

# An Evaluation to Compare Software Product Line Decision Model and Feature Model

Liana B. Lisboa<sup>1</sup>, J. Jenny Li<sup>2</sup>, P. Morreale<sup>2</sup>, D. Heer<sup>2</sup> and D. M. Weiss<sup>3</sup>

<sup>1</sup>*Aplicateca, Bahia, Brazil*

<sup>2</sup>*Department of Computer Science, Kean University, Morris, NJ, U.S.A.*

<sup>3</sup>*Department of Computer Science, Iowa State University, Ames, IA, U.S.A.*

**Keywords:** Software Product Line, Product Family Scoping, FAST, FODA, Decision Model, Feature Model.

**Abstract:** A key issue in defining a product line is specifying the allowable set of products that will be produced using product line assets, i.e., the scope of the domain. This paper conducts an evaluation to compare two different approaches for defining domain scope, decision model as defined in the Family-oriented Abstraction, Specification, Translation (FAST) process and the feature model as defined in the Feature-Oriented Domain Analysis (FODA) process. The comparison is based on applying the approaches to two examples, one a textbook example and the other to a product line we maintain on an open source website, in order to identify guidelines for improving the identification and representation of a software family. Our conclusion is that decision model includes both commonality and variability definition at software architecture level and thus it is more suitable for larger product line with a significant number of commonality and variability.

## 1 INTRODUCTION

Software Product Line Engineering is a novel software engineering technology taking advantage of the commonality and expected variability among members of a software family (Weiss, 2013). A key issue in defining a product line is specifying the allowable set of products that will be produced using product line assets. We call this the scope of the product line. If the scope is too wide, it becomes difficult to develop a set of assets that can be efficiently (re)used to produce members of the product line. If the scope is too narrow, it is not worth the investment to create a set of reusable assets. Accordingly, it is important to have a systematic approach to defining the scope of a family.

Several different approaches to defining scope have evolved over the last few years (Frakes, 1998) (Moon, 2005). Approaches such as Family-oriented Abstraction, Specification, Translation (FAST) (Weiss, 1999) use an assumption-based approach, in which one specifies the assumptions about what is common to products and what may vary among products. The resulting specification is called a decision model to define commonality and select the

set of allowed values of variability in a product line.

Approaches such as Feature-Oriented Domain Analysis (FODA) (Kang, 1990) use a feature model, configured as a graph, with annotations, that defines the features available for products and specifies options, alternatives, and exclusions among features. Its focus is on variability.

Both approaches use their specifications to guide the creation of the architecture for the product line and to guide the process of producing a product. As an attempt to improve both approaches, and understand better their relative advantages in practice, we conducted an evaluation study to compare their usage in two product lines. Researchers in the past had attempted to compare the two approaches as reported in (Czarnecki, 2012) and (Schmid, 2011). Their comparison did not take into consideration of decision model's ability to represent commonality besides variability and decision model's emphasis on software architecture. The main goal of our comparison was to identify guidelines for improving the identification of a software family using either feature model or decision model on a software architecture level and for selecting either one of them depending on the characteristics of a family.

Furthermore, the paper also provides a set of procedures to switch from the decision model to the feature model, which was not discussed in any existing papers. Interchangeability is very important because a product line might grow and its development situation may change as well, in which case one model might become more advantageous to such an extent that the architectures would decide to alter modelling approach mid-way through a product line development.

In another related work, Wartik and Prieto-Diaz (1992) define a framework for comparing reuse-oriented domain analysis approaches, in order to obtain a common conceptual ground, to establish a way to determine which approach best suits each needs, and to study the feasibility of a unified approach. Ferré and Vegas (1999) survey domain analysis methods, with the aim of finding an explanation for the diversity of domain modelling techniques and studying which techniques the methods can use to represent reusable elements. They are different from our work of focusing on decision model and feature model which are the two most prominent techniques for analysing and specifying product line domains. These studies focused on defining a framework or process as a way of comparing the guidelines of the domain analysis processes. However, this paper is the first to evaluate and compare specific examples of applications for each approach.

The remainder of this paper is organized as follows. Section 2 gives a more detailed discussion of both approaches, and Section 3 describes the two product lines where they were applied. Section 4 discusses the results of the two evaluation studies. Section 5 concludes the evaluation and points to future research directions.

## 2 TWO-METHOD OVERVIEW

In this section, we present an overview of the two methods, including their description, guidelines and generated artefacts.

