Selection, Evaluation and Generation of Test Cases in an Industrial Setting: a Process and a Tool

Nicolas Guelfi[1], Benoît Ries[1,2]

[1] *Laboratory of Advanced Software Systems*
University of Luxembourg
L-1359 Luxembourg, Luxembourg
{nicolas.guelfi,benoit.ries}@uni.lu

[2] *I.E.E.*
ZAE Weiergewan
L-5326 Contern, Luxembourg
benoit.ries@iee.lu

Abstract

The test phase in safety-critical systems industry is a crucial phase of the development process. Some companies of these industries have their own test methods which do not reuse the notions available in the theory of software testing or model driven engineering. This paper reports on an experience in a testing process improvement made inside a safety-critical systems company in order to improve the quality of the test phase improvement. We present the initial situation, the objectives, the proposed process and the tools that are used to support it. In particular, we show that the most efficient improvements were achieved concerning the test process definition and in allowing a tailored and precise delimitation of the system's elements to be tested.

1. Introduction

Software testing, as defined in the Software Engineering Body of Knowledge [1], consists of the dynamic verification of the behaviour of a program on a *finite set of test cases*, suitably *selected* from the usually infinite execution set *against the expected behaviour*. We consider [14] that model-based testing consists of specifying a selection of test cases based on a model of the expected behaviour. The complexity lies in the capability of the test process in helping the test engineer to better reach a delimited (cf. Section 4.1) and verifiable (cf. Section 4.2) test set w.r.t. the test objectives and project specificities that also impose to take into account different verification viewpoints (cf. Section 4.1.2).

I.E.E. [5] is an international company leading in the offering of sensor-based systems. Products range from car seat pressure mats to 3D vision sensing solutions. The developed products are safety-critical and as such benefits from a rigorous test process definition that has to be tool supported.

In this paper, we present a model-based testing process focusing on the test selection activities as an improvement to the current testing process of the I.E.E. company.

2. Process improvement

2.1. Initial state of practice

Embedded software systems as such, introduce a bias toward system testing by the customers and suppliers. An important part of the companies that develop embedded software systems were previously developing non programmable physical systems. The first bias is that customer tests focus on physical attributes and thus do not cover the attributes introduced for the purpose of the embedded software. Secondly, in the same way another bias is also introduced for the requirements specification, which may have important consequences in our context of specification-based testing. Especially, the supplier will neglect its own requirements and tests, both coming from its development and test platforms.

Functional tests that are performed at IEE have an unknown coverage, and the amount of testing performed is usually defined by the amount of time available in the schedule.

The testing that is performed first focuses on software structural tests, of a white box nature, and then skips straight to the testing of the entire system as a black box. This causes a high risk of software functional defects remaining, which are hard to detect when testing the system as a whole. It is important to notice that there is a high temptation for the suppliers and customers to shy away from a more formal definition of the complete software test space since it

might provide evidence that the sufficient testing could not be achieved exhaustively in the time available.

2.2. Requirements for process improvement

Improving product quality can be done by following a verification process proposing the execution of a tractable test phase for which test cases are selected in order to address explicit quality objectives. If this method used simple software requirement models as an input, this would also add more weight to the requirements analysis phase of a project, with corresponding benefits to the project and especially to the subsequent design phase.

By focusing this method at the software boundary, there would be a big reduction in hidden software defects, and potentially, a big reduction in the total time taken to discover and remove defects from the system as a whole.

The test time itself would become much more deterministic, and hence allow an increased chance that regression tests would be allowed for in the scheduling of changes to the system.

Thus our proposal for process improvement tries to address the three objectives:

- Based on a simple software requirements model.
- Includes a precise description of the test space.
- Offers a precise knowledge of the test coverage.

3. Industrial case study

Throughout this paper, we will illustrate our process with the *PersonCounting* (PeCo) case study taken from a concrete industrial system developed at I.E.E.

The PeCo application observes a certain field of view (FOV), a virtual cube with an edge length of about 3 meters, to provide the three following features:

- count all persons entering and exiting the FOV
- track persons as long as they are in the FOV
- discriminate "persons" from other mobile, or immobile, objects.

Figure 1 shows a display of the PeCo application. The Distance Map view shows the raw image received from the observed field of view and the 3D-Model shows the interpretation of the image. Finally, two counters compute respectively; the number of persons currently in the field of view and the total number of persons who have walked through the field of view.

