Product Line Engineering: The State of the Practice

Andreas Birk, *sd&m*

Gerald Heller, *Hewlett-Packard*

Isabel John and Klaus Schmid, *Fraunhofer Institute for Experimental Software Engineering*

Thomas von der Maßen, *University of Aachen*

Klaus Müller, *Robert Bosch GmbH*

Software product lines can be very powerful for ensuring quality, economic efficiency, and manageability of software system families. SPLs are relevant to large industrial enterprises that want to better manage their software-intensive systems' development. They can also provide small start-ups with unique and striking business models. However, SPL technology can be challenging.

We aim to shed light on SPL practices in today's software industry. Over two and a half years, our work group, which met under the umbrella of the German Computer Society (a sister organization of the IEEE), has investigated and compared SPL practices. The work group comprises members from five organizations:

- Hewlett-Packard, a global IT vendor
- Robert Bosch GmbH, a leading supplier of automotive electronics
- Software Construction Group, University of Aachen, representing a large supplier of industrial energy solutions
- Market Maker Software, a small software developer for online stock market information management, represented by Fraunhofer IESE
- sd&m (software design & management), a renowned software house

Moreover, most work group members had experience from more than one product family or business department, giving us a rather broad overview of the state of SPL development in industrial practice. (In the rest of the article, we omit company references to preserve the anonymity of sensitive information.)

An SPL is a set of software-intensive systems sharing a common, managed set of features.[1] These features satisfy a particular market segment or mission's specific needs and are

Software product line technology can be difficult to implement and maintain. A work group comprising five organizations has investigated and compared SPL practices. The results detail the state of SPL practice.

Table 1

Factors for characterizing product line development organizations

	Organization 1	Organization 2	Organization 3	Organization 4	Organization 5
Organizational characteristics					
Employees in business unit	1,000	1,000	10	250	1,000
Number of development sites	8	3	1	3	8
Market orientation	Market segment	Customer projects	Customer projects	Customer projects and market segment	Customer projects
Hardware embedding	Software only	Embedded system	Software only	Embedded system	Software only
Development of a core platform	No	Yes	No	Yes	Yes
Product line characteristics					
Number of products	High	Very high	Medium	High	High
Performance requirements	Strict	Strict	Loose	Strict	Strict
Stability	Low	High	Medium	High	High
Architecture/implementation	Components	Framework/components	Framework	Framework	Framework/components

developed from a common set of core assets in a prescribed way. Our work group set out to better understand how software organizations can successfully set up and manage SPLs. We identified four key practice areas:

- Organization and support practices
- Practices that balance platform versus client interests
- Requirements engineering practices
- Architectural practices

For each area, the work group identified and compared the various practices their organizations employed. They also contrasted them with published reports of SPL practices.[1–3] The result, which we report here, details the state of SPL practice.

Organizational characteristics

Table 1 characterizes the five organizations and their SPL projects. In addition to these five projects, we accessed about 10 more project contexts for comparison and refined our observations and conclusions.[4] This allowed for a wide spectrum of detailed investigations.

The organizational context illustrates the various product line situations the study covered. We characterize context in terms of these factors:[5]

- *Number of employees in the business unit* defines how many employees are in the part of the organization that applies SPL software development and includes categories 1 through 10, 11 through 250, and 251 through 1,000.
- *Number of development sites* involved in SPL development includes the categories 1, 2 through 3, and 4 through 8.
- *Market orientation* defines whether the organization targets a specific market segment without a specific customer in mind or addresses individual customer projects.
- *Hardware embedding* is either an embedded system or pure software.
- *Core platform development* indicates whether reusable assets and the final product are developed in different organizational entities.

The product line characteristics describe the product lines we studied in more detail:[6]

- *Number of products* expresses the average number of different software products that are developed from the SPL during one year.
- *Performance requirements* describe whether system performance has high priority for development. They can be *strict*, if performance is expected to be a high-priority design and implementation constraint, or *loose* if not.
- *Stability* describes the degree to which we can expect the domains relevant to the product line to change in the foreseeable future.
- *Architecture* or *implementation* characterizes how organizations use the implementation approach for the product line. Possible values are *component-based ap-*

To support product line development from an organizational viewpoint, we found it key to demonstrate a convincing business case.

proach, *object-oriented framework*, *domain-specific language*, and *other*.

