Towards Multi-Objective Test Case generation for Variability-Intensive Systems [⋆]

Ana B. Sánchez, Sergio Segura and Antonio Ruiz-Cortés

Department of Computer Languages and Systems, University of Seville,
Av Reina Mercedes S/N Seville, Spain
{anabsanchez,sergiosegura,aruiz}@us.es

Abstract. Testing variability-intensive systems is a challenge due to the potentially huge number of derivable configurations. To alleviate this problem, many test case selection and prioritization techniques have been proposed with the aim of reducing the number of configurations to be tested and increasing their effectiveness. However, we found that these approaches do not exploit all available information since they are mainly driven by functional information such as the feature coverage. Furthermore, most of these works are focused on a single-objective perspective (e.g. features coverage), which could not reflect the real scenarios where several goals need to be met (e.g. features coverage and code changes coverage). In this context, we identify an important challenge, to take advantage of all available system information to guide the generation of test cases. As a first step towards a solution, we propose to study all this information with special emphasis on non-functional properties and address the test case generation as a multi-objective problem. Also, we describe some open issues to be explored that we hope have an important impact on future evaluations.

Keywords: Multi-objective test generation, extra-functional attributes

1 Introduction

The ability of a software system to be configurable and adaptable to different environments and users's needs is known as *variability*. Software applications exposing a high degree of variability are usually referred to as *Variability-Intensive Systems (VISs)*. Testing VISs is a challenge due to the potentially huge number of configurations under test. As an example, Debian Wheezy [1] has more than 37,000 packages that can be combined to form millions of different configurations of the operating system. This makes exhaustive testing of VISs infeasible, that is, testing every single configuration is too expensive in general. Furthermore, the software development process usually has to deal with limited available resources and time, which makes that testers must choose the right tests to find

[⋆] This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project TAPAS (TIN2012-32273) and the Andalusian Government projects THEOS (TIC-5906) and COPAS (P12-TIC-1867).

as many faults as possible at the right time. Typical approaches for VIS testing use a model-based approach [10], that is, they take an input feature model representing the VIS and return a valid set of features (i.e. configurations) to be tested, i.e. a test suite. In particular, two main strategies have been adopted: test case selection and test case prioritization. *Test case selection* [9, 11] aims at reducing the test space by selecting an effective and manageable subset of configurations to be tested. *Test case prioritization* [5, 12] schedules test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal, e.g. detecting faults as soon as possible. Both strategies are complementary and are often combined.

Selection and prioritization can be driven by multiple objectives. Typical ones are those based on combinatorial testing where test cases are selected in a way that guarantees that all combinations of t features are tested [11]. The properties of feature models have also been used to select and prioritize test suites in several approaches [6, 12]. Recently, some works have pointed out that not all features are equally relevant and take into account user preferences by assigning weights to the features [8]. Non-functional data such as the number of changes in the code or the number of known defects are recognized as good indicators of the error-proneness of software components [13, 15]. In fact, these data have been used in some works to accelerate the detection of faults by testing first those components with a higher fault propensity [4]. Analogously, non-functional data such as cost or number of users may be good indicators of the impact of certain faults. For instance, the number of downloads of a certain module in an open-source VIS could be used to prioritize test cases running first those exercising the most popular modules. This would allow us to start debugging and correcting faults as soon as possible minimizing the impact on the largest portion of users.

Challenges. Current VIS approaches for test case selection and prioritization are mainly driven by functional information and basic user preferences. However, although this has shown to be effective, these works fail to exploit non-functional information such as the number of code changes or the features size, which are good indicators of their fault propensity and the impact of faults [13]. As a related problem, most contributions for VIS testing use a single objective approach [5, 11], that is, they either aims to maximize a goal (e.g. feature coverage) or minimize another (e.g. suite size) but not both at the same time. Other works [14] combine several goals into a single objective by assigning them weights proportional to their relative importance. While this may be acceptable in certain scenarios, it may be unrealistic in others where users may wish to study the trade-offs among multiple objectives where all of them are equally important. Finally, most works on VIS testing evaluate their approaches in terms of performance (e.g. execution time) using synthetic data [7]. Thus, there is a serious lack of works reporting the results of testing approaches with real data which would provide researchers and practitioners with a better insight into the effectiveness of the testing techniques.

Solution overview. We address these challenges guiding the generation of test cases for VISs using both functional and non-functional attributes in a multi-objective perspective. More specifically, we inspired us on a case study based on a real VIS system presented by some of the authors in [13] to suggest several examples of objective functions using non-functional information. These functions consider the order of the test cases in the suite enabling a combined selection and prioritization approach. We plan to conduct our evaluation using real data extracted from the Drupal framework [13].

