TOWARDS STRATEGIC DESIGN REUSE BY LEVERAGING COMMONALITY AND MANAGING VARIABILITY

Shihong Huang  
Dept. of Computer Science & Engineering  
Florida Atlantic University  
Boca Raton, FL 33431  
+ 1 561-297-1275  
shihong@cse.fau.edu

Ravi Shankar  
Dept. of Computer Science & Engineering  
Florida Atlantic University  
Boca Raton, FL 33431  
+ 1 561-297-3470  
ravi@cse.fau.edu

Jan Mangs  
Dept. of Computer Science & Engineering  
Florida Atlantic University  
Boca Raton, FL 33431  
+ 1 561-929-4894  
jmangs@fau.edu

Abstract – Increasing software productivity and shortening the software development cycle have become critical to the success of an organization. Some commonly used strategies to shorten the software development cycle while simultaneously increasing programmer productivity and product quality include improving software processes, deploying new technologies, and reusing software artifacts. This paper presents a ongoing work that focuses on strategic design reuse by leveraging commonality and managing variability in design artifacts. It utilizes the methodologies of Model Driven Engineering (MDE) and Software Product Lines (SPL). This work is part of a larger effort focused on migrating towards model-driven engineering approach for effective enterprise software development.

Keywords: model-driven engineering, software product lines, reuse, design, UML

1. INTRODUCTION

Software has become ubiquitous in almost every sector of life. Some of the universal business goals include producing high quality software, quick time to market, maintaining low cost, and satisfying customers’ needs.

Increasing software productivity and shortening the software development cycle have become critical to the success of an organization. This paper proposes a methodology to support this goal through the strategic reuse of design artifacts.

In particular, we propose to develop a methodology for identifying commonality and variability in core design assets as represented by UML diagrams. We will also develop a series of tools that realize the methodology (implemented as a series of Eclipse plug-ins). This work is part of a larger effort focused on migrating towards a model-driven engineering approach for effective enterprise software development. Precise models enable automation and productivity.

Some of the commonly used strategies to shorten the software development cycle while simultaneously increasing programming productivity and product quality include improving software process, deploying new technologies, and reusing software artifacts. These three aspects are briefly discussed in the following sections.

1.1 Improving Software Process

Process improvement is a series of actions taken to identify, analyze, and improve existing process within an organization to meet new goals and objectives. Process improvement has been shown to be an effective way to lower costs, improve productivity and quality, and deliver products and services on time. Process improvement has proven to be a successful method of achieving
some of these goals. The benefits of adopting a process improvement program such as the SW-CMM are well documented. However, there is a limit as to how far one can go with changing the management process without changing the engineering process.

1.2 Deploying New Technologies

New technologies, such as automation, also have an important role to play in achieving some of the goals listed above. Codification of best practice in a particular application domain in a tool is a tried-and-true technique for reducing the complexity of the problem space for less-experienced engineers. A difficulty with this approach is the cost needed to train the engineers to use the new tools in an effective manner.

As illustrated in [6], software component reuse was the focus on considerable attention in the 1990s. Unfortunately, results from this strategy fell short of expectations and did not achieve significant ROI and business value. This was due in part to the nature of code (component) reuse, which is low-level, fine-grained, and opportunistic. There are inherent limitations to this type of reuse. However, reuse as an improvement strategy is still a promising avenue to explore – just need to be at a different (higher) level of abstraction: design.

1.3 Reusing Software Artifacts

Low-level components can be made reusable, but their reuse potential is closely linked to the implementation domain. True reuse value is better achieved when the artifact being reused is coarser-grained (e.g., frameworks) than relatively simple components that often lack the context provided by an architecture. Furthermore, opportunistic reuse relies on ad-hoc scavenging from a repository. Greater benefit can be achieved through pre-planned strategic reuse – something that can be realized through the use of software product lines.

2. SOFTWARE PRODUCT LINES AND MODEL DRIVEN ENGINEERING

2.1 Software Product Lines

A software product line (SPL) 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 that are developed from a common set of core assets in a prescribed way” [6]. Software product lines can help facilitate high-level, strategic design reuse by leveraging commonality and managing variability across multiple applications with a common architecture. In this way, product lines are a direct response to the perceived shortcomings of lower-level reuse methods that were described in the previous section.

SPLs have already proven themselves successful in several applications domains. For example, Cummins Inc. produces diesel engine control systems. They have over 20 product groups with more than 1,000 separate engine applications. By adopting a product-line approach, product cycle time was slashed from 250 person-months to a few person-months, and build and integration time was reduced from one year to one week [2]. They found that their quality goals were exceeded and that their customer satisfactions was higher than before.

In the Command and Control (C2) domain, Raytheon developed a ground-based spacecraft command and control system called the Control Channel Toolkit using a product-line approach [2]. They found that quality increased by 10x, incremental build time was reduced from months to weeks, software productivity increased by 7x, development time and cost decreased by 50%, and product risk was significantly decreased.

