Towards identifying evolution smells in Software Product Lines

Klaus Schmid[1], Rainer Koschke[2], Christian Kröher[1], Dierk Lüdemann[2]

[1]University of Hildesheim, Institute of Computer Science, Software Systems Engineering, Marienburger Platz 22, 31141 Hildesheim, Germany
{schmid|kroeher}@sse.uni-hildesheim.de

[2]University of Bremen, Institute of Computer Science, Softwaretechnik, Am Fallturm 1, 28359 Bremen
koschke@informatik.uni-bremen.de, dierk@tzi.de

Abstract: As more and more companies shift to a product line approach, supporting the evolution of software product lines becomes increasingly important. While today already significant work exists along the lines of quality analysis for software product lines, there is much less work that addresses the evolution scenario. In this paper, we briefly describe different categories of approaches for identifying problems in product lines. Based on this we describe a new research direction for identifying problems in product line evolution scenarios.

1 Motivation

Software evolution is known to be a difficult problem in software engineering. Often errors are introduced into the software as part of the evolution; the difficulty of evolution is compounded by the fact that developing an increment often requires modifying many different parts of the software in a well-coordinated fashion. Any mistake in this may lead to defects. While this is well-known to be a problem in traditional single-system development, it is even more of a problem for software product lines [Sc09, LSR07]. Three reasons lead to the increased complexity in this situation:

- Product lines are longer lived and evolve as long as any of their constituent products evolve.
- Modifications will typically impact a range of products, but not all products. However, the impact on individual products is often not clear when making changes. Thus, what amounts to a correction for one product, may introduce a defect into another one.
- A product line encompasses more artifacts of a larger size than any individual product. Moreover, a product line will typically contain a variability model in addition to the other typical development artifacts [SJ04].

All these issues taken together emphasize the need for supporting the evolution of software product lines in a way that reduces the probability of defects. Unfortunately, today there exists only limited support for identifying problems introduced in product line evolution. In this paper, we will briefly outline some ways to identify problems that are introduced as part of product line evolution.[1]

Further, we will make the assumption in this paper that a product line has already been established. We will not discuss the problem of identifying, for example, copy-and-paste reuse and integrating the corresponding variants into a product line [KFBA09].

2 Approaches to Analyze Defects

Our focus in problem identification is in particular on the (semi-)automatic detection of problems that may exist or be introduced in product lines. This direction of research is, of course, not new. Analysis of product lines has been extensively discussed in the literature [BSR10], however, mostly not with a focus on analyzing evolution, but rather with a focus on scrutinizing a single state of a product line.

From this starting point, we can identify three major directions relevant to analyzing quality problems in product lines.

Variability Model Problems

The first category of approaches is to simply analyze the variability model for any obvious problems. This kind of research has significant history in the field of product lines [BSR10]. Some examples of such an approach are

- Analyze any inconsistencies in the variability model (this would imply that no consistent product may be derived)
- Dead feature analysis (a dead feature is a feature, which can never be selected as this would lead to a contradiction)

The key characteristic of such an approach is that it analyzes only the variability model without taking the realization artifacts of the product line into account at all.

Variability Coordination Problems

However, the variability model is not the only source of variability information. Rather the variability realization, i.e., the code may contain variability information as well. For example, an implementation using preprocessors like the C-preprocessor (which will also be of prime interest to the EvoLine-project) may imply dependencies among

[1] Based on this motivation and along the ideas outlined in this paper, the EvoLine-Project was created, which is part of the German Research Foundation (DFG) Priority Programme SPP 1593.

variabilities [SLBWC11]. If an explicit variability model exists, the dependencies should conform to the variability model. Moreover, we can perform the same analysis, which we discussed in the previous section as well on this implied variability model.

Where do these implications come from? Imagine, for example, that conditional compilation is used realizing two features A and B. However, each condition for B is contained within code, which is conditional on A. Thus, we can derive an implied dependency: B→A. Besides the preprocessor code also the build system, i.e., make-files, may be the source of dependencies.

If on the other hand the variability model permits a configuration B∧¬A we know that variability model and code are mutually inconsistent. While we cannot say which one is correct, we know there is a underlying problem either in the variability model or in the implemation that should be identified and corrected.

