Modeling and Using Product Line Variability in Automotive Systems

Steffen Thiel and Andreas Hein, *Robert Bosch Corporation*

Automotive systems provide a broad spectrum of services that fundamentally improve passenger comfort, safety, economy, and security. Parking assistance or adaptive cruise control systems make it easier to operate a car in various driving situations, thus reducing drivers' workload and increasing their comfort. Safety-related systems, such as automatic stability or airbag control, help drivers avoid or reduce the impact of accidents. Fuel economy systems lower emissions and increase fuel efficiency, while security systems protect the car from unauthorized manipulation.

Product lines offer a promising approach to automotive system development because they permit strategic reuse of core assets. However, to gain significant economies of scope, variability must be systematically considered throughout the development process.

An automotive system typically consists of dedicated processors, software, and interfaces that let the system measure, manipulate, and otherwise interact with its external environment. Designers optimize such systems to reflect specific application characteristics. Apart from a system's desired functionality, automotive system designers must consider many possibly conflicting qualities and constraints. Developing an automotive system can thus involve hundreds or thousands of variants, adding to the existing engineering complexity. Whereas variability has typically been addressed on a case-by-case basis in late development phases, designers now need a managed, systematic approach to the ever-increasing number of variants.

Product lines provide this systematic approach, along with a special focus on variability among related products. As we discuss here, systematic planning and continuous variability management is a prerequisite for effective product lines. We've developed an approach to modeling and utilizing variability to support the efficient creation of product variants. Our approach is based on experiences with several industrial case studies at Bosch. Before describing them, we explain how product line development meets the major design challenges in the automotive system domain.

Product line development

Automotive systems typically have thousands of requirements, but some are especially important. Many automotive systems are real-time systems with strict temporal requirements that result from the internal control loops. Thus, a computation's correctness depends, in part, on its timeliness.

0740-7459/02/$17.00 © 2002 IEEE

Furthermore, designers must guarantee the safety and reliability of the automotive system software and embedded computer, even under harsh conditions including excessive heat or cold, vibration, shock, power supply fluctuations, water, and corrosion. Another essential quality of automotive systems is their availability, or readiness for use. Maintainability might also be important. Legacy software, for example, might have to run on replacement hardware. Finally, security is critical: Developers must be able to guarantee that the system software cannot be easily manipulated.

Although many of these challenges involve comprehensive research and analysis, most have been solved by technical means. However, providing such solutions in a way that is both cost-effective and allows a short time-to-market remains challenging for both traditional and platform-based development.

Traditional development

Today's automobiles use many automotive systems. Luxury cars, for example, can include more than 80 electronic control units that operate as single, partly networked systems. In these systems, the software portion is often highly adapted to the underlying hardware and implements fixed, very specific functions (such as adjusting seats or lifting windows).

Although companies might have considered the development of unifunctional entities cost-effective in the past, it is hardly so when we consider the total functionality of the car's automotive systems. The disproportionate hardware costs, along with excessive software development and maintenance costs for the various automotive systems, make the conventional "one at a time" approach singularly unattractive. Moreover, the restricted reusability that results from binding software functionality to dedicated hardware—as well as the additional packaging, power consumption, and electromagnetic interference—now make it difficult to profitably engineer automotive systems in the traditional way.

Platform-based development

To overcome these problems, the industry recently began integrating automotive functions on powerful multipurpose platforms that replace mechanical and electronic components with intelligent software solutions. For example, companies now use a common platform for infotainment systems (including, for example, a radio, CD player, and navigation system[1]) and safety systems (including parking assistance and precrash detection[2]). Although adopting a platform-oriented development permits additional services, more flexibility, and shared hardware use, cost-effectiveness and time-to-market have still not been addressed. Consequently, the effort required to develop more complex platform software is not fully compensated by the hardware cost savings.

The product line approach

Despite their high volume, automotive systems nonetheless have numerous variations due to differences among customers, price, and technology. Therefore, a strategic reuse approach that guarantees economies of scope is indispensable. We can achieve this strategic reuse by adopting a product line approach to platform-based development.

