



# Successful Software Product Line Practices

**John D. McGregor**, *Clemson University*

**Dirk Muthig**, *Lufthansa Systems*

**Kentaro Yoshimura**, *Hitachi*

**Paul Jensen**, *OverWatch Textron*

**A** software product line 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 in place.<sup>1</sup> Organizations adopting product development strategies that include a software product line have achieved impressive results, reducing product cycle time and increasing productivity by an order of magnitude.

Major organizations such as Cummins, Philips HealthCare, Hewlett Packard, and others have successfully applied these techniques.

The software product line strategy is a blend of business and technical actions that lets an organization satisfy a wide range of customers, gain leverage with suppliers, meet the threats of substitute products, and deter other companies seeking to enter the market. The strategy is robust over a wide range of technologies, domains, and organizations of different structures, cultures, and goals. Service-oriented architectures, agile development methods, and open source business models have all played roles in successful product line organizations.

### Software Product Line Differences

We'll follow the path of a successful product line organization and use a brief commonality and

variability analysis (see the sidebar "A Bit of Terminology") to characterize successful product line organizations. Successful software product line organizations differ in many ways.

First, the size of the product line and its individual products can vary. With reuse percentages running above 50 percent, a product line organization will typically recoup the extra cost of making assets reusable after two or three products. In this context, even product lines of a few products can be very profitable. Some organizations build a few large products that take months or even years, while others build many small products that each take a matter of hours.

Second, the organization's structure and agility can vary. Some successful product line organizations are self-contained within a business unit while others span business units, contract with strategic partners, or form consortia out-

side the existing organizations. Even large organizations, such as Philips Healthcare and Hewlett Packard, have developed innovative approaches to integrating the efforts of various internal and external organizational units into a product line organization.

Finally, different product lines use different production methods. Some software product line organizations have successfully used traditional programming languages and largely manual development techniques, while others have automated portions of the code generation for their products using model-driven techniques. Various aspects of agile development methods have been integrated into product line practices. Successful product line organizations intentionally select techniques and tools, models, and processes that match their goals.

## Software Product Line Commonalities

Several elements are common to successful software product lines.

First, the scope of the product line is well defined, but not rigidly so. The definition of which products belong to the product line—its scope—provides the context within which many other decisions are made. For example, the decision about whether a particular module should be designed to accommodate a certain variation is made by determining whether the scope definition permits a product that would need that variation to be in the product line. It's critical to clearly define the scope of the product line, but it's equally critical to realize that the scope will change over time.

Second, the organization uses a common software product line architecture as the basis for each product. This architecture is a reference that guides production and describes those portions of products that are common and those that vary from one product instantiation to another. The architecture provides the basis for exploiting commonality and managing variation.

Third, commonality is sufficiently defined to realize the economies of scale. Commonality lowers costs and increases productivity through the repeated use of assets. The software product line strategy can significantly improve productivity if many different products can share assets. Economies of scale are realized via the same factoring of required behaviors as economies of scope.

Fourth, variation is sufficiently well managed to realize the economies of scope. Managing variation reduces the time required to meet the

## A Bit of Terminology

Although there's great diversity within the software product line community, a number of terms are in common use.

**Core asset:** This is an artifact that's designed with sufficient configurability for use in multiple products. For example, a software architecture can be designed to be the reference architecture for the product line. Each product architecture is instantiated using that architecture.

**Scope:** The scope of the software product line is determined by the capabilities and qualities available for a product definition. Including too much in the product line's scope requires assets that could only be used in a few products, while narrowing the scope too much results in a product line that might not attract sufficient customers.

**Variation point:** A variation point is a design decision that identifies where products can vary from one another. The product specifier chooses among multiple capabilities to determine a product's definition.

**Variant:** Each choice that can be made at a variation point is a variant.

**Commonality and variability analysis:** This analysis examines proposed capabilities and determines which will be shared by all products and which will only be included in some of the products.

needs of a diverse audience by using preexisting, configurable assets. Product line requirements must be factored sufficiently so Core asset developers can understand the variety of behaviors, which must be supported by each asset, and can translate that variety into appropriate variation mechanisms in the Core assets.

Finally, the organization is structured and operated to facilitate building reusable assets and building products using those assets. Each role brings a unique perspective to bear on the organization's activities. Core asset development requires a broad perspective that encompasses issues across the entire scope of the product line. Product building requires a focused perspective that gives highest priority to the activities needed to construct the product. The activities of these two roles are coordinated by a management team that views the organization's capability to produce products as its most important asset.

