Chapter 18
Recommending Mechanisms for Modularizing Mobile Software Variabilities

Márcio Ribeiro
Federal University of Pernambuco, Brazil

Pedro Matos Jr.
Federal University of Pernambuco, Brazil

Paulo Borba
Federal University of Pernambuco, Brazil

ABSTRACT

Software Product Lines (SPLs) encompass a family of software systems developed from reusable assets. One issue during SPL maintenance is the decision about which mechanism should be used to restructure variabilities aiming at improving the modularity of the SPL artifacts. Due to the great variety of mechanisms (inheritance, configuration files, aspect-oriented programming), selecting the incorrect ones may produce negative effects on the cost to evolve the SPL. To reduce this problem, the authors propose a Decision Model to help developers to choose mechanisms to restructure variabilities in SPLs. Their model was developed based on two domains: mobile test scripts and J2ME games. When using the model, developers may improve the variabilities' modularity, the SPLs design, and remove bad smells such as cloned code.

INTRODUCTION

Software Product Lines encompass a family of software-intensive systems developed from reusable assets (known as core assets). By reusing such assets it is possible to construct a large scale of products through specific variabilities defined according to customers' requirements (Pohl, Böckle, & van der Linden, 2005). On the other hand, implementation activities become more complex because they also have to realize variabilities (Patzke & Muthig, 2002).

In this context, reasoning about how to combine both core assets and product variabilities is

DOI: 10.4018/978-1-61520-655-1.ch018

a challenging task (Anastasopoulos & Gacek, 2001). In other words, the challenge consists of understanding the available mechanisms - such as Inheritance, Configuration Files, Aspect-Oriented Programming (Kiczales, et al., 1997), and so forth - for realizing variability and knowing which of them fits best for a given variability (Patzke & Muthig, 2002). Previous work (Kolb, Muthig, Patzke, & Yamauchi, 2005) (Alves, Matos Jr, Cole, Borba, & Ramalho, 2005) has structured product line variabilities by using only one mechanism. Because each mechanism has strengths and weaknesses, not all variabilities were well structured when considering modularity criteria, for example.

Due to the great variety of available mechanisms, selecting an incorrect mechanism may produce negative effects on the cost to maintain the SPL (Ribeiro, Matos Jr, Borba, & Cardim, 2007). For example, cloned code and concerns not modularized may appear, affecting independent evolution of SPL artifacts, increasing developer's effort, and consequently decreasing productivity when evolving the SPL.

The problem of combining core assets and SPL variabilities exists not only at source code, but also in other artifacts such as requirements and tests. In order to reduce the aforementioned problems, we have defined a Decision Model encompassing two different domains. Such model is supposed to help developers on the task of choosing mechanisms to restructure variabilities in SPLs. To construct our decision model, we analyzed real variabilities found in two different domains: Motorola mobile phone test scripts and J2ME games from Meantime Mobile Creations[1]. The test variabilities analyzed were handled by using *if-else* statements whereas the J2ME games variabilities are implemented using conditional compilation. The motivation to restructure them is that both approaches do not provide modularity at all; variabilities are not separated from core assets. Our model aims at suggesting mechanisms so that these variabilities can be modularized, being important to improve the product lines' design.

The main contributions of our work are:

a. A Decision Model for improving guidance of developers in SPL maintenance. Existing models (Anastasopoulos & Gacek, 2001) (Patzke & Muthig, 2002) consider a high level approach that rely basically on feature types. In contrast, we provide a decision model which is code-centric and more fine-grained. Because we consider not only the feature type, but also the exactly variability location at the source code and some criteria, our recommendations may be more precise. In addition, we discuss some particularities and similarities between the domains when defining the model. Given the different natures of them, we believe our model may be applied in other domains as well;
b. In order to analyze the advantages and disadvantages of each mechanism, we have calculated some metrics (Sant'Anna, Garcia, Lucena, & von Staa, 2003). Because those metrics are supposed to deal with single products, we have defined new metrics to encompass SPLs, representing another contribution of our work.

MOTIVATING EXAMPLES

In this section we provide some motivating real examples extracted from Motorola mobile phone test scripts and J2ME games from Meantime. The variabilities are handled by using *if-else* statements and conditional compilation so that there is no software modularity in those artifacts.

Example 1: Test Script Variability

The first variability kind analyzed in this work is at the End of Method Body. Our example of this test variability (Figure 1) consists of two optional features implemented at the end of the *procedures* method. Hence, four instances of the product line