SIMPLIFIED DESIGN OF TEST CASES BASED ON MODELS

Guntis Arnicans, Vineta Arnicane
University of Latvia
guntis.arnicans@lu.lv, vineta.arnicane@lu.lv

Complexity of model-based testing

Over the past decade a growing trend to use model-based testing approach is observed. Unfortunately, the industry rare exploits this approach. The main causes are complexity of this method, the need for a good specialists and the initial high consumption of resources. We offer pick up from the model-based testing approach the simplest ideas and techniques, and use them at least in the design of test cases. We offer additional recommendations that allow to improve the testing skills for the majority of testers. Our offer could be a useful in early stages of software development and testing, for instance, when the tester often uses an exploratory testing.

One of the primary problems to be solved by the testers is a design of test cases. Tests can be designed at different conceptual levels. After the design of test cases, they must be specified by giving the exact input values and scenarios how to feed the system under test with these values. Then the system is tested and results are evaluated. The more test cases we produce, the more resources are consumed. Design of test cases is essential effort with aim to reduce this resource consumption.

Model-based testing is based on three issues:

1. a formal model that describes the behavior of the software;
2. a test generation algorithm or criteria;
3. tools that support a test infrastructure (test cases generation, management, implementation, evaluation of results, etc.).

Model-based testing assumes that most of the steps are automated, except for the first model-building step that specifies software behavior and features. Model might be created by using different formalisms [1, 2], for example, UML state diagram, class diagram, sequence diagrams, finite state machines. Chosen formalism or language is a very important and must be taken into account before creating of the model, as it directly affects the quality of testing. It is important to integrate the software development process and model-based testing approach. Most of model-based testing approaches are not empirically verified and used in the industry. Some of the reasons for rare use of model-based testing are its natural complexity, the need to have good knowledge of formal testing methods, difficulty to translate data generated by the model to final test cases if the model is designed at high conceptual level, need of automation. In addition, the complexity arises from the fact that the method calls for a model that describes the whole system, and it must be consistent and correct.

Simplification of model-based testing for designing of test cases

Instead of building one large formal model, we recommend to develop several small models. Any model can be described by its informal notation. Few simple principles are applied to obtain design of test cases.

- **Create many small models instead of one large**. If we look at model as a graphical diagram then count of nodes have to be less than 10 because it is difficult for human brain to keep in mind more objects at a time.
- **Create models from the various aspects**. The more aspects are used in testing, the more testing is diverse and can reveal different problems. This principle gives opportunity from one side to hold models not too big, from the other side – do testing differently, from different points of view, with different goals and methods.
- **Create several models for the chosen aspect**. It is preferable that different testers create their own model for given aspect. Each tester has own viewpoint to the task. This diversity can better provide testing team with various opinions what software has to do or is doing. If inconsistency between test cases is discovered, then additional analysis and investigation have to be done.
- **Model can be created informally.** The formalism is not a primary goal. Tester may use such notation that is more convenient for aspect and his own skills. We offer to draw

graphs by giving necessary meaning to nodes and edges [3]. Nodes and edges may be of different type, and attributes may be associated with any node or edge.

- **Model may be incomplete.** Our main goal is obtain designs for test cases. Basically this task has to be performed without automation because model is created without formalism. Tester takes into account any imperfection that he or she notices (author usually knows how the model is created and what kinds of the problems it contains).
- **Models are created for various levels of abstraction**. The model can, for instance, describe a part of source code (low level) or basic properties or functioning of the whole system. The higher is conceptual level, the more it expresses a user viewpoint of system (one element of model can be a long scenario of activities while testing process).
- **Test cases are designed at logical level**. The actual input values of test case are assigned before testing. If model is of high conceptual level, then a blueprint or sketch of test case is obtained from this model. And blueprint serves as a class of many actual test cases.
- **Test generation is based on well known graph coverage criteria**. Tester can adapt any coverage criteria from the structural testing method (node coverage, edge coverage, node pair coverage, in-out edge pair coverage for node, path coverage with or without cycles, etc.). If any node or edge has attributes and constraints, then principles of equivalence class testing and boundary testing methods can be applied, too.
- **Breadth testing principle is used to design and organize the test cases**. Let us assume that we have many different small models. From each one we can design test cases. At first we advise to execute tests that are easily obtainable and cover many aspects. For instance: 1) 90% node coverage in each model; 2) 90% edge coverage in each model; 3) 90% edge pair coverage for each node in each model; 4) testing of cycles; 5) testing of attributes; 6) other criteria.
- **Create library of exploited models and patterns**. Accumulate models and create patterns basically for learning purpose. We do not expect that very specific informal models can be widely reused. But samples are very important in order to educate testers and teach them think creatively – what aspects can be modelled and what notation is understandable and useful.
- **Hand-drawn model also is worth**. Significant part of people more quickly create models by hand using their own notation that seems convenient for them. Usually the colours are used. It is not worth to spend time to obtain nice drawings for informal model from which we cannot directly generate test cases. Scan the image of model and use it as a picture.
- **Try and throw away**. Do not afraid to quickly develop a small model, generate first set of test cases, try it, and throw it away later, if the test set is not good enough or the model does not conform to new version of software or requirements.

Conclusions

Our approach is especially beneficial for exploratory testers [4] at early stages of software development or testing. The “classical” model-based testing is more appropriate for critical projects at the latest development stages when changes in requirements and software are minor. We suggest simplification the using of models to a level at which the most of testers understand proposed method, can apply it, and does not lose the essence of model-based testing – model reveals the most important issues of system and helps us to look at the system from various points of view.

Acknowledgement

This work has been supported by the European Social Fund Project No. 2009/0216/1DP/1.1.1.2.0/09 /APIA/VIAA/044.

References

1. Baker, P., Dai, Z. R., Grabowski, J., Schieferdecker, I., Williams, C., Model-Driven Testing: Using the UML Testing Profile, Springer, 2007.
2. Utting, M., Legeard, B., Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2006.
3. Beizer, B., Black-Box Testing: Techniques for Functional Testing of Software and Systems, Wiley, 1995.
4. Whittaker, J. A., Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison-Wesley Professional, 2009.