Program Families

Min Deng and Ji Zhang
CSE870 Advanced Software Engineering MiniProject
Computer Science and Engineering
Instructor: Dr. B. Cheng
Michigan State University
Contact Information: chengb@cse.msu.edu

April 12, 2002

Abstract

In this paper we summarize one of David Parnas' papers, “On the design and development of program families.” In his paper, Parnas introduced the original idea of Program Families. This idea has a tremendous impact on computer techniques today and a lot of work has been done based on this idea. We introduce three papers that are based on this idea to show its impact. The first paper is about a process named Commonality Analysis that can be used to identify the commonalities and variabilities of potential family members. The second paper is about a special component called an adaptable component that can represent a component family by varying parameters of adaptability. The third paper introduces two economic models that justify the use of the product line approach. We also present a paper that uses Parnas’ idea but does not cite his paper. This uncited paper introduces a scoping method that can be used in the product line approach.

Keywords: family, product line, adaptable, domain.

1 Overview of Program Families

This section describes the work that David Parnas did in the area of Program Families [1]. The title of his paper indicates us that it is about how to design and develop program families.

Parnas’ paper answers two questions, “What is a program family?” and “Why people should study a program family instead of individual programs?” A set of programs is considered a family, when it is the case that in order to study this set, it is necessary to study the common properties among the elements of the set first, and then study the specific properties of the individual family members. Different versions of programs and programs applicable to various platforms are examples of program families. The members of a program family share several common features; therefore we can first study these commonalities (common features), and then cover the variabilities (implementation details) of each program. This approach is a more efficient way to study, develop and maintain program families than studying each individual program respectively from the beginning.

There are three techniques used to design and develop program families. One is the traditional technique called “sequential development” and the other two are new techniques: “stepwise refinement” and “specification of information hiding modules”. The main difference between the traditional approach and the new approaches is the entities that are reused when developing new family members. In the traditional approach, the first family member is developed to the “working” stage, and other members are developed by modifying complete working stage programs. This approach will bring some inappropriate characteristics from the ancestor members to the descendant members, and as a result, the descendants will have performance deficiencies. In the new techniques, the reused entities are intermediate designs. When developing new members in the family, we no longer modify complete programs, but derive the new family members from the intermediate designs. The main difference between these two new techniques is that in the Stepwise Technique the intermediate representations are incomplete programs, while in the Specification Technique the intermediate representations are specifications. Therefore in the Stepwise Technique each stage refines the result of earlier stages and tends to narrow down the variations of family members. While in the Specification Technique, the goal of each stage is to make it possible to postpone the decisions about the program and leads to a broader program family. As these two new methods are complementary in software development, they can be combined together.

Parnas’ Program Family Model has a tremendous impact on today’s techniques. There are numerous software products developed as families. One example is the Microsoft Office Family. Different products of one Microsoft
Office version, such as Word 2000, Excel 2000, etc., are various family members. Different versions of the same Microsoft Office product, such as Word 95, Word 97 and Word 2000, are also members of the family. These family members have commonalities but they are different. Sun JDK is also a product family. It has different versions from JDK 1.0 to JDK 1.4 and various versions for different platforms: Solaris, Windows and Linux. All these versions of JDK can be considered a program family.

Today we have another popular technique termed “product line”, which has a similar meaning to “program family”. A product line is a group of products that share a set of commonalities that meet the requirements of the market [2]. The objective of the product line approach is to decrease the cost of the products thus increase the competence of the product line provider. How to develop product lines (product families) is an active research topic nowadays. CMU/SEI sponsored the First Software Product Line Conference (SPLC1) in 2000, with 185 participants and tens of technical reports and papers [3]. SEI has developed a Framework of Software Product Line Practice [4]. In this framework, the key aspects of a product line are identified. Some practical techniques to develop a scalable product line are recommended.

The Family-oriented, Abstraction, Specification and Translation (FAST) process [5] is a software product line technique that concerns how to define and construct the environments for building family members. This process has been practiced in Lucent Technologies.