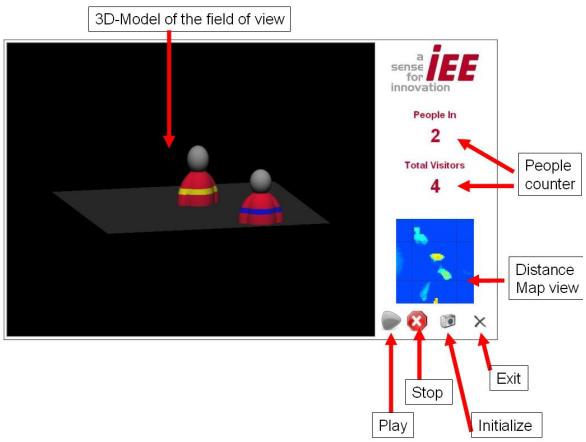

Figure 1. PersonCounting display

It is complex to test the PeCo application because of the variety of test that may occur. In particular, it is complex to define details criteria of what is a valid or invalid "persons". In this paper, we abstract away these intricate details and characterize a "person" from its height (z-axis value) which should be higher than 1,20 meter. In the remaining, persons will also be referred to as object of interest (shortly, OOI).

4. Process workflow description

We propose a model-driven test selection process that fulfils the three requirements identified in the previous Section 2.2. This process comprises the following artefacts:

- A modelling language for the analysis phase based on UML2 [12] class diagram and protocol state machines.
- A test selection language that describes precisely the system's state space to be tested. A test model is produced from the requirements model and the test constraints are specified with a test selection language.
- A set of metrics that help to evaluate the test selection coverage, thanks to a formal semantics given to the analysis and test models in terms of the Alloy [6] formal language.

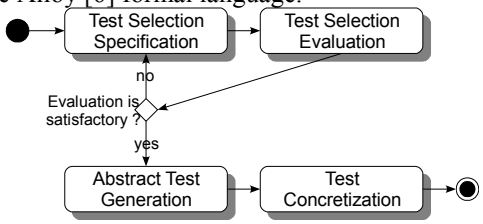

Figure 2. Process workflow

Prior to specifying the test selection, the software analyst understands and describes the customer requirements in an analysis specification. For the

purpose of this testing process, we propose this specification should contain at least a requirements model composed of the two following UML2 diagrams:

- a class diagram that describes (1) the attributes characterizing the system under test (SUT), (2) the events that the SUT can receive from its environment and (3) the data types that are passed by the events and used to characterize the system attributes. Figure 3 describes the data of the PeCo case study.

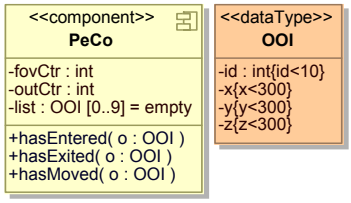

Figure 3. Requirements model: data

- a protocol state machine that specifies the allowed order and the conditions for calling the SUT's received events (precondition) as well as the expected changes in terms of changes of system attributes (postcondition). Figure 4 describes the behaviour of the PeCo case study.

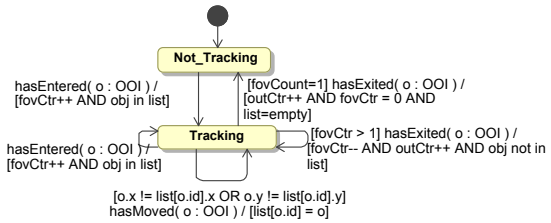

Figure 4. Requirements model: behaviour

4.1. Test selection specification

Constraint-based testing has been used for more than a decade now [2]. We proposed to the IEE verification team to combine model-driven testing with constraint-based testing. In that framework, the constraints are expressed on the model of the expected behaviour, i.e. the requirements model. The first step of our approach (cf. Figure 2) is the specification of *test constraints* on the requirements model. The aim of these test constraints is to reduce the number and/or length of test cases that would be exhaustively generated from the requirements model.

4.1.1. Test selection language. Test selection languages are defined to help structure and ease the understanding of test specifications. For instance Ostrand and Balcer who have defined TSL [13] in the context of the category-partition test selection technique.

We define a small test selection language in the context of our process for the IEE company that is composed of a restricted number of instructions. Its purpose is to express in a structured way the test selection specification and to enable automatic interpretation for automation of the abstract test cases generation phase.