### 2.1 Decision Model

The FAST process defines a product line as a family designed to take advantage of common aspects and predicted variability. The FAST process seeks to reduce the delay to production of the first family member by introducing systematic methods for defining the scope of a product line, for creating a way to describe products in the product line, and for

generating products from their descriptions. Once the initial investment is made, the time to produce products may be quite short. The initial investment consists of the effort needed to produce a product line engineering environment, and may take a year or more of effort because it is often applied to very large product line. Thereafter, the product generation can be automated. In our industrial case, it takes less than one staff/hour to generate a product member once the product line environment is in place.

A commonality analysis is a systematic way of gaining such confidence and of deciding what the scope of the family is, i.e., what are the potential family members. The FAST commonality analysis process creates a decision model designed to satisfy the following goals.

- 1) Define the scope of the family by specifying assumptions about what is common to family members (commonalities) and what is variable among family members (variability and its values).
- 2) Provide a precise statement of how much variability is accommodated by the product line by specifying what the allowable set of values is for variability. Each such specification is called a parameter of variation of a decision model.
- 3) Define common terminology for the family by creating a dictionary of terms for the family.
- 4) Provide a basis for constructing the architecture for the product line that can be used to produce its products. For example, variability is usually mapped to one module in the architecture.
- 5) Provide a basis for automating the generation of members of the family by specifying three elements: (a) a partial ordering among the parameters of variation that dictates the order in which values for the parameters must be chosen, (b) constraints among parameters of variation that limit the allowable choices of values, and (c) a mapping between parameters of variation and modules in the product-line's architecture, known as the system composition mapping.

The ordering and constraints among parameters of variation and the system composition mapping together provide a way to construct a model of what decisions to be made to specify a product and what modules are needed to construct it, which is called a decision model.

### 2.2 Feature Model

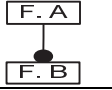
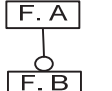
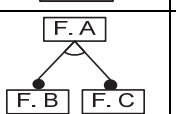
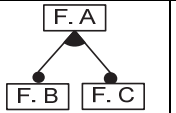
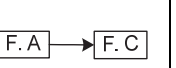
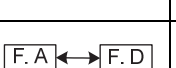
Feature modelling was proposed as part of the Feature-Oriented Domain Analysis (FODA) method

(Kang, 1990) with the objective of representing the variable features identified in a domain. Kang et al. define a set of activities and artefacts for the domain analysis process in FODA. The artefacts include the feature model, the *context model*, which defines which applications are part of the domain; the *dictionary*, which contains domain terminologies; the *entity-relationship* and *functional models*, and the *architectural model*. We focus here on the feature model since it is the main asset for our comparison. The variability represented in the feature model are characterized through features, which are defined as distinguishable characteristics relevant to some stakeholders in the domain.

The most common way of representing the feature model is through a hierarchical view, usually a tree, where the features become more detailed as the levels increase in the tree. In order to map the domain features with their variability, it is necessary to analyse the information sources, which can be *textbooks*, *existing applications* and *domain experts*. Thus, the roles are the *domain experts*, *domain analysis*, *users* and *requirements analysts*.

The feature model is usually composed of a root feature and sub features. The possible sub feature types are **mandatory**, **optional**, **alternative** and **or** (Czarnecki, 2000). The model can also include composition rules, which are divided into **exclusion** and **implication**. In Kang et al.'s definition, the rules are represented textually. Here we present them as dependency relationships in Table 1 notations.

Table 1: Feature model notations.

Type	Notation	Description
Mandatory		A mandatory feature will always be present in the product
Optional		The feature may or may not be present in the product
Alternative		Exactly one feature of the set is selected for the product
Or		At least one feature of the set is selected for the product
Implication		If the source feature (F.A) is selected, then the destination feature (F.C) is also selected for the product
Exclusion		Both features cannot together be in the same product

Besides the feature representation, the process also includes documentation that defines the features including synonyms, description and information sources, and the composition rules between the features. Once the feature model activity is finished, its artefacts can be used to derive functional models, such as activity and state charts, the domain architecture, and the product selection.

There are several extensions for the Feature Model proposed in FODA. Perhaps the best known is from (Czarnecki, 2000). They extend the feature notation with the "or" feature and propose a new set of documentation for each feature with *rationale*, *priority*, *constraints* and *binding sites and models*. Other extensions are PLUSEE (Gomaa, 2004) and PLUSS approach (Eriksson, 2005). Feature models have also been used to support a diversity of different techniques, like legacy systems (Kang, 2005), constraints programming (Benavides, 2005) and grammars (Batory, 2005).