Besides product lines, there are many other fields, such as Domain Engineering (DE), Component-Based Software Engineering (CBSE), etc., derived from Parnas’ Program Family Model [6]. DE is a process to analyze, design and implement reusable components that can be efficiently used in a certain application domain [7].

Parnas also stressed the importance of information hiding. It is to make the functional specifications of the behavior of modules externally visible, while hiding the implementation details. Today the importance and advantages of information hiding are recognized. It has been developed as one of the key design principles in Object-Oriented programming languages.

2 First Related Paper

This section presents Weiss’ work [8] that extends Parnas’ Program Family Model [1].

Parnas’ paper tells us the definition of families and methods to design and develop families. In his paper, Weiss extended Parnas work and presented a process named Commonality Analysis to identify the commonalities and variabilities among family members.

The Commonality Analysis is an early part of the Family-oriented Abstraction, Specification and Translation (FAST) [5] process, which has been practiced in Lucent Technologies. The Commonality Analysis is an analytic process done before designing a family. The objective of this process is to identify both the commonalities and the variabilities among the potential family members in a systematic way, thus to decrease the cost to maintain and expand the family members in the future.

Usually the Commonality Analysis process is to hold a series of meetings. These meetings must include a moderator, who is an expert in the FAST process, a recorder, who will record the meeting details, and several participants, who are experts in family analysis. From this process they intend to obtain the Commonality Analysis Document, which includes three central parts: terminologies, commonalities and variabilities. This document will be about 25-50 pages, excluding appendices.

The Commonality Analysis process includes five stages: prepare, plan, analyze, quantify and review. During the prepare stage, the moderator assures the necessary preconditions for continuing the process. In the plan stage, the moderator and family analysis experts discuss the objectives, steps and expected results of the process until reaching an agreement. The analyze stage is critical. In this stage, the moderator and family analysis experts will analyze the terminologies, commonalities and variabilities of the family and these three parts can be done in parallel. In the quantify stage, the parameters of variabilities are quantified. After these four stages, the Commonality Analysis Document should have been generated. In the review stage, the Commonality Analysis Document will be reviewed by experts outside the analysis meetings.

Normally the Commonality Analysis is related to the requirement analysis stage of software development process, but sometimes it can also produce ideas about the design and implementation stages.

The whole effort for the Commonality Analysis lasts approximately 24 staff weeks. Although the effort is high, it is worthwhile because it decreases the effort needed to design and develop family members in the future, and increases the competence of companies.

In a word, Weiss’ paper extends Parnas’ family model by presenting the Commonality Analysis to identify the terminologies, commonalities and variabilities of a family. The result of its practice in Lucent Technologies shows that this technique is applicable and useful.
3 Second Related Paper

This section describes Campbell’ work [9] that extends Parnas’ Program Family Model [1].

Campbell’s paper presents us a concept “adaptable components” to model component families. An adaptable component is a special reusable component that represents a family of similar components. It achieves adaptability by using a group of parameters, therefore we can derive a series of instances from one adaptable component, and one adaptable component virtually represents numerous potential component members.

There are two reuse models. One is the usual reuse model, i.e. reuse occurs by selecting one appropriate component from a library of components, customizing and then constructing a new component. The other is adaptable component based reuse model, i.e. we can get a family of components by varying the parameters of an adaptable component. Compared with the usual reuse model, it is less time-consuming, less space-consuming and more efficient.

To develop an adaptable component is more costly than constructing one component instance, but the benefit can be seen in the future when additional component family members are to be developed.

This technique has been implemented in the software called Metaprogramming Text Processor (MTP), which has been developed to define and instantiate adaptable components in Prosperity Heights Software [10].

4 Third Related Paper

This section describes Geppert and Roessler’s work [11] that extends Parnas’ Program Family Model [1].

In [11], Geppert and Roessler stated that starting a product line process needs more initial investment than starting a simple project without using product line approach because this approach needs more effort on market investigation, planning, training, etc. Hence the product line approach is not always beneficial and we need a theory to justify the use of the product line approach. Geppert and Roessler addressed this issue in their paper.

