A Test Strategy for Configurable Software Systems Using Machine Learning

Fischer Ferreira

Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
fischerjf@dcc.ufmg.br

Abstract. Configurable software systems are software systems that can be adapted or configured according to a set of features to increase reuse and productivity. The testing process is essential because configurations that fail may potentially hurt user experience and degrade the reputation of a project. However, testing configurable systems is very challenging due to the number of configurations to run with each test, leading to a combinatorial explosion in the number of configurations and tests. Currently, several testing techniques and tools have been proposed to deal with this challenge, but their potential practical application remains largely unexplored. Our main goal is to investigate test strategies for configurable systems and provide a method to improve the approaches presented in the literature. To achieve this goal, we divided our work into seven steps. We conducted a systematic mapping study for understanding the state-of-the-art testing approaches for configurable systems. We are also proposing a dataset with 30 configurable software systems, an extensive test suite, failures found, and source code metrics. Furthermore, we performed a comparative study with test tools from different strategies. Finally, we introduce a new test strategy for configurable systems. At the moment, we have a prototype tool to support our test strategy for configurable systems. In future purposes, we want to evaluate the efficacy and efficiency of our approach to observing known failures. The preliminary results of the work indicate that testing configurable software systems is an interesting research topic with many opportunities and gaps in the literature. A method that benefits from metrics extracted from the source code of the configurable system could be a promising approach. We believe that our approach can better the efficiency and efficiencies of the state-of-the-art testing tools for configurable systems.

Keywords: Testing Configurable Systems · Machine Learning

1 Introduction

Configurable systems are software systems that can be adapted or configured according to a set of features (configuration options). Configurable systems offer numerous options to fit specific customer needs [18], and developers may activate or deactivate options to address a diversity of deployment contexts and usages.

To ensure that all configurations correctly compile, build, and run, developers usually spend considerable effort testing their systems because configurations that fail may hurt potential users, and degrade the reputation of a project [8].

Software testing is a key component for ensuring that all configurations work properly. However, testing configurable systems is more challenging than testing monolithic systems. While in monolithic software systems there is only one configuration, for configurable systems we need to run all tests in several different configurations, which leads to a combinatorial explosion of configurations and tests. Therefore, testing thoroughly, against all configurations, is a cost practice. Alternatively, a popular strategy used in industry is to run the tests for a default configuration. This approach is efficient, but it can miss bugs [7, 14].

Besides those two cases (testing only the default configuration or exhaustively testing all configurations), several approaches for testing configurable systems have been proposed [13, 15, 17]. Some of them consider only the Feature Model (FM)[1] [10–12] in order to generate products to be tested (Combinatorial Interaction Testing). However, they may explore configurations not reached by tests. Other approaches [17] take the code (test or app) into account in addition to FM, and dynamically explore all reachable configurations from a given test. However, such dynamic techniques only explore configurations related to testing. Even with a large number of testing techniques and tools for configurable systems [15, 17], their potential practical application remains mostly unexplored. We still lack a deeper understanding of the effectiveness of test strategies for configurations systems for finding failures in the configurable system with a test suite.

This paper presents a proposal for a Ph.D. thesis in Computer Science. Our main goal is to provide more effective strategies for testing configurable software systems. For this purpose, we designed seven study steps including systematic mapping study on detection of test strategies for configurable software systems, empirical investigation of detection strategies, proposing a dataset with 30 configurable software systems, a comparative study with test tools from different approaches, the definition of a new test strategy and implementation of this strategy. Finally, in future purposes, we want to evaluate the efficacy and efficiency of our approach to observing known failures. Additionally, we report the main results achieved with the study.

2 Background

2.1 Variability Encoding Overview

Configurable systems have long been studied by the software product line engineering community [16] and represent a configurable set of systems that share a common, managed set of options (or features) to address specific needs. Among the strategies to introduce variability in software systems, variability encoding has drawn practitioners' attention since developers only need to *annotate*

[1] The *Feature Model* defines valid feature combinations or configurations.

variation points on their existing systems. Thus, developers simply activate or deactivate features to address different deployment contexts. For short, while annotating variation points, developers should create a configuration file where they determine options that are going to be active in a target variation.