## The Literature

Since the previous *IEEE Software* special issue on software product lines in 2002, successes have multiplied, the community has broadened, and the experience base has diversified. There were numerous success stories in 2002, but they tended to originate from the research departments of large, technical companies. For example, Steffen Thiel and Andreas Hein illustrated the use of variability in automotive systems.<sup>2</sup> Frank van der Linden provided a view of a cooperative research program among several companies and research

**Successes  
have multiplied,  
the community  
has broadened,  
and the  
experience  
base has  
diversified.**

universities.<sup>3</sup> Ari Jaaksi described Nokia's research into using a product line approach to build browsers for cellular telephones.<sup>4</sup> Issues related to introducing product line concepts into organizations were major concerns. Linda Northrop's article "SEI's Software Product Line Tenets" and Klaus Schmid and Martin Verlage's article "The Economic Impact of Product Line Adoption and Evolution" provided practical advice based on experience.<sup>5-6</sup> The "Point/Counterpoint" discussion by Paul Clements and Charles Krueger considered a fundamental strategic issue: whether to build assets before or as products are built.<sup>7-8</sup> Kyo C. Kang, Jaejoon Lee, and Patrick Donohue illustrated the emergence of specialized technical approaches by describing the feature modeling method for specifying products.<sup>9</sup> Since that special issue there's been a special issue of the *Communications of the ACM*, and the proceedings of the annual Software Product Line Conference continues to provide an important venue for software product line research.

In 2010, the software product line context has matured and broadened considerably. There are success stories about software product lines in the production departments of small as well as large companies, a variety of business models, product lines of product lines, and complex ecosystems of interdependent suppliers that support the product line.

The Software Product Line Conference (SPLC) has become an annual conference run by an international steering committee and rotating among diverse parts of the world. There are a growing number of specialized product line venues beyond SPLC including the Practical Product Lines conference and a research workshop on software product lines at the International Conference on Software Engineering 2010.

Product line practices have matured to the point where standards can be identified and institutionalized. In late 2009 the Object Management Group approved an RFP for a standard variability modeling language. Standards for tools and methods for software product lines have been proposed to ISO/IEC JTC1 and are under discussion.

Other communities have expressed interest in software product line engineering. *The Journal of Systems and Software* included a special issue on integrating agile and product lines practices in 2008. In November 2009, the Product Managers View, an online community of product managers, produced a series of webinars introducing software product lines to their community.

## **This Special Issue**

The articles in this special issue address many of the aspects of software product line development we've identified.

In "Clearing the Way for Software Product Line Success," Lawrence Jones and Linda Northrop draw on 15 years of software product line experience at the Software Engineering Institute ranging from companies of less than 50 people to global corporations, including numerous instances of applying diagnostic instruments. They identify two key problems that many organizations have when initiating their first software product line.

Experiences with the first generation of successful product lines have led to proposals for new approaches. Jan Bosch describes a compositional approach to product line development that addresses perceived problems with the evolution of assets over the life time of the product line.

Software product line development is architecture-centric. Jaejoon Lee and Gerald Kotonya describe the influence of service-oriented architectures on software product line development.

Isabel John presents the commonality and variability extraction (CAVE) technique. CAVE aims to reduce the need for domain experts to obtain the information needed for scoping during product line initiation by using existing product documentation to create initial models.

Kannan Mohan, Balasubramaniam Ramesh, and Vijayan Sugumaran analyze factors that affect the integration of product line and agile development methods. They use experience from complex adaptive systems to describe the integration and provide an additional analysis of a previously published case study of successful integration.

## **Successful Software Product Line Organizations**

Here, we present five vignettes of successful software product line practices. Each has a different story to tell about the context in which the product lines were implemented and what made them successful.

### **Cummins**

In 1994, Cummins adopted a software product line approach, creating the Core product line from existing software assets. The initial product line was a base set of components shared via source code with each product. Application teams then tailored this base software to meet their specific needs. While this approach success-

fully delivered new products rapidly, concerns arose around the maintenance expense and sustainability of the architecture as the code bases diverged for products over time.

