Feature Mining From a Collection of Software Product Variants

Rafat AL-msie’deen\textsuperscript{1}, Abdelhak D. Serial\textsuperscript{1}, Marianne Huchard\textsuperscript{1}, Christelle Urtado\textsuperscript{2}, Sylvain Vauttier\textsuperscript{2} and Hamzeh Eyal Salman\textsuperscript{1}

\textsuperscript{1} LIRMM / CNRS & Montpellier 2 University, Montpellier, France
\{Al-msiedee, Abdelhak.Serial, huchard, eyalsalman\}@lirmm.fr
\textsuperscript{2} LGI2P / Ecole des Mines d’Alès, Nîmes, France
\{Christelle.Urtado, Sylvain.Vauttier\}@mines-ales.fr

1 Reverse Engineering Software Product Lines

Similarly to car manufacturers who propose a full range of cars with common characteristics and numerous variants and options, software development might entail to segment users’ needs and propose to them a software family to choose from. Such software family is called a software product line (SPL) [1]. A SPL is usually characterized by two sets of features: the features that are shared by all products in the family, called the SPL’s commonalities, and, the features that are shared by some, but not all, products in the family, called the SPLs variability. These two sets define the mandatory and optional parts of the SPL. Software product line engineering (SPLE) focuses on capturing the commonalities and variabilities between several software products that belong to the same family.

In order to provide a more subtle description of the possible combinations of optional features (e.g., some optional feature might exclude another and require a third one), SPLs are usually described with a de-facto standard formalism called a feature model. A feature model characterizes the whole software family. It defines all valid feature sets, also called configurations. Each valid configuration represents a specific product, either it be an existing product or a valid product-to-be.

Software product variants are seldom developed in a disciplined way from scratch. Alternatively, ad hoc reuse techniques such as copy-paste-modify are used on the software’s code until some point where the need to discipline the development by adopting a SPLE approach raises. Expected benefits are to improve product maintenance, ease system migration, and the extracted features may lead to the production of new products. In order to capitalize from the existing code, reverse engineering is needed but manual analysis of the existing software product variants to discover their features is time-consuming, error-prone, and requires substantial efforts. Automating feature mining from source code would be of great help.

In literature, surprisingly, the reverse engineering of features (or feature model) from source code is seldom considered [2]. Existing approaches mine features from a single software product variant, while we think it is necessary to consider all available variants at a time [3].
2 A Three Step Process to Mine Features from Code

Feature location in OO source code consists in identifying the object-oriented building elements (OBEs) that implement a particular feature across software product variants. The OBE we consider are packages, classes, attributes, methods and their body. We assume that a feature can be mapped to one and only one set of OBEs: each feature has a unique implementation for the whole product family.

In order to mine features from the OO source code of software variants, we propose a three step process and rely on both Formal Concept Analysis (FCA) [4] and Latent Semantic Indexing (LSI) [5] techniques. Our approach:

1. extracts OBEs from each software product variant by parsing its code.
2. uses FCA to build a lattice from OBEs and software product variants that hierarchically groups OBEs from the software product variants into disjoint, minimal partitions. This classification provides us with two OBE sets: Common OBEs (that are shared by all variants and can be found on the top node of the lattice) and variable OBEs (that are shared by several but not all variants and appear at the bottom of the lattice).
3. clusters OBEs into features. Each OBE set is analyzed using LSI and FCA techniques to mine the optional and mandatory features based on the lexical similarity between OBEs.

We have implemented this three step approach and evaluated its produced results on a collection of ten ArgoUML products. The results showed that most of the features were identified [6]. In our future work, we plan to combine both textual and semantic similarity measures to be more precise in determining feature implementation. We also plan to use the mined common and variable features to automate the building of the studied software family’s feature model.

References