Three Cases of Feature-Based Variability Modeling in Industry*

Thorsten Berger[1], Divya Nair[1], Ralf Rublack[3], Joanne M. Atlee[4], Krzysztof Czarnecki[5], and Andrzej Wąsowski[5]

[1] University of Waterloo, [2] University of Leipzig, [3] IT University of Copenhagen

Abstract. Large software product lines need to manage complex variability. A common approach is variability modeling—creating and maintaining models that abstract over the variabilities inherent in such systems. While many variability modeling techniques and notations have been proposed, little is known about industrial practices and how industry values or criticizes this class of modeling. We attempt to address this gap with an exploratory case study of three companies that apply variability modeling. Among others, our study shows that variability models are valued for their capability to organize knowledge and to achieve an overview understanding of codebases. We observe centralized model governance, pragmatic versioning, and surprisingly little constraint modeling, indicating that the effort of declaring and maintaining constraints does not always pay off.

1 Introduction

Many modern systems contain an increasing amount of *variability* to tailor systems for different customers and hardware. Variability can be realized using a wide range of mechanisms including static and dynamic configuration parameters, components, frameworks, and generators. Variability-rich systems range from large industrial product lines [12,32,1] to prominent open-source software, such as the Linux kernel [7] with over 11,000 configuration options—aka *features* [26].

Variability in these systems has to be managed. Variability modeling, the discipline of describing variability in formal representations—*variability models*—is one of the key techniques to deal with complex variability. Variability models, such as feature [26,14] or decision [35,16,13] models, provide abstractions of the variabilities present in software. They allow engineers to scope systems and to plan their evolution; they can also be used for system configuration and derivation using automated tools, such as configurators and generators.

However, variability modeling, as any modeling layer, comes at a cost. Models have to be created and maintained, tools introduced, developers trained, and possibly the organization restructured. These costs may outweigh any realized benefit, such as a high degree of automation or decreased time-to-market—two benefits often emphasized in the literature. But surprisingly, although hundreds of publications target variability modeling techniques [10,22,9], little is known

* Partially supported by ARTEMIS JU (grant n°295397)

about actual practices in the industry. This scarcity of published empirical data impedes research progress and the improvement of methods, languages, and tools.

We attempt to address this gap with an exploratory case study of variability modeling in three companies. Our objective is to provide contextualized empirical data on practices, and to elicit perceived strengths and weaknesses of variability modeling. The analysis of each case is guided by three research questions:

- *How are variability models created and evolved (**RQ1**)?* We investigate modeling practices, such as strategies to identify features and to modularize, evolve and scale models. We also gather core characteristics of the models.
- *What are the benefits (**RQ2**) and what are the challenges (**RQ3**) of variability modeling?* We identify technical, organizational, and commercial values and challenges of modeling, as experienced and perceived by practitioners.

To put this empirical data into context, we also inquire organizational structures supporting the practices, and elicit scales, architectures, and technologies of the respective software product lines.

This case study is part of our ongoing effort to improve the empirical understanding of variability modeling. We previously surveyed companies in their use of variability modeling [5] and conducted semi-structured interviews with eight of them. In the present work, we select three companies and describe and analyze them in-depth. Our selection represents a broad range of development scales from very small (two developers) to ultra-large (100 development teams); comprises domains that commonly use variability modeling (automotive, industrial applications/energy, and eCommerce [5]); and covers all product-line adoption strategies (proactive, extractive, and reactive [27]). In contrast to quantitative research, our goal is not too reach any statistically significant deductions, but to describe the practices that were successful in three heterogeneous cases. We provide rich descriptions of three selected cases rather than analyzing all interviews, such as using Grounded Theory [20], which is the subject for future work.

We proceed as follows. Sect. 2 introduces variability modeling and related work. Sect. 3 describes our methodology. Sect. 4 presents results for all cases. Sect. 5 compares the cases, Sect. 6 discusses threats to validity, and Sect. 7 concludes.

2 Background and Related Work

We previously studied variability modeling in systems software [6]. That study revealed the significance of feature and decision modeling concepts in languages conceived by practitioners. It also showed that additional concepts (such as defaults, visibility conditions, derived features) are needed to scale modeling. Interestingly, the models had very different characteristics (size, shape, constraints) than models considered in research. In the present work, we strive to gain insight into how the results from the previous study relate to industrial practices. Our preceding survey [5] showed that feature models were among the most popular notations, but also that a wide range of notations and tools is used. It also confirmed the existence of large models—which have been reported before [39,38,29], but without any further characteristics, such as the use or complexity of constraints.

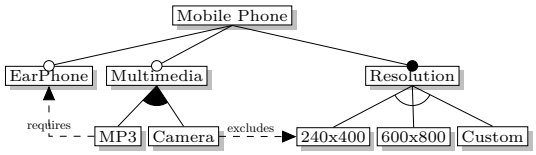

Fig. 1. Simple feature model (adapted from [18])

In the present study, all of our subjects use feature models. These are hierarchical structures of features, together with constraints that restrict valid feature combinations. Fig. 1 shows the model of a Mobile Phone. It always (solid dot) has the feature Resolution and optionally (hollow dot) has the features EarPhone or Multimedia, or both. Multimedia is an OR group (select at least one) and Resolution an XOR group (select exactly one). Further constraints reside in the hierarchy (child-parent implication) and in additional cross-tree constraints (*requires* and *excludes*). For instance, MP3 requires an Earphone. In practice, constraints may be more complex. Some languages support rich constraints (e.g., arithmetic) over features with non-Boolean values, such as numbers and strings [31,6].

