Functional Testing of Feature Model Analysis Tools. A First Step *

Sergio Segura, David Benavides and Antonio Ruiz-Cortés
Department of Computer Languages and Systems
University of Seville
Av Reina Mercedes S/N, 41012 Seville, Spain
{sergiosegura, benavides, aruiz} AT us.es

Abstract

The automated analysis of Feature Models (FMs) focuses on the usage of different logic paradigms and solvers to implement a number of analysis operations on FMs. The implementation of these operations using a specific solver is an error-prone and time-consuming task. To improve this situation, we propose to design a generic set of test cases to verify the functionality and correctness of the tools for the automated analysis of FMs. These test cases would help to improve the reliability of the existing tools while reducing the time needed to develop new ones. As a starting point, in this position paper we overview some of the classifications of software testing methods reported in the literature and study the adequacy of each approach to the context of our proposal.

1. Introduction

The analysis of an FM consists on the observation of its properties. Typical operations of analysis allow finding out whether a FM is void (i.e. it represents no products), whether it contains errors (e.g. feature that can not be part of any products) or what is the number of products of the software product line represented by the model. In particular, the analysis is generally performed in two steps: $i)$ First, the model is translated into a specific logic representation (e.g. Constraint Satisfaction Problem (CSP), [4]), $ii)$ Then, off-the-shelf solvers are used to automatically perform a set of analysis operations on the logic representation of the model [5].

The available empirical results [6, 7] and surveys [5] in the context of the automated analysis of FMs suggest that there is neither an optimum logic paradigm nor solver to perform all the operations identified on FMs. As a result of this, many authors propose enabling the analysis using different paradigms such as constraint programming [4], propositional logic [3, 16] or description logic [9, 15].

Implementing the operations for the analysis of FMs using a specific solver is not a trivial task. The lack of specific testing mechanisms in this context difficult the development of tools and reduce their reliability. To improve this situation, we propose to design a set of generic test cases to verify the functionality and correctness of the tools for the automated analysis of FMs. These test cases would help to improve the reliability and robustness of the existing tools while reducing the time needed to develop new ones. As a starting point, in this position paper we narrow the search for a suitable testing technique, test adequacy criteria and test data generation mechanism to be used for our tests. For that purpose, we overview some of the classification of software testing methods reported in the literature and study the adequacy of each approach to the context of our problem.

The remainder of the paper is structured as follows: In Section 2 we introduce some common classification of testing techniques and evaluate the adequacy of each approach for our purposes. Some general classes of test adequacy criteria and some argumentation about whether they are appropriate for our proposal are presented in Section 3. In Section 4 we study different mechanisms for the generation of test data. Finally, we overview our evaluation of the different approaches and describe our future work in Section 5.

2. Selection of testing techniques

Testing techniques can be classified according to multiple factors. Next, we describe some of them and evaluate the adequacy of each approach to the context of our proposal.

- **Knowledge of the source code.** According to our knowledge about the source code of the program un-

*This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472) and the Andalusian Government project ISABEL (TIC-2533)

der test, tests can be classify as *white-box*, *black-box* or *grey-box* test cases [8, 10, 11, 12, 17]. Black box test cases are those in which no knowledge about the implementation is assumed. These test cases are based exclusively on the inputs and outputs of the system under test. White-box (or, alternatively, *glass-box*) test cases consider the entire source code of the program while grey-box test cases only consider a portion of it.

We want our proposal to be suitable to test any tool for the automated analysis of FMs independently of the logic paradigm or solver it uses for the analysis. Therefore, we will need a black box testing technique in which only the input and output of the tools are considered.

- **Source of information used.** According to the source of information used to specify testing requirement, testing techniques can be mainly classified as *program-based*, *specification-based* and *interface-based* [8, 10, 17]. Program-based testing approaches base on the source code of the program to create the tests. Specification-based techniques based on the specification (i.e. requirements) of the program to identify features to test. Finally, interface-based test cases specify testing requirements in terms of the type and range of the inputs without referencing any detail of the program or the specification.

  Since we have decided to used black box testing techniques we can only consider the approaches not assuming any knowledge about the source code, that is, specification- and interface-based testing. On the one hand, specification-based testing appears as a suitable strategy since we assume that the tools for the automated analysis of FMs share a common functional specification for the analysis operations. On the other hand, we presume interface-based is not suitable for our proposal since it only deals with the information provided by the interface without considering any functional aspect of the software under test.

- **Execution-based vs Non-execution based.** Testing can be perform either by running the program to test or without running it through the usage of so-called *software inspections* [10, 12]. Non-execution based methods usually require to examine the source code of the program and the documentation that accompanies it. Thus, we consider that an execution-based technique is probably the best option to our purpose since we do not assume any knowledge about the source code.

