Reasoning about Product-Line Evolution using Complex Feature Model Differences

Johannes Bürdek[1], Timo Kehrer[2], Malte Lochau[2], Dennis Reuling[3], Udo Kelter[3], Andy Schürr[2]

Abstract: In this work, we report about research results on the Reasoning about Product-Line Evolution using Complex Feature Model Differences, published in [Bü15]. A (software) product-line constitutes a long-term investment and, therefore, has to undergo continuous evolution to meet constantly changing requirements. Initially, product-line evolution leads to changes of the variability model, typically a feature model, due to its central role in the product-line paradigm. Thus, product-line engineers are often faced with the problems that (1) feature models are changed ad-hoc without proper documentation, and (2) the semantic impact of changes is unclear. We propose a comprehensive approach to tackle both challenges. For (1), our approach compares the old and new version of the diagram representation of a feature model and specifies the changes using edit operations on feature diagrams. For (2), we propose a novel approach for reasoning about the semantic impact of diagram changes. We present a set of edit operations on feature diagrams, where complex operations are primarily derived from evolution scenarios observed in a real-world case study. We demonstrate the applicability of our approach with respect to the case study, and evaluate its scalability concerning experimental data sets.

Keywords: Software Evolution, Model-Driven Engineering, Software Product Lines, Features

1 Summary

Software product line engineering constitutes a well-suited paradigm to cope with the inherent diversity and long-living nature of modern software systems. During *domain engineering*, the scope of a product line is defined in a variability model.The FODA method is a widely used and extensively investigated variability modeling approach. A *feature model* defines the set of supported features and valid feature combinations, also called configuration space. A *feature diagram* is a visual notation of a feature model. It represents the set of domain features as nodes and contains further notational constructs to denote different kinds of (logical) constraints among those features. Although product lines are, by design, well-prepared to meet diverse customers' needs, one cannot always anticipate all future requirements. For example, new features or new valid feature combinations could be added. In this paper we deal with changes in the feature model itself. The modifications may include adding and removing features and constraints as well as arbitrary complex

[1] Technische Universität Darmstadt, Fachgruppe Echtzeitsysteme, Merckstr. 25, 64283 Darmstadt, Email: {buerdek,lochau,schuerr}@es.tu-darmstadt.de
[2] Universität Siegen, Praktische Informatik, Hölderlinstr. 3, 57076 Siegen, Email: {kehrer,dreuling,kelter}@informatik.uni-siegen.de

Acknowledgements: This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP 1593: Design For Future - Managed Software Evolution.

changes to the feature diagram.
Feature model modifications are often conducted without proper documentation, e.g., by recording somehow the changes. Even if so, it is typically not obvious how changes in a feature diagram impact the configuration space. Certain changes can be classified w.r.t. their effect on the set of valid feature combinations [TBK09]: A *refactoring* leaves this set unchanged, i.e., it transforms a feature diagram into a semantically equivalent diagram. A *generalization* (*specialization*) enlarges (shrinks) the set of valid feature combinations. Knowing these effects is very useful since further development work, e.g., adapting the set of test data for the SPL, can be omitted completely in case of refactorings or simplified in the other cases. The changes whose effect can be classified in this way are typically „small" and „local" in the sense that only a few nodes and edges in the feature diagram are affected and that these items are connected. If several of these small changes are applied to a model then their combined effect, as visible by comparing the old and new version of the diagram, mostly cannot be classified using one of the above categories, i.e. the overall difference has to be classified as *arbitrary edit* [TBK09]. It is desirable to split the complete difference between two feature diagrams into smaller changes, each change being the effect of a so-called edit step. An edit step consists of an edit operation and concrete arguments. The edit steps should have two properties: (a) they use edit operations which are offered by typical visual editors for feature diagrams or they are local reorganisations which are often applied by developers, and (b) they can be classified as refactoring, generalization or specialization, possibly depending on conditions on the context of their application.
The main contribution of [Bü15] is an approach for comparing two feature diagrams and for automatically splitting the difference between them into edit steps which have the above properties (a) and (b). Our differencing approach is state-based, i.e., we assume the old and new version of a feature model to be available, but no information about editing processes. A part of our contribution is a catalogue of edit operations and refactorings typically applied to feature models. Amongst others, this catalogue was derived from evolution scenarios observed in a real-world case study from the automation engineering domain. Concerning the classification of edit operations and their concrete applications, we propose a logic-based formal framework for reasoning about their semantic impact. We have evaluated the applicability and feasibility of our approach and provide results obtained from an empirical evaluation with artificial data sets in order to demonstrate the scalability of the approach when applied to larger feature models. As future work we plan to use the derived catalogue of edit operations for mutation testing [Re15].

References

[Bü15] Bürdek, Johannes; Kehrer, Timo; Lochau, Malte; Reuling, Dennis; Kelter, Udo; Schürr, Andy: Reasoning About Product-line Evolution Using Complex Feature Model Differences. Autom. Softw. Eng., 23(4):687–733, 2016 (first online: 12 October 2015).

[Re15] Reuling, Dennis; Bürdek, Johannes; Rotärmel, Serge; Lochau, Malte; Kelter, Udo: Fault-based Product-line Testing: Effective Sample Generation Based on Feature-diagram Mutation. In: Proc. of SPLC. SPLC '15. ACM, pp. 131–140, 2015.

[TBK09] Thüm, Thomas; Batory, Don; Kästner, Christian: Reasoning About Edits to Feature Models. In: Proceedings of the 31st International Conference on Software Engineering (ICSE). IEEE Computer Society, pp. 254–264, 2009.