Both the organizational and the SPL characteristics show that the investigation covered a heterogeneous set of samples that varied considerably. This gave us a rich collection of case studies for in-depth investigation. We discussed our organizations' SPL practices in the light of SPL's perceived implementation difficulties. We held more than 10 one-day meetings over more than two years. Observing changes and improvements during this time period gave us another interesting opportunity to assess the applied SPL practices' effectiveness.

When comparing the five SPL projects' characteristics, we can see how much organizational factors and business environment shape SPL practices. For instance, one organization develops its product line concurrently at different worldwide locations. It not only develops the product line within the organization itself but also uses a couple of subcontracted R&D organizations distributed around the globe. This high distribution can aggravate communication between developers, hindering effective development of reusable components. In another organization, the organizational units for all product line development are on the same site, simplifying strong information exchange among the complementary product lines' contributors. Owing to the organization's small size, it can use a simple organizational structure: a single organization manager is responsible for the product line's domain and application engineering.

Other factors that shape an R&D organization's SPL approach are technological environment and product-related factors. They can cause issues that are different from those resulting from organizational and business-related factors. For instance, in some organizations with many products and relatively instable domains, the following challenges became increasingly apparent and motivated the need for a solid product line approach: Products started to overlap in functionality, customers who bought more than one product faced consistency and efficiency issues, and development and maintenance costs increased.

Product line engineering practices

We identified the various factors related to SPL development and used them to characterize our organizations and their product lines. This initial investigation built a broad enough common basis for comparing the different organizations and their SPL approaches. It let us evaluate product line engineering practices and identify experiences about SPL development.

Organization and support practices

Introducing product line development to an organization significantly impacts the entire organization. It can fundamentally change development practices, organizational structures, and task assignments. These changes can in turn impact team collaboration and work satisfaction. So, organizations must plan and manage them carefully, mitigating possible adverse effects.

You can organize SPL development in two ways:

- *Within product teams*: The same developers and teams who develop products on the basis of core assets are temporarily assigned to developing or evolving these core assets.
- *In a separate SPL team*: One team develops the core assets while other teams develop products. The SPL team can be permanent—being responsible for initial asset development and later evolution—or it can exist only until it develops the initial collection of core assets. Product developers would then perform further platform evolution.

When you assign SPL development as an additional task to product development teams, you risk spoiling the SPL core by focusing it too strongly on the next product to come. When you create a separate SPL team, the reusable components might not sufficiently address the current products' needs or might not be aligned with the current schedule.

In our work group's investigation, only one organization chose a permanent separation of SPL and product development. Another developed core assets as a second product development task. The other three developed their reusable assets in temporarily specialized teams, while doing further maintenance and evolution in association with product development. Organizational setup also changes as the SPL matures. When you derive the SPL from existing products, you might initially develop some components of the product line in product development teams. However, when more and more products are using these components, the teams might move to a different organization.

This distribution pattern matches our observations throughout industry. Few organizations perform fundamental restructuring into separated SPL and product organizations. Most want to avoid the risk of such deep changes, although this could limit the advantages of SPL development. One reason to avoid these changes might be to maintain proven work structures and organizational culture. In some cases, a company's business model suggests avoiding permanent restructuring—namely, when product customization is the main business and the volume of SPL maintenance doesn't justify a separate team.

Regardless of asset and product development's basic organization, you must ensure an optimal fit of the SPL core with product development needs. This requires a sophisticated balance of SPL autonomy and consideration of product concerns. You should not align the SPL core too closely with individual product needs but must ensure timely implementation of relevant product features. The required measures are equally important for every software organization that performs SPL development.

To support product line development from an organizational viewpoint, we found it key to demonstrate a convincing business case and identify an appropriate organizational structure. In particular, if an organization establishes a separate SPL team, it must internally justify the team's platform continually.

