Support of Design Reuse by Software Product Lines: Leveraging Commonality and Managing Variability

Dr. Shihong Huang

Computer Science & Engineering
Florida Atlantic University

shihong@cse.fau.edu

Abstract

One of the goals of the Motorola/FAU "One Pass to Production" (OPP) project is to drastically shorten the software development cycle. This research supports this goal through the reuse of design artifacts by software product lines. In particular, we propose to develop a method for identifying commonality and variability in core design assets (as represented by UML diagrams).

Keywords: software product lines, reuse, design, UML

1. Motivation

Some of the common strategies to shorten the software development cycle while simultaneously increasing programming productivity and product quality include:

- Process improvement
- New technology
- Software reuse

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.

Automation also has 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-experiences 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 Figure 1, 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 at a different (higher) level of abstraction: design.

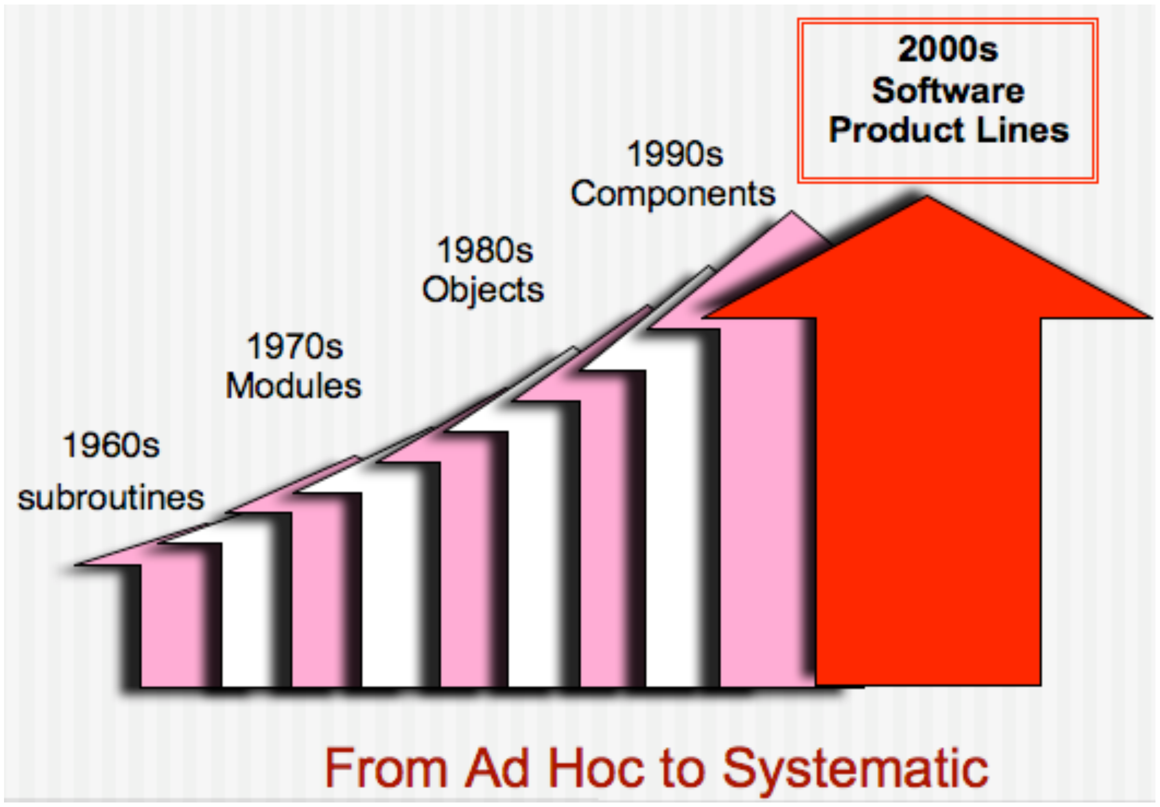

Figure 1: Software Reuse History (from [6])

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

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]. As illustrated in Figure 2, 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 quality 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.

