2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

Lost in Source Code: Physically Separating Features in Legacy Systems

Extended Abstract

Jacob Krüger
Otto-von-Guericke-University Magdeburg, Germany
jkrueger@ovgu.de
Harz University of Applied Sciences Wernigerode, Germany
jkrueger@hs-harz.de

***Abstract*—Feature-oriented programming allows developers to physically separate and reuse features via composition. This promises several benefits compared to other reuse approaches, for instance, easier traceability and maintenance. However, due to their simplicity cloning and annotation-based product lines are established in practice. We aim to reduce risks and costs of migrating towards composition, lowering the adoption barrier. This includes *i)* processes, *ii)* migration approaches, and *iii)* assessing advantages and disadvantages. Overall, we will facilitate integrating physical separation into legacy applications.**

***Index Terms*—software product line, extraction, migration**

I. Introduction

Software developers face the need to customize their systems to customer requirements [1]. They reuse existing software artifacts to reduce corresponding costs. In practice, two approaches are commonly used. Firstly, with *clone-and-own*, an existing system is copied and then customized to new requirements [1, 7]. While this approach is simple to apply, costs for maintenance and customization increase with each clone, due to change propagation and bug fixing [1]. Secondly, *annotation-based* product lines, especially the `C` preprocessor, are used to implement variability in a common code base [1, 8, 12]. However, variability is only virtually separated, resulting in scattering and tangling of code (a.k.a. `#ifdef hell`), which hampers maintenance and readability [12]. We refer to any system developed with these approaches as *legacy system*.

Missing physical separation of features is one potential problem of both approaches. A feature represents a concern from conceptual down to implementation level [1]. Due to physical separation, e.g. with feature-oriented programming, compositional product lines enable systematic reuse and customizing based on modularization [1, 8].

Still, migrating legacy systems towards composition is rarely applied, because companies fear risks and costs of refactoring their systems [2, 3, 9]. We aim to facilitate and systematize the migration of features towards physical separation. Overall, we want to support the introduction of composition into practice, reducing companies' adoption barrier [3]. To this point, we address the following research questions:

1) **How can we migrate legacy systems towards compositional product lines?**
   We aim to define processes and tools to guide the separation of features depending on the legacy system.
2) **How can we separate code into feature modules?**
   We want to identify patterns in legacy code that can be separated with variant-preserving transformations [6].
3) **How suitable are these approaches for practice?**
   We will conduct empirical studies to investigate several characteristics of our approach.

II. Related Work

Feature location aims to identify variability in a system [4]. Several approach were proposed but most focus on a single system implemented without a variability mechanism [4, 5]. For cloned products, we aim to locate features in one system and map them in others to identify a common base.

We summarize two refactoring types as *variant-preserving transformation*:

- Variant-preserving migration (from stand-alone to product line) towards feature-oriented programming is mostly considered for a single system [6]. While some works discuss approaches and potential patterns [11], others show that there are still open challenges [2].
- Variant-preserving mapping (from annotation to composition and vice versa) towards feature-oriented programming is rarely considered in research [6]. Several authors argue that combining annotation and composition promises benefits [8] and is essential for step-wise migrations [2].

We aim to extend these works to define processes and patterns for migrating legacy systems towards feature-oriented programming. In particular, we will investigate problematic code structures and integrated tooling [2].

Some *empirical studies* address the implementation of feature-oriented programming [2, 6, 11, 13]. However, to our knowledge these studies are rarely applied, limited to single cases, and are missing industrial evaluations. We aim to address these points and extend the existing body of knowledge on benefits of physically separated features by conducting further studies.

III. Contributions

We conceptually sketch our approach in Figure 1. For our work, we assume the following three conditions to be true:

- **Existing legacy systems:** The prerequisite of our approach is that either cloned or annotated legacy systems

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.46

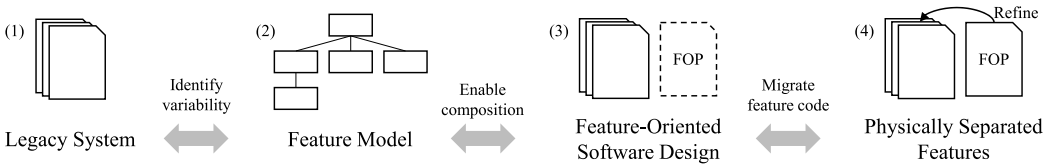

Fig. 1. Sketch of physical separation in legacy systems.

with a set of features exist (Figure 1 Step 1). However, variability is not necessarily mapped or managed, wherefore we aim to semi-automatically locate and map features (Figure 1 Step 2). To this point, we propose to adopt feature location [4] and code clone detection [5].

