Towards Tool-Support for Evolutionary Software Product Line Development

Benjamin Klatt, Klaus Krogmann
FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14, 76149 Karlsruhe, Germany
{klatt,krogmann}@fzi.de

1 Introduction

Software vendors often need to vary their products to satisfy customer-specific requirements. In many cases, existing code is reused and adapted to the new project needs. This copy&paste course of action leads a multi-product code-base that is hard to maintain.

Software Product Lines (SPL) emerged as an appropriate concept to manage product families with common functionality and code bases. Evolutionary SPLs, with a product-first-approach and an ex-post product line, provide advantages such as a reduced time-to-market and SPLs based on evaluated and proven products [3].

In addition to the advantages, the evolutionary derivation of an SPL from existing products approach requires challenging tasks to enforce such a transformation: i) The existing software including the differences between the derivates need to be understood. ii) Furthermore, the common SPL architecture has to be understood and the products have to be refactored into a core with variation points as well as variations that can be instantiated for the derived products.

There are existing approaches to introduce and maintain evolutionary SPLs [5],[6]. Most of them start with the requirement specifications and only few consider the analysis of existing software artifacts. The latter often include hidden variation points that might be missed by considering requirements only as described by Schütz et al. [7]. However, Schütz et al. also state that the software analysis and finding of the right variation architecture is often too complex and expensive to be done manually by the developers.

In this paper, we present an integrated, tool-supported reverse-engineering, variation analysis, and refactoring approach which transforms similar products to an evolutionary SPL. Introducing automation in that evolutionary SPL approach provides a high potential to reduce the effort of analyzing the products under examination and to support choosing the most appropriate SPL architecture. Companies will benefit from this effort reduction and are supported in introducing an SPL.

Our approach is related to the one of Frenzel and Koschke [4], but considers custom SPL characteristics, respects the component-based architecture paradigm (e.g. explicit interfaces) and gives the freedom to choose from multiple possible variation point designs which are evaluated by the approach.

In the following, we provide an insight to the individual steps of the proposed process as well as an outlook on how we plan to evaluate the approach and on the future development.

2 Integrated Process

As illustrated in Figure 1, the process starts with an analysis of multiple existing products using code analysis and reverse-engineering techniques to gain an architecture design-space implicitly described by the SPL candidate products. Afterwards, we support a semi-automated assessment & selection process to iteratively choose the most appropriate SPL architecture and desired variation points to be realised. Finally, when a specific SPL architecture has been selected, an impact analysis is performed and a refactoring support is provided.

3 Software Code Analysis

The goal of the software analysis is to analyse multiple products under examination to identify possible SPL cores, variations, and thus possible SPL architectures. This analysis is separated into three steps. First, code repositories and artifacts are analyzed to identify differing and common code. Then, reverse-engineering techniques are tailored for those two parts and applied on them. On the one hand, common stable parts are reverse-engineered in a more coarse grained fashion with focus on interfaces to the variable parts. On the other hand, the variable parts are reverse-engineered in more detail to gain a basic architecture of their internals. Finally, further code analysis techniques are

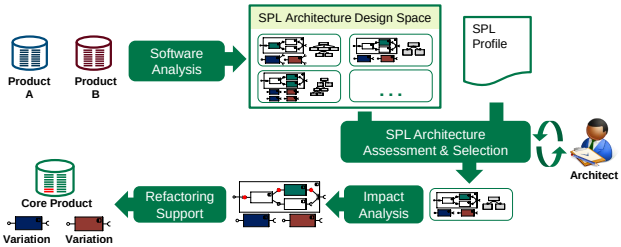

Figure 1: Integrated SPL Adoption Process

used to lay the foundation for the evaluation of possible SPL architectures in the following step (i.e. establish a code model which allows for example code metrics).

The analysis has to determine an architectural model identifying the common and the variable parts of the application. This model will not only include a single SPL architecture but alternatives from which in the next step software architects can select appropriate variation points.

