A methodology to recover feature models from object-oriented source code

Ra’Fat Ahmad Al-Msie’Deen, Abdelhak-Djamel Seriai, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, Hamzeh Eyal-Salman

To cite this version:

A methodology to recover feature models from object-oriented source code

Ra'fat AL-MSIE'DEEN*, Abdelhak Djamel SERIAI*, Marianne HUCHARD*, Christelle URTADO**, Sylvain VAUTTIER**, Hamzeh EYAL SALMAN*

*LIRMM, UMR 5506, CNRS and UM2, Montpellier, France
**LIGP, Ecole des Mines d’Alès, Nîmes, France
(Christelle.Urtado, Sylvain.Vauttier)@mines-ales.fr

A methodology to recover feature models from object-oriented source code

A methodology to recover feature models from object-oriented source code

Software Product Line Engineering

A Software Product Line is a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way [1].

SPLE Software Product Line Engineering focuses on capturing the commonalities and variations between several software products that belong to the same family. Capturing variants is the key activity that distinguishes SPLE from other software development approaches; it is called variability management.

Motivations

SPLEs are often designed after a number of product variants have been implemented using ad hoc techniques such as copy / paste / modify.

In order to migrate software products which are deemed similar to a product line, it is necessary to detect common features and variations between a set of product variants. Reverse engineering a feature model from existing software is a challenging activity.

Creating manually a feature model for an existing system is time-consuming, error-prone, and requires substantial effort from a modeler [3]. Automatic extraction of feature models from source code would improve product maintenance, ease system migration, and the extracted feature model may lead to the production of new products.

In recent years, a lot of work on reverse engineering has addressed the extraction of feature models from different artifacts but not from source code for a set of software product variants except [6].

Here, we present a methodology to extract a feature model from O.O. source code for a set of product variants to support the migration process from conventional software development to SPLE.

Overview of our Methodology

To our aim is the extraction of feature models that represent variations among a set of product variants, and enable to calculate product configurations using formal concept analysis (FCA).

We investigate products in which variability is represented by the names of packages, classes, attributes, methods, and bodies of methods (i.e. different choices for algorithms, methods invocation). We also investigate products in which variability lies in the contents of packages, classes, attributes, and methods.

Motivation: Model-based variability provides both a declarative and a more detailed formalism for each software product variant and considers variation in both names and contents.

Methodology Steps:

Step 1: O.O. source code is analyzed to extract source code elements (packages, classes, methods, attributes) for all product variants.

Step 2: Commonalities and variations are extracted for all product variants using FCA. FCA is a mathematical method that provides a way to identify "meaningful groupings of objects that have common attributes" [5]. We have tested it on a mobile media case study and obtained promising results.

Step 3: Variation to feature mapping model is defined. The aims of this model are to identify block of variations and atomic block of variations. Each block of variation in the model contains one or more atomic block of variations and each atomic block of variation represents one and only one feature. Source code elements have one or more source code variation, and each block of variations have one or more source code variation.

Step 4: Feature model elements such as root node, mandatory / optional features, parent features, cross-tree constraints or parent (group of features) constraints are extracted. Mandatory features appear on the common block, and optional features appear on the block of variations, the feature model constraints appear in block of variations. For example, if a single block has two features, it means that one requires the other. Finally, the feature model is synthesized.

Our methodology is based on 4 steps. We already have implemented step 1 extract source code elements and step 2 extract commonalities and variations. We have tested them on some standard case study and obtained promising results. We are still working on steps 3 & 4.

Related Work

Kryssel et al. [4] propose an approach to extract feature diagrams using FCA from incidence matrix that contain matching relation as input. It shows the parts of a set of function–block-oriented models that describe different controllers of a DC motor. Loesch et al. [5] present a new method based on FCA to analyze the realized variability in a software product line, and construct a lattice that provides a classification of the variability of feature models derived from the product line. Züld et al. [6] propose an automatic approach for feature identification from source code for a set of product variants. This approach assumes that the product variants use the same vocabulary to name packages, classes, attributes and methods in its source code.

References


Source code analysis

Source code analysis plays an important role in our methodology. Our analysis technique provides both a declarative and detailed information for each software product variant and considers variation in both names and contents.

Overview of our methodology

Methodology Steps:

Step 1: O.O. source code is analyzed to extract source code elements (packages, classes, methods, attributes) for all product variants.

Step 2: Commonalities and variations are extracted for all product variants using FCA. FCA is a mathematical method that provides a way to identify "meaningful groupings of objects that have common attributes" [5]. We have tested it on a mobile media case study and obtained promising results.

Step 3: Variation to feature mapping model is defined. The aims of this model are to identify block of variations and atomic block of variations. Each block of variation in the model contains one or more atomic block of variations and each atomic block of variation represents one and only one feature. Source code elements have one or more source code variation, and each block of variations have one or more source code variation.

Step 4: Feature model elements such as root node, mandatory / optional features, parent features, cross-tree constraints or parent (group of features) constraints are extracted. Mandatory features appear on the common block, and optional features appear on the block of variations, the feature model constraints appear in block of variations. For example, if a single block has two features, it means that one requires the other. Finally, the feature model is synthesized.

Our methodology is based on 4 steps. We already have implemented step 1 extract source code elements and step 2 extract commonalities and variations. We have tested them on some standard case study and obtained promising results. We are still working on steps 3 & 4.

Related Work

Kryssel et al. [4] propose an approach to extract feature diagrams using FCA from incidence matrix that contain matching relation as input. It shows the parts of a set of function–block-oriented models that describe different controllers of a DC motor. Loesch et al. [5] present a new method based on FCA to analyze the realized variability in a software product line, and construct a lattice that provides a classification of the variability of feature models derived from the product line. Züld et al. [6] propose an automatic approach for feature identification from source code for a set of product variants. This approach assumes that the product variants use the same vocabulary to name packages, classes, attributes and methods in its source code.

References