- **Testing level.** Testing can be performed at different levels, mainly: *unit*, *integration* and *system level* [2]. Unit testing focus on the verification of isolated component or modules. Integration testing exposes defects in the interfaces and interaction between integrated modules. Finally, system testing test the whole system to verify whether it fulfill the requirements or not.

  We intend to verify the functionality of each analysis operation separately in order to be as accurate as possible when informing about defects. Consequently, we consider that unit tests are the most suitable approach for our needs.

3. Selection of test adequacy criteria

Adequacy criteria define what constitutes a good test. Zhu *et al.* [17] identify three generic classes of test adequacy criteria. Next, we introduce them and check whether they are appropriate for our context of application.

- **Fault-based.** This criterion measures the adequacy of a test according to its ability to detect faults. Typical strategies using this criterion are *error seeding* or *program mutation testing* [17]. In these approaches, artificial faults are introduced into a copy of the program (or one of the programs) under test. Then, the software is tested and the number of artificial faults detected is counted. The proportion of artificial faults detected determines the adequacy of the test.

  We consider that the usage of this kind of adequacy criterion would be suitable for our proposal since the main goal of our tests is to expose defects in the tools for the automated analysis of FMs. For instance, once the test were ready, several mock implementations of the analysis operations including artificial faults could be developed and used to measure the ability (i.e. adequacy) of our tests to find them.

- **Error-based.** Error-based testing requires test cases to check the software on certain error-prone points [17]. This way, this criterion measures the ability of the tests to verify that the software does not deviate from its specification in a typical way.

  Our experience in the analysis of FMs has leaded us to identify several error-prone points when designing tools for the automated analysis of FMs. These are especially related with the usage of feature cardinalities and the detection of dead features [13]. Thus, we consider this adequacy criterion is also a suitable option to take into account for our purpose.

- **Structurally-based.** A structurally-based criterion requires the design of a test that covers a particular set of elements (e.g. statements) in the structure of the program or the specification. This test adequacy criterion is mostly used with program-based testing [10]. Thus, we do not find it particularly useful for our tests.

4. Selection of mechanisms for test data generation

We identify different options to generate the test data, namely:

- **Automated *vs* manual.** The generation of test data can be either manual or automated [10]. On the one hand, the automated generation of test data is faster and usually enables the generation of bigger and more complex program inputs. On the other hand, the manual generation simplifies the creation of customized tests.

  We consider that both approaches are suitable for our approach and that they could even being combined. This way, the test inputs could be composed by two groups of automatically- and manually-generated FMs.

- **Random *vs* systematic.** Test data can be generated either randomly or systematically. On the one hand, random test data generation approaches relies upon a random number generator to generate test input values [10]. On the other hand, systematic approaches follow some pre-defined criteria to generate the data.

  Once again, we consider that a combined approach would be the most suitable option for our tests. For instance, a set of randomly-generated FMs could be first used to test the tools using different size and forms of the input. Then, a second group of FMs could be systematically designed to exercise some specific error-prone points in the tools under test. In this context, we already have a tool, the FAMA[1] framework [14], providing support for the automated generation of random FMs.

5. Overview and open issues

Figure 1 illustrates an FM summarizing the factors (i.e. features) we considered for the selection of an adequate testing techniques, test adequacy criteria and test data generation mechanism. Cross-tree constraints are not included for simplicity. The studied features are mainly based on the decomposition of different testing techniques and adequacy criteria proposed by Binder [8], Kapfhammer [10] and Zhu et al. [17]. However, we remark that the usage of other classifications of software testing methods could also be feasible.

Filled features in Figure 1 illustrate the set of configurations that we evaluated as adequate for our proposal. These configurations helped us to refine our contribution and motivated some of the main research questions to be addressed in our future work, namely:

[1] http://www.isa.us.es/fama

- **What specific technique should be used to define the tests?** We concluded that we should use a black box, execution- and specification-based technique for unit testing. However, there still exist a vast array of specific techniques fulfilling these criteria such as *equivalence partitioning*, *boundary-value analysis* or *decision table testing* [11, 17]. Selecting one of these techniques and adapting it to our context of application is one of our major challenges.

- **What specific technique should be used to measure the adequacy of the tests?** We concluded that using a fault-/error-based test adequacy criterion is probably the best option for the context of our problem. However, once again we must still select one of the many available techniques to measure test adequacy using those criteria such as *error seeding*, *program mutation* or *perturbation testing* [17].

- **How should the test cases be specified?** The specification of the test cases should be rigorous and clear in order to provide all the information needed (e.g. inputs, expected outputs, etc.) in a widely accepted format. For that purpose, we plan to revise the literature of software testing and the related standards [1].