Listing 1 presents a fragment of source code in a configurable systems named Companies. In this example, the method *getTotal* returns a string containing a calculated value. If the feature TOTAL_WALKER is active, the method returns the value calculated by the TotalWalker class. If the feature TOTAL_REDUCER is active, the method returns the value calculated by the TotalReducer class. On the other hand, if the features TOTAL_WALKER and TOTAL_REDUCER are not active, no value will be calculated.

```
1  public String getTotal() {
2     String value = "";
3     if (Configuration.TOTAL_WALKER) {
4         TotalWalker walker = new TotalWalker();
5         walker.postorder(currentValue);
6         value = Double.toString(walker.getTotal());
7     } else if (Configuration.TOTAL_REDUCER) {
8         TotalReducer total = new TotalReducer();
9         double valueDouble = total.reduce(currentValue);
10        value = Double.toString(valueDouble);
11    }
12    return value;
13 }
```

Listing 1.1. Variability Encoding Example

2.2 Testing Configurable Systems

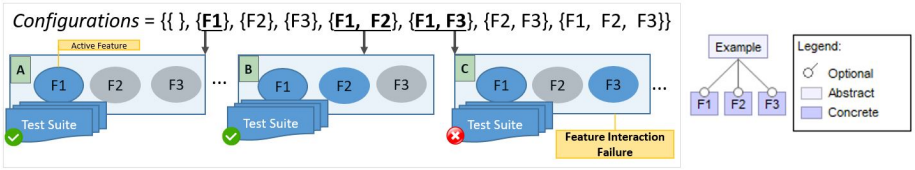

Fig. 1. Feature Interaction Failure

A major challenge for developers of configurable systems is to ensure that all configurations correctly compile, build, and run. Even when each feature performs well individually, interactions among them are prone to happen and introduce unexpected behavior to the system [2]. Feature interaction is a well-known problem for configurable systems and enforces the need for test suites covering all potential feature interactions. Figure 1 presents an example of 3 features of an illustrative configurable system and all the possibilities of combining the three features. In addition, Figure 1 shows the A, B, and C configurations submitted to the test suite. We see that both F1 and F3 features are active in the C configuration and that the test suite reports failure. However, the test suite does

not find failures for configurations A and B. The interaction between the F1 and F3 features causes the feature interaction failure. Above a certain amount of features, it is infeasible to test all possible feature combinations. This way, researchers and practitioners have to choose somehow the configurations they want to test. However, it is not trivial to know which priority configuration.

3 The Proposed Approach

This Ph.D. project aims to provide more effective strategies for testing configurable software systems. Therefore, the goal of this research is:

To evaluate testing strategies in the context of testing configurable systems in the perception of software developers and researchers with expertise in software testing to propose a new test strategy for configurable systems.

To achieve this goal, we propose a seven step research design (Figure 2). We discuss each study step as follows.

Fig. 2. Study Steps

Step 1: We performed a Systematic Mapping Study [5] to survey the literature on test strategies for configurable systems. Our goal was to identify and analyze tools reported and used in the literature for testing configurable software systems. This step provided state-of-the-art testing tools for configurable systems.

Step 2: We designed a dataset composed of 30 configurable systems with extensive test suite [6]. The goal was to establish a standardized dataset for executing and comparing test strategies for configurable systems. As a contribution, we provided the research community with the first dataset for configurable systems with an extensive test suite and reported feature interaction failures.

Step 3: We evaluated testing tools in two comparative studies. In the first study [5], we conducted a comparative study to evaluate two testing tools found (VarexJ and SPLat). We decided to focus our research on tools that implements dynamic testing techniques, that is, to test all possible configurations of the configurable software systems. In the second study, we designed and performed a comparative empirical study of four state-of-the-art combinatorial interaction testing tools (IncLing [1], ICPL [11], Chvatal [10], and Random [4]). This strategy uses techniques to test only a representative subset of all possible configurations.

Step 4: We investigated the dispersion of failures found by the test suites over the dataset subjects. To support our analysis, we identified classes and features in each system with at least one failure.

Step 5: We are working on a machine learning algorithm to support the identification of priority configurations for testing. Based on software metrics,

a machine learning based strategy provides the capability of understanding the properties present on the configurable systems (CS), and based on them, it performs failure detection. Figure 3 presents the steps in the proposed method divided into two main parts Model Building and Model Application.