When a software company is conducting a product line project, the main procedure is this: first, the software company invests money to start up the product line, and then expects the net return in the future. When a number of family members are developed, the product line approach begins to pay back. The point at which the product line approach begins to pay back is called the payback point. The position of this point depends on the characteristics of the project. The more the products are produced each year, the nearer the payback point is. In economics, this procedure can be depicted by the Discounted-Cash-Flow (DCF) Model. In this model, the value of an investment is a function of three factors: cash, time and risk.

The DCF Model fits very well in some cases. One example is the cell phone product line, where many models of new cell phones are produced every year. If we regard each model as a member in the product family, the payback point is expected in the near future. In this case, we can accurately predict the position of the payback point, and the managers have more confidence in the use of the product line approach.

But the DCF Model does not fit well in some other situations. When producing an evolutionary product, such as PBX (a private telephone switchboard), which evolves gradually over a long period of time and results in several releases instead of new models. If we consider the various releases of this product a family, the family members will spread out over a long period of time, therefore the payback point could be located far in the future. In this case there are many uncertainties, e.g. we cannot predict what kind of product features will be needed in the future. As a result, there will be more risk to develop this kind of product by using the product line approach.

If we use the DCF Model to depict the PBX development procedure, this kind of product will not look like an appropriate candidate of the product line approach. If we change our perspective and regard a possible evolutionary path as a product line, the conclusion will be different. For example, in Figure 1, at first we are at point 0, and predict that there will be four possibilities in release 2. As a result, when designing release 1, we make it flexible enough to support the four possibilities of release 2. By doing this, we need more effort to design release 1, but we make it easier to shift from release 1 to release 2. We do not really develop all the family members, but create the options or flexibility. In this case, the DCF Model does not fit very well because it does not take the option into consideration. Hence other economic models that take option into account are needed to justify the use of the product line approach in an evolutionary product.

Today another widely accepted economic model is the Black-Scholes Option-Pricing (BSOP) Model. This model takes option into account and is used in stock markets. Because the stock price is very volatile, two investors may completely disagree with each other on either buying a share of stock or not, but both of them will agree that the right (option) to buy a stock is valuable. This BSOP Model can accurately value the options in stock markets.
Geppert and Roessler stated that the BSOP Model can be applied to product lines approach, and it will be more appropriate than the DCF Model. We can easily map the characteristics of the product line approach into the variables of the BSOP Model.

The authors also indicated that although the BSOP Model is better than the DCF Model, it is still not the best model for product lines because it is designed for the stock markets. Therefore further research on finding better models is needed.

5 Indirectly Related Paper

This section presents Schmid’s work [12] that extends Parnas’ work [1] on Program Families, but it does not cite his original work.

Schmid stated that the product line approach is a method to produce product families in a more efficient way. However, it is not always profitable to develop the whole project with the product line approach because at the start point the product line approach needs more initial investment than traditional approaches. Some parts of the domain are directly related to software reuse while others are not, therefore appropriate scoping is important. If the scope is too large, it will add cost and complexity to the project, and as a result, the project will fail. If the scope is too small, then it cannot reflect the benefit of using the product line approach.

Later the author proposed the following six requirements of an applicable scoping method.

1. It must provide detailed economic information for the proposed scope;
2. It should address the risk of performing product line with a certain scope;
3. The scoping activity should consume only a small amount of resource;
4. It should be integrated with the overall product line development approach;
5. It should provide the detailed guidance, so that people other than developers of the method can perform it;
6. It needs to be tailorable.

Schmid argued that up to now there is no scoping method that fulfills all these requirements. Then he proposed a product line scoping solution. This solution includes three components: product line mapping, domain based scoping and feature based scoping. The product line mapping component provides the description of the product line, including the functionalities of the system that are parts of the product line. This component also identifies and provides the descriptions of domains. The domain based scoping component involves evaluating the domains identified by the product line mapping component. The feature based scoping component uses further refined and more organization-specific criteria to evaluate expected benefits of reusing each feature.

The product line approach is a practical way to apply program families, so it is derived from the original idea of Parnas’ Program Families. The objective of Schmid’s work is to propose a scoping method for the product line approach, therefore his work is an extension of Program Families. Schmid should have cited Parnas’ paper [1].

References