A software product line is a set of software-intensive products that share a common, managed feature set that satisfies the specific needs of a particular market segment. Product line development proceeds from a common set of core assets in a prescribed way.[3]

Economies of scope imply a mass-customization ability,[4] which in turn requires a systematic consideration of variability throughout product line development. Paradoxically, the latter is often dismissed as secondary. Nonetheless, as we now describe, this variability is crucial to achieving effective product lines.

Modeling product line variability

Developing product line products differs from developing single products in that variability is an inherent part of the modeling. This does not mean that common software engineering practices are obsolete. Rather, we must both extend these practices and develop new ones.[5]

Variability affects all product line artifacts, from requirements to code. Clearly, we need specific solutions to support the specific customer needs that motivated the variation. However, in current practice, designers often give variability incidental treatment. They typically introduce it during late design or implementation and express it, for example, through myriad compiler switches. Moreover,

Developing product line products differs from developing single products in that variability is an inherent part of the modeling.

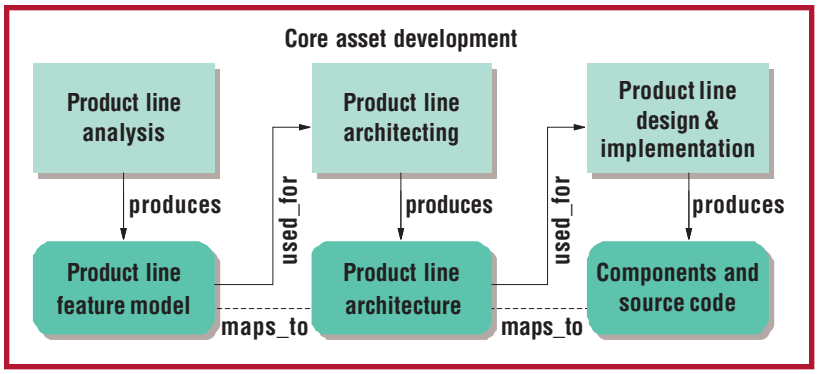

Figure 1. Managing variability throughout core asset development. The figure shows a representative set of processes (light green boxes) and artifacts (dark green boxes).

designers often introduce a variation point based on heuristics or expert knowledge. The documentation of variable requirements addressed by a variation point is often implicit, making the variation's rationale hard to identify. With such an approach, product customization—and especially the integration of new features—is complex and error-prone. Therefore, handling variability late in development eliminates the company's ability to achieve significant economies of scope.

Our approach addresses these problems by systematically and continuously incorporating variability throughout product line engineering. We must introduce and refine variability during core asset development and reflect variability in the production artifacts.[6] Figure 1 shows a representative set of processes and artifacts; a more comprehensive overview is available in the literature.[3]

Feature model

The feature model is an essential result of product line requirements analysis.[7–10] It captures product line members' functional and nonfunctional capabilities, as well as their commonalities and variabilities. It also provides various stakeholders with a valuable view of the product line. For example, customers can use the feature model to gain an understanding of the product line's functionality, while system architects and product engineers use it to drive the development of product variants.

Figure 2 shows a simplified example of a feature model for a car periphery supervision product line. CPS systems provide passenger comfort and safety functions based on sensors that detect objects in the vehicle environment.

The feature model structures CPS product line capabilities into a tree that shows designers which variants to create and a constraint network that coordinates their combination. For clarity, we've omitted the constraints network in Figure 2, which shows only the tree structure:

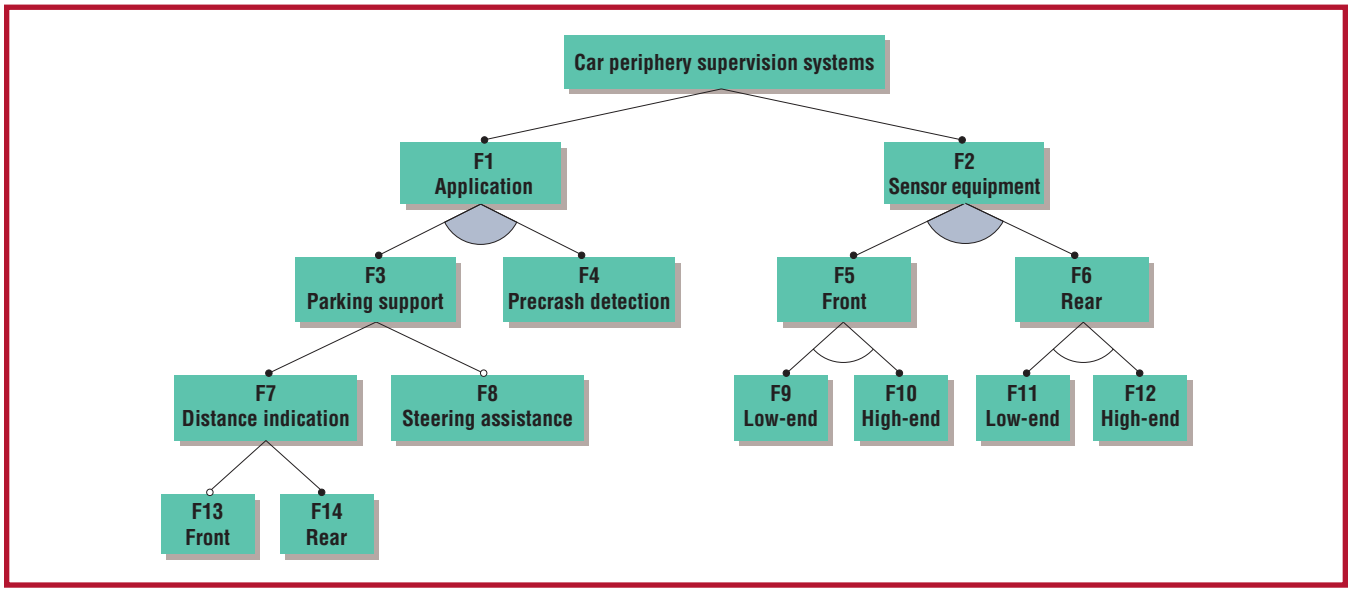

Figure 2. A feature model for a product line that provides car periphery supervision, which uses sensors to detect objects in the vehicle's environment. "F" indicates a feature. Filled arches indicate or-features and empty arches indicate alternative features; filled circles indicate mandatory features and empty circles indicate optional features.[10]