4 Assessment & Selection

This part of the process makes use of the previously generated design-space as well as an SPL profile. In an SPL profile, the architect specifies desired characteristics the target SPL has to conform to. We classify the characteristics into generally applicable characteristics and system specific ones. The former can be applied to any SPL. They include characteristics such as the intended SPL maturity level [3] or API usability requirements of the later SPL. The latter describe characteristics specific to the products under study. They can include constraints and preferences such as to not split specific parts of the system or to prefer fewer large variation points over many smaller ones for specific parts of the system.

As an initial step of the assessment & selection process, the design-space is reduced as much as possible based on SPL profile constraints that directly limit the possible SPL architecture alternatives.

Next, the architect can select single variations from the SPL architecture alternatives in the design space. In general, a single independent variation as well as a group of dependent ones can be solved with different designs of variation points. An architect is involved to select the most appropriate one. To support his decision, the SPL profile is considered and metrics are calculated to rate the available alternatives. However, this can only be a recommendation and an architect might want to select not the best rated alternative but one that better matches his intention. Our approach enables him to select his favorite, supported by the automated assessment.

To handle the possibly high number of alternatives, the assessment & selection process can be distributed over multiple iterations. Within a single iteration, the architect only needs to decide about a single indepetendent variation or a group of dependent ones. The iterative approach furthermore faces the problem of dependencies among possible variations (i.e. later iterations do no include variations which are invalid due to prior decisions). Which of the variations are dependent or independent has to be determined from the code analysis.

As a result of the assessment & selection, the architect determines all variations and the selected variation point designs from the finally chosen SPL architecture.

5 Impact Analysis & Refactoring

To complete a single process cycle, an impact analysis based on the selected SPL architecture is performed. The aim of this step is to identify all parts of the original software products that are affected by the new architecture. Even code areas that are not directly related to a variation area might be affected because of shared code between them. To enable such an analysis, we utilise code traces we previously obtained during the initial code analysis and reverse-engineering process.

Refactoring support can be derived from the results of the impact analysis. This support should enable developers to change the existing products to meet the new product line architecture and introduce variation points. This support can be either on a task level as supported by frameworks such as Mylyn [1] or can be integrated in a development environment such as the Eclipse Java or C++ development tools.

6 Outlook

Currently, we are evaluating existing analysis tools such as SiSSy [8] and SoMoX [2] with respect to their suitability for our approach. We will then combine them with other code analysis techniques such as clone detection and code history analysis to tailor an appropriate reverse-engineering and code variation detection. Furthermore, candidates of test systems are studied for their appropriateness as case studies. We will seek to identify systems that have been manually refactored to product lines at least in certain aspects. We will apply our approach to older versions of these systems and compare our results with their newer and refactored versions. There is also ongoing work on defining a meta-model appropriate to describe the results of the software analysis step (i.e. design space, variations, and variation points).

References

[1] Mylyn - Task & ALM Framework. http://www.eclipse.org/mylyn/.

[2] SoMoX – SOftware MOdel eXtractor. http://www.somox.org.

[3] J. Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Software Product Lines*. 2002.

[4] P. Frenzel, R. Koschke, A. Breu, and K. Angstmann. Extending the reflexion method for consolidating software variants into product lines. In *WCRE 2007*, 2007.

[5] J. Meister. *Produktgetriebene Entwicklung von Software-Produktlinien am Beispiel analytischer Anwendungssoftware*. PhD thesis, Carl von Ossietzky Universität, Oldenburg, Germany, 2006.

[6] M. Riebisch and S. Bode. Design Decision Support for Evolvability and Variability. In *Proceedings of the 12. Workshop Software-Reengineering (WSR'10)*, 2010.

[7] D. Schütz. Variability Reverse Engineering. In *Proceedings of the EuroPLoP 2011*, 2009.

[8] O. Seng, F. Simon, and T. Mohaupt. *Code Quality Management*. dpunkt Verlag, Heidelberg, 2006.