Eli Whitney revolutionized the manufacturing of rifles using interchangeable parts. Henry Ford did the same for automobiles, integrating the idea of interchangeable parts and an assembly line. A growing number of software development organizations are adopting approaches that emphasize proactive reuse, interchangeable components, and multiproduct planning cycles to construct high-quality products faster and cheaper. Standard methods, referred to as software product line or software family practices, have developed around these approaches. This special issue focuses on the technical, managerial, and organizational activities related to introducing these practices.
Introducing product line practices into an organization typically involves many levels of the enterprise. Organizations have employed two main types of strategies when introducing software product lines: heavyweight and lightweight. With heavyweight strategies, a product line’s initial product costs significantly more than an initial product in single-system development (see Figure 1). However, after approximately three products, the product line has lower cumulative costs. In contrast, lightweight strategies require minimal upfront investment. The initial cost falls somewhere between the single-product cost and heavyweight cost (see Figure 1). The tradeoff for lower upfront costs is that it takes longer to reduce cumulative costs.

Early adopters of these two approaches have been rewarded with significant improvement in productivity, time to market, and product costs. Companies such as Cummins, which took a heavyweight approach, and Nokia, with its lightweight approach, have reported successful experiences.1

The keys to success

Three ideas are common to most successful product line efforts: exploring commonality among products to proactively reuse software artifacts, encouraging architecture-centric development, and having a two-tiered organizational structure.

Exploring commonalities and variabilities

Software developers have been interested in building multiple systems with similar requirements for years. In 1976, David Parnas stated, “We consider a set of programs to constitute a family whenever it is worthwhile to study programs from the set by first studying the common properties of the set and then determining the special properties of the individual family members.”2 This definition focuses on a set of programs’ common properties, regardless of how you implement the programs. More recently, the Software Engineering Institute’s Product Line Practices initiative has defined a software product line as “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.”3 This definition focuses on the common properties of products and the common technologies used to implement programs.

A product line’s scope is specified such that the products have a high degree of commonality. A product line organization realizes economically significant reuse, which so many earlier reuse approaches failed to deliver, by exploiting this commonality to achieve economies of scope. Product lines use proactive approaches that plan the use of assets in multiple products rather than ad hoc approaches that reuse assets only if they happen to be suitable.
Besides the commonalities, products in a product line also vary in their features. To allow for controlled variation, analysts and software architects introduce so-called variation points on each level of abstraction (for example, on the level of the requirements, architecture, and runtime system). As experienced in the major European product line initiatives, Esaps (www.esi.es/esaps) and Café (www.esi.es/café), product line variability has two major dimensions: a time dimension, which deals with the evolution of software artifacts over time, and a space dimension, which deals with software’s different behaviors (for example, a software artifact used in multiple products might need to exhibit different behaviors).4

The degree to which an organization must create a comprehensive, robust set of assets before delivering its first product varies from one initiation method to another. In heavyweight initiation schemes, the organization creates or acquires assets to satisfy the specifications in the product line architecture before creating the products. In lightweight initiation schemes, the organization mines assets from existing products and products currently in production.

**Architecture-centric development**

The software architecture is key to a software product line’s success. An architecture that specifies all of the products identifies and communicates the regions of commonality and points of variation. A product team creates the product-specific architectures by specializing the product line architecture to achieve specific functional and quality requirements.

The software architecture creation process includes techniques that let architects make design choices that enhance certain qualities of the system and degrade others. The architects design the product line architecture to support the range of values for each quality attribute necessary to represent all products. Developers then construct each product by selecting reusable assets that result in the appropriate quality values.

The degree to which an organization must formally define an inclusive, complete software architecture before delivering its first product varies from one approach to another. For example, reference architectures in specific domains can provide both a starting point for the product line architecture and support for early decisions about the first product if the product line architecture is not sufficiently complete.

