SBSE4VM: Search Based Software Engineering for Variability Management

Roberto E. Lopez-Herrejon
Systems Engineering and Automation
Johannes Kepler University Linz, Austria
roberto.lopez@jku.at

Alexander Egyed
Systems Engineering and Automation
Johannes Kepler University Linz, Austria
alexander.egyed@jku.at

***Abstract*—SBSE4VM is an ongoing Lise Meitner Fellowship project sponsored by the Austrian Science Fund (FWF) that runs for two years. The driving goal of the project is to explore the application of Search Based Software Engineering techniques to reverse engineer, evolve, and fix inconsistencies in systems with variability.**

***Index Terms*—Software Product Lines; Feature Orientation; Product Line Evolution; Search Based Software Engineering; Fixing Inconsistencies**

I. Introduction

Because of economical, technological and marketing reasons today's software systems are more frequently being built as *Software Product Lines (SPLs)* where each product implements a different combination of *features* – increments in program functionality [1]. The effective management and realization of *variability* – the capacity of software artifacts to vary [2] – is crucial to reap the benefits of SPL practices. Among these benefits are: increased software reuse, faster and easier product customization, and reduced time to market.

An important challenge in *Software Product Line Engineering (SPLE)* is guaranteeing that all the desired software products (i.e. distinct feature combinations) can in fact be realized from the existing set of artifacts in a SPL. Verifying this guarantee is a non-trivial task because of the typically large number of software products and the inherent complexity of the artifacts involved. This guarantee is even more important when considering variability evolution, whether in effectively managing changes to an existing SPL or in extracting variability out of existing system variants – that may or may not have some common development history – to create a new SPL. Consequently, when this guarantee does not hold, it is vital to provide software designers with sets of fixes or other variability modeling and implementation alternatives to deal with this lack.

Search-Based Software Engineering (SBSE) is an emerging discipline that focuses on the application of search-based optimization techniques to problems in Software Engineering [3]. Among the techniques SBSE relies on are basic search algorithms such as hill climbing or simulated annealing, and evolutionary computation techniques such as genetic algorithms or genetic programming. SBSE techniques permit the elicitation of multiple software designs or module refactoring that can be evaluated and assessed using different software quality metrics as fitness metrics. Additionally, such techniques are generic, robust and scalable making them suitable for addressing the challenges of variability management. The driving goals of the project is thus to perform a comprehensive review and analysis of SBSE techniques for variability management and to provide adequate tool support to empirically validate their effectiveness and adequacy in various case studies.

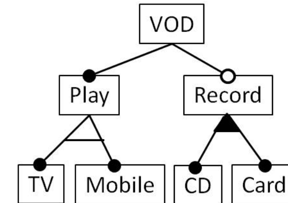

Fig. 1: Feature Model Example

II. Variability Modeling Basics

Variability modeling specifies all meaningful and legal feature combinations in a SPL, and its de facto standard are *Feature Models (FM)* [4]. Figure 1 shows a feature model example of a hypothetical product line of Video-On-Demand systems. Features are depicted as labeled boxes and are connected with lines to other features with which they relate, collectively forming a tree-like structure. The root feature of a SPL is always included in all programs, in this case the root feature is `VOD`. A feature can be classified as: *mandatory* if it is part of a program whenever its parent feature is also part (e.g. `Play`), and *optional* if it may or may not be part of a program whenever its parent feature is part (e.g. `Record`). Mandatory features are denoted with filled circles while optional features are denoted with empty circles both at the child end of the feature relations denoted with lines. Features can be grouped into: *inclusive-or* relation whereby one or more features of the group can be selected and *exclusive-or* relation where exactly one feature can be selected. These relations are depicted as filled arcs and empty arcs respectively.

In Figure 1, feature `Record` with its children features `CD` and `Card` is an example of inclusive-or, whereas feature `Play` with children `TV` and `Mobile` form an exclusive-or. Additionally, there are constraints called *cross-tree constraints* that cannot be depicted directly on a feature diagram and

represent more complex relations between features [4].