Showing a convincing business case. What are the essential launch conditions for a product line initiative? We've observed two triggers in practice: the set-up, based on the personal conviction of key personnel, or the more disciplined approach—the business case is systematically developed to introduce the technology. Although the latter is preferable, in practice we've observed that sufficient management buy-in is a key quality to making a product line successful. The latter trigger usually plays only a secondary role. So far, we have yet to see a product line development where the overall introduction occurs on the basis of a thorough cost-benefit analysis. However, in a long-running product line development—where protests against it might occur repeatedly—justifying the product line approach through cost-benefit analysis is particularly necessary.

Identifying an organizational structure. When establishing an independent domain engineering unit, you always risk that it will "develop a life of its own." Will the assets you develop still focus on future products down the line? Will they be available in time for the products? This is probably the most-often-reported problem from product line development, especially in the case of divisional work split. That's because this kind of platform team is usually quite distant from the customers, losing the required understanding of customer perspective.

Another direct consequence of creating a platform team along with several project teams is that the number of communications increases. The platform team must confer with the various project teams, a trade-off of the project teams' requirements must be made, and so on. This might also easily lead to a situation where people want to move away from a product line approach. This kind of negotiation usually starts when deciding which requirements should become core assets and which ones should be product specific (see the section on balancing platform versus client interests).

Justifying the platform team. The two problems just mentioned might lead to a situation where the platform team itself is the problem. In that case, you might perceive the team as failing to serve the customer, creating delivery problems, and generating overhead.

Depending on the overall criteria for introducing SPL development, the justification might take different forms. If cost-benefit considerations drive SPL development, the only approach is to continuously monitor the cost advantages that a platform can generate. Consideration on the customer side can also generate platform development. For example, the customer might want to have its various products based on a single platform to reduce total ownership cost (for example, through minimizing training and support costs).

While in the second case justifying SPL development is relatively easy, the first case usually calls for repeated justification. This seems straightforward if an adequate measurement program is already in place. However, appropriate measurement systems are usually not

Another direct consequence of creating a platform team along with several project teams is that the number of communications increases.

The prioritization process deals mainly with responsibility and acceptance.

available. So, most justification efforts must rely on a combination of indirect cost-benefit indicators and argumentation.

Balancing platform versus client interests

SPL platform development's goals differ from product development's goals, which client interests drive. While platform development must provide a consistently high-quality platform, product development must meet delivery dates and customer requirements. So, with every SPL development you must decide whether to integrate a given requirement into the platform or into an individual product only. The right date and order of requirements implementation is essential to SPL development's effectiveness.

Integrating requirements into the common platform. In a product line situation, many projects simultaneously depend on the core assets or the common platform. So, the sequence in which you integrate requirements into the platform becomes key. You must ensure that the required functionality that a future product will reuse is already part of the platform by the time it's needed.

When analyzing solutions to manage platform requirements, you must

1. Establish a prioritization process in the organization that everybody adheres to
2. Find prioritization criteria that, if applied, lead to optimizing the organizational benefit

As a prerequisite, you must establish a process for identifying platform requirements and for scheduling results for customer projects. The prioritization process deals mainly with responsibility and acceptance. Either a single key manager is responsible (for the various products and the platform concurrently) or an architectural board serves as a decision body.

To accept decisions concerning requirements integration, an organization's members must understand how decisions are made, and the relevant stakeholders must be involved. So, the product management team, the platform team, and the various product development teams must share a process for requirements prioritization in a product line context. Typically, organizations can enforce communication using job rotation. This lets the platform development stakeholders (developers and managers) better understand the product needs. Other approaches include discussion forums, frequent meetings, or tool support (current requirements status or accessible decision lists). Acceptance increases if an organization establishes measures to show benefits and problems. Examples for measurements are "number of products in which the functionality and its solution appear," "efforts for realization and integration," or "risk-versus-benefit estimates."

Measurements enforcing acceptance by staff and management lead to the second type of criteria. In the SPL context, introducing a prioritization process involves the following:

- *Product requirement prioritization*: Which requirements should you introduce into which product?
- *Platform requirement prioritization*: Which requirements should you introduce into the platform and when?
- *Product development planning and organization*: In which sequence should you integrate requirements into the platform? When should individual product developments reuse those requirements?