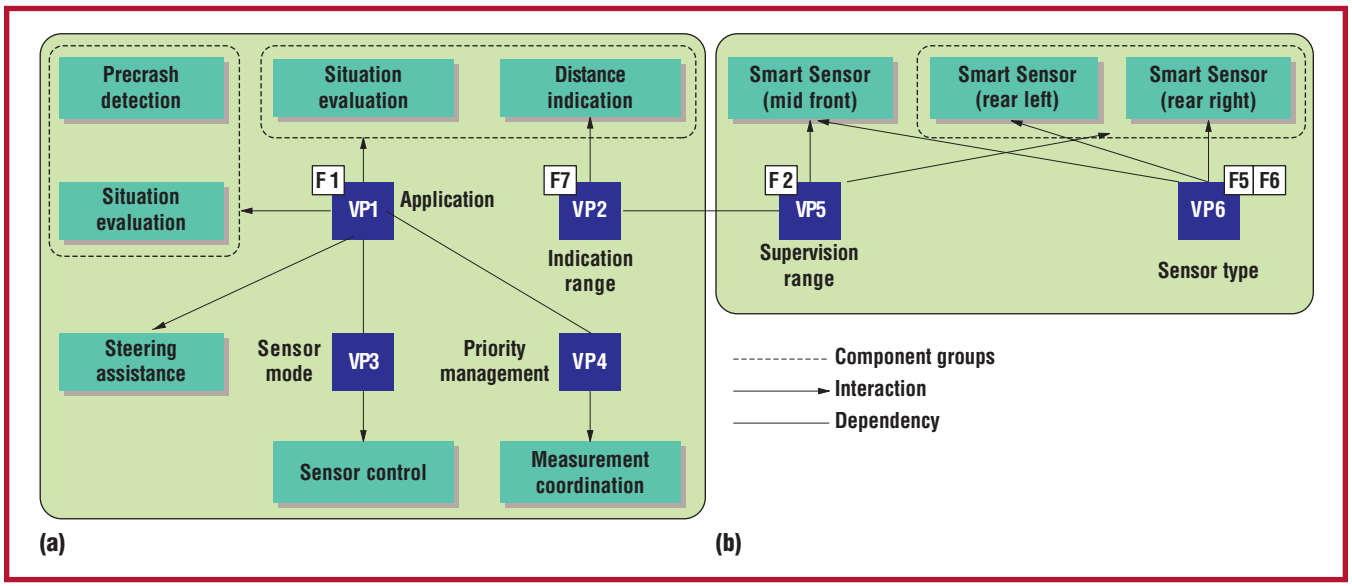

Figure 3. Variability in the (a) logical architecture and the (b) physical architecture. The variation points satisfy the variability among the requirements in Figure 2. "F" indicates a feature and "VP" a variation point.

- The *application* branch is concerned with the intended CPS system functionality.
- The *sensor equipment* branch shows the hardware variants required to realize the sensor platform on which the functionality is based.

As Figure 2 illustrates, a CPS system includes at least one of two applications: parking support and precrash detection. Parking support basically consists of rear-distance indication; we can enhance it with front-distance indication and steering assistance. We can define sensor equipment for the car's front or rear, applying either a low- or high-end variant for each.

Product line architecture

Architecture is the first design artifact that places requirements into the solution space. Designers typically organize the architecture description into multiple architectural views. Each view represents the target system from a particular perspective while addressing one or more stakeholder concerns.

With a product line, the architecture must also capture design element variability.[3,6,11] Architectural variability represents alternative design options that could not be bound during architectural modeling. Designers often express this variability as a set of architectural variation points that show (part of) the architectural solution to variable features.

A feature model does not, however, imply a specific design; rather, it hints about where designers must pay special attention to structuring an architecture—for example, with respect to configurability.[11] Nevertheless, configurability is unlikely to be the only attribute a designer must consider during architecture design. In the automotive context, performance, safety, and reliability also play important roles, as mentioned earlier. The final architecture must consider all functional and nonfunctional requirements, including qualities and design constraints.

Figure 3a and b shows the variability in the logical and the physical views of the CPS product line architecture. At the architectural level, we introduce variation points to satisfy the variability among the requirements in Figure 2. We characterize each variation point by specifying how and when a variation point applies. As the solid lines between them show, variation points might depend on each other to define consistent component configurations.

The logical view contains four variation points. As the arrows indicate, variation point 1 affects a logical component, steering assistance, and two component groups. Variation points 2, 3, and 4 affect only individual logical components (distance indication, sensor control, and measurement coordination, respectively). Additionally, there are dependencies between variation points 1 and 3, 1 and