In response to those concerns, Cummins designed a second-generation product line, Core 2, to support a wide range of diesel and alternative-fueled engines for a wide range of markets and a broad domain of customer features, engine configurations, and emissions levels. Specialized tools were designed to manage the product line, apply its assets to new products, and ensure the code base is maintained as a common asset throughout the product life cycle. First introduced on a product in 2004, the Core 2 product line has reduced product cost through a strong use of common assets. The broad existing base of common assets also means that time to market for new product is significantly reduced because much of a new product's software is already written. Common assets also ensure that Cummins can maintain a common feature set, as well as a common "look and feel" across its product line.

Overall, the conversion to Core 2 has been a success for Cummins. Compared to Core, Core 2 supports more than three times as many products with 25 percent fewer software developers per product. The product line and its toolsets continue to evolve and add support for new products and markets while supporting systems of increasing complexity. For further reading, see the work of Scott Decker and Jim Dager.<sup>10</sup>

### **Hewlett-Packard**

Hewlett-Packard's consumer and small and medium business inkjet printers and all-in-ones have been using the Owen software product line for more than ten years. Owen, which is an embedded software, had its beginning in 1997 when the San Diego and Vancouver divisions decided to cooperate on a common architecture for print-engine firmware developed in Vancouver and leveraged by San Diego. Owen has grown from the first year supporting two products to supporting 20–25 new products each year. Peter Toft and his colleagues first described how Owen started out as a firmware cooperative.<sup>11</sup> A cooperative is "an autonomous collection of projects, voluntarily united to meet their common needs and aspirations."<sup>12</sup> Originally, projects would choose to join Owen and choose which code to accept; they were encouraged to make code changes so other members could benefit from them.<sup>11</sup> Over the years, changes in business objectives have resulted in the expecta-

tion that all inkjet products use Owen, that all products share a common code base, and code changes are always done in the interest of Owen (and hence the overall business) and not for a specific product.

The Owen architecture is a component-based architecture. Each component can require or provide services (via interfaces) to the rest of the system. Components in related functional areas are grouped into larger subsystems. One of Owen's key subsystems has been the "framework," which provides services like persistent storage, resource and job management, and system startup/shutdown and power orchestration.

Over the years, many subsystems have emerged that tend to fall into two categories—an end-user function (print, scan, fax, copy, photo, and so on) or an infrastructure subsystem that serves many others (connectivity, security, and so on). The subsystems focused on end-user functions tend to evolve more (in response to evolving product or customer needs) whereas infrastructure subsystems tend to be more stable.

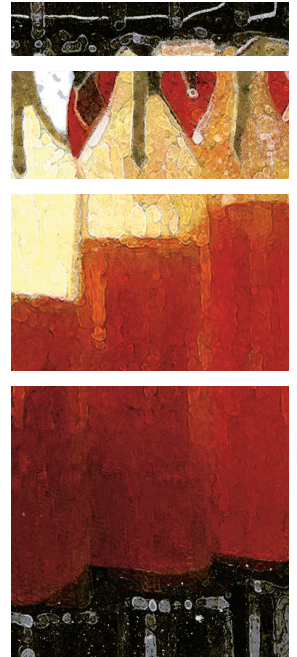
A key factor that's made Owen successful has been a continued focus on decoupling. This is partially enforced in the Owen architecture by not allowing cyclic dependencies between components. At a subsystem level, Owen seeks to have clean, well-defined subsystem interfaces and architectural rules around which subsystems can build on top of other subsystems.

In the coming years, Owen will stretch further in terms of the number and types of products it supports as well as their complexity, and also in terms of development agility and higher development efficiencies. Owen will accomplish this by adopting and refining more agile practices and evolving the architecture as required.

### **Hitachi**

Hitachi is a leading global electronics conglomerate that offers a wide range of products including medical systems, automotive components, and consumer products. The size of software development organizations range from a few developers to hundreds of developers depending on product domains. Hitachi noticed that there's no one-size-fits-all approach. One of the most important factors is the scalability of the software product line engineering (SPLE) approach.

In the case of small to medium-scale organizations, Hitachi often faces adoption challenges. For example, we applied SPLE to a medical device development department.<sup>13</sup> The business unit found it difficult for the department to set up a Core asset



## About the Authors



**John D. McGregor** is an associate professor of computer science at Clemson University, a partner in Luminary Software, and a visiting scientist at the Software Engineering Institute. His research interests include software product lines, systems engineering, and software architecture. McGregor has a PhD in mathematics from Vanderbilt University. Contact him at [johnmc@cs.clemson.edu](mailto:johnmc@cs.clemson.edu).