Various possible criteria exist. However, in the context of product line development, we should make some variations to these criteria owing to the available reuse potential's impact.[7]

Strong pilot client influence. Although a pilot client is important to successfully establish SPL, implementing its specific requirements might conflict with the SPL development strategy. To avoid letting the SPL focus too narrowly on the pilot client, these tactics are useful:

- While working with the pilot client, never lose sight of the overall domain. For instance, perform domain analysis before or during pilot-client-driven platform development. We discovered that all the organizations in the investigation performed some kind of domain analysis in this way. For large-scale industrial product development, up-front SPL investments are not always possible. In these cases, you should perform domain analysis in parallel with platform development for the pilot client. The analysis will then form the basis for well-focused platform extensions in the context of future product instances.

- If you design the platform on the basis of the pilot client's needs, walk through the platform's features and explicitly document expected deviations other clients require.
- Develop a sound vision of the product line and clearly communicate it throughout the organization.
- Ensure that platform components are sufficiently generic and well encapsulated. This generally strengthens platform applicability to future projects. However, don't make components too generic or complex. For instance, avoid unnecessarily rich component interfaces. Rather, extend the interfaces later when needed.

Realization of platform requirements in products. Owing to the product team's milestones, it often cannot wait for the platform team to implement all their requirements. So, product teams must often implement platform requirements as product requirements. This means that two products implement and maintain the same requirement twice. Also, transferring previously implemented product features into the platform later will be expensive. We discovered that most of the work group organizations fought such problems with organizational measures:

- Minimize application engineering. Ensure that feature teams perform as much development as possible, while client-specific teams derive the final products by integrating platform components.
- Perform systematic product line scoping to clarify which requirements you'll implement in the platform. On the basis of this clarification, actively enforce implementing these requirements in the SPL platform only.
- Establish some mechanism of job rotation between platform and product development. This creates awareness among the developers about where a requirement is best implemented.
- Install an architecture review board that fulfills cross-sectional functions and mediates across product and platform development. The board should be responsible for the overall architecture and decide how and where to implement requirements.
- Enforce the development of explicit architectural models that include clear definitions of their semantics.

Requirements engineering practices

A precise requirements engineering process—a main driver for successful software development—is even more important for product line engineering. Usually, the product line's scope addresses various domains simultaneously. This makes requirements engineering more complex. Furthermore, SPL development involves more tasks than single-product development. Many product line requirements are complex, interlinked, and divided into common and product-specific requirements. So, several requirements engineering practices are important specifically in SPL development:

- Domain identification and modeling, as well as commonalities and variations across product instances
- Separate specification and verification for platform and product requirements
- Management of integrating future requirements into the platform and products
- Identification, modeling, and management of requirement dependencies

The first two practices are specific to SPL engineering. The latter two are common to software development but have much higher importance for SPLs.

Issues with performing these additional activities can severely affect the product line's long-term success. During the investigation, we found that most organizations today apply organizational and procedural measures to master these challenges. The applicability of more formal requirements engineering techniques and tools appeared rather limited, partly because such techniques are not yet designed to cope with product line development's inherent complexities. The investigation determined that the following three SPL requirements engineering practices were most important to SPL success.

Domain analysis and domain description. Before starting SPL development, organizations should perform a thorough domain analysis. A well-understood domain is a prerequisite for defining a suitable scope for the product

Many product line requirements are complex, interlinked, and divided into common and product-specific requirements.

Not knowing the SPL architecture's capabilities inevitably leads to the architecture not being fully used.

line. It's the foundation for efficiently identifying and distinguishing platform and product requirements.

Among the five participants in our investigation, three explicitly modeled the product line requirements. The others used experienced architects and domain experts to develop the SPL core assets without extensive requirements elicitation. Two organizations from the first group established a continuous requirements management that maintained links between product line and product instance requirements. The three other organizations managed their core assets' evolution using change management procedures and versioning concepts. Their business did not force them to maintain more detailed links between the requirements on core assets and product instances.

The impact of architectural decisions on requirements negotiations. A stable but flexible architecture is important for SPL development. However, focusing SPL evolution too much on architectural issues will lead to shallow or even incorrect specifications. It can cause core assets to ignore important SPL requirements so that the core assets lose relevance for SPL development. Organizations can avoid this problem by establishing clear responsibilities for requirements management in addition to architectural roles.

