# Supporting Software Product Lines Development: FLiP – Product Line Derivation Tool

Sérgio Soares
University of Pernambuco
sergio@dsc.upe.br

Fernando Calheiros
Meantime Mobile Creations
fernando.calheiros@cesar.org.br

Vilmar Nepomuceno
Meantime Mobile Creations
vilmar.nepomuceno@cesar.org.br

Andrea Menezes
Meantime Mobile Creations
andrea.menezes@cesar.org.br

Paulo Borba
Informatics Center — UFPE
phmb@cin.ufpe.br

Vander Alves
Fraunhofer IESE
vander.alves@iese.fraunhofer.de

## Abstract

With the growing academic and industrial interest in Software Product Lines, one area demanding special attention is tool support development, which is a pre-requisite for widespread software product lines practices adoption. In this demo, we present FLiP, a suite of tools consisting of 3 modules: a refactoring tool that implements code transformations for extracting product variations from Java classes into AspectJ aspects, a module that integrates with a Feature Model tool for updating a software product lines feature model accordingly to code transformations, and a building module that creates the final products. FLiP has been designed and tested in the context of industrial-strength mobile game software product lines.

***Categories and Subject Descriptors*** D.2.3 [*Software Engineering*]: Coding Tools and Techniques—Object-oriented programming

***General Terms*** Design, Languages

## 1. Introduction

The extractive and the reactive Software Product Line (SPL) (3) adoption strategies (5) involve, respectively, bootstrapping existing products into a SPL and extending an existing SPL to encompass another product. In both cases, product line refactorings (2) are useful to guide the SPL derivation process by extracting product variations and appropriately structuring them. They also help to assure the safety of the whole process by preserving SPL configurability, i.e., the resulting SPL has at least the same instances than the initial set of independent products being bootstrapped or the initial SPL being extended.


*OOPSLA'08,* October 19–23, 2008, Nashville, Tennessee, USA.
ACM 978-1-60558-220-7/08/10.

In single system refactoring, ensuring safety and effectiveness of a refactoring process already requires automation, in practice. In the SPL context, where complexity increases due to the need to manage a high number of variants, such support becomes even more indispensable. In this context, we describe FLiP, which is a suite of tools implemented as Eclipse plugins. FLiP has been designed and tested in the context of mobile game SPLs.

FLiP's refactoring tool (FLiPEx) implements code transformations (2) for extracting product variability from Java classes into AspectJ aspects (4). An aspect modularizes variant code related to a feature and allows plugging into the SPL core alternative variability for that feature. This refactoring module interacts with FLiP's feature manager (FLiPG), which integrates with Feature Model (FM) tools (1) for updating a SPL FM accordingly to code transformations. This, for example, might turn mandatory into optional features. Finally, FLiP's building system (FLiPC) interacts with the manager module, which is responsible for using the information stored to build the final products.

The refactoring process is presented to the user in the form of a wizard with which the user interacts to provide all the information required to perform the refactoring. After selecting the code to be refactored, the user is presented with a list of suggested refactorings; after selecting the refactoring, s/he selects or creates the features to which the code to be extracted belongs, and then chooses the destination aspects and associates them with the selected features. The possibility of selecting several destination aspects to which the generated AspectJ construct is copied helps the user to develop different implementations of the same variation.

## 2. Mobile Games

Mobile games (and mobile applications, in general) must adhere to strong portability requirements. This stems from business constraints: in order to target more users, owning

different kinds of devices, service carriers typically demand that a single application be deployed in a dozen or more platforms. Each platform generally provides vendor-specific Application Programming Interfaces (APIs) with mandatory or optional advanced features, which the developer is likely to use in order to improve game quality. In addition, devices have memory and display constraints, which further require the developer to optimize the application. In either case, adapting the game for each platform is mandatory. We analyzed and managed the specific kinds of variations arising from platform variation, where platform means a combination of vendor-specific API and hardware constraints (2). These kinds of variation tend to be considerably fine-grained such that they generally crosscut the game core and are tangled with other kinds of variation, suggesting AOP as a suitable candidate for modularizing them.