### 3 EVALUATION STUDY

We conducted two case studies to compare the Feature Model and the Decision Model approaches. The first study, which we used as a pilot case, was the Floating Weather Station (FWS) family detailed in the FAST process book (Weiss, 1999). The second was the open-source software product line project PolyFlow testing tool family (Li, 2013).

The comparison starts from the Decision Model approach for both cases. And then the existing artefacts were used as a base to generate the feature model. For the generation of the feature models in the project, we used the ToolDay (Tool for Domain Analysis) (Lisboa, 2007) for tool support. The tool uses a somewhat different notation for representing the *alternative* and *or* features groups than the one shown in Table 1. Figure 1 shows the ToolDay notation. The *GroupId* tag in the relationships identifies the features for this group.

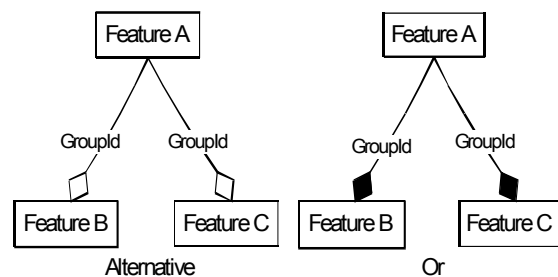


Figure 1: ToolDay for "Alternative" and "Or".

### 3.1 FWS Family

The Floating Weather Station Family is described in Chapter 5 of (Weiss, 1999). This example is used to exemplify the Decision Model of FAST process. For our comparison we used only the artefacts of the Commonality Analysis phase. The case was chosen because it describes a simplified version of a real system. Since it is a small domain, it was easier to test our comparison process.

Floating Weather Station (FWS, Figure 2) buoys are deployed at sea and periodically report the current wind speed via messages sent by radio. Each member of the FWS family contains an on-board computer that controls the operation of the buoy while it is at sea. The purpose of the commonality analysis is to provide the following capabilities for the FWS family of buoys:

- A way to specify the configuration of a particular buoy.
- A way to generate, for a specified buoy configuration, the software that controls a buoy while it is at sea.

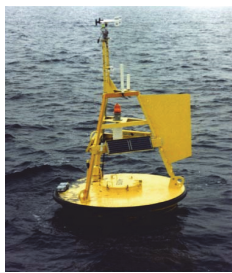


Figure 2: A Picture of a Floating Weather Station.

Floating Weather Stations interact with systems that are equipped to receive the signals transmitted by the onboard radio transmitter. Such systems may be shipboard, ground-based, or satellite-based. The software for the FWS domain interfaces with the sensors that the FWS uses to monitor wind speed, and with the transmitter that the FWS uses to send messages. Figure 3 shows these interactions and gives a brief indication of the nature of the interface. For example, the Sensor domain receives commands from the FWS software and sends data to it.

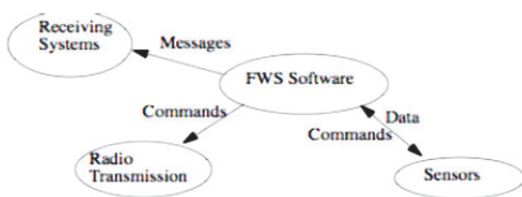


Figure 3: FWS Domain Interactions.

Table 2: Commonality and Variability of FWS Family.

Commonality	
C1	FWS transmits messages with the wind speed at fixed intervals
C2	The wind speed is calculated as a weighted average of the sensors reading
C3	FWS has one or more sensors for wind speed
C4	FWS has a radio transmitter
Variability	
V1	The formula for computing the wind speed from the sensors reading can vary.
V2	Types of msg can vary in content and format
V3	The transmission period of message can vary
V4	The number of wind speed can vary
V5	The resolution of wind speed sensors varies
Parameters of Variation and Their Values	
PV1	Weight applied to high-resolution sensors reading: 1 – 100
PV2	Type of message : Short or long message
PV3	Transmit period: 1 – MaxTransitPeriod
PV4	Maximum number of sensors: 2 – 20

The first step for the comparison was to understand the domain to be modelled. The information sources for that were the Commonality Analysis documents. Furthermore, some products were identified through web search engines. Afterwards, the commonality analysis document was analysed. It consists of 6 commonality, 9 variability and 13 parameters of variations for the variability. Table 2 shows some of the items identified.