Semantic Variability Problems

While the two categories above relied on a simple analysis of variability model information, we can easily go further and analyze semantic information from the code more deeply. Some examples of such an approach are

- *Type inconsistencies*: pre-processor code may even modify type information. This can lead to situations where most, but not all, instances of the product line are type-correct. [KGREOB11]

- *Data-flow issues*: similarly, code manipulation may lead to some instances where necessary variables are not initialized. [RQBTBS11]

There are many examples of these kinds of analysis. Effectively all analysis approaches for single system code that help to detect code smells can be lifted to the product line situation. Of course, the introduction of the product line basis leads to a combinatorial explosion. This turns even a problem like determining type-correctness, which is not regarded as a major challenge for single systems, into a complex problem that also requires significant computation time.

3 Evolutionary Analysis

The problems described above have already been studied to a significant degree in product line engineering. However, these types of analysis have typically not been made in an evolution context. If we take the perspective of an evolution problem, the description changes slightly. Instead of asking whether a certain model is, for example, consistent, we ask whether an atomic change (typically a change as described as a change set in a commit-operation) introduces a problem.

On the one hand, this makes the situation more complex as it increases the amount of information that is available and potentially needs to be taken into account even further (besides the product line information also the change set).

On the other hand, we can decide to focus on the change itself. That is, we assume that the product line was correct before the change and we are only interested in the change itself. We can phrase this problem as:

If we assume the product line PL was initially correct and consistent, and we perform a change C, yielding a product line PL', can we then deduce that PL' is consistent and correct? Thus, instead of asking:

correct(PL'), we ask for *correct*(PL) → *correct*(PL').

As an individual change set is typically much smaller than the product line, we can assume that this will usually lead to significantly reducing the amount of information that needs to be taken into account. However, this amount of information will typically be significantly larger than the change set C alone, as it needs to be interpreted in context.

All taken together, we regard it as a valid research hypothesis that the evolutionary approach will actually lead to significantly more efficient decision procedures to identify any potential evolution issues (evolution smells) in the product line. As positive side-effect this will focus the feedback only on those issues that were introduced in the last iteration, thus providing directly relevant feedback.

4 Conclusion

In this paper, we introduced the concept of *product line evolution smells*, which extends existing product line analysis approaches to the evolution case. We also showed how we expect to optimize existing analyses by making them more efficient through incremental semantics. We expect the evolutionary approach to provide faster results to the developer, thus making the analysis more useful. We also aim at making the results more relevant by focusing on those parts that have recently changed. However, the concepts described above are the basis for research in progress. Thus, whether these advantages can actually be realized is currently open and will be studied further in the EvoLine-project.

5 Aknowledgements

This work was partially supported by the EvoLine-Project, funded by DFG within the SPP1593. Any opinions expressed herein are solely by the authors and not of the DFG.

References

[BSR10] Benavides, D.; Segura, S. & Ruiz-Cortes, A. Automated Analysis of Feature Models 20 Years Later: A Literature Review, Information Systems, 2010, Vol. 35, No.6, 615-636.

[KFBA09] Koschke, R.; Frenzel, P.; Breu, A. & Angstmann, K. *Extending the Reflexion Method for Consolidating Software Variants into Product Lines*, Software Quality Journal, Vol. 17, No. 4, pp. 331–366, 2009.

[KGREOB11] Kästner, C.; Giarrusso, P.; Rendel, T.; Erdweg, S.; Ostermann, K. & Berger, T. Variability-aware parsing in the presence of lexical macros and conditional compilation. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA),* 805-824, 2011.

[LSR07] van der Linden, F.; Schmid, K.; Rommes, E.; Product Lines in Action, Springer, 2007.

[RQBTBS11] Ribeiro, M.; Queiroz, F.; Borba, P.; Toledo, P.; Brabrand, C. & Soares, S. On the impact of feature dependencies when maintaining preprocessor-based software product lines. In *International Conference on Generative Programming and Component Engineering*. pp. 23-32, 2011.

[Sc09] Schmid, K.: Verteilte Evolution von Produktlinien: Herausforderungen und ein Lösungsansatz, 1. Workshop Design For Future - Langlebige Softwaresysteme (L2S2), FZI Karlsruhe, 2009.

[SJ04] Schmid, K.; John, J.; A Customizable Approach to Full Lifecycle Variability Management; Sci-ence of Computer Programming, Vol. 53, No. 3, pp. 259-284, 2004.

[SLBWC11] She, S.; Lotufo, R.; Berger, T.; Wasowski, A.; Czarnecki, K.; Reverse engineering feature models. International Conference on Software Engineering, 461-470, 2011.