**Kentaro Yoshimura** is a researcher at Hitachi. His research interests include software product line engineering, software repository mining, and software visualization. Yoshimura has a PhD in information science and technology from Osaka University. Contact him at [kentaro.yoshimura.jr@hitachi.com](mailto:kentaro.yoshimura.jr@hitachi.com).



**Dirk Muthig** is chief platform architect in the domain of passenger services at Lufthansa Systems. He previously worked at the Fraunhofer Institute for Experimental Software Engineering, focusing on product line engineering (the PuLSE method). Muthig has a PhD in computer science from the Technical University of Kaiserslautern. Contact him at [dmuthig@lhsystems.com](mailto:dmuthig@lhsystems.com).

**Paul Jensen** is the chief architect at Overwatch Systems, an operating unit of the Textron Corporation. His research interests include software product lines and cloud computing in government defense environments. Jensen has a PhD in physics from the University of Texas at Austin. Contact him at [pjensen@overwatch.textron.com](mailto:pjensen@overwatch.textron.com).



development team to adopt SPLE. To do this, the business unit organized a cross-product development-team organization as a champion team. The team is in charge of decision making on Core asset issues. Each product development team member does actual development tasks. The overhead of coordinating with the product development team is affordable considering the customer-oriented advantages of this method.

For a large-scale organization that's already produced a large number of products, Hitachi developed a method to evolve its Core asset based on the product release history.<sup>14</sup> The product configuration transactions are analyzed statistically to extract configuration constraints such as co-change patterns. The constraints are imported into the Core asset and applied for future product configurations.

Sharing the SPLE experiences across business units is crucial for success. Hitachi has set up

workshops where engineers report their experiences and researchers integrate SPLE knowledge as a Core asset.

### OverWatch

Overwatch Systems focuses on the development and fielding of multidiscipline data analysis software systems. Areas of expertise include data fusion, all-source analysis, signal intelligence acquisition and analysis, sensor network technology, and visualization. In 2003, the company adopted a software product line approach, producing the Overwatch Intelligence Center software product line over a period of several years. As of 2009, multiple members of the software product line have been fielded to combat environments including a signals intelligence collection and analysis system and two all-source intelligence analysis systems.

From 2003 to 2009, Overwatch Systems' software product line has grown from two to ten products. In that same time period, the company's revenue has grown by a factor of 3.6. Management believes that this couldn't have been achieved without the speed and reduced costs that a software product line approach enabled. This success has been keyed by the development of a flexible product line software architecture, but tempered by continuing difficulties related to delivering products to the government from a company-owned software product line. At the most fundamental level, these struggles involve a lack of complete control over the alignment of product features, delivery schedules and quality requirements with multiple, disparate government organizations.

The future of Overwatch Systems' software product line lies in embracing the concepts of composite applications and a government cloud-computing environment. In a cloud-computing environment, computing, resources, and applications are available as services through the network with new functionality emerging constantly. Composite applications let the user create applications from preexisting functions to satisfy new requirements quickly. By embracing these concepts, the Overwatch Intelligence Center software product line will move the assembly and testing of product line members from a centralized organization to the end user in the field. For further reading, see the work of Paul Jensen.<sup>15</sup>


### SystemForge

SystemsForge is a software product line for developing e-commerce, content management, and

other custom Web applications. It's been used to build over 200 Web applications over the last four years.

Initially, we developed a component-based solution with reusable components for common functionality such as shopping carts, event calendars, and content management. Over time, the number of configuration options became unmanageable, so we moved to a domain specific modeling solution with domain specific languages (DSLs) for describing controller, view, and model functionality including DSLs for describing object relationships and contextual validation rules. The problem with DSLs was that to build a comprehensive e-commerce system with 30–40 distinct business objects, 15–20 controllers and 50–60 distinct views took a day or two (which was too slow), even though 80 percent of the functionality was common among projects.