Based on the above information, we started to define how the commonality and variability could be mapped to a feature model. The FAST commonality analysis process advises grouping commonality and variability according to their similarities, starting with the categories external interfaces and behaviour; this was helpful advice. For example in the FWS case, the commonalities C1, C2 are grouped as Behaviour, and C3 and C4 are grouped as Devices. Variability V1, V2 and V3 belong to Behaviour and V4 and V5 to Devices. Thus, we can map these groups as a high level in a feature model.

The values defined in the parameters of variation (PoV) were used for defining the variants in the model. Whenever the value set had a small number of options, such as PV2 in Table 2, which has two options: Short or Long, they could be mapped to the variable features – *Optional*, *Alternative* and *Or*. For PV2, it is not possible to have both message types in the FWS application. Thus, it can be mapped to an *Alternative* feature group. If more than one option in

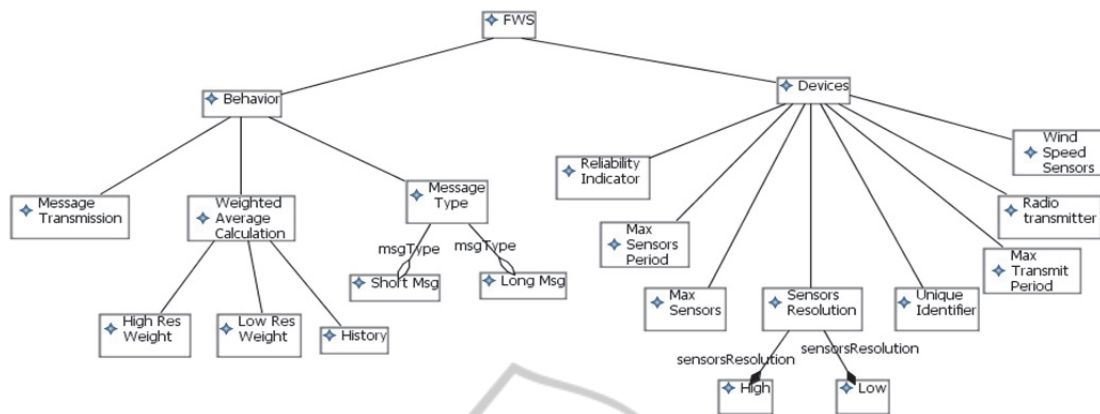


Figure 4: Feature Model for the FWS Family.

the PoV is possible, it is equivalent to an *Or* feature group; and parameters whose values indicate that the functionality may be either included or excluded are considered *Optional* in the feature model.

Besides variability, decision model includes also the definition of commonality which is essential in define the scope of a family or the domain of a product line. Feature model doesn't explicitly represent commonalities, but we could treat commonality as mandatory features in the feature model. This is only possible when the number of commonality is smaller than variability, otherwise the resultant feature model diagram will be too large to handle. Mixing commonality and variability may also hinder reader's ability to comprehend the model and thus we decided not to convert commonalities.

Figure 4 presents the final feature model for the FWS domain, excluding commonality because we use mandatory variability for other purpose as explained later. Figure 4 shows that FWS' root feature is the *FWS* feature. It has the sub features *behavior* and *device*, which were extracted from the groups defined in the document. The sub features that belong to the *alternatives* (sub features of *Message Type*) and *ors* (sub features of *Sensors Resolution*) sets were extracted from the Parameters of Variation table.

However, the majority of the PoV have a large range of values, for instance *2 to 20* (PV4) or *1 to 100* (PV1). These variations were represented as a mandatory feature, since all of the values are greater than or equal to one, and we can use the *constraints attribute* in the documentation to specify the value range, rather using explicit *Or* to select from the 100 choices. Mandatory is similar to commonality.

As it can be seen from this example, our procedure of converting decision model to feature model is quite straightforward. It should also work for the other way around of converting a feature

model to a decision model. We hope our procedure will become a standard way for such a conversion. This example also shows that feature model works well for specifying the variability of small families.

Another reason why feature model works well in this example is that this example does not include any kind of dependency constraints. Feature model only supports two kinds of constraints: constraint attribute to specify a range and "and"/"or" constraints among adjacent features. This limitation does not affect its ability in specifying small product lines. But for very large product lines with very complex constraints, its capacity might not be sufficient, which we will show through an example in the next subsection.