- **Which criteria should be followed to generate the input FMs?** The criteria for the generation of input data usually depend on the testing technique used. However, these techniques are often designed to work with numeric values and not with complex structures as FMs. Thus, the adaptation of these criteria to the context of FMs is a key challenge to decide the number, size and form of the input FMs to be used in our tests.

- **How could our proposal be integrated into the FAMA framework?** FAMA (FeAture Model Analyzer) is an extensible framework for the automated analysis of FMs integrating different logic paradigms and solvers. Our final goal is to integrate our proposal into this framework in order to enable the automated testing of the tools for the automated analysis of FMs.

References

[1] Draft IEEE Standard for software and system test documentation (Revision of IEEE 829-1998). Technical report, 2007.

[2] L. Baresi and M. Pezzè. An introduction to software testing. *Electronic Notes in Theoretical Computer Science*, 148(1):89–111, 2006.

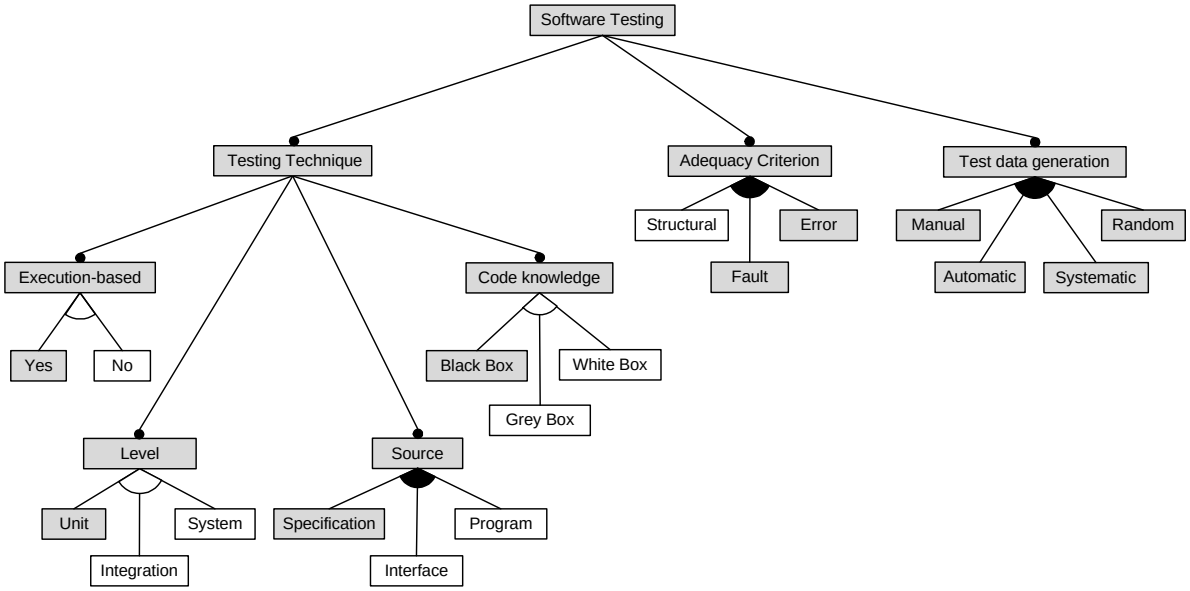

Figure 1: Feature-based overview of software testing methods

[3] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference, LNCS 3714*, pages 7–20, 2005.

[4] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.

[5] D. Benavides, A. Ruiz-Cortés, P. Trinidad, and S. Segura. A survey on the automated analyses of feture models. In *Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*, 2006.

[6] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. In *Managing Variability for Software Product Lines: Working With Variability Mechanisms*, 2006.

[7] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Using java csp solvers in the automated analyses of feature models. *LNCS*, 4143:389–398, 2006.

[8] R. V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[9] S. Fan and N. Zhang. Feature model based on description logics. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 1144–1151. 2006.

[10] G. Kapfhammer. *The Computer Science Handbook*, chapter Software Testing. CRC Press, 2nd edition, June, 2004.

[11] G. J. Myers and C. Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[12] S. Schach. Testing: principles and practice. *ACM Comput. Surv.*, 28(1):277–279, 1996.

[13] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software, in press*, 2007.

[14] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. Fama framework. In *Proceedings of the 12th International Software Product Line Conference (Tool demonstration)*, 2008.

[15] H. Wang, Y.F. Li, J. un, H. Zhang, and J. Pan. Verifying Feature Models using OWL. *Journal of Web Semantics*, 5:117–129, June 2007.

[16] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. In J. Davies, editor, *ICFEM 2004*, volume 3308, pages 115–130. Springer–Verlag, 2004.

[17] H. Zhu, P. Hall, and J. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, 1997.