The work group participants reported that a suitable organizational tool for balancing requirements and architecture is roundtable meetings in which requirements engineers, lead architects, and marketing and sales personnel discuss SPL implementation. Also, integrating the architects into customer negotiations will solve many problems that can arise from conflicting requirements. Another measure is to effectively document requirements and architectural vision so that product marketing and SPL architects can understand each other and agree on implementation.

Effective tool support. We often discussed tool support for SPL requirements engineering during the investigation. Because requirements engineering for SPL can become highly complex, effective tool support is important. Existing tools don't satisfactorily support aspects such as variability management, version management for requirements collections, management of different views on requirements, or dependency modeling and evolution. So, an SPL organization must design custom solutions for these issues. Specifically, the two participants in the investigation that had established continuous requirements management had to maintain expensive customization and support infrastructures for their tool environment. The other organizations tried to avoid these costs by mitigating insufficient tool support through organizational measures such as strict staging of the requirements specification.

Architectural practices

The decision to develop an SPL often arises from the expected benefit of the platform, which should provide a common architecture for all the product line's members. All products should fit into the provided architecture and benefit from it. Unfortunately, the common architecture's functionality, interfaces, and constraints are usually abstract and complex. Not all the development organization's members or teams will understand them well.

Not knowing the SPL architecture's capabilities inevitably leads to the architecture not being fully used. So, requirements that developers have already implemented into the platform might be reimplemented for various products. The multiple implementation of a requirement leads first to an overhead—which is linked to avoidable, possibly high costs—and then to a useless platform because its capabilities are not exploited.

A major problem arises if the multi-implemented requirements are constrained by other requirements so that the SPL architecture becomes unstable. Again, implementing (original) platform requirements in products will erode the platform and reduce its advantages.

Ensuring that a product line adequately uses architectural advantages requires that three capabilities be in place.

Required capabilities. First, you should explicitly define and document the architecture. Good documentation of architecture, platform features, and generic interfaces is important for the product teams to understand the reusable assets. From the participants' experience, organizations must complement measures for architecture documentation with organizational methods, such as establishing an architectural role that is responsible for defining, communicating, and maintaining the architecture.

You should describe architecture using well-established notations such as the Unified Modeling Language, and the architecture description should cover all relevant architectural views and use clearly defined semantics. You should supplement it with architectural scenarios that present the architecture from a system use perspective. Often, use cases and textual descriptions serve that purpose.

Second, you should communicate the architecture to stakeholders. Simply defining and documenting the architecture is insufficient; you must properly and adequately communicate it as well. Actively disseminating this information is key. You must address the various stakeholder needs and give them the necessary information. You must also find an adequate notation to communicate this information to stakeholders such as marketing or sales personnel who are not apt at reading technical notations.

Third, you must enforce exploitation of the available architecture. To enforce the architectural principles, you must install responsible roles. This can be a lead architect or a whole architecture review board. This process must start early; when new projects are under negotiation, you must ensure that they're compatible with the existing architecture.

Architectural roles. We concluded that you can support all three success factors when you define and deploy explicit architectural roles with clear responsibilities in the organization. The following four roles match the experiences of most of the work group participants:

- Product architect
- Product line architect
- Domain architect
- Component architect

Each architect has a clear idea of what to document. For instance, the product line architect concentrates on architectural style and principles and describes the boundary between framework and product. The component architect describes component capabilities and the component's relationship to other relevant components. The architects are responsible for traceability between (product) requirements and architecture solutions. They should also communicate the architecture to the various stakeholders and inform them about architectural changes.

During our investigations, it became clear that certain practices are more common and widespread across the industry than others. For example, organizations tend to avoid dedicated organizational restructuring toward separate product line organizations. Rather, most organizations prefer to develop an SPL core

About the Authors

Andreas Birk is a consultant and software engineering professional at sd&m (software design and management). His special interests include software engineering methods, knowledge management, and requirements engineering. He received his Dr.-Ing. in software engineering and his Dipl.-Inform. in computer science and economics from the University of Kaiserslautern. He's a member of the IEEE Computer Society, the ACM, and the German Computer Society. Contact him at sd&m AG, Löffelstraße 46, D-70597 Stuttgart, Germany; andreas.birk@sdm.de.