**Two-tiered organization**

An organization using product line practices is structured to facilitate two fundamental roles: development of reusable assets and development of products that use those assets. The first role includes all products while the second focuses on a single product. Organizations accomplish this division of responsibility in a variety of ways—some have teams dedicated to each role and others use the same people for both.

Product line practices affect many parts of an organization. Roles such as product planning or marketing will change from a single-product focus to working with a set of related products and can take advantage of the product line approach. Planning a set of products at one time as opposed to individually over time produces economies of scale.

The degree to which organizational changes must occur before delivering the first product varies. In heavyweight approaches, the organization assigns specific teams to produce assets such as the architecture and components. In lightweight approaches, the organization might create the first few products as usual and then use mining efforts to extract product line assets after the fact.

**Issue overview**

In the Point/Counterpoint department, Paul Clements of the Software Engineering Institute argues the advantages of thorough planning and design before constructing products. Charles Kreuger of BigLever Software counters this view by supporting an approach that develops reusable assets while products are being constructed.

The first two articles survey the major initiatives in developing product line practices. In “SEI’s Software Product Line Tenets,” Linda M. Northrop describes a comprehensive set of practices that have evolved from the experience of initiating product lines in a variety of industrial domains and government agencies. In “Software Product Families in Europe: The Esaps & Café Projects,” Frank van der Linden summarizes the challenges faced and lessons learned when introducing software product lines in European companies.
The next two articles address general issues related to introducing a product line. “The Economic Impact of Product Line Adoption and Evolution,” by Klaus Schmid and Martin Verlage, describes decisions that must be made when initiating a product line and presents economic arguments that help guide the decision-making process. “Feature-Oriented Product Line Engineering,” by Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe, describes the central role that product features can play in product line development. Using feature-oriented domain analysis and the feature-oriented reuse method, the authors illustrate how to use features as the guiding principle behind product development.

The final two articles report on industrial experiences gained when introducing a software product line. “Modeling and Using Product Line Variability in Automotive Systems,” by Steffen Thiel and Andreas Hein, describes the pervasiveness of variability when operating a product line. The article presents a study of the systematic planning and variability management that occurs in a product line to achieve strategic reuse. “Developing Mobile Browsers in a Product Line,” by Ari Jaaksi, provides a look inside an organization as it evolves to meet the demands for faster, cheaper development. The article provides a valuable set of lessons that the team learned during the evolution of the organization.

References

About the Authors
John D. McGregor is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests are software product lines and component-based software engineering. He received his PhD from Vanderbilt University. Contact him at the Dept. of Computer Science, Clemson University, Clemson, SC 29634; johnmc@lumsoft.com.

Linda M. Northrop is director of the Product Line Systems Program at the Software Engineering Institute. Her research interests include software architecture, software product lines, and the predictable assembly of certifiable components. She received her undergraduate degree from LeMoyne College and her graduate degrees from the State University of New York and the University of Rochester. Contact her at the Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA 15213-3890; lmn@sei.cmu.edu.

Salah Jarrad is vice president of Software and Systems at MMCD, Panasonic Wireless Design Center. Contact him at MMCD, Panasonic Wireless Design Center, 1226 NorthBrook Pkwy, Suwanee, GA 30024; sjarrad@panasonicatlanta.com.

Klaus Pohl is full professor for software systems engineering and the director of the Institute for Computer Science at the University of Essen. His research interests include scenario-centered requirements management and requirements-driven software product line development. He received a degree in computer science from FH Karlsruhe, Germany, a degree in information systems from the Uni. Konstanz, Germany, and a PhD in computer science from the Technical University of Aachen, Germany. Contact him at Software Systems Eng., Univ. of Essen, Altendorferstr. 97-101, 45117, Essen, Germany; pohl@cs.uni-essen.de.

Check it out
A Free CD-ROM with Your CG&A Subscription

This supplemental CD will contain peer-reviewed multimedia content such as 2D and 3D simulations and animations, standalone interactive tutorials, and demonstrations of application examples. The CD will not duplicate any current electronic or print content.

Subscribe today!
http://computer.org/cga