In total, this feature model denotes 8 valid feature combinations. An example is the program that can only play videos on TV is defined with features `VOD`, `Play` and `TV`. As another example, the program (`VOD` always included) that play videos (`Play`) on mobile sets (`Mobile`) that can be recorded (`Record`) on a memory card (`Card`). It should be noted that typical feature models contain hundreds if not thousands of features yielding large numbers of potential feature combinations [5].

III. Project Overview

The SBSE4VM[1] project started in August 2012 and will run until July 2014. The funding is approximately 153K+ Euros provided by the Austrian Science Fund (FWF Der Wissenschaftsfonds) agency under the Lise Meitner Program[2]. This is an ongoing funding program that promotes mobility and collaboration between Austria and the rest of the world. It supports foreign researchers willing to pursue one or two years of work at an Austrian research institution. The goals of the Lise Meitner program are: *i)* strengthening of the quality and the scientific know-how of the Austrian scientific community, and *ii)* creating international contacts. The first author is the recipient of the fellowship and the second author is the head of the host Austrian institute.

IV. Scientific Problems Addressed

In this section we provide a more detailed overview of the three problems that the proposed work aims to address.

A. Fixing Inconsistencies in the Presence of Variability

The main source of inconsistencies is the discrepancy between the variability that is modeled, using a feature model, and the variability that is actually realized. For example, consider the consistency rule that if a sequence diagram has a message `m` targeting an object of class `C`, then the class diagram of class `C` must contain method `m`. Figure 2 provides an illustration of this rule. Consider message `store` in Figure 2(a) that has been identified as belonging to feature `CD` of our feature model in Figure 1. For this rule instance to be consistent there must be a method `store` defined in class `Streamer` at least in every possible feature combination where feature `CD` is selected. The question is: *in what features should method `store` be defined to make this rule instance consistent?*

Let us enumerate the possibilities: 1) in feature `CD`, 2) in feature `Record`, 3) one definition in feature `TV` and one in feature `Mobile`, 4) in feature `Play`, and 5) in feature `VOD`. This set of possibilities can be expanded by three additional considerations. The first consideration is the assumption that the inconsistency is caused by the use of an incorrect message name. In our case, instead of message `store` perhaps the designer intended to use message `start`. If so, this method is defined in feature `Record` in Figure 2(b), and

[1]Project website http://www.sea.uni-linz.ac.at/sbse4vm/

[2]http://www.fwf.ac.at/en/projects/meitner.html

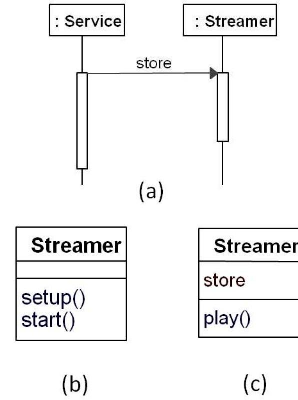

Fig. 2: Inconsistency Fixing Example: (a) Feature CD, (b) Feature Record, (c) Feature Play

because feature `Record` is always selected when feature `CD` is selected then the rule instance would be consistent. The second consideration is the assumption that the inconsistency is caused by the incorrect target of the message. This would mean that other classes, besides `Streamer`, can be the target and thus they would need to define the required method. The third consideration is the assumption that the inconsistency is caused by incorrect feature ascription of the message, meaning that perhaps the message should be in a different feature other than `CD`. Notice that these latter two assumptions open up new sets of fixing alternatives.

Additionally, the interplay between consistency rule instances must also be taken into account. This is because it may be the case that fixing a rule instance may cause an inconsistency in another rule instance. For example, let us assume that class diagrams must adhere to a consistency rule that states that classes must have distinct names for their attributes and methods. Take now feature `Play` as shown in Figure 2(c). The fifth alternative mentioned above for fixing our message rule instance, adding the method `store` in feature `Play`, would indeed make consistent the message rule instance but at the price of making inconsistent the distinct names rule instance of class `Streamer`.

