

Assessing Modularity of Feature Models with ACNs

Kanwarpreet Sethi Sunny Huynh Yuanfang Cai

Drexel University

{kss33, sunny, yfcai}@cs.drexel.edu

1. The Problem

Feature oriented programming (FOP) is an emerging approach to simplifying construction of software in product lines (Prehofer 1997). FOP works by developing the base program for the common part of a product line and individual features for each variation. After that, the base program code is automatically merged to produce a fully functional software system. In Model Driven Development (MDD), programs are defined in high level domain specific modeling languages and automatically transformed into code for the target platform. Trujillo et. al. combined FOP and MDD into FOMDD (Trujillo et al. 2007) so that code can be automatically generated for individual features by specifying high level models and automatically compose the individual features together to form a whole product line.

The problems is that since the features are developed independently of each other and automatically composed together, modularity of the resulting composed code cannot be guaranteed. The modularity of these products may be affected further by duplicated code in the individual features. This duplication not only increases the size of the executable and increases build times (Liu and Batory 2004) but also adversely affects modularity. With the advent of Model Driven Development, there is a need to assess the modularity of design models before they are transformed to source code to identify potential issues with modularity of the end products.

2. Our Idea

Our idea is to formalize the transformation of a software product line FOP model using a high-level design model that can be used to assess design modularity. After that, instead of using traditional coupling and cohesion metrics, we plan to use net option value (NOV) (Baldwin and Clark 2000) to assess the modularity of the resulting system.

The reasons to model and assess the transformation of FOMDD, instead of assessing the source code modularity after the source code is automatically generated, are as follows. First, after the source code is generated, the code that was cloned, shared or extended become undistinguishable at the source code level. As a result, the decisions about which parts represent commonality, and thus should remain stable, and which parts reflect variations that can be changed independently, are all blurred. Second, the higher-level model should be more abstract and smaller in scale, so that the user can understand system modularity without looking into the code in detail.

We plan to NOV analysis instead of coupling and cohesion metrics because the NOV model captures how modularity in design creates values, the essence of software product lines. Modularity has been shown to add value to software in terms of real options (Baldwin and Clark 2000) as it allows for investment of a superior replacement for a module or keep the current implementation, if that's the better choice. Hence, in a product line, modularity allows sustenance of the architecture and design for ease of maintenance and growth of the product line in terms of features supported.

According to Baldwin and Clark, *design rules* decouple a system, and create independent *modules*. Design rules are stable design decisions that decouple otherwise coupled decisions. The analogous of design rule in a product line is the commonality among features.

As long as the shared common parts remain stable, the independent features provide a portfolio of options that can be substituted with better versions. We believe that NOV can be a better metric for product lines.

3. The Approach

Our propose to capture FOMDD model transformation using an *Augmented Constraint Networks* (ACN) (Cai et al. 2007), which can be used to automatically generate a *Design Structure Matrix* so that the modular structure of a system can be visualized, and NOV value can be calculated (Cai et al. 2007).

The core of an ACN is a constraint network that consists of variables representing design decisions and logical constraints representing the dependencies among design decisions. An ACN also contains a *dominance relation* that formalized the notion of design rules as a pair-wise *cannot – influence* relations. Based on the constraint network, we have formally defined what it means by "one decision depends on the other". The dominance relation allows for automatic decomposition of the system into *modules*, each representing an independent task assignment.

A DSM is a square matrix in which columns and rows are labeled with variables, and the matrix is populated with a pair-wise dependence relation that can be automatically derived from an ACN. In order to reveal the modular structure of a design, the user can aggregate all the design rules (commonalities) into the first block of the matrix. After that, the user can assess if all other blocks along the diagonal (modules) are independent from each other (there are no off-block dependencies). After that, the NOV value can be computed according to the number of modules, their sizes and their dependency relations.

We plan to model each feature and base model with a complementary ACN model. Within each feature, we model all the dimensions as variables, and model the relations within and among features as constraints. Similar to how features are picked to synthesize the final constructed software, the individual ACN models of the features can be combined to form the ACN for the final software product, with every possible model combination having a complementary ACN model.

Take the small calculator generator in (Batory et al. 2004) as an example. The base model consists of a *clear()*, an *enter*, and three global variables, and the extended parts include *add()* and *sub* functions. We

thus use five design variables to model the five dimensions in the base. We also use another two ACNs to model the two extensions, each having one dimension. To assess the modularity properties of the resulting system, we combine these ACNs together. Depending on how the FOMDD translator works, the dependencies among these three ACNs should be automatically detected. For example, all the extended functions will depend on the three global variables, which should be automatically detected and transformed.

The challenge is to combine individual feature ACNs to compose the full ACN, we would need an algorithm similar to algorithms used in FOP (such as AHEAD (Batory et al. 2004)) to combine individual features to produce the final product. However, ACNs being more abstract models, might be easier to combine. Since ACN uses logical expressions as the computational core, when ACN combination is needed, inconsistent constraints should be detected and resolved, if possible. Each feature's interaction with the base model and other features will need to be analyzed to construct a well formed full ACN. Once this ACN is produced by combining selected feature ACNs, we can assess the modularity of this model using NOV metrics (Cai et al. 2007), as well as other metrics such as volatility and concern scope/impact.

References

- Carliss Y. Baldwin and Kim B. Clark. *Design Rules, Vol. 1: The Power of Modularity*. MIT Press, 2000.
- Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, June 2004.
- Yuanfang Cai, Sunny Huynh, and Tao Xie. A framework and tools support for testing modularity of software design. In *22nd IEEE/ACM International Conference on Automated Software Engineering*, November 2007.
- Jia Liu and Don Batory. Automatic remodularization and optimized synthesis of product-families. In *3rd International Conference on Generative Programming and Component Engineering*, October 2004.
- Christian Prehofer. Feature oriented programming: A fresh look at objects. In *11th European Conference on Object Oriented Programming*, 1997.
- Salvador Trujillo, Don Batory, and Oscar Diaz. Feature oriented model driven development: A case study for portlets. In *29th International Conference on Software Engineering*, May 2007.