Gerald Heller is a senior software engineering consultant at Hewlett-Packard in Germany. He has worldwide responsibility for the requirements engineering process at HP's largest software organization. His research interests include collaborative, component-based development. He received his PhD in computer science from Friedrich Alexander University of Erlangen in Germany. Contact him at Hewlett Packard GmbH, Schickardstraße 25, D-71034 Böblingen, Germany; gerald.heller@hp.com.

Isabel John is a researcher and consultant in software product lines at the Fraunhofer Institute for Experimental Software Engineering. Her main interests include requirements engineering for product lines, scoping, and legacy integration into product lines. She received her Diplom, in computer science from the University of Kaiserslautern. Contact her at the Fraunhofer Inst. for Experimental Software Eng., Sauerwiesen 6, D-67661 Kaiserslautern, Germany; isabel.john@iese.fraunhofer.de.

Thomas von der Maßen is a member of the Software Construction Group and a PhD student at the University of Aachen. His research interests include requirements engineering for software product lines, especially the modeling of variability and tool support. He received his Diplom in computer science from the University of Aachen. Contact him at Research Group Software Construction, RWTH Aachen, Ahornstraße 55, D-52074 Aachen, Germany; vdmass@cs.rwth-aachen.de.

Klaus Müller is an internal consultant for requirements engineering and organizes the knowledge transfer between business units for corporate research and development at Robert Bosch, Stuttgart. His research interests include self-assessment methods and mastering process improvement. He received his PhD from the Technical University of Aachen. Contact him at Robert Bosch GmbH, Robert-Bosch Straße 2, D-71701 Schwieberdingen, Germany; klaush.mueller@de.bosch.com.

Klaus Schmid is the department head for requirements and usability engineering at the Fraunhofer Institute for Experimental Software Engineering, where he has worked on several projects that transferred product line engineering concepts to industrial environments. His research interests include requirements engineering and product line development and institutionalization. He received his PhD in computer science from the University of Kaiserslautern. Contact him at the Fraunhofer Inst. for Experimental Software Eng., Sauerwiesen 6, D-67661 Kaiserslautern, Germany; klaus.schmid@iese.fraunhofer.de.

in a task-force effort. Later, they attempt to evolve their core SPL assets in the context of their product- or customer-specific projects.

It also became clear that SPL practices depend on each other and on contextual factors, such as the development organization's structure, product characteristics, and business models. Selecting one practice raises the need for others to achieve leverage effects or compensate for that practice's drawbacks.

Overall, we observed a clear trend toward SPL adoption. SPLs are appearing more and more frequently throughout industry, and they're growing more and more complex. Product line engineering is a key strategic technology for software organizations to attain and maintain unique competitive positions. However, it also became clear during the investigations that we must overcome many obstacles before effective SPL engineering becomes wide-spread. Further systematic collection and provision of SPL experience is necessary.

References

1. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002.
2. J. Bosch, *Design and Use of Software Architectures*, Addison-Wesley, 2000.
3. D.M. Dikel, D. Kane, and J.R. Wilson, *Software Architecture: Organizational Principles and Patterns*, Prentice Hall, 2000.
4. A. Birk, "Three Case Studies on Initiating Product Lines: Enablers and Obstacles," *Proc. PLEES 2002 Product Line Engineering Workshop*," Frauenhofer IESE, 2002, pp. 19–25.
5. K. Schmid and C. Gacek, "Implementation Issues in Product Line Scoping," *Software Reuse: Advances in Software Reusability*, LNCS 1844, Springer-Verlag, 2000, pp. 170–189.
6. J. Bayer et al., "PuLSE: A Methodology to Develop Software Product Lines," *Proc. Symp. Software Reusability* (SSR 99), ACM Press, 1999, pp. 122–131.
7. K. Schmid, "A Comprehensive Product Line Scoping Approach and its Validation," *Proc. 24th Int'l Conf. Software Eng.* (ICSE 02), ACM Press, 2002, pp. 593–603.