In summary, even for this simple example, several fixing alternatives have to be elicited and carefully analyzed. More realistic scenarios typically contain dozens of consistency rules, across multiple software artifacts in hundreds or even thousands of feature combinations. Clearly, automated and robust support is crucially needed to cope with this complexity. This is precisely where we argue that SBSE techniques can be leveraged. Ultimately, the selection among the fixing choices will be done by the software designer. SBSE techniqes can be used to obtain and represent the different fixing alternatives, and to flexibly quantify (through different fitness metrics) their suitability and thus help the designer with his/her selection.

B. Reverse Engineering of Variability

The most common scenario for SPL development in practice is depicted on the left part of Figure 3. It shows several

related systems (i.e. they offer some similar functionality) that are developed mostly independently, although they may have shared some artifacts at some point in their history. This approach is called *product-centric* [6], and works fine upto a certain number of distinct products depending on product domains, the development organization and its software engineering practices. After such number, simply adding new independently-developed systems is no longer feasible either because of managerial, economical or technical reasons. It is then that a SPL approach is the premier alternative for effectively coping with the complexity of the variability in the existing products.

The transition to a SPL approach is a not a minor undertaking because it requires a significant investment to identify, extract, and reify the variability across all the artifacts involved as depicted on the right part of Figure 3. In this figure, the software artifacts (e.g. requirements or implementation) capture the variability identified in accordance with the feature model. The proposed work focuses on two crucial issues of reverse engineering in this approach: feature model extraction and feature refactoring.

Extracting feature models that capture the variability present in the product-centric products is a cornerstone in SPL development. Several techniques have been proposed to distill feature models from scripts or other documentation [7], [5]; however, they rely heavily on the designer's intuition or domain knowledge. In contrast, our work proposes using feature sets – the combination of features as shown in [8] – in tandem with consistency constraints for the extraction of feature models. It should be noted that feature sets can be incomplete and uncertain; for instance, the developer may not know all the features in each product or decide not to integrate all products at once but instead to take a *reactive approach* (see [6]) whereby products are incrementally integrated into the SPL infrastructure that is being developed. It is these incompleteness and uncertainty that make SBSE techniques an appealing option because they permit realizing different alternative feature models, to represent the desired feature sets, that can be fine tuned by using consistency constraints as a basis for metrics to quantify their suitability.

Feature refactoring [9], can be regarded as a form of software clustering along feature functionality. Its ultimate goal is to modularize features such that they can be composed or selected in all the combinations denoted by a feature model and yield working systems. Like before, current approaches in this topic make a hefty demand of the developer's domain knowledge and assume some basic familiarity with the implementation [10], [11], [9]. But most importantly; in these pieces of work, only one refactoring alternative is ever considered. In this regard, we propose to exploit SBSE techniques to provide developers with multiple refactoring alternatives that take into account also software quality measurements in the form of extensions and adaptations to standard Object-Oriented metrics and feature model metrics [12].

C. Variability Evolution

Like any other software project, a Software Product Line if successful is deemmed to evolve. But in contrast with standard traditional single systems, evolution can occur in either of the two core SPL processes (domain engineering or application engineering) that have to be consistent with each other. Next we present the main evolution scenarios that our work proposes to study.

Feature model evolution. Of crucial importance is the evolution of the feature models because they determine the set of valid features combinations that must be realizable. An example of feature model evolution is changing the type of relation among features. For instance, assume that now the VOD systems of the product line can play both on TV and mobile sets. This means that now there is an inclusive-or relation between feature `Play` with features `TV` and `Mobile`. This seemingly simple change actually creates other four new possible feature combinations in addition to those 16 denoted by our feature model of Figure 1. Other common feature model changes are adding and removing features, moving features across the feature hierarchy, and changes in cross-tree constraints. All these changes may turn consistent rule instances into inconsistent, delete rule instances, or create new rule instances that need to be validated.