We then moved to a hybrid model using feature modeling for selecting common functionality. Instead of using the feature models to configure components, each node on the feature tree represented  $O.n$  statements in each of the DSLs and the feature model allows us to passively generate a first cut of the application described in the model, view, and controller DSLs. We then customize the DSL statements with unique requirements for a specific project and use a combination of subclassing and AOP for adding custom code while still allowing for active regeneration of code from the DSL statements. The main issue we're now focusing on is DSL evolution so we can evolve our metamodels and automatically transform existing projects as backwards compatibility isn't an option indefinitely and versioning of DSLs becomes unwieldy over time. We're developing tooling for automatically transforming DSL statements based on metamodel transformations. We're also doing research on the best approaches to handle validation of DSL statements and generate of meaningful tests for generated code. For further reading, see the work of Peter Bell.<sup>16–18</sup>

**T**he articles in this special issue, the vignettes, and recent conference papers are part of a growing body of knowledge and experience on how to successfully implement software product lines in various contexts to meet different business goals. Organizations are pushing the envelope by creating new organizational structures and introducing technologies that haven't been used in a product line context. 

## Acknowledgements

We thank Zachary Schwab and Randy Lyvers for the Cummins vignette; Jacob Refstrup, Holt Mebane, and Joe Bauman for the Hewlett-Packard vignette; Kentaro Yoshimura and Yasuaki Takebe for the Hitachi vignette; and Peter Bell for the SystemForge vignette.

## References

1. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.
2. S. Theil and A. Hein, "Modeling and Using Product Line Variability in Automotive Systems," *IEEE Software*, vol. 10, no. 4, 2002, pp. 66–72.
3. F. van der Linden, "Software Product Families in Europe: The Esaps and Café Projects," *IEEE Software*, vol. 10, no. 4, 2002, pp. 41–49.
4. A. Jaaksi, "Developing Mobile Browsers in a Product Line," *IEEE Software*, vol. 10, no. 4, 2002, pp. 73–80.
5. L. Northrop, "SEI's Software Product Line Tenets," *IEEE Software*, vol. 10, no. 4, 2002, pp. 32–40.
6. K. Schmid and M. Verlage, "The Economic Impact of Product Line Adoption and Evolution," *IEEE Software*, vol. 10, no. 4, 2002, pp. 50–57.
7. P. Clements, "Being Proactive Pays Off," *IEEE Software*, vol. 10, no. 4, 2002, pp. 28–31.
8. C. Krueger, "Eliminating the Adoption Barrier," *IEEE Software*, vol. 10, no. 4, 2002, pp. 29–31.
9. K.C. Kang, J. Lee, and P. Donohue, "Feature-Oriented Product Line Engineering," *IEEE Software*, vol. 10, no. 4, 2002, pp. 58–65.
10. S.G. Decker and J. Dager, "Software Product Lines Beyond Software Development," *Proc. 11th Int'l Software Product Line Conf. (SPLC 07)*, IEEE CS Press, 2007, pp. 275–280.
11. P. Toft, D. Coleman, and J. Ohta, "HP Product Generation Consulting, A Cooperative Model for Cross-Divisional Product Development for a Software Product Line," *Proc. 1st Software Product Lines Conference (SPLC 1)*, P. Donohue, ed., Kluwer Academic Publishers, 2000, pp. 111–132.
12. P. Toft, "Hewlett-Packard: The HP Owen Firmware Cooperative—A Software Product Line Success Story," *Software Product Lines*, 2004; [www.softwareproductlines.com/successes/hp.html](http://www.softwareproductlines.com/successes/hp.html).
13. Y. Takebe et al., "Experiences with Software Product Line Engineering in Product-Development-Oriented Organizations," *Proc. 13th Int'l Software Product Line Conf. (SPLC 2009)*, Software Eng. Inst., 2009, pp. 275–284.
14. K. Yoshimura et al., "Factor Analysis Based Approach for Detecting Product Line Variability from Change History," *Proc. 5th Working Conf. Mining Software Repositories (MSR 08)*, 2008, pp. 11–18.
15. P. Jensen, "Experiences With Software Product Line Development," *Crosstalk*, vol. 22, no. 1, 2009, pp. 11–14.
16. P. Bell, "A Practical High Volume Software Product Line," *Proc. Conf. Object-Oriented Programming Systems Languages and Applications (OOPSLA 07)*, ACM Press, 2007, pp. 994–1003.
17. P. Bell, "Automating the Transformation of Statements in Evolving Domain Specific Languages," Domain-Specific Modeling Workshop, Conf. Object-Oriented Programming Systems Languages and Applications (OOPSLA 07); [www.dsmforum.org/events/DSM07/papers/bell.pdf](http://www.dsmforum.org/events/DSM07/papers/bell.pdf)
18. P. Bell, "DSL Evolution," InfoQ; [www.infoq.com/articles/dsl-evolution](http://www.infoq.com/articles/dsl-evolution).



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.