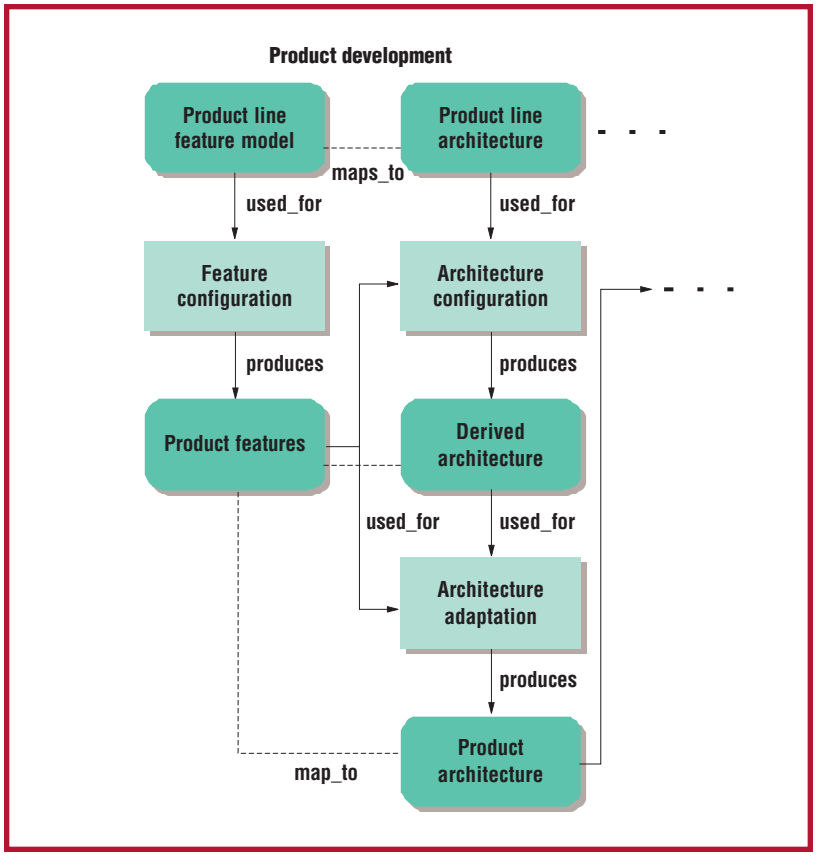

Figure 4. Interaction between the feature and architecture models to utilize variability. Light green boxes represent processes; dark green boxes represent artifacts.

4, and 2 and 5. Variation point 2 parameterizes the indication range software, whereas variation points 5 (supervision range) and 6 (sensor type) map the corresponding elements to the hardware platform. Thus, variation points 5 and 6 are part of the CPS's physical view, describing the sensor equipment that the specified functionality requires.

Variability also affects other architectural views, including the process and deployment views, which we discuss elsewhere.[11]

Other work products

The feature model and architecture documentation represent only a portion of the work products required for product line development. The feature model represents product line members' particular capabilities, while the product line architecture offers the overall structure for realizing these capabilities. To create product line members, designers must consistently refine design solutions for realizing both the common and variable features during detailed design and implementation.

Not all variable features will inevitably affect the architecture's overall organization. Rather, designers encapsulate some variability from the architectural viewpoint, and it first appears at a more detailed level. In automotive systems in particular, there are problems (and variations among them) that designers can adequately address through component design or code constructs. Examples include algorithmic conversions of feedback control activities, software code encryption to prevent unauthorized tuning, or runtime data and instruction compression for optimizing memory efficiency.

As designers refine the architecture during design and implementation, the number of variation points usually increases because the mechanisms must ultimately be realized through constructs at a lower abstraction level. Nevertheless, the concrete solutions we use to implement a variation point must conform to the conventions defined in the architecture. To control this process, establishing adequate traceability links—which reveal the rationale behind a code-level variation—is vital.

Using product line variability

As we mentioned earlier, work products created during product line development do not exist in isolation. Rather, they relate to each other as designers refine and realize requirements stepwise from analysis to code. Stepwise refinement includes the variability identified first in the feature model, then implemented in architectural variation points. Designers should explicitly map features to the corresponding architectural variation points. In principle, this mapping shows how the architecture's variability mechanisms contribute to the realization of the feature model's variability.

In Figure 3, we denote this mapping using feature identifiers attached to the architectural variation points. For example, the options associated with the application feature (F1) affect the architectural variation point 1. This point is associated with the corresponding logical components in the product line architecture. Variation points 3 and 4 are only indirectly affected by feature variants (through their relationship to variation point 1). Variation point 6 depends on both the feature specification of the front (F5) and rear (F6) sensor equipment. This example raises two major points:

- The correspondence between features and architectural variation points is rarely 1:1 (F5 and F6 both map to variation point 6, for example).