Evolution of variability realization. Software artifacts with variability are also subject to evolution. A first case is when evolution does not modify the semantics of features. For example, standard refactorings such as method renaming could be used to improve factors like code readability and their changes remain contained at the syntactic level. A second case is when feature semantics is altered; for example, by moving a code or model fragment that realizes some functionality from one feature to another.

Both types of scenarios have an impact on the consistency of the SPL because they may trigger the creation of new rule instances, and deletions or modifications of existing ones. SBSE techniques can be leveraged here to elicit the consistency impact that different evolution scenarios may produce. In this way, designers could analyze the ripple effect of changes and their impact before they are actually committed or realized.

V. PROJECT GOALS

The overall goal of the proposed work is to perform a thorough evaluation of the potential of SBSE techniques for supporting the management of consistency and evolutionary problems in Software Product Line development. The proposed project aims to achieve the three following specific goals:

- Perform a comprehensive review and analysis of SBSE techniques, with special focus on Genetic Algorithms and Genetic Programming, to identify the possible alternatives for tackling the identified SPL development problems and characterize their pros, cons and trade-offs involved.
- Implement adequate tooling support that facilites the application of the identified SBSE techniques.

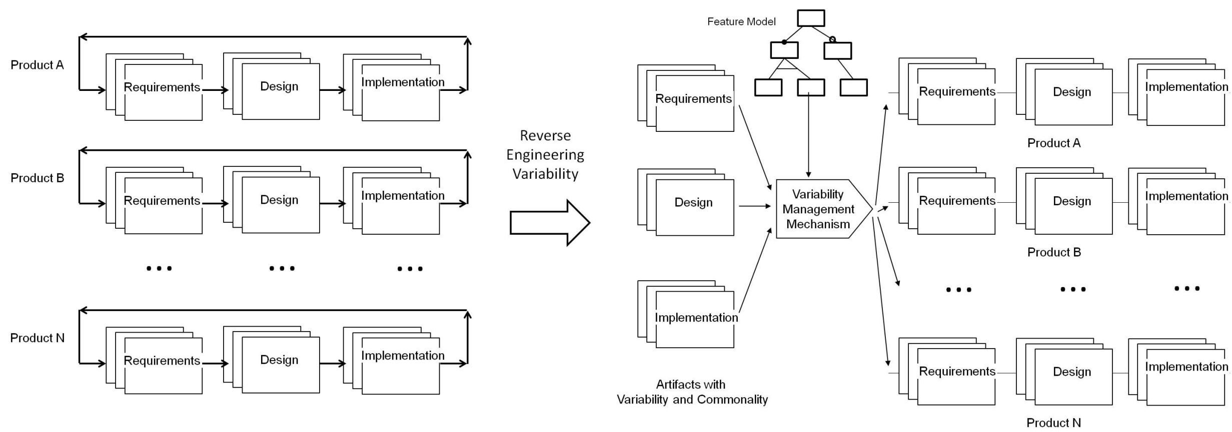

Fig. 3: Reverse Engineering Variability

- Conduct an assessment of the selected SBSE techniques with several cases studies of distinct sizes and application domains.

VI. Project Relevance and Related Projects

The three main topics of the proposed work – inconsistency fixing, reverse engineering, and evolution – are themes relevant to the CSMR community. We complimentary study the impact of variability on these three topics.

This project is a result of our previous EU-funded project C2MV2 [13], which studied how variability impacted consistency checking and provided mechanisms to detect inconsistencies. The current project goes a step further into fixing the inconsistencies once they have been detected.