### 3.2 PolyFlow

FAST Decision Model analysis was applied to the development of one of Avaya's research product lines, now an open source called PolyFlow (Li, 2013). It is a family of test tools that minimally provide the capability to execute test cases and calculate associated coverage measures. Initially, the tool operated only on Java programs and calculated line coverage and slicing measures. As it became more popular, users requested features such as automated test generation, testing of C and C++ programs, and execution on different platforms, such as Windows, Linux, and VxWorks. It soon became clear that PolyFlow was a family of test tools that should be developed as a product line. More descriptions of the PolyFlow family can be found in (Devine, 2012), and (Li, 2008).

The first step of the PolyFlow product-line process was to conduct a commonality analysis that identified the common aspects as well as the variability for the family. The PolyFlow product family includes **34** commonality and **41** variability,

each variability having an associated parameter of variation that has from 2 to 6 values. An example commonality is that all family members must derive the *control flow graph* of the program under test and be able to display it for the user. Two variability examples are the programming language of the system under test, such as *Java* or *C*, and the type of *error path identification* mechanisms.

Table 3: Partial PolyFlow Commonality, Variability, and Parameters of Variation.

Commonality	
C1	Parsing
C2	Program Structure Graph
C3	Error Path Identification
C4	Reporting
Variability	
V1	Programming Languages of target system
V2	Java code types
V3	Ways to identify error path
V4	Testing Platforms
Parameters of Variation and Values	
PV1	Language: Java, C/C++, C#
PV2	Java code type: Java source, Java byte code
PV3	Error path: Keyword, user-defined
PV4	Platform: *nix, VxWorks, Windows, Switch

Table 3 shows a few commonalities, variability and parameters of variation of the PolyFlow product line. Table 3 includes four commonalities among the product family members: *Parsing*, *Program Structure Graph*, *Error Path Identification* and *Reporting*. All product family members must provide those four functions. Table 3 also shows four variability, *Language*, *Java Type*, *Error Path Identification*, and *Platform*, each of which has a list of valid parameter values. For example, the language can be *Java*, *C*, or *C++* (PV1 in Table 3), and the platform can be any of *\*nix*, *VxWorks*, *Windows*, or *Switches* (PV4 in Table 3). Please note that this is only a very small selection from PolyFlow family.

Based on the available documents and on the guidelines defined in the pilot, we built the feature model. *PolyFlow(eXVantage* in the diagram) was considered the root feature, and *parsing* and *program structure graph* some of its first sub features, as shown in in Figure 5, with details later.

For the feature *Parsing*, the model included the commonality *Parse/AST* and the variability *Target Language*, which was further detailed in an alternative relationship between *Java* and *C/C++*.

The *Java* feature has two options, *source-code* and *byte-code*. The *Error Path Identification* feature can have more than one form of identification, which is represented by the *Or* feature group with: *User Defined Words* and *Exception Handling*.

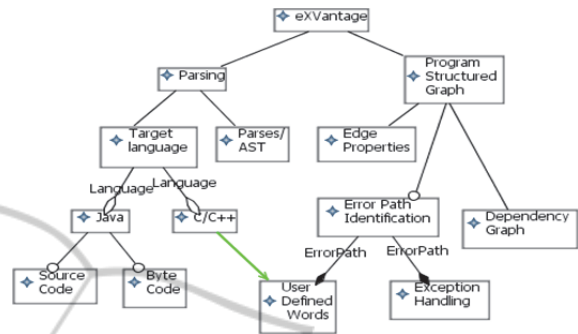


Figure 5: Fragment of PolyFlow Feature Model.

One characteristic of this domain that was not present in the FWS family is the existence of dependencies among the variability, i.e. complex constraints. Only the simple constraints *If (V1 is C/C++) then V3 is User-Defined Keywords* is represented in the feature model fragment in Figure 5, as a relationship arrow between the features *C/C++* and *User Defined Keywords*. Relationship arrows are difficult to represent constraints involving more than three features in the diagram.

The reason why we selected only a subset of PolyFlow decision model is that we found that feature model is more suitable for smaller product lines. If we include more commonality and variability from PolyFlow, we wouldn't be able to fit the feature model into this paper. On the other hand, with a tabular form for decision model on an architecture level only, it can represent large families because of the length of the table can grow without affecting the content. A complete set of PolyFlow decision model can be found in (Li, 2013).

