Guiding Feature Asset Mining for Software Product Line Development

Thomas Eisenbarth Daniel Simon

Universität Stuttgart
Breitwiesenstraße 20–22
70565 Stuttgart, Germany
{eisenbarth, simon}@informatik.uni-stuttgart.de

1 Introduction

Software product line architectures promise significant benefits over traditional architectures such as shorter time-to-market, shorter and cheaper development cycles, and higher exploitation of the reuse potential at hand. While the ideas and concepts of product lines are well suited for developing new products, it is not obvious if and how one can apply this technology in the presence of legacy software.

Migrating legacy software systems to a product line provides ways for extending and developing successful products and offers a chance to protect and preserve a company's former investments. The legacy artifacts have been designed under significantly different circumstances and typically for just one single application domain. Therefore turning legacy software into a software product line requires a new design aware of the old product's key assets—assuming reuse really pays. Consequently, reengineering efforts have to address a number of issues unique to product line development.

The product line development frameworks of Bosch [5] and the SEI [11] recognize a need for integration of existing assets. Bergey et al. [4, 12] discuss how to exploit the reuse potential by means of traditional reverse engineering methods and business process decisions in detail. Another effort integrating legacy assets into a product line is described by Bayer et al. [3] in the context of PuLSE [2].

Our goal is to develop techniques that respect the specific reverse engineering prerequisites of product lines. Recently, we introduced techniques that help to derive feature-component maps [9] by means of mathematically founded concept formation.

2 What's different?

A number of techniques have recently been developed that try to analyze legacy software by focusing on features of the software. They are all based on dynamic and static analyses as well as software metrics [6, 7, 8, 9, 10, 13, 14, 15].

When analyzing legacy source code in product line development context, some aspects that will be discussed in the following gain prominent importance.

Early indication whether further investigation will pay By looking at results of the early analysis steps over the system, early judgment about the lucrativeness of feature asset mining is enabled. Our techniques [7, 8, 9, 10] are incremental as they support judgment after each round whether it is benefical to take the next step or cancel the reuse effort.

No overall architecture recovery of the legacy software Our new analysis techniques need no complete recovery of the legacy code's architecture. Our techniques incorporate domain knowledge of both users and programmers rather than focusing on the source code only. Our method is opportunistic because it allows the product line engineer to analyze the old code just as far as needed. The support for partial analyses or quick and cheap ad-hoc decisions is an import aspect since complete results usually require much time. (Why should someone recover an architecture, just to decide he doesn't need it anyway?)

Analysis according to the legacy software's features We prepare legacy software for reuse in software product lines by (a) feature location and (b) connector recovery for handling the communication in the software artifact. On the one hand side, we try to grasp the functional features. On the other hand side, the communication between functional features organized via data or control structures in the source is revealed so that the communication with the other parts of the legacy system becomes explicit.

To gain the desired information, dynamic and static analyses complement each other. Further, metrics that measure the disparity, concentration, and dedication of features to program parts are computed [15]. Our domain oriented approach serves the goals of product line engineering: the analysis is not limited to classic programming paradigms but focuses on the user's needs. Indeed, even the users of the system may easily assist the product line engineer in analyzing the code assets by giving hints on how-to-use the system (this might range from typical usage to extremely rare cases).

Incremental shift towards product lines If the introduction of product line technology is too disruptive because the legacy system at hand is too large and too complex and the product is business critical to the company, then cautious and incremental shift towards product line methods is advisable.

We believe that our techniques can support the integration of development personal and avoid wrapping and decay of the competence encoded in the software. The proposed proceeding for that case is

I As long as the structure of the legacy software prevents the identification of sensible variation and commonality points, restructure according to findings of feature analysis.

II Identify variabilities and commonalities. Based upon that, identify interfaces.

3 Outlook

Currently, the Bauhaus system [1] at the Universtät Stuttgart provides means for incremental architecture recovery and code validation. To satisfy the specific needs of product line engineering, we are extending Bauhaus by the mentioned techniques for spotting software features. In conjunction with information about intra-software communication gained by connector recovery, we hope to find methods that empower software engineers to quickly find interfaces and aid their quest for reusable legacy assets. Cheap and quick reuse can help in convincing companies to switch to product line architectures initially. Further, if the legacy asset is on no account reusable, our approach will help to save much effort.

References

[1] The New Bauhaus Stuttgart. Available at `http://www.bauhaus-stuttgart.de`, 2001.

[2] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99)*, pages 122–131, Los Angeles, CA, USA, May 1999. ACM.

[3] Joachim Bayer, Jean-François Girard, Martin Würthner, Jean-Marc DeBaud, and Martin Apel. Transitioning Legacy Assets to a Product Line Architecture. In *Proceedings of the Seventh European Software Engineering Conference (ESEC'99)*, Lecture Notes in Computer Science 1687, pages 446–463, Toulouse, France, September 1999.

[4] John Bergey, Liam O'Brien, and Dennis Smith. Mining Existing Software Assets for Software Product Lines. Technical Report CMU/SEI-2000-TN-008, Software Engineering Institute (SEI), Carnegie Mellon University, May 2000.

[5] Jan Bosch. *Design & Use of Software Architectures*. Addison-Wesley and ACM Press, 2000.

[6] Kunrong Chen and Václav Rajlich. Case Study of Feature Location Using Dependence Graph. In *Proceedings of the International Workshop on Program Comprehension*, pages 241–249, Limerick, Ireland, June 2000. IEEE Computer Society Press.

[7] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Herleitung der Feature-Komponenten-Korrespondenz mittels Begriffsanalyse. In *Proceedings of the 1. Deutscher Software-Produktlinien Workshop*, pages 63–68, Kaiserslautern, Germany, November 2000.

[8] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Aiding Program Comprehension by Static and Dynamic Feature Analysis. In *Proceedings of the International Conference on Software Maintenance*, page To appear., Florence, Italy, November 2001. IEEE Computer Society Press.

[9] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Derivation of Feature-Component Maps by Means of Concept Analysis. In Pedro Susa and Jürgen Ebert, editors, *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 176–179, Lisbon, Portugal, March 2001.

[10] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Feature-Driven Program Understanding Using Concept Analysis of Execution Traces. In *Proceedings of the International Workshop on Program Comprehension*, pages 300–309, Toronto, Canada, May 2001. IEEE Computer Society Press.

[11] Linda M. Northrop. A Framework for Software Product Line Practice. Available at `http://www.sei.cmu.edu/plp/framework.html`, 2001.

[12] Nelson Weiderman, John Bergey, Dennis Smith, and Scott Tilley. Can Legacy Systems Beget Product Lines? *Lecture Notes in Computer Science*, 1429, 1998.

[13] Norman Wilde, Michelle Buckellew, Henry Page, and Václav Rajlich. A Case Study of Feature Location in Unstructured Legacy Fortran Code. In Pedro Susa and Jürgen Ebert, editors, *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 68–75, Lisbon, Portugal, March 2001.

[14] Norman Wilde and Michael Scully. Software Reconnaissance: Mapping Program Features to Code. *Journal of Software Maintenance: Research and Practice*, 7:49–62, January 1995.

[15] W. Eric Wong, Swapna S. Gokhale, and Joseph R. Hogan. Quantifying the Closeness between Program Components and Features. *The Journal of Systems and Software*, 54(2):87–98, October 2000.