2.org (M.J. Escalona), risoto@us.es (M. Mejías).

http://dx.doi.org/10.1016/j.jss.2015.08.001
0164-1212/

This study is organized as follows: Section 2 presents the background that lists the terminology used in order to introduce the reader to the concrete context of the paper. Next, Section 3 offers a brief summary regarding related work on functional test cases and the most used testing techniques. Section 4 explains the approach by introducing the process of functional test cases generation from functional requirements under a MDT perspective in the first subsection; the second one analyzes in detail the set of used metamodels; and the third one defines transformations. Then, Section 5 details how the approach is implemented and provides the reader with a concrete example and practical references of the approach. To conclude, Sections 6 and 7 state the original contribution of this work together with the relevant conclusions and ongoing work.

2. Model-Driven engineering

As mentioned before, this paper focuses on the application of MDE in functional test cases generation. This section provides the reader with the lexicon and tools used along the text.

Two important elements must be stuck out in MDE environment: ***metamodels*** and ***transformations***.

Metamodels normalize the information used for generating test cases. A metamodel defines the constructor and the relations with constraints allowing models design with a valid semantics. Metamodels enable combining different concrete syntaxes (as textual and graphical syntaxes exposed in the motivational example) using the same lexicon and manipulating the same semantic artifacts.

A ***transformation*** is a relation between elements in a source metamodel and elements in a target metamodel. Therefore, executing transformation helps build a group of elements in target models to conform to their metamodels, using the information from a set of elements in source models, which must also agree with their metamodels. In this case, the agreed point guarantees that the transformation can be performed. Transformation elements define a systematic process regardless of any tool or programming language.

In this paper, UML (Unified Model Languages) (Object Management Group, 2011) class diagrams define the metamodel presented in the next sections, while QVT (Query-View Transformation language) (Object Management Group, 2010) identifies transformations. We select both solutions, as they are well-known standards. UML is proposed by OMG (Object Management Group) and it is frequently used in MDE for defining metamodels. OMG also suggests that QVT should be the standard for specifying transformations among models. Even though other possibilities are available to run our work, like ATL (ATL Transformation Language, 2015), we prefer using QVT since it has provided us with successful results in preceding projects (García-García et al., 2013).

QVT defines two main syntaxes for defining transformations: ***QVT-Relational*** and ***QVT-Operational***. *The former* is a declarative language similar to SQL. *The latter* defines operators and control structures from classic imperative languages. We have selected QVT-Operational for this paper because some of the transformations outlined in the subsequent section use a pathfinder algorithm.

A transformation in QVT is decomposed into a set of ***mapping operations***. A ***mapping*** is a relation between one or more source elements and one or more target elements.

Other elements used for defining transformations in QVT are ***queries*** and ***helper***. Both elements are operations that perform a computation and provide a result. A *helper* may have side effects on the given parameters, whereas a *query* has no side effects.

This paper also introduces the concept of ***direct mapping***. It defines a relation between one source element and one target element and a relation 1:1 between the attributes of the source element and the target element. Attributes from source and target elements have the same names and types. From this perspective, generating test cases from functional requirements becomes a problem when defining metamodels for functional requirements and test cases, and creating a set of transformations to get concrete test cases from particular functional requirements. Metamodels and transformations needed for producing functional test cases from functional requirements are presented in the next sections.

3. Related work

As the contextualization of our problem concerns, this section presents related work and it is divided in two parts. Section 3.1 summarizes the bibliography related to functional test cases generation from functional requirements. Then, Section 3.2 describes the two main techniques found in the literature previously studied.

3.1. State-of-the-art

At the time of writing this paper, there are two main surveys, (Escalona et al., 2011) and (Denger and Mora, 2003), studying existing approaches dealing with generating functional test cases from functional requirements. This section summarizes their most relevant conclusions.

These two considered surveys are specific for functional test cases, although other relevant comparative studies in this sense were published, such as Anand et al. (2013), where prominent techniques for automatic generation of software test cases are compared.

Conclusions from Denger and Medina's survey point out that the authors of the approaches do not follow any standards when defining templates (for functional requirements). On the contrary, each approach uses its own templates and formats. Another conclusion from that report is that none of the approaches uses path analysis techniques and, as a previous step, the approaches build a more formal representation of functional requirements.

Escalona's survey, developed by the same authors who write this article, concludes like the previous one. They mainly agree that many of the existing approaches have to formalize requirements as a first step to generate functional test cases, because of the use of text templates and colloquial language to define functional requirements. This is a mandatory step in approaches that offer a high degree of systematization and demand supporting tools. However, it can be pointed out that some approaches offer a systematic way, or even automatic ways to generate more formal models to automate the process. In this case, this is possible because requirements are described in natural language, therefore, they are metamodels and some transformations from this description enable translating requirements in natural language into activity diagrams.

Escalona's survey, published at the end of 2011, cites 24 approaches; the oldest dates back to 1988 (Category-Partition Method) and the newest to 2009. Denger's, published in 2003, cites 12 approaches; the oldest from 1988 (it is the same approach used in Escalona's survey) and the newest from 2002.

Below, there are some examples of the approaches included in Denger's and Escalona's surveys and new approaches not included in any surveys in order to update the current situation.

Hartmann et al. (2004) start their approach with functional requirements written in natural language. The result is a set of functional test cases obtained from a coverage criterion based on combinations that support Boolean propositions.

Binder's book (Binder, 2000) describes the application of the Category-Partition Method to use cases. Categories are any point in which the behavior of the use case may be different in two realizations of the use case. This application is named Extended Use Case Pattern.

In addition, Ibrahim et al. (2007) offer a tool, called GenTCase, which generates test cases automatically from a use case diagram enriched with every use case tabular text description.