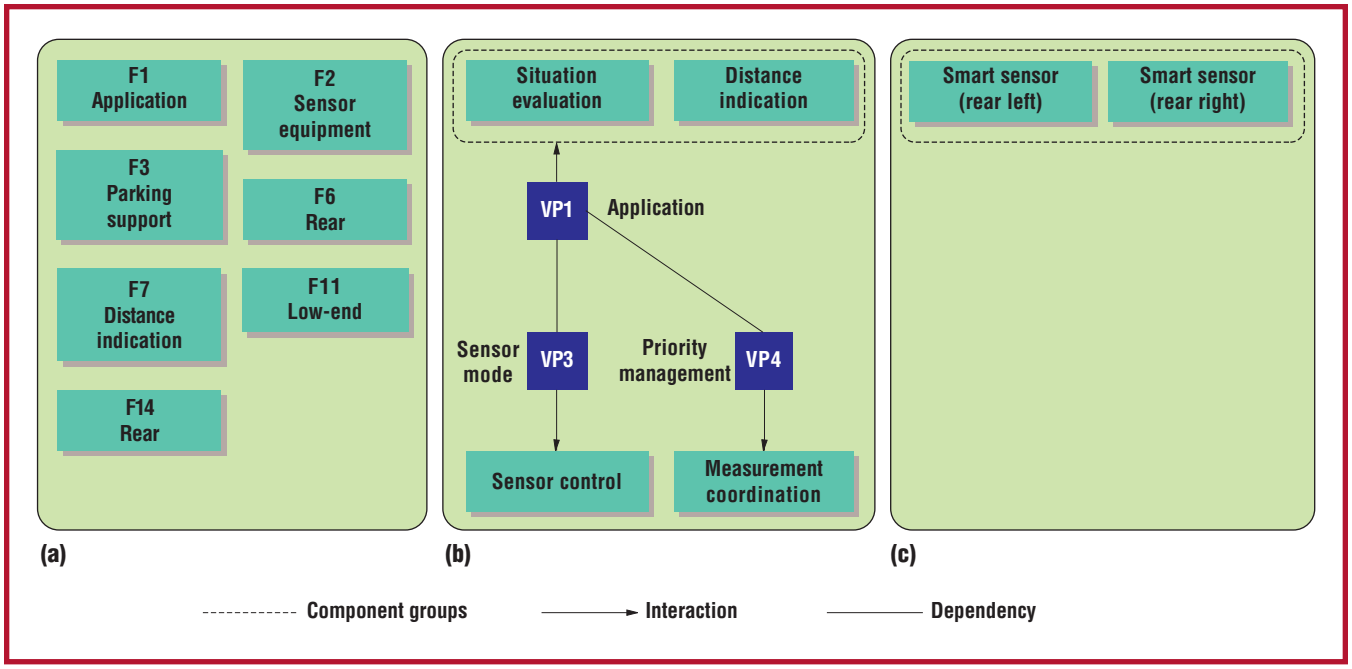

Figure 5. CPS parking support variant, including (a) product features, (b) the related variation points in a portion of the logical view, and (c) a portion of the derived physical view, which contains no variability and can thus be input directly into the product architecture.

- The architectural variation points do not introduce new variability; rather, they realize the feature model's variability. Variation points 3 and 4 do not contradict this statement; they simply support variation point 1 and F1, respectively.

The traces between the features and the architecture not only help stakeholders understand how designers have realized product line variability, but also can be effectively used for product derivation. We therefore propose an extension to the concept of feature modeling to support the development of product line members, starting from their feature specifications.[8,12] The basic idea is to choose among the different feature options and resolve the variability according to customer needs.

For example, we might specify a low-end parking support that displays distances to rear obstacles by selecting the features F3 and F11 (see Figure 2). Such selections must be consistent with the relationships among features. Following feature selection, a product developer can use a configuration tool to propagate the selections to the architecture. Figure 4 shows how the feature and architecture models work together.

The product line's feature model serves as a starting point for deriving a product. In the feature configuration process, we use the feature model to specify products in terms of features. Then, in the architecture configuration process, we use the specified product features to bind the corresponding variation points in the product line architecture by using the traceability between the product line feature model and architecture. The result is a derived architecture that conforms to the product features. The architecture also serves as the basis for potential customizations in the architecture-adaptation process, which yields the actual product architecture.

To support product architecture maintenance, product developers should preserve the traces between the derived architecture and the corresponding product features. The reason for this is that product developers eventually must adapt an architecture to add features specific to only a few products. The decision about what is built "inside" and "outside" a product line is based on business considerations that shape the product line's scope. In practice, new features that are not directly supported by the current product line architecture still must be included to satisfy all customer requirements. However, slight adaptations are acceptable as long as those features require only small, local changes to the architecture and don't negatively affect the overall architectural quality. Generally, management must make an explicit decision about whether to include such "extra features" at the product or product line level.