The language includes instructions for specification of static as well as dynamic test constraints. Static test constraints are expressed on the class diagram elements that define the types of input domain of the SUT. These types are used for all the parameters values of the system events. Expressions for this type of constraints look like "**select** n **values for** evt(param) **where** cond" where the logical condition cond is used to characterize the set of selected evt parameters values from which n values will be chosen. If n is not specified all values satisfying the condition are selected.

Dynamic test constraints aim at reducing the length of execution traces specified in the analysis model and are expressed on elements of the protocol state machine. Expressions for this type of constraints look like "**repeat** n **times event** evt" and request that the number of calls to evt in a test is n.

4.1.2. Multiple test stakeholders. The test selection is a step that involves a number of stakeholders. In the context of our case study inside the IEE company, we have identified and formalized five actors who are likely to participate in the definition of the test specification.

During the elicitation of functional requirements of the system to be developed, the *Customer* often express some required test that the system will have to pass. In the PeCo application, the customer expressed a particular critical case when two persons walk in and out of the field of view being very close to each other (e.g. holding hands).

The *Quality engineer* analyses potential failures of the system resulting in a so-called Failure Modes and Effects Analysis (FMEA) [4] document. FMEA of the PeCo system includes the following potential failure mode: an OOI changing height around the threshold of 1,20m. For instance, when a person in the FOV ties his shoe laces.

The *Product-line manager* records previous failure in the field from the same products family and trace these failures to test cases that were not previously selected and that should now be included. For instance, the system may not perform a correct discrimination for men with beards longer than 20 cm.

Late in the system development process, the *Manufacturer* may express some specific system tests. The manufacturer tests must be described in the test selection phase and performed by the company before manufacturing phase. In the PeCo system, the manufacturer tests will include that the alignment of the image is correct after manufacturing.

An *Integrator* (more generally called, *Installation engineer*) is responsible for integrating the manufactured product into its physical environment. In our case study, in some cases, the camera and associated PeCo application will actually cover a narrower FOV than the 3 meters. For instance, when it is installed in a narrow corridor.

Our test selection language aims to express all the test constraints coming from these different stakeholders. For instance, the test constraint coming from the *Integrator* when the system is installed in a 2-meters wide corridor, would constrain the three events to be called with object that have x-values between 50 and 250:

select values for hasEntered(obj) **where** obj.posX >50 and obj.posX < 250

select values for hasMoved(obj) **where** obj.posX >50 and obj.posX < 250

select values for hasExited(obj) **where** obj.posX >50 and obj.posX < 250

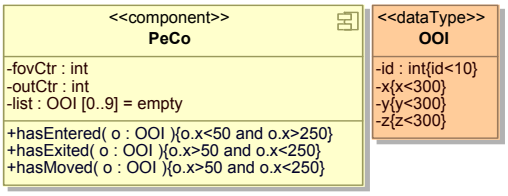

Figure 5. Test model: data

4.1.3. Test model generation. A test model (made of a class diagram and a protocol state machine) is produced from applying the test constraints to the requirements model. We have automated this test model production, with the Kermeta [10, 11] model transformation engine. Each constraint from the test selection specification is implemented as a Kermeta model transformation. They are all applied incrementally on the input requirement model.

The application of the test selection, given in Section 4.1.2., results in the test model shown in Figure 5 where the three events have been decorated with the test constraints. The behaviour being unchanged, see Figure 4.

4.2. Test selection evaluation

The purpose of this step is to evaluate the test selection (specified in the previous step) before the tests are executed on the SUT. In order to evaluate the test selection, we propose computing the values of some metrics in order to help the *test manager* in deciding if the test selection will be covering sufficient behaviour of the system. We identified the following metrics that may be used for this evaluation.

- *State coverage* that refers to how many (and which) states can be reached from the initial state of the test model's protocol state machine.
- *Transition coverage* in a similar way to the state coverage, it refers to how many transitions can be reached from the initial state of the test model.
- *Event coverage.* It represent how many (and which) events can still be accepted by the protocol state machine after test selection.
- *Maximum number of repetition for each event*. It represent how many times each event can be accepted at most by the protocol state machine after test selection.
- *Data definition domain size*. How many possible values can be derived from each datatype of the test model.
- *Multiple preconditions coverage*. When precondition is made of a disjunction of conditions, this metrics informs about how many of the single conditions can be true at least once in all possible execution of the test model.

The test constraint from the Integrator given in Section 4.1.2. obviously reduces the value of the metric *Data definition domain size* on the test model.