- **Step-wise migration:** We will utilize a step-wise and semi-automatic process to facilitate the risks of migrations and allow ongoing assessments [2, 9, 10]. For this reason, we consider to migrate only selected features to refine the system (Figure 1 Steps 3 and 4), enabling incremental migration towards physically separated features.
- **Benefits of physical separation:** In advance, we assume that physically separated features are beneficial to a certain degree. Several works discuss the complementary advantages of cloning, annotations, and composition [1, 8], but empirical studies are rare [13]. Thus, we aim to assess our approach as well as to which extend physical separation is useful.

Overall, we will describe a process to migrate legacy systems towards feature-oriented programming. To facilitate its application, we propose to semi-automatically locate variability and migrate code based on an incremental approach.

IV. Evaluation and Results

We plan to combine the following three evaluation approaches:

- **Open-Source Case Studies:** Several open-source systems developed with clone-and own or annotations exist, partly originating from industry. Due to the open access, we aim to use such systems for case studies to define and assess our approach, especially to adopt feature location and identify migration patterns.
- **Industrial Case Studies:** With industrial partners, we aim to evaluate our processes in practice, especially its costs and benefits. As a result, we can describe best practices or pitfalls that companies may face during such migration processes and identify improvements.
- **Empirical Studies:** In addition to case studies, we aim to conduct experiments and interviews, for instance with industrial and open-source developers, to assess benefits of physical separation of features. The results help to further assess benefits of physically separating features and our approach.

With these strategies, we aim to credibly and comprehensively evaluate our approach. This way, we can further improve the defined processes.

V. Conclusions

Summarized, our goal is to define processes for migrating legacy systems towards feature-oriented programming. To our knowledge, this is the first systematic approach to implement such migrations based on source code analysis and transformation. We aim to combine and integrate existing works from several domains, such as feature location, to extend and customize them for our processes.

Acknowledgment

This research is supported by DFG grant LE 3382/2-1. I thank Gunter Saake and Thomas Leich for their guidance.

References

[1] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines*. Springer, 2013.
[2] F. Benduhn, R. Schröter, A. Kenner, C. Kruczek, T. Leich, and G. Saake. Migration from Annotation-Based to Composition-Based Product Lines: Towards a Tool-Driven Process. In *SOFTENG*, pages 102–109. IARIA, 2016.
[3] P. C. Clements and C. W. Krueger. Point/Counterpoint: Being Proactive Pays Off/Eliminating the Adoption Barrier. *IEEE Softw.*, 19(4):28–30, 2002.
[4] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature Location in Source Code: A Taxonomy and Survey. *J. Softw. Evol. Process*, 25(1):53–95, 2013.
[5] S. Duszynski, J. Knodel, and M. Becker. Analyzing the Source Code of Multiple Software Variants for Reuse Potential. In *WCRE*, pages 303–307. IEEE, 2011.
[6] W. Fenske, T. Thüm, and G. Saake. A Taxonomy of Software Product Line Reengineering. In *VaMoS*, page 1–8. ACM, 2013.
[7] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed. Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In *ICSME*, pages 391–400. IEEE, 2014.
[8] C. Kästner and S. Apel. Integrating Compositional and Annotative Approaches for Product Line Engineering. In *McGPLE*, pages 35–40. University of Passau, 2008.
[9] J. Krüger, W. Fenske, J. Meinicke, T. Leich, and G. Saake. Extracting Software Product Lines: A Cost Estimation Perspective. In *SPLC*, page 354–361. ACM, 2016.
[10] J. Krüger, I. Schröter, A. Kenner, C. Kruczek, and T. Leich. FeatureCoPP: Compositional Annotations. In *FOSD*, pages 74–84. ACM, 2016.
[11] R. E. Lopez-Herrejon, L. Montalvillo-Mendizabal, and A. Egyed. From Requirements to Features: An Exploratory Study of Feature-Oriented Refactoring. In *SPLC*, pages 181–190. IEEE, 2011.
[12] F. Medeiros, C. Kästner, M. Ribeiro, S. Nadi, and R. Gheyi. The Love/Hate Relationship with the C Preprocessor: An Interview Study. In *ECOOP*, pages 495–518. Schloss Dagstuhl, 2015.
[13] J. Siegmund, C. Kästner, J. Liebig, and S. Apel. Comparing Program Comprehension of Physically and Virtually Separated Concerns. In *FOSD*, pages 17–24. ACM, 2012.