Science of Computer Programming 93 (2014) 65–85

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

On the use of feature-oriented programming for evolving software product lines — A comparative study

Gabriel Coutinho Sousa Ferreira[a], Felipe Nunes Gaia[a], Eduardo Figueiredo[b], Marcelo de Almeida Maia[a,*]

[a] *Federal University of Uberlândia, Brazil*

[b] *Department of Computer Science, Federal University of Minas Gerais, Brazil*

HIGHLIGHTS

- We provide an open benchmark for the analysis of modularity in evolving SPLs.
- We provide a quantitative and qualitative evaluation of SPL variability mechanisms.
- FOP and DP have shown better adherence to the Open–Closed Principle than CC.
- In general, FOP was more effective tackling feature modularity degeneration.

ARTICLE INFO

Article history:
Received 15 February 2012
Received in revised form 7 October 2013
Accepted 29 October 2013
Available online 15 November 2013

Keywords:
Software product lines
Feature-oriented programming
Variability management
Design patterns
Conditional compilation

ABSTRACT

Feature-oriented programming (FOP) is a programming technique based on composition mechanisms, called refinements. It is often assumed that feature-oriented programming is more suitable than other variability mechanisms for implementing Software Product Lines (SPLs). However, there is no empirical evidence to support this claim. In fact, recent research work found out that some composition mechanisms might degenerate the SPL modularity and stability. However, there is no study investigating these properties focusing on the FOP composition mechanisms. This paper presents quantitative and qualitative analysis of how feature modularity and change propagation behave in the context of two evolving SPLs, namely WebStore and MobileMedia. Quantitative data have been collected from the SPLs developed in three different variability mechanisms: FOP refinements, conditional compilation, and object-oriented design patterns. Our results suggest that FOP requires few changes in source code and a balanced number of added modules, providing better support than other techniques for non-intrusive insertions. Therefore, it adheres closer to the Open–Closed principle. Additionally, FOP seems to be more effective tackling modularity degeneration, by avoiding feature tangling and scattering in source code, than conditional compilation and design patterns. These results are based not only on the variability mechanism itself, but also on careful SPL design. However, the aforementioned results are weaker when the design needs to cope with crosscutting and fine-grained features.

1. Introduction

Software Product Lines (SPLs) [17] are known to enable large scale reuse across applications that share a similar domain. The potential benefits of SPLs are achieved through a software architecture designed to increase reuse of features in

* Corresponding author. Tel.: +55 3432394306.
E-mail addresses: gabriel@mestrado.ufu.br (G.C. Sousa Ferreira), felipegaia@mestrado.ufu.br (F.N. Gaia), figueiredo@dcc.ufmg.br (E. Figueiredo), marcmaia@facom.ufu.br (M. de Almeida Maia).

0167-6423/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.scico.2013.10.010

several SPL products. There are common features found on all products of the product line (known as mandatory features) and variable features that allow distinguishing between products in a product line (generally represented by optional or alternative features). Variable features define points of variation and their role is to permit the instantiation of different products by enabling or disabling specific SPL functionality.

As in any software life cycle, changes in SPLs are expected and must be accommodated [30]. When it comes to SPLs, these changes have even more impact, since changes to attend new stakeholder requests [17], may affect several products. In an ideal scenario, the introduction of new features on an SPL should be conducted by inserting components that encapsulate new or enhanced features [11], minimizing ripple effects of changes.

Variability management is a key factor to be considered when evolving SPLs. Several mechanisms, whether annotative or compositional [34], support variability management. Examples of variability mechanisms are FOP refinements [12,14], conditional compilation [2,5], and object-oriented design patterns [27]. To be considered effective, these mechanisms must guarantee the SPL architecture stability and, at the same time, facilitate future changes. In order to ensure these requirements, the variability mechanisms should minimize changes and should not degenerate modularity. In other words, variability mechanisms should support non-intrusive and self-contained changes that favor insertions and do not require deep modifications in existent components. These requirements are related to the Open–Closed principle [42], which states that "software should be open for extension, but closed for modification". This principle can be achieved with mechanisms that add new artifacts to extend the system functionality, but minimize the amount of modifications in current code.

Our work targets to find out how variability mechanisms behave in terms of modularity and change propagation on specific SPL change scenarios. In this context, this paper presents two case studies that evaluates comparatively three mechanisms for implementing variability on evolving software product lines: conditional compilation (CC), object-oriented design patterns (DP) and feature-oriented programming (FOP). This investigation extends our preliminary work [22] and focuses on the evolution of two software product lines, called WebStore and MobileMedia (Section 3). We choose these SPLs because they were available to us and have been used in previous studies with similar purpose [16,24]. Altogether, we considered five versions of WebStore SPL and seven versions of MobileMedia SPL.

In this study, we analyzed and compared the implementation of variability mechanisms to evolve two SPLs, using a pure FOP language (Jak) [14] and other two OO-based programming techniques. This work evaluated the compositional mechanisms available in FOP by using the other two variability techniques as baseline. The SPL implementation assessment was based on modularity and change propagation metrics recurrently used to quantify separation of concerns and change impacts [16,18,26,47,52]. Moreover, our study contributes to build up a body of knowledge that allows the comparison of AHEAD and other FOP or non-FOP approaches.

This paper extends the previous SBLP paper with two major contributions, as follows.

- A new case study using the MobileMedia SPL; our preliminary work relies only on the WebStore SPL. MobileMedia is larger than WebStore not only in terms of number of components but also with respect to the variety of change scenarios. Therefore, this new case study helped us to (i) increase the results reliability, (ii) come up with new findings, and (iii) reduce threats to study validity.
- We also provide more detailed data analysis and a deeper discussion about the new findings. The analyses, that now considered data collected from both SPLs, reinforced the findings from the first case study and revealed several new ones. For instance, based on the MobileMedia case study, we observed that the SoC (*Separation of Concerns*) metrics tend to be less discriminative on larger systems.

  Therefore, the novel contributions of this extended paper are threefold.
- The development of public benchmark data with 113,152 data points concerning four feature modularity metrics extracted from two SPLs implemented with three different variability mechanisms in 12 different versions.
- The qualitative and quantitative analysis framework for change propagation and feature modularity metrics that can be reused in further replications of this study.
- Discussion and observations based on the obtained data about the role and the singular applicability of each variability mechanism in the context of evolving software product lines.

The rest of this paper is organized as follows. In Section 2, the implementation mechanisms used in the case study are revisited. Section 3 presents the study setting, including the target SPLs and their respective change scenarios. Section 4 analyzes change measures through different releases. In Section 5, the modularity of WebStore and MobileMedia SPLs are quantitatively analyzed and discussed. Section 6 presents the threats to validity of this study. Section 7 presents related work. Finally, Section 8 concludes this paper.

2. Variability mechanisms for software product lines

This section revisits some concepts about the three techniques evaluated in this study: conditional compilation (CC), object-oriented design patterns (DP) and feature-oriented programming (FOP). We choose conditional compilation and design patterns because these are the state-of-the-practice options adopted in SPL industry [5,42]. Although there are other approaches that could be used to represent the feature-oriented paradigm [43], we chose AHEAD because it has been widely studied [8,10,12,14,34].