4.3. Abstract test generation and concretization

The purpose of the *abstract test generation* step is to interpret the test model into a set of sequences of event calls coming with their parameter values. This is a classical step in model-driven testing. In our process, we advocate an exhaustive generation of test cases from the test model; if the exhaustive generation is not possible (e.g. for timing reasons) there shall be an additional iteration of the test selection process, i.e. the test selection specification must be further constrained and evaluated until the exhaustive generation from the test model is satisfactory.

The example test selection, given in the last paragraph of Section 4.1.2., on its own does not provide a significant enough reduction in the test to be performed. Therefore, the test model should be further constrained. This can be achieved by combining test constraints identified by further test stakeholders, see Section 4.1.2., for example the quality engineer.

Concerning the tool support for this step [3], we use the Alloy Analyzer [7] which has already been used successfully for test generation from Alloy models [8, 9].

Our process ends with a test concretization phase in which, test engineers associate with each event, parameters values of the generated tests a concrete value of the physical embedded system under test. In our case study, it means to define the 3D images that represent the tracked objects (i.e OOI).

5. Conclusion

We have identified gaps in the current testing process of the company and proposed a model-driven testing process that represent an important improvement for IEE. The main advantages of our proposed approach are (a) the specification of the system is the main artefact to decide for further test activities; (b) a domain specific language is used for test selection specification (c) the test model specifies the tests that will be effectively performed with the same notation as the requirements model (d) metrics are available for the verification of the test selection (e) the overall process is tool-supported.

The process fulfills the process improvements requirements set by the company. The first feedback from the company confirmed the expected contribution. The company especially approved (1) the simplicity of the requirements modeling notation (restricted UML2 class diagram and protocol state machine) (2) the usefulness of the test selection language to precisely delimit the test space to be exercised on the SUT (3) the evaluation of the test selection, enabling an awareness of the coverage of the selected test space.

Before its widespread adoption IEE will go through a complete evaluation on a medium size pilot project.

6. Acknowledgments

The authors would like to thank David Wiseman from IEE for his help on the case study description.

7. References

[1] A. Bertolino, ``knowledge area description of software testing - SWEBOK," Tech. Rep., Joint IEEE-ACM Software Engineering Coordinating Committee, 2000. www.swebok.org.

[2] A. Gotlieb, B. Botella, and M. Rueher, ``Automatic test data generation using constraint solving techniques," *SIGSOFT Software Engineering Notes*, vol. 23, no. 2, pp. 53-62, 1998.

[3] N. Guelfi, and B. Ries, "A Semantics of UML2 Class Diagrams and Procotol State Machines in Alloy for Test Selection Analysis", submitted to *The International Conference on Software Engineering and Formal Methods*, 2008.

[4] "Analysis techniques for system reliability - procedure for failure mode and effects analysis (FMEA)," std IEC 60812 ed. 1.0 b, 1985.

[5] "International Engineering Electronics", www.iee.lu.

[6] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. MIT Press, March 2006.

[7] D. Jackson, I. Schechter, and I. Shlyakhter, ``ALCOA: The Alloy constraint analyzer," in *The International Conference on Software Engineering*, June 2000.

[8] S. Khurshid and D. Marinov, "TestEra: Specification-based testing of java programs using SAT," *Automated Software Engineering Journal*, vol. 11, October 2004.

[9] S. Khurshid, D. Marinov, I. Shlyakhter, and D. Jackson, ``A case for efficient solution enumeration," in *The International Conference on Theory and Applications of Satisfiability Testing*, 2003.

[10] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel, ``Weaving executability into object-oriented meta-languages," in *The International Conference on Model Driven Engineering Languages and Systems*, 2005.

[11] P.-A. Muller, F. Fleurey, D. Vojtisek, Z. Drey, D. Pollet, F. Fondement, P. Studer, and J.-M. Jezequel, ``On executable meta-languages applied to model transformations," in *The Model Transformations In Practice Workshop*, 2005.

[12] OMG, ``Unified Modeling Language: Superstructure, version 2.1.1," Full Specification formal/07-02-05, Object Management Group, 2007.

[13] T. J. Ostrand and M. J. Balcer, ``The category-partition method for specifying and generating functional tests," *Communications ACM*, vol. 31, no. 6, pp. 676-686, 1988.

[14] M. Utting, A. Pretschner, and B. Legeard, ``A taxonomy of model-based testing," Tech. Rep. ISSN 1170-487X, University of Waikato, April 2006.