## 3. Refactorings

Contrary to the proactive approach, which is more like the waterfall model, we rely here on a combination of the extractive and the reactive approaches. There are a number of reasons for this. First, small to medium-sized organizations, which still want to benefit from SPLs, cannot afford the high cost incurred in adopting the proactive approach. Second, in domains such as mobile game development, the development cycle is so short that proactive planning cannot be completed. Third, there are risks associated in the proactive approach, because the scope may become invalid due to new requirements.

Our method first bootstraps the SPL and then evolves it with a reactive approach. Initially, there may be one or more independent products, which are refactored in order to expose variations to bootstrap the SPL. Next, the SPL scope is extended to encompass another product: the SPL reacts to accommodate the new variant. During this step, refactorings are performed to maintain the existing product, and a SPL extension is used to add a new variant. The SPL may react to further extension or refactoring.

## 4. Instance generation

After each refactoring, information about this process is stored, such as new aspects generated by the extraction and the features related with these aspects. The tool currently used for managing this information is Pure::Variants (1), but other tools could be used to manipulate the information, since FLiP is not dependent to this particular tool. All this information is compiled to generate product Line instances that will be used by FLiP's build system to create the final products.

## 5. Architecture

An overview of the FLIP tool suite's architecture, built upon the Eclipse plugin platform, is presented in Figure 1. The figure shows the relationship between FLiP's modules, the

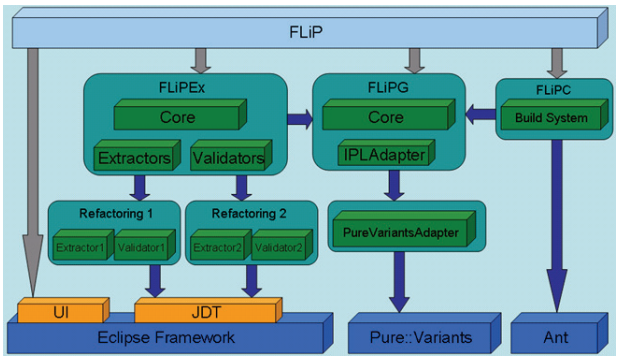


**Figure 1.** FLiP Architecture.

Eclipse framework, and a Feature Model tool. FLiP was built with extensibility as one of its most important design features.

All parts of FLiPEx (the refactoring module) are independent plugins. Core is the main plugin and gives support to the Extractors and Validators. Each refactoring has an Extractor responsible for removing code from a Java class and creating corresponding AspectJ code that inserts the variation back at its original site. Each Extractor has corresponding Validators, which are responsible to check if the selected code meets all the preconditions of its refactoring (2).

The FLiPG (feature manager) module provides integration with the underlying feature model tool. This integration is made using the available plugin that provides an implementation of the adapter interface that specifies which services FLiPG needs from the feature model tool, such as retrieving the list of features from the feature model, the components in the configuration knowledge, and updating both.

Finally, FLiPC (the builder tool) is the module responsible for actually generating the products' releases. FLiPC has a build system that uses Ant and information retrieved from FLiPG about the SPL products.

## Acknowledgments

This research was partially sponsored by CNPq and FINEP, and supported by Meantime and CESAR.

## References

[1] *Pure::Variants*. http://www.pure-systems.com.

[2] Vander Alves et al. Extracting and evolving mobile games product lines. In *SPLC'05*, pages 70–81. Springer-Verlag, 2005.

[3] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[4] Gregor Kiczales et al. Getting Started with AspectJ. *Communications of the ACM*, 44(10):59–65, October 2001.

[5] Charles Krueger. Easing the transition to software mass customization. In *PFE'01*, pages 282–293, 2001.