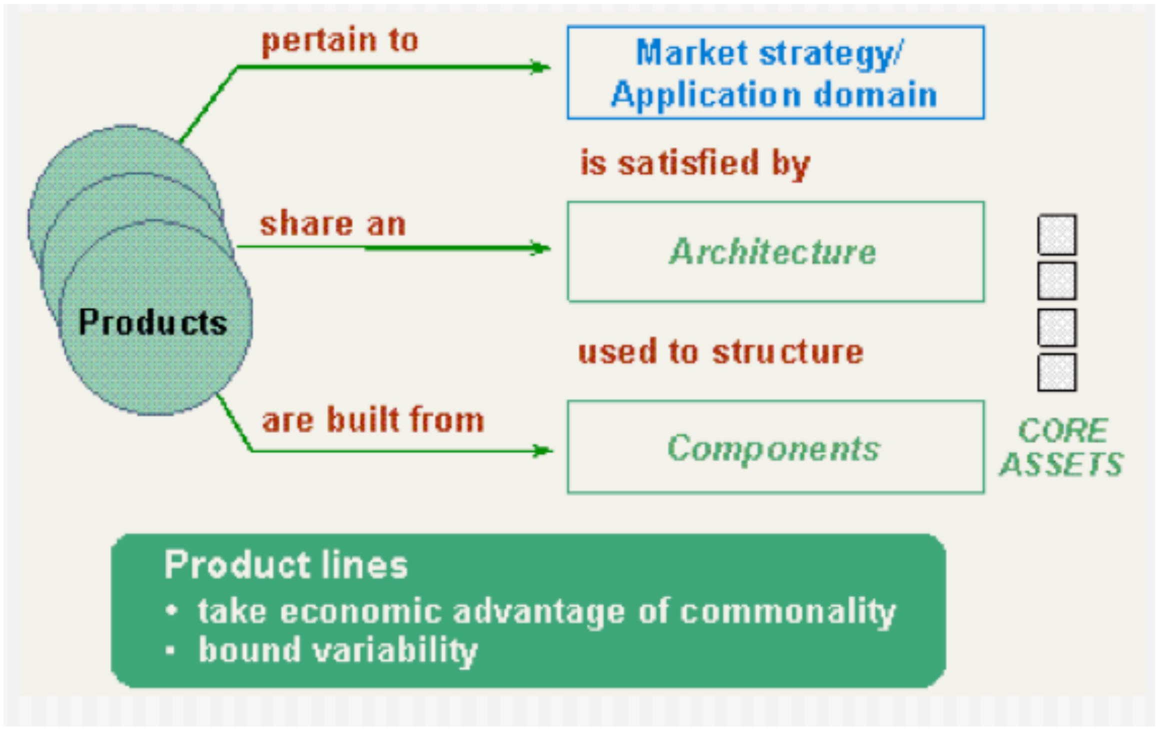

Figure 2: Software Product Lines (from [6])

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.

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.

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.

3.1 Goals

The primary goal of this research is to develop a method for identifying commonality and variability in core design assets.

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.

3.2 Method

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.

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].

3.3 Schedule

This project is planned for a one-year duration. The project will begin on August 1, 2006 and end July 31, 2007. It is hoped that when the project nears completion, discussions can begin for anticipated follow-on work.

Project Phase	**Activity**	**Dates**
Start	Project begins	August 1, 2006
Phase I	Survey of related work	August 1 – October 31, 2006
Phase II	Development of C&V measures for UML diagrams	November 1, 2006 - March 31, 2007
Phase III	Assessment of measures for model problem	April 1 - June 30, 2007
End	Wrap-up; Final report	July 1-31, 2007

References

[1] Anquetil N. and Lethbridge T. “Experiments with Clustering as a Software Remodularization Method,” 235-255. In *Proceedings of the IEEE 6th Working Conference on Reverse Engineering* (WCRE’99). IEEE Computer Society Press. October 1999.

[2] Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*. Addison Wesley Professional, 2001

[3] Everitt, B. *Cluster Analysis*. Heineman Educational Books, London, 1974.

[4] Heie, A. “Global Software Product Lines and Infinite Diversity.” In *Proceedings of the 2nd Software Product Line Conference*. August 19-22, 2002; San Diego, CA.

[5] http://www.sei.cmu.edu/architecture

[6] http://www.sei.cmu.edu/productlines

[7] http://www.softwareproductlines.com

[8] Huang, S. “UMLIO: Development of an Assessment Instrument to Evaluate the Fidelity of UML Diagram Tools in Support of Design Reuse.” Research project, Florida Atlantic University, June 2006.

[9] Jaaksi, A. “Developing Mobile Browsers in a Product Line.” *IEEE Software*, pp. 73-80. July/August, 2002.

[10] Müller, H. A; Orgun, M. A; and Tilley, S.: “A Reverse-Engineering Approach to Subsystem Structure Identification.” *Journal of Software Maintenance: Research and Practice*, 5:181-204, 1993.

[11] Ning, J.; Engbets, A.; and Kozaczynski, W. “Recovering Reusable Components from Legacy Systems.” *Proceedings of the Working Conference on Reverse Engineering*, pp. 64-72. IEEE Computer Society Press, 1993

[12] Tilley, S.; Murphy, S.; and Huang, S. “5th International Workshop on Graphical Documentation: Determining the Barriers to Adoption of UML Diagrams” (GDOC 5: Sept. 21, 2005; Coventry, UK). Held in conjunction with *The 23rd International Conference on Design of Communication* (SIGDOC 2005: Sept. 21-23, 2005; Coventry, UK).

[13] Wiggerts T. “Using Clustering Algorithms in Legacy Systems Remodularization,” pp 33-43. In *Proceedings of the IEEE 4th Working Conference on Reverse Engineering* (WCRE’97). IEEE Computer Society Press. October 1997.