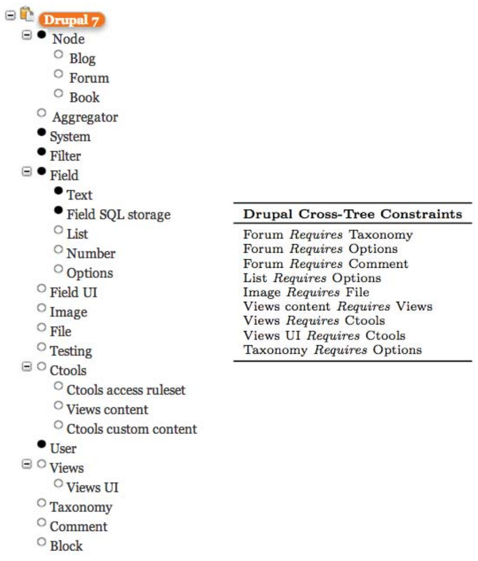
2 Motivating example

We propose to take advantage of the Drupal framework as a motivating VIS to illustrate and evaluate our approach. Drupal is a highly modular open-source web content management framework written in PHP with more than 14,000 modules that can be composed to form valid configurations of the system [3]. More importantly, the Drupal Git repository and the Drupal issue tracking systems are public and inexhaustible sources of valuable functional and non-functional information about the framework and its modules.

Fig 1 depicts the feature model of Drupal v7.23. Nodes in the tree represent features where a feature corresponds to a Drupal module. The model includes the core modules of Drupal, modelled as mandatory features (filled circle icon), plus some optional modules, modelled as optional features (empty circle icon). In addition to the tree, some cross-tree constraints are also depicted. These are of the form `X` requires `Y` which means that configurations including the feature X must also include the feature Y. A *Drupal configuration* is a combination of features consistent with the hierarchical and cross-tree constraints of the model. In total, the Drupal feature model represents 96,768 different Drupal configurations. For more information about how the model was constructed see [13].

In this paper, we propose modelling the non-functional data extracted from the Drupal framework as feature attributes in a feature model. The data were obtained from the Drupal git repository and the Drupal issue tracking system and refer to a period of one year, from September 30th 2012 to September 29th 2013 [13]. In particular, we propose using the following attributes:

– *Feature size.* Number of Lines of Code (LoC) of the source code associated to the feature. The sizes range between 326 LoC and 70,372 LoC (see [13]).
– *Code changes.* Number of commits made by the contributors to the feature in the Drupal git repository. The number of changes ranges from 0 to 43.
– *Single faults.* Number of faults in the feature reported by the users in the Drupal issue tracking system. Faults were collected for two consecutive versions of the framework v7.22 and v7.23. The number of faults found ranges between 0 and 157 (see [13]).
– *Integration faults.* List of features for which integration faults have been reported in the Drupal issue tracking system. For instance, the interaction between the modules `Blog` and `User` caused a fault in the Drupal system according to the case study in [13].



Drupal Cross-Tree Constraints
Forum *Requires* Taxonomy
Forum *Requires* Options
Forum *Requires* Comment
List *Requires* Options
Image *Requires* File
Views content *Requires* Views
Views *Requires* Ctools
Views UI *Requires* Ctools
Taxonomy *Requires* Options

Fig. 1. Drupal v7.23 feature model

– *Tests*. Number of assertions of the features, i.e. a group of methods that check for a condition and return a Boolean, according to the work [13].

3 Basic objective functions

In this section, we present some examples of objective functions based on non-functional attributes. An objective function is a function that represents a goal to optimize. The functions could receive two input parameters, an attributed feature model representing the VIS under test with the non-functional data as feature attributes (afm) and an ordered test suite (ts). As a result, the functions would return a real value measuring the quality of the suite with respect to the optimization goal. These functions would take into account the order of the test cases in the suite. We define the objective functions as follows:

Maximize the number of changes. Changes in the code are likely to introduce faults [15]. Thus, the number of changes in the code of a feature may be a good indicator of its error proneness and could help us to predict faults in later versions of the system. Considering this, we propose an objective function to maximize the number of changes covered by a test suite, giving a higher priority to those test cases including the features with a higher number of changes. Function 1 would calculate the number of code changes, indicated as features attributes, covered by the features included in each test case of the suite ts.