Search Based Software Engineering is a thriving research area [14]. There have been several past and ongoing projects that have exploited this paradigm. For example, the EU-Funded project *EvoTest Evolutionary Testing for Complex Systems* [15] applied SBSE testing techniques to various domains such automotive. A recent project *FITTEST* [16], aims at applying SBSE techniques to testing the Future of the Internet. A recent UK-lead project has proposed a new vision, the GISMOE challenge [17], whereby the software non-functional properties are optimized via SBSE techniques.

Acknowledgements

This research is partially funded by the Austrian Science Fund (FWF) project P21321-N15 and Lise Meitner Fellowship M1421-N15.

References

[1] D. S. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling step-wise refinement," *IEEE Trans. Software Eng.*, vol. 30, no. 6, pp. 355–371, 2004.

[2] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques," *Softw., Pract. Exper.*, vol. 35, no. 8, pp. 705–754, 2005.

[3] M. Harman, P. McMinn, J. Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Empirical Software Engineering and Verification*, ser. Lecture Notes in Computer Science, B. Meyer and M. Nordio, Eds. Springer Berlin Heidelberg, 2012, vol. 7007, pp. 1–59. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25231-0_1

[4] D. Benavides, S. Segura, and A. R. Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, 2010.

[5] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *ICSE*, R. N. Taylor, H. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 461–470.

[6] C. W. Krueger, "Easing the transition to software mass customization," in *PFE*, ser. Lecture Notes in Computer Science, F. van der Linden, Ed., vol. 2290. Springer, 2001, pp. 282–293.

[7] K. Czarnecki and A. Wasowski, "Feature diagrams and logics: There and back again," in *SPLC*. IEEE Computer Society, 2007, pp. 23–34.

[8] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Reverse engineering feature models from programs' feature sets," in *WCRE*, M. Pinzger, D. Poshyvanyk, and J. Buckley, Eds. IEEE Computer Society, 2011, pp. 308–312.

[9] R. E. Lopez-Herrejon, L. Montalvillo-Mendizabal, and A. Egyed, "From requirements to features: An exploratory study of feature-oriented refactoring," in *SPLC*. IEEE, 2011, pp. 181–190.

[10] J. Liu, D. S. Batory, and C. Lengauer, "Feature oriented refactoring of legacy applications," in *ICSE*, L. J. Osterweil, H. D. Rombach, and M. L. Soffa, Eds. ACM, 2006, pp. 112–121.

[11] S. Trujillo, D. S. Batory, and O. Díaz, "Feature refactoring a multi-representation program into a product line," in *GPCE*, S. Jarzabek, D. C. Schmidt, and T. L. Veldhuizen, Eds. ACM, 2006, pp. 191–200.

[12] E. Bagheri and D. Gasevic, "Assessing the maintainability of software product line feature models using structural metrics," *Software Quality Journal*, 2010.

[13] R. E. Lopez-Herrejon and A. Egyed, "C2mv2: Consistency and composition for managing variability in multi-view systems," in *CSMR*, T. Mens, Y. Kanellopoulos, and A. Winter, Eds. IEEE Computer Society, 2011, pp. 347–350.

[14] Y. Zhang, "Repository of publications on search based software engineering. http://crestweb.cs.ucl.ac.uk/resources/sbse_repository."

[15] "Evotest evolutionary testing for complex systems," http://evotest.iti.upv.es/.

[16] T. E. J. Vos, P. Tonella, J. Wegener, M. Harman, W. Prasetya, E. Puoskari, and Y. Nir-Buchbinder, "Future internet testing with fittest," in *CSMR*, T. Mens, Y. Kanellopoulos, and A. Winter, Eds. IEEE Computer Society, 2011, pp. 355–358.

[17] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark, "The GISMOE challenge: Constructing the pareto program surface using genetic programming to find better programs (keynote paper)," in 27th *IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, Essen, Germany, September 2012.

[18] T. Mens, Y. Kanellopoulos, and A. Winter, Eds., *15th European Conference on Software Maintenance and Reengineering, CSMR 2011, 1-4 March 2011, Oldenburg, Germany*. IEEE Computer Society, 2011.