Figure 5 shows the product features, the corresponding variation points in the logical view, and the derived physical architecture. Because we selected F3 and F11, we must select other features as well. F7 and F14, for

Work products created during product line development do not exist in isolation.

example, are mandatory parts in every parking-support application (see Figure 2, for example). The derived logical view consistently excludes steering assistance and precrash detection but still contains variation points that must be resolved as part of the product architecture design. The derived physical view contains no variability after resolving variation points 5 and 6 according to the feature selections, so we can use it as direct input to the product architecture.

We are currently working on concepts and techniques to extend and improve variability modeling and management for industrial applications. Our research areas include representation issues and modeling and traceability guidelines for different development phases, as well as tools that exploit variation points to support efficient product creation. In addition to these topics, we are working to refine our system-engineering processes to make product line development more effective.

References

1. P. Motz et al., "Mobile Media Open Computing Platform," *Proc. In-Vehicle Software 2001* (SP-1587), SAE 2001 World Congress, Soc. of Automotive Engineers, Warrendale, Pa., 2001, pp. 135–153.
2. S. Thiel et al., "A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems," *Proc. In-Vehicle Software 2001* (SP-1587), SAE 2001 World Congress, Soc. of Automotive Engineers, Warrendale, Pa., 2001, pp. 43–55.
3. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, Boston, 2001.
4. C.W. Krueger, "Easing the Transition to Software Mass Customization," *Software Product Family Engineering*, F. van der Linden, ed., Lecture Notes in Computer Science no. 2290, Springer-Verlag, Berlin, 2002, pp. 282–293.
5. S. Thiel, "On the Definition of a Framework for an Architecting Process Supporting Product Family Development," *Software Product Family Engineering*, F. van der Linden, ed., Lecture Notes in Computer Science no. 2290, Springer-Verlag, Berlin, 2002, pp. 125–142.
6. F. Bachmann and L. Bass, "Managing Variability in Software Architectures," *Proc. Symp. Software Reusability: Putting Software Reuse in Context*, ACM Press, New York, 2001, pp. 126–132.
7. G. Chastek et al., *Product Line Analysis: A Practical Introduction*, tech. report CMU/SEI-2001-TR-001, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 2001.
8. A. Hein, J. MacGregor, and S. Thiel, *Configuring Software Product Line Features*, tech. report 2001-14, Dept. of Computer Science, Univ. of Karlsruhe, Karlsruhe, Germany, 2001, pp. 67–69.
9. K.C. Kang et al., *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, tech. report CMU/SEI-90-TR-21, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1990.
10. K. Czarnecki and U.W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, Boston, 2000.
11. S. Thiel and A. Hein, "Systematic Integration of Variability into Product Line Architecture Design," to be published in *Proc. 2nd Software Product Line Conf.* (SPLC2), Springer-Verlag, Berlin, 2002.
12. A. Hein, M. Schlick, and R. Vinga-Martins, "Applying Feature Models in Industrial Settings," *Software Product Lines: Experience and Research Directions*, P. Donohoe, ed., Kluwer Academic Publishers, Boston, 2000, pp. 47–70.

About the Authors

Steffen Thiel is project leader for product line improvements in the Software Technology Department of Robert Bosch Corporate Research and Development in Frankfurt, Germany. At Bosch, he was responsible for the European product line research project ESAPS and is now leading Bosch activities for its successor, CAFÉ. Prior to his product line activities, he developed intelligent vehicle information systems at Bosch. His research interests include requirements and feature analysis, quality-driven architectural design, product derivation, and evolution. He received a diploma in computer science from the Technical University of Darmstadt. He is a member of the IEEE Computer Society. Contact him at Robert Bosch Corp., Corporate Research and Development, Software Technology, P.O. Box 94 03 50, D-60461, Frankfurt, Germany; steffen.thiel@de.bosch.com.

Andreas Hein is a member of the scientific staff at the Corporate R&D Department for Software Technology at Robert Bosch Corporate Research and Development in Frankfurt, Germany. He has worked on several European software product line projects, including PRAISE, ESAPS, and CAFÉ. In addition to product lines, his research interests are in software engineering practices and configuration systems. He received a diploma in computer science from the Technical University of Darmstadt. Contact him at Robert Bosch Corp., Corporate Research and Development, Software Technology, P.O. Box 94 03 50, D-60461, Frankfurt, Germany; andreas.hein1@de.bosch.com.