## 4 RESULT ANALYTICS

The goal of our study is to compare the usage of the two product line scoping methods, decision model and feature model. So far the previous section has shown that the two are interchangeable for small product lines. So any evaluation of either model should imply the feasibility for the other model. In this section, we will take a look at their difference and the impacts of the difference to the users. Please note that our comparison presents an user point of view on whether and how to use the two models.

They differ on how these activities should be carried out formally.

The strengths of the Decision Model include:

- 1) Precise specifications of commonalities and variability: the tabular format gives sufficient space to add or remove them, as well as specifying them. This property of decision model makes it suitable for defining product lines with high quality requirements, such as safety-critical ones.
- 2) Support for expression of (sometimes complex) constraints among parameters of variability: a constraint section in the table gives room to specify any kind of constraints.
- 3) Expression of binding time: the sequence in decision model indicates the order the variability should be selected due to some constraints and each has a column used to specify its binding time.
- 4) Guidance for constructing product architecture: This character permits a direct mapping from parameters to components used to build products, thereby allowing automated generation of products, which is a key property of FAST process.

More work on evaluation of decision model is still needed. In our trial, it took a long time for us to get a complete decision model because the product line is so complex. Even though the product generation time is reduced substantially, the effort needed to produce a decision model is quite large: in our PolyFlow case study, it took over 1 staff/year. This commonality analysis time to create decision model has no counter-part in conventional software development time, which hinders our ability to compare them quantitatively, even though we did a study of quality improvement recently (Devine, 2012).

The feature model process strengths, including its extensions (Czarnecki, 2000), are:

- simple visual representation;
- provides a clear view of the full domain; and
- has several optional fields for documenting (such as, description, priorities and constraints) every feature in the model.

These strengths make it perfect for feature modeling of small and light weight product lines, where all features can be viewed clearly in one simple visual representation. However, those strengths could also be a drawback for very large systems. The feature model process does not incorporate complex constraints among features and has not been proven to scale to large systems. In fact, in our PolyFlow

trial, we gave up trying to convert the entire decision model to feature model because it became too complex to maintain.

During the comparison, one of the most difficult representations of variability in the commonality analysis for the feature model is when the value set of the Parameters of Variation involve a great variety of options – such as PV1 in Table 2 where the weight for the high-resolutions sensors can vary from one to one-hundred. This type of variability is hard to represent as feature groups (*Alternatives* or *Or* types), because of the quantity of features the group would have. Our innovative solution is to map the values as an attribute in the feature (Czarnecki, 2000). Therefore, these attributes can have constraints defining their value range. In the PV1 example, the range would be from 1 to 100.

Furthermore, the binding time, easily represented in the commonality analysis, and included in (Czarnecki, 2000), is not easily identified in a feature model. Setting the binding time as an attribute in the variability features would make it more visible. During the comparison, we also found that it was possible to identify requirements for tool automation in both processes. They were:

- Features, commonality, variability and traceability among documents (both processes);
- Consistency checker (both processes);
- Visibility level (FODA);
- Complex constraints among features (FODA), such as those proposed in (Batory, 2005); and,
- Automatic generation of reports (FODA).
- Automatic product generation (FAST)

Overall we found that both feature model and decision model are very useful for defining product lines. The key guideline in selecting them is that feature model works better for small product lines and decision model works nicely for large system with complex constraints and high quality requirements such as safety-critical product lines.

## 5 CONCLUDING OBSERVATION

We have provided here a qualitative evaluation of the decision model of FAST process and the feature model of FODA processes based on existing analyses of two domains: one an example and the second a real industrial domain.

The decision model has a textual and tabular view of the domain's commonalities and variability, while the feature model has a visual representation. However, both processes have the same goal, which

is to identify the common and variable characteristics of the domain and to document it in order to be used in the architecture definition and as a basis for the generation of product line members. Conversely, the processes achieve this outcome in different ways. The decision model seems to confer advantages in precision, and its ability to express binding times and constraints on the value spaces for variability. Its model representation uses simply assumptions over the domain, representable as predicates, for the commonality and the variability. One result is that the decision model can be viewed as the basis for creating a domain specific language.