One application area that has enjoyed considerable success through software product lines is mobile phones. Indeed, mobile phones are an ideal platform for software product lines, since there is a high degree of commonality across all models, yet there is also a significant amount of variability in terms of the user features and the implementation characteristics. For example, different phones will have a varying number of keys, with varying display sizes, offering a varying sets of features. There is a need to run multiple operating systems (e.g., Symbian, Windows Mobile, Linux), support multiple protocols (e.g., CDMA, GSM), and support multiple national languages (e.g., English, Chinese, Spanish) [4].

In one much-publicized case study [9], Nokia documented how they adopted a product-line approach to Web browsers for their cell phones. Although there were costs associated with the introduction of product lines into their processes, they found the benefits outweighed the drawbacks by a measurable amount. One of the keys to their
success was retroactively identifying artifacts that were common across multiple products, and clearly delineating these common features from release-specific variabilities.

2.2 Model-Driven Engineering

Software Product Lines approaches directly support Model Driven Engineering, which is one of the primary philosophies to shorten software life cycle and in the meantime increase software quality. In Alan Brown’s article [14] he shows the modeling spectrum from code-only to model-only approach in software development in the context of MDE. Model-driven engineering has become one of the best approaches for effective enterprise software development. Precise models enable automation and productivity.

3. COMMONALITY AND VARIABILITY

At the heart of software product lines are the notions of commonality and variability. Artifacts that share a sufficient number of common features are candidates to become part of the “core assets” of the product line. Those artifacts that fall outside of this commonality threshold are not part of the core assets, but instead are specific to a single product due to variances in key features.

The benefit of identifying design commonality and variability is to support multiple target platforms. Platform independent models (PIM) (commonality) capture the functionality of a system but don’t depend on a specific implementation platform, so it is easier to switch between platforms. Common functionality is modeled and transformed to be used to generate implementations for the different platform (variability), therefore, reducing the cost of supporting multiple platforms and ensuring consistency across them.

In order to determine whether or not an artifact is a candidate for promotion to a core assets, one must first determine how to measure similarity between the artifacts. One must also set a threshold for this measure, to distinguish between common and variable assets.

For design artifacts, such as UML diagrams, similarity measures can take into consideration a wide variety of characteristics. These include classic design quality indicators such as coupling and cohesion, graphical style and spatial layout information, graph-theoretic measures (e.g., connectivity), application- and domain-specific design patterns, and so on.

4. RESEARCH METHODOLOGY

4.1 Goal

The primary goal of this research is to develop a method for identifying commonality and variability in core design assets – leveraging commonality and managing variability (LCMV).

Towards this end, as a secondary goal we will develop distance measures that capture the relative “sameness” of a collection of high-level software designs.

These measures will be used to identify common features that cut across multiple designs, and isolate variabilities that are specific to unique designs. The measures will be highly tailorable by the end-user, so that they can be applied to specific application areas with more refinement.

4.2 Methodology

In order to make the problem more tractable, we will assume that design artifacts are represented as UML diagrams. For legacy systems where such diagrams are not available, a pre-processing stage will be needed to recreate such designs from source code and other artifacts [11]. The UML representation is needed because it offers a standardized representation of design that is understandable by software engineers yet is also machine processable.

The methodology will be based on three different analysis techniques.

- The first is graph-based. This approach does not incorporate domain knowledge, but does leverage the extensive body of research from graph theory and tool support for graph matching.
- The second approach is text-based. It analyzes the XMI representation of the UML design artifacts, leveraging previous work in areas such as pattern recognition, data mining, and machine learning.
- The third approach incorporates spatial layout information and stylistic guidelines in the models that capture design intent.

Using UML also builds upon our existing strengths in the area. For example, we have been working for some time on developing style guidelines for UML diagrams in the context of graphical documentation [12]. More recently, we began
work on a project to develop an assessment instrument to evaluate the fidelity of UML diagram tools in support of design reuse [8].

In developing the design distance measures, we will leverage previous work in related areas wherever appropriate. Such areas include cluster analysis [3][13], pattern recognition, data mining, machine learning, reverse engineering [10], and software clustering and decomposition via coupling and cohesion metrics [1].

The tools that implement the design reuse methodology will be implemented as a series of Eclipse plug-ins. They will be end-user programmable so that they afford extensive customization and guidance by the software engineer. For example, the granularity of the matching between design artifacts will be tunable, from very rough matches based solely on graph connectivity, to more exact matches that incorporate characteristics of the UML diagrams (e.g., relationships types, methods, naming conventions). The tools will first be implemented to support UML use case and class diagrams.

5. SUMMARY

Model driven engineering (MDE) is a style of software development where the primary software artifacts are models from which code and other artifacts are generated. One of the primary goals of One Pass to Production (OPP) project is to shorten software life cycle. Precise models enable automation and productivity. Software Product Line (SPL) approach directly supports MDE by identifying the core assets among family products, rather than focusing a single product, so the production of new products are a matter of days or weeks. LCMV takes the first step of identifying design commonality and variability to support multiple target platforms, with the final goal to migrating towards a model-driven engineering approach for effective enterprise software development.

REFERENCES


