Science of Computer Programming 79 (2014) 70–85

Contents lists available at ScienceDirect

Science of Computer Programming

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

# FeatureIDE: An extensible framework for feature-oriented software development

Thomas Thüm [a,*], Christian Kästner [b], Fabian Benduhn [a], Jens Meinicke [a], Gunter Saake [a], Thomas Leich [c]

[a] University of Magdeburg, Magdeburg, Germany
[b] University of Marburg, Germany
[c] METOP GmbH, Magdeburg, Germany

ARTICLE INFO

*Article history:*
Received 24 January 2011
Received in revised form 31 May 2012
Accepted 4 June 2012
Available online 21 June 2012

*Keywords:*
Feature-oriented software development
Software product lines
Feature modeling
Feature-oriented programming
Aspect-oriented programming
Delta-oriented programming
Preprocessors
Tool support

ABSTRACT

FeatureIDE is an open-source framework for feature-oriented software development (FOSD) based on Eclipse. FOSD is a paradigm for the construction, customization, and synthesis of software systems. Code artifacts are mapped to features, and a customized software system can be generated given a selection of features. The set of software systems that can be generated is called a software product line (SPL). FeatureIDE supports several FOSD implementation techniques such as feature-oriented programming, aspect-oriented programming, delta-oriented programming, and preprocessors. All phases of FOSD are supported in FeatureIDE, namely domain analysis, requirements analysis, domain implementation, and software generation.



## 1. Introduction

Feature-oriented software development (FOSD) is a paradigm for the construction, customization, and synthesis of software systems [4]. A *feature* is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system [22]. The basic idea of FOSD is to decompose software systems into features in order to provide configuration options and to facilitate the generation of software systems based on a selection of features. A software product line (SPL) denotes the set of software systems that can be generated from a given set of features [17].

FeatureIDE is an Eclipse-based framework to support FOSD. The main focus of FeatureIDE is to cover the whole development process and to incorporate tools for the implementation of SPLs into an integrated development environment (IDE). FeatureIDE's architecture eases the development of tool support for existing and new languages for FOSD and thus reduces the effort needed to try out new languages and concepts.

Currently, the development of FeatureIDE focuses on teaching and research. FeatureIDE is used in software engineering lectures in Austin, Magdeburg, Marburg, Namur, Passau, Santa Cruz, and Torino. Before we implemented FeatureIDE, our students had to learn how to use several command-line tools, each with different parameters and output, whereas they are often used to working with modern IDEs such as Eclipse. With FeatureIDE, we provide a coherent user interface and automate tasks which previously required complex tool chains. We envision that FeatureIDE can also be used productively in future and serve as an open-source alternative to commercial product-line tools [36,12].

* Corresponding author.
*E-mail address:* tthuem@ovgu.de (T. Thüm).

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2012.06.002

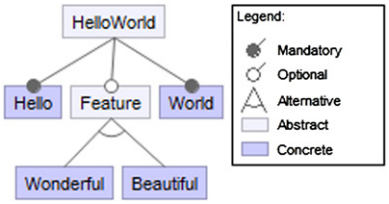


**Fig. 1.** A simple feature model modeling an SPL of Hello World programs. The features *Hello* and *World* are mandatory, and simply print the feature's name. The features *Wonderful* and *Beautiful* are alternatives, but they are not required. This SPL contains three valid Hello World programs.

FeatureIDE supports several implementation techniques for FOSD, and others can be integrated with low costs. The user interface for different implementation techniques is almost identical. Hence, FeatureIDE is especially qualified for teaching and for comparing SPL implementation techniques with respect to their applicability for the development of SPLs.

FeatureIDE underwent several changes since the initial development in 2004. In 2005, we presented a prototypical version of FeatureIDE [28]. At that time, FeatureIDE was just a front end for the programming language Jak of the AHEAD tool suite [9]. The development of this tool support was costly for a research language, but we earned positive feedback from other universities using FeatureIDE for teaching. Hence, we made this effort reusable for the FOSD implementation tools FeatureHouse [5] and FeatureC++ [6], and presented FeatureIDE as a tool framework for FOSD [25].

We present recent developments of FeatureIDE such as improved usability, new functionalities, and the newly integrated FOSD languages AspectJ [26], DeltaJ [37], Antenna [34], and Munge [31]. Furthermore, we discuss the effort of extending FeatureIDE, and describe how FeatureIDE is implemented and tested, while reporting pitfalls and opportunities interesting for other academic tool builders. Developers of Eclipse plug-ins get insights, as we share our lessons learned with FeatureIDE.

## 2. Feature-oriented software development

FOSD can be used to plan and implement SPLs (referred to as domain engineering) as well as to select features and generate customized programs (application engineering). FeatureIDE supports the FOSD process, and we distinguish between the following four phases.

1. *Domain analysis.* The aim is to capture the variabilities and commonalities of a software-system domain, which results in a feature model.
2. *Domain implementation.* Implementing all software systems of the domain at the same time, while mapping code assets to features.
3. *Requirements analysis.* Requirements are mapped to the features of the domain, and features needed for a customized software system are selected, resulting in a configuration.
4. *Software generation (or composition).* A software system is *automatically* built, given a configuration and the domain implementation.

Domain implementation and software generation highly depend on each other. An *SPL implementation technique* describes how features are mapped to implementation artifacts and how to generate customized software systems. In this section, we introduce feature models, configurations, and SPL implementation techniques currently supported by FeatureIDE.

### 2.1. Feature modeling and configuration

In SPLs, not all combinations of features are considered valid and lead to useful software systems. A *feature model* defines the valid combinations of features in a domain [22]. Feature models have a hierarchical structure, whereas each feature can have subfeatures [17]. The graphical representation of a feature model is a *feature diagram*, and an example is shown in Fig. 1. Connections between a feature and its group of subfeatures are distinguished as *and*-, *or*-, and *alternative*-groups [8]. The children of *and*-groups can be either *mandatory* or *optional*. A feature is *abstract* if it is not mapped to implementation artifacts, and *concrete* otherwise [40]. A feature model may also have cross-tree constraints to define dependencies which cannot be expressed otherwise. A *cross-tree constraint* is a propositional formula over the set of features that is usually shown below the feature diagram.

Feature models are a common notion for variability, and their semantics is as follows: the selection of a feature implies the selection of its parent feature. Furthermore, if a feature is selected, all mandatory subfeatures of an *and*-group must be selected. In *or*-groups, at least one subfeature must be selected, and in *alternative*-groups, exactly one subfeature has to be selected. Finally, all cross-tree constraints must be fulfilled.

A *configuration* is a subset of all features defined in the feature model. A configuration is *valid* if the combination of features is allowed by the feature model (i.e., if it fulfills the semantics of groups and all cross-tree constraints). Otherwise, the configuration is called *invalid*.