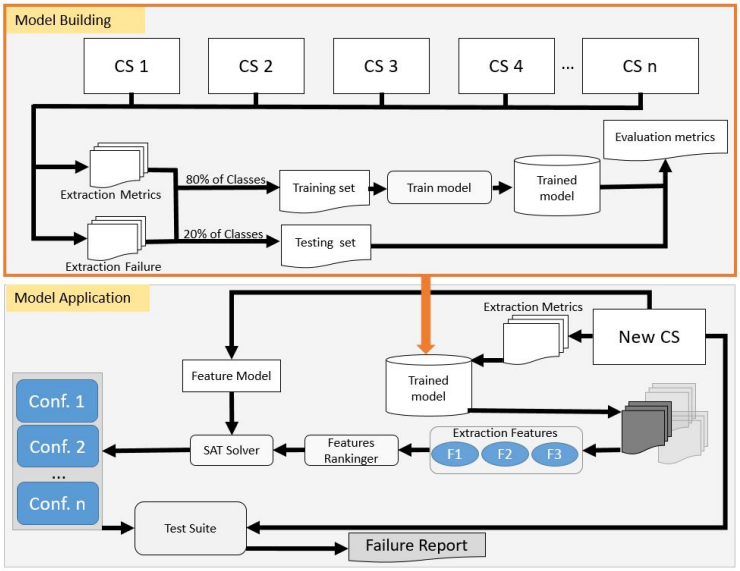

Fig. 3. Proposed Strategy

Model Building: The model building demonstrates the training part of our model using a machine learning algorithm (classifier). Each instance is formed by a set of metrics (Extraction Metrics) and a label (Extraction Failure) to be predicted 1 or 0 indicating the occurrence or not of the failure, respectively.
Model Application: Using the trained model, it is possible to discover the classes of the configurable system source code that are prone to failures. Once we discover these parts of the source code, it is possible to determine the related active features. Using a SAT Solver, we will assemble all the configurations in which the features are present. Finally, for these configurations found, we submit the test suite.

Step 6: We plan to integrate our tool with FeatureIDE. FeatureIDE is an Eclipse-based IDE that supports all phases of the development of configurable systems [19].

Step 7: After the development of our testing strategies in Step 6, we can conduct experiments to evaluate the effectiveness and efficiency of our approach to finding failure in configurable systems.

The steps 1 to 5 (colored in green in Figure 2) have been completed. We are currently executing step 6, the development of a testing tool for configurable systems (colored in yellow in Figure 2). Step 7 (colored in red in Figure 2) is

related to the evaluation of the proposed test strategy and tool. Therefore, this final step is scheduled to be executed after step 6 is complete.

4 Preliminary Results

In Step 1, we conducted a Systematic Mapping Study [5] on testing tools for configurable systems. We obtained a set of 34 tools. We defined these characteristics based on previous studies: graphical user interface, prototype, online, plugin, free for use, open-source, user guide available, solution examples available, supported language, tested, code level test, feature model check, feature interaction, and tool Web site. We also compared tools in terms of test strategies. We found the following test strategies for configurable systems in the analyzed papers. Sound Testing Techniques is used to test all possible configurations of the configurable software systems. Combinatorial Interaction Testing uses techniques to test only a representative subset of all possible configurations.

In Step 2, we first proposed a test-enriched dataset with 30 configurable software systems. This dataset was created based on a literature review and further implementation of test suites to improve code coverage. We created 727 test cases for 20 configurable systems and further 90 tests for the three systems that we extended. Our dataset has a total of 3 182 tests. As means for quality assurance, we created tests until the have a coverage of 70% and kill at least 40% of mutants. The final dataset has 30 systems varying in domains, size, variability, and test suite size. We are confident that this dataset will benefit practitioners and also be useful for researchers comparing testing approaches and methods.

In Step 3, we observed that VarexJ and SPLat presented distinct results for efficiency while testing the target systems. VarexJ found more errors than SPLat for the majority of the target systems. In the second part of our study, we also conducted a comparative study of tools (IncLing [4], ICPL [11], Chvatal [10], and Random [4]) to compute recall, precision, and F-measure.

In Step 4, our failure report shows that 254 failures occurred in 151 different configurations, and that 70 test cases failed. All failures that we found involve more than one feature, so, we call them feature interaction failures. In addition, 14 out of the 30 systems of our test-enriched dataset have failures. Our analysis of the dispersion of failures shows that in 11 of the 14 systems that failed, all failures are in the same class. In other words, feature interaction failures are normally concentrated in one (or few) class(es) of configurable systems. We hope that this failure report and these preliminary analyses support practitioners on understanding the failures that happen in their systems.

5 Related Work

Many approaches have been proposed in the literature to test configurable systems [15, 17]. Some approaches generate configurations for testing, considering only the feature model [15]. Other strategies implement a dynamic technique that explores all reachable configurations from a given test [17]. Our method

ranks software components identified priority configurations for testing. Some approaches in the literature that use machine learning techniques to prioritize configurations for testing configurable systems consider only the Feature Model [3]. However, the main difference regarding our approach is that through our machine learning, we can find the priority points in the source code for testing, and we discover configurable that manipulate these components of the source code. Other strategies use software metrics and machine learning to identify failures in monolithic systems [9]. We propose to use the technique to detect failures in configurable systems. We expect that this approach is useful for testing configuration systems with greater efficiency and efficiency than state-of-the-art.

