Science of Computer Programming 96 (2014) 230–253

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines

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

[a] *Federal University of Uberlândia, Brazil*
[b] *Federal University of Minas Gerais, Brazil*

HIGHLIGHTS

- Variability mechanisms are systematically evaluated in the evolution of SPLs.
- FOP and AFM have shown better adherence to the Open-Closed Principle than CC.
- When crosscutting concerns are present, AFM are recommended over FOP.
- Refactoring at component level has important impact in AFM and FOP.
- CC compilation should be avoided when modular design is an important requirement.

ARTICLE INFO

Article history:
Received 1 April 2013
Received in revised form 28 February 2014
Accepted 10 March 2014
Available online 31 March 2014

Keywords:
Software product lines
Feature-oriented programming
Aspect-oriented programming
Aspectual feature modules
Variability mechanisms

ABSTRACT

Feature-Oriented Programming (FOP) and Aspect-Oriented Programming (AOP) are programming techniques based on composition mechanisms, called refinements and aspects, respectively. These techniques are assumed to be good variability mechanisms for implementing Software Product Lines (SPLs). Aspectual Feature Modules (AFM) is an approach that combines advantages of feature modules and aspects to increase concern modularity. Some guidelines on how to integrate these techniques have been established in some studies, but these studies do not focus the analysis on how effectively AFM can preserve the modularity and stability facilitating SPL evolution. The main purpose of this paper is to investigate whether the simultaneous use of aspects and features through the AFM approach facilitates the evolution of SPLs. The quantitative data were collected from two SPLs developed using four different variability mechanisms: (1) feature modules, aspects and aspects refinements of AFM, (2) aspects of aspect-oriented programming (AOP), (3) feature modules of feature-oriented programming (FOP), and (4) conditional compilation (CC) with object-oriented programming. Metrics for change propagation and modularity were calculated and the results support the benefits of the AFM option in a context where the product line has been evolved with addition or modification of crosscutting concerns. However a drawback of this approach is that refactoring components' design requires a higher degree of modifications to the SPL structure.

* Corresponding authors.
E-mail addresses: felipegaia@mestrado.ufu.br (F.N. Gaia), gabriel@mestrado.ufu.br (G.C.S. Ferreira), figueiredo@dcc.ufmg.br (E. Figueiredo), marcmaia@facom.ufu.br (M.A. Maia).

http://dx.doi.org/10.1016/j.scico.2014.03.006
0167-6423/© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Software Product Lines (SPLs) refer to an emerging engineering technique that aims to provide the systematic reuse of common core and shared modules in several software products [12]. Optional features define points of variability and their role is to permit the differentiation of products in a specific software product line (SPL). SPLs products share the same application domain and have points of variability among them. The adoption of SPLs presents as potential benefits the increased product's quality and development productivity, which are achieved through the systematic reuse of features in different products [12].

During an SPL life cycle, change requests are not only inevitable, but also highly frequent [22] since they target several different products. These change requests must be accommodated since they include demands from multiple stakeholders [16].

Variability mechanisms play a crucial role when considering evolving SPLs. They must guarantee the architecture stability and, at the same time, facilitate future changes in the SPL. Therefore, variability mechanisms should not degenerate modularity and should minimize the need of future changes. Ideally, all evolutionary tasks in an SPL should be conducted through non-intrusive and self-contained changes that favor insertions and do not require deep modifications into existing components. The inefficacy of variability mechanisms to accommodate changes might lead to several undesirable consequences related to the product line stability, including invasive wide changes, significant ripple effects, artificial dependencies between core and optional features, and the lack of independence of optional code [17,32].

In our previous study [15], we analyzed and compared variability mechanisms to evolve SPLs, using FOP, Design Patterns and Conditional Compilation. The evaluation was also based on change propagation measures and modularity metrics. In that work, the result was mostly favorable for FOP mechanism. It is important to consider that crosscutting concerns were not considered in the subject system analyzed in that study. This work has as main goal the better understanding of how contemporary variability mechanisms contribute to the mentioned SPLs evolution practices. To this aim, this paper presents two case studies that evaluate comparatively four mechanisms for implementing variability during the evolution of product lines: conditional compilation (CC), feature-oriented programming (FOP), aspect-oriented programming (AOP), and aspectual feature modules (AFM). This work is an extension of a previous work [19], which was carried out only with one SPL, called WebStore. In this work, we include five releases of another SPL called MobileMedia [17,45]. This SPL 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. Moreover, we extended significantly the amount of data, providing *quantitative* and *qualitative* analysis of the measured data in greater depth. The quantitative analysis refers to interpretation of collected measures related to *stability* and *modularity*. The analysis of stability considers measures of change impact [45], while the analysis of modularity relies on Separation of Concern metrics [39]. The qualitative analysis is concerned with the interpretation and reasoning of the possible factors that influenced the quantitative results.

The analysis presents novel results that support the benefits of choosing between AFM and FOP when an SPL has many optional features. In this case, class refinements adhere more closely to the Open-Closed principle [33]. In addition, these mechanisms cope well for features with no shared code and facilitate the instantiation of different products. However, FOP is not suitable for crosscutting concerns and design refactoring with AFM causes a higher number of modifications in SPL components. The results also demonstrate that CC may not be appropriate in SPL evolution context when modularity of features is an important concern. For example, the inclusion of new features usually increases tangling and scattering of others features.

In Section 2, the implementation mechanisms used in the case study are presented. Section 3 describes the study settings, including the target SPL and change scenarios. Section 4 analyzes changes made in the WebStore and MobileMedia SPLs and how they propagate through its releases. Section 5 analyzes the modularity of both SPLs with specific concern-related metrics. Section 6 provides an overall discussion of results. Section 7 presents some limitations of this work. Section 8 presents related work and points out directions for future work. Finally, Section 9 concludes this paper.

2. Variability mechanisms

This section presents some concepts about the four techniques evaluated in the study: conditional compilation (CC), feature-oriented programming (FOP), aspect-oriented programming (AOP), and aspectual feature modules (AFM). Our main goal is to compare the different composition mechanisms available to understand their distinct strengths and weaknesses. Although CC is not a new variability mechanism, we decided to include it in this study because it is still a state-of-the-practice option adopted in SPL industry, and can serve as a fair baseline for comparison [1,41,6].

2.1. Conditional compilation

The Conditional Compilation (CC) approach studied in this work is a well-known annotation-based technique for handling software variability [1,4,25]. It has been used in programming languages like C for decades and it is also available in object-oriented languages such as Java. Basically, the preprocessor directives indicate pieces of code that should be compiled or not, based on the value of preprocessor variables. The major advantage of this approach is that the code can be marked at different granularities, from a single line of code to a whole file.