The primary advantage of feature models is their visual appeal. For simple domains where there are few constraints among variability, they help the domain engineer to visualize the domain for better understanding and maintenance. The visual representation must be supplemented with attribute definitions as domain complexity increases. Also as domain size and complexity increase, the visual diagram will require manipulation, such as panning and zooming to preserve the ability to see the entire domain at one time.

Both approaches benefit from the ability to impose structure on the domain and both may become unwieldy as domain size and complexity increases. But overall they are the state-of-the art technology to define the scope of a software product line. Our case study shows that they work for both cases and the guideline for selection is to use feature model for small product lines and decision model for larger ones with precision requirements. The future research direction of this work would be to compare the two as applied to other complex cases and collect quantitative measurements. In addition, it would be desirable to do the decision model based on an existing feature model real case, even though the one way procedure was given in the paper.

## REFERENCES

- Weiss, D. M., 2013. Software Product Line Hall of Fame. [http://www.sei.cmu.edu/productlines/plp\\_hof.html](http://www.sei.cmu.edu/productlines/plp_hof.html).
- W. B. Frakes, R. Prieto-Díaz, and C. J. Fox, 1998. "DARE: Domain Analysis and Reuse Environment," *Annals of SW Eng.*, vol. 5, no. 1998, pp. 125-141.
- M. Moon, K. Yeom, and H. S. Chae, 2005. "An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line," *IEEE Transactions on SW Eng.*, vol. 31, no. 7, pp. 551-569.
- D. Weiss and C. T. R. Lai, 1999. Software Product-Line Engineering: A Family-Based Software Development Process: Addison-Wesley, 1999, pp. 448.
- K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, 1990. "Feature-Oriented Domain Analysis (FODA) Feasibility Study," *Technical Report CMU/SEI-90-TR-21*, SEI, CMU, Pittsburgh.
- K. Czarnecki, and et al., 2012. "Cool features and tough decisions: a comparison of variability modeling approaches", *VaMoS'12 Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Sys.*, pp 173-182, NYC, NY, USA.
- K. Schmid, and et al., 2011. "A comparison of decision modeling approaches in product lines", *VaMoS'11 Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pp 119-126, New York, NY, USA.
- S. Wartik and R. Prieto-Díaz, 1992. "Criteria for comparing reuse-oriented domain analysis approaches," *International Journal of Software Engineering and Knowledge Eng.*, vol. 2, no. 3, pp. 403-43.
- X. Ferré and S. Vegas, 1999. "An Evaluation of Domain Analysis Methods," *4th Int. Workshop on Evaluation of Model Methods in Sys. Ana. & Des.*
- K. Czarnecki and U. Eisenecker, 2000. *Generative Programming – Methods, Tools, and Applications: Addison-Wesley*, pp832
- H. Gomaa and M. E. Shin, 2004. "Tool Support for Software Variability Management and Product Derivation in Software Product Lines," *Workshop on Software Variability Management for Product Derivation, SPLC*, Boston, USA.
- M. Eriksson, J. Börstler, and K. Borg, 2005. "The PLUS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations," *SPLC*, Rennes, France, pp. 33-44.
- K. C. Kang, M. Kim, J. Lee, and B. Kim, 2005. "Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets – a Case Study," *SPLC*, Rennes, France, pp. 45-56.
- D. Benavides, P. Trinidad, and A. Ruiz-Cortes, 2005. "Automated Reasoning on Feature Models," *Conf. on Advanced Information Systems Engineering (CAiSE)*, Portugal, pp. 491-503.
- D. Batory, 2005. "Feature Models, Grammars, and Propositional Formulas," *Software Product Lines Conference (SPLC)*, Rennes, France, pp. 7-20.
- Li, J. J., 2013. <http://www.trustie.net/projects/project/show/PolyFlow>
- L. B. Lisboa, V. C. Garcia, E. S. Almeida, and S. L. Meira, 2007. "ToolDAY - A Process-Centered Domain Analysis Tool," *Brazilian Symposium on Software Engineering (SBES) - Tools Session*, João Pessoa, Paraíba, Brazil, pp. 54-60.
- Devine, T. R., Goseva, K. Krishnan, S., Lutz, R. R. and Li, J. J., 2012. "An empirical study of pre-release software faults in an industrial product line", *Proc. of IEEE ICST2012*, April.
- Li, J. J., Slye, H., Trung, D. and Weiss, D. M., 2008. "Decision-model-based Code Generation for PLE", *Proc. of IEEE SPLC2008*.