6 Conclusion and Future Work

This paper presents a proposal for a Ph.D. thesis in Computer Science that focuses on the test strategy for configurable systems. We aim to introduce new test strategies with machine learning algorithms to support the identification of priority configurations for testing, in the expectation of proposing a test strategy with greater effectiveness and efficiency than the tools present in the literature. We present an overview of the thesis and also discuss some preliminary results of previous studies. As future work, we intend to conclude the last study steps, publish the study results as papers for the community, complete the study in parallel with Step 6 and Step 7, and finish this Ph.D.'s in March 2021 with the thesis defense.

References

1. Al-Hajjaji, M., Krieter, S., Thüm, T., Lochau, M., Saake, G.: Incling: Efficient product-line testing using incremental pairwise sampling. In: Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. pp. 144–155. GPCE 2016, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2993236.2993253, http://doi.acm.org/10.1145/2993236.2993253
2. Apel, S., Kolesnikov, S., Siegmund, N., Kästner, C., Garvin, B.: Exploring feature interactions in the wild: The new feature-interaction challenge. In: In 5th International Workshop on Feature-Oriented Software Development (FOSD). pp. 1–8 (2013)
3. Devroey, X., Perrouin, G., Cordy, M., Samih, H., Legay, A., Schobbens, P., Heymans, P.: Statistical prioritization for software product line testing: an experience report. Software & Systems Modeling (SSM) pp. 153–171 (2017)
4. FeatureIDE: featureide tool, https://featureide.github.io/, Accessed 15-nov-2019
5. Ferreira, F., Diniz, J.P., Silva, C., Figueiredo, E.: Testing tools for configurable software systems: A review-based empirical study. In: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems. pp. 1–10 (2019)
6. Ferreira, F., Vale, G., Figueiredo, E., Diniz, J.P.: On the proposal and evaluation of a test-enriched dataset for configurable systems. In: Proceedings of the 14th

International Working Conference on Variability Modelling of Software-Intensive Systems. pp. 1–10. VAMOS '20 (2020)
7. Greiler, M., Deursen, A.v., Storey, M.A.: Test confessions: A study of testing practices for plug-in systems. In: Proceedings of the International Conference on Software Engineering. pp. 244–254. IEEE (2012)
8. Halin, A., Nuttinck, A., Acher, M., Devroey, X., Perrouin, G., Baudry, B.: Test them all, is it worth it? assessing configuration sampling on the jhipster web development stack. Empirical Software Engineering (ESE) pp. 674–717 (2019)
9. Jiang, Y., Cuki, B., Menzies, T., Bartlow, N.: Comparing design and code metrics for software quality prediction. In: Proceedings of the 4th international workshop on Predictor models in software engineering. pp. 11–18 (2008)
10. Johansen, M.F., Haugen, Ø., Fleurey, F.: Properties of realistic feature models make combinatorial testing of product lines feasible. In: 14th International Conference on Model Driven Engineering Languages and Systems (MODELS). pp. 638–652 (2011)
11. Johansen, M.F., Haugen, Ø., Fleurey, F., Eldegard, A.G., Syversen, T.: Generating better partial covering arrays by modeling weights on sub-product lines. In: International Conference on Model Driven Engineering Languages and Systems. pp. 269–284. Springer (2012)
12. Krieter, M., T., S.T., Saake, L.M., Al-Hajjaji, G.: Incling: efficient product-line testing using incremental pairwise sampling. In: 15th Proceedings of the ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE). pp. 144–155 (2016)
13. Liebig, J., von Rhein, A., Kästner, C., Apel, S., Dörre, J., Lengauer, C.: Scalable analysis of variable software. In: Proceedings of the Foundations of Software Engineering. pp. 81–91. ACM (2013)
14. Machado, I., McGregor, J., Cavalcanti, Y., Almeida, E.: On strategies for testing software product lines: A systematic literature review. Information and Software Technology (IST) pp. 1183 – 1199 (2014)
15. Medeiros, F., Kästner, C., Ribeiro, M., Gheyi, R., Apel, S.: A comparison of 10 sampling algorithms for configurable systems. In: Proceedings of the International Conference on Software Engineering. pp. 643–654. ACM (2016)
16. Pohl, K., Metzger, A.: Software product line testing. Information and Software Technology (IST) pp. 78–81 (2006)
17. Souto, S., d'Amorim, M., Gheyi, R.: Balancing soundness and efficiency for practical testing of configurable systems. In: In 39th Proceedings of the International Conference on Software Engineering (ICSE). pp. 632–642 (2017)
18. Svahnberg, M., Van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Software: Practice and experience (SPE) pp. 705–754 (2005)
19. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming (SCP) pp. 70–85 (2014)