$$nchanges(afm, ts) \tag{1}$$

Maximize the number of faults. Earlier studies show that the detection of faults in an application can be accelerated by testing first those components that showed to be more error-prone in previous versions of the software [4]. Thus, we propose an objective function to maximize the number of faults detected by a test suite giving a higher value to those test cases that detect more faults faster. Function 2 would return the number of faults found in the features of a previous system version that are detected by the test cases of the suite ts.

$$nfaults(afm, ts) \tag{2}$$

Maximize the features size. The size of a feature, in terms of its number of Lines of Code (LoC), can provide a rough idea of the complexity of the feature and its error proneness [13]. Based on this, we propose a novel objective function to maximize the size of the features covered by a test suite, giving priority to those test cases covering higher portions of code faster. Function 3 would return the number of LoC of the features included in each test case of ts.

$$size(afm, ts) \tag{3}$$

Minimize the number of test cases. The number of test assertions in the features of a test case may be used to estimate the test case execution cost. Hence, we propose an objective function to minimize the execution cost of a suite by minimizing the number of assertions of its test cases. Thus, the test cases containing fewer assertions will have a higher priority to be tested. Function 4 would calculate the number of assertions of the features included in the tests cases of ts.

$$ntests(afm, ts) \tag{4}$$

4 Test case generation as a multi-objective problem

In Multi Objective Problems (MOP), algorithms have to optimize two or more objectives (sometimes conflicting). A typical example of a MOP is the Knapsack problem, where the goal is to minimize the weight of a sack while maximising the profit (by placing items into the sack). Considering the objective functions previously mentioned, we propose use them in a multi-objective perspective to drive the generation of effective test suites from a feature model representing a VIS. In a future assessment of our idea, we will have to focus our attention on the possible combinations of the different objective functions. As an example, we could maximize the number of changes covered and the number of faults detected by a suite plus minimize the size of the suite. Another example could be maximize the features size in order to test earlier the more complex configurations and minimize the number of test assertions to reduce the cost of testing.

5 Open issues

Throughout this work, we have identified some open issues to be explored in order to perform our approach, namely:

Combination of objective functions. The definition of new objective functions based on non-functional information could lead to a whole new world of possibilities to help us to optimize the test case generation of VISs. Thus, the key of this issue is to achieve the right combinations of the optimization goals.

Algorithms to resolve the multi-objective problem. In the literature there exist different algorithms to address the multi-objective problem. Among the most popular proposals are the evolutionary algorithms and the mathematical programming algorithms. Thus, the key of this open issue is to choose the best option to resolve the multi-objective problem.

References

1. Debian Wheezy. `http://www.debian.org/releases/wheezy/`, accesed April 2014.
2. D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analyses of feature models 20 years later: A literature review. *Information Systems*, 2010.
3. D. Buytaert. Drupal Framework.`http://www.drupal.org`, accessed in Oct 2013.
4. E. C. P. Cristian Simons. Regression test cases prioritization using failure pursuit sampling. In *ISDA*, 2010.
5. X. Devroey, G. Perrouin, M. Cordy, P.-Y. Schobbens, A. Legay, and P. Heymans. Towards statistical prioritization for software product lines testing. In *International Workshop on Variability Modelling of Software-Intensive*, 2014.
6. J. Ferrer, F. Chicano, and E. Alba. Evolutionary algorithms for the multi-objective test data generation problem. *Software Practice and Experience*, 2012.
7. J. Ferrer, P. Kruse, F. Chicano, and E. Alba. Evolutionary algorithm for prioritized pairwise test data generation. In *Genetic and Evolutionary Computetion Conference (GECCO)*, 2012.
8. M. F. Johansen, O. Haugen, F. Fleurey, A. G. Eldegard, and T. Syversen. Generating better partial covering arrays by modeling weights on sub-product lines. In *International Conference MODELS*, 2012.
9. R. Lopez-Herrejon, F. Chicano, J. Ferrer, A. Egyed, and E. Alba. Multi-objective optimal test suite computation for software product line pairwise testing. In *IEEE International Conference on Software Maintenance*, 2013.
10. S. Oster, A. Wubbeke, G. Engels, and A. Schurr. *A Survey of Model-Based Software Product Lines Testing*, chapter 13, pages 338–381. 2011.
11. G. Perrouin, S. Oster, S. Sen, J. Klein, B. Budry, and Y. le Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, 2011.
12. A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. A comparison of test case prioritization criteria for software product lines. In *IEEE International Conference on Software Testing, Verification, and Validation*, 2014.
13. A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. The drupal framework: A case study to evaluate variability testing techniques. In *8th International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, 2014.
14. S. Wang, S. Ali, and A. Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2013.
15. S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. In *Software Testing, Verification and Reliability*, 2012.