Variability modeling is a core activity in software product line engineering (SPLE) [12]. Although detailed industrial experience reports on SPLE exist, only few focus variability modeling. The "Software Product Line Hall of Fame" [1], a catalog of SPLE case studies [37], and a practice-oriented book [28] contain information on adoption practices, organizational structures, and architectures, but offer little insight into the use of variability models, their sizes and contents, and the techniques used to build them. In fact, recent literature reviews on the evaluation of variability modeling lament the lack of empirical work on this topic [11,9,22,10]. Exceptions are industrial experience reports. Grünbacher et al. [21] emphasize that techniques need to be customized to the organizational context in which they are used; Reiser et al. [33] request compliance constraints for the same purpose; Riebisch et al. [34] point out the use of feature models by non-software developers; Gillan et al. [19] identify a lack of documented methodologies to create feature models. These reports are complementary to our study, but cannot provide a coherent picture. Finally, variability modeling can be seen as an instance of model-driven development (MDD). Hutchinson et al. [24,25] study MDD practices and experiences in industry. They reveal success factors for applying MDD, such as incremental adoption, organizational commitment, and integration with existing development processes. While these results are relevant to variability modeling, we strive to gain insights specific to variability modeling.

3 Methodology

We conducted semi-structured interviews with knowledgeable representatives from eight organizations identified in our previous survey [5]. In this paper, we explore three of these cases in depth. Our selection criteria were that the cases (i) represent a wide range of organizational sizes, (ii) stem from domains that most frequently apply variability modeling according to the survey, and (iii) cover all of the three common adoption strategies: proactive, extractive, and reactive [27].

Table 1. Variability model characteristics, variable artifacts, and variability mechanisms

	consulting company	component producer	car manufacturer
notation	feature model	feature model	semi-structured feature lists
tools	CaptainFeature	pure::variants	TeamCenter (prev. Excel)
modularization	single model	single model	hierarchy of models
model sizes (features, approx.)	40	1,100	top level: 300–500, intermediate level: up to 800, low-level: up to 3000
feature types	100% Boolean	95% Boolean, 5% integers and strings	100% Boolean
feature kinds	mandatory, optional	mandatory, optional	mandatory, optional
model hierarchy depths	5–6	3–4	2–3
cross-tree constraints	none	very few	none
custom relations	none	*recommended*	marketing relevance
variable artifacts	code, help system database schema	code (requirements and test cases planned)	code, logical design blocks, components, Simulink models
variability mechanisms	custom preprocessor and code generator	C preprocessor and dynamic parameters	C preprocessor and dynamic parameters
feature-to-artifact mapping	hard-coded in generator (imperative)	pure::variants' family model, feature Makefiles	informal textual descriptions and architecture diagrams

Each of the interviews lasted one hour on average. We allowed the interviewees to speak freely, but assured coverage of the following five topics: *Context* of variability modeling, including organizational structure, variability mechanisms, programming languages, and technologies; *Practices (RQ1)* used to create and evolve models, including roles and responsibilities of the actors involved; *Characteristics* of models, including size, shape, modeling elements, and richness of constraints; *Benefits (RQ2)* of variability modeling; and *Challenges (RQ3)* experienced. The interviews were recorded, transcribed, and analyzed by extracting information relevant to the research questions.

4 Results

We report the results in a structured narrative form together with interpretations. For each case, we provide the *context*; then in the first subsection the *practices (RQ1)*; in the second subsection the *benefits (RQ2)*; and in the third subsection the *challenges (RQ3)* of variability modeling. Interview quotes are prefixed with **A**, **B**, and **C** for the respective subjects, and our questions with **Q**.

Table 1 summarizes the *characteristics* of the respective models, the types of artifacts whose variability they describe, and variability mechanisms.

4.1 Consulting Company

Our first subject is a small ($\leq$ 50 employees) consulting company delivering customized web-based e-commerce and enterprise applications. The company specializes in MDD of software solutions for customers. We interviewed a department lead, acting both as a software architect and developer. Our case study focuses on a Java web-shop system that was in production for 2–3 years. Its purpose was to explore the potential of generator-based SPLE using variability management and modeling solutions originating from research—including feature

models, a feature-model configurator, and a code generator framework. The latter is the main variability mechanism: it conditionally compiles source files after preprocessing them with a home-grown preprocessor. The development can be characterized as follows:

- *Research-driven*: The company followed a textbook approach to variability modeling. It adopted practices mainly originating from [14], using a feature model and a code generator for product derivation.
- *Small-scale*: The development team comprised two developers, both working on the code, the generator framework, and the feature model (40 features).
- *Prototype-based*: The company started with a prototype to experiment and to gain experience with software product lines and feature modeling. The prototype went into production and was sold to six customers.
- *Fully platform-oriented*: All artifacts are integrated into one platform. New customer requirements are always realized within the platform.
- *Re-active*: The product line and the feature model are the result of decomposing an initial product into features.

Modeling Practices (RQ1). The company developed a feature model with the goal of configuring and deriving products automatically. It used the relatively simple tool CaptainFeature [3], which had usability issues, but no better tool existed in 2002. The interviewee emphasized the preference for having a tool that supported the exact graphical notation of feature models (Fig. 1). This representation of variability could be handled sufficiently well for a small model of 40 features, using the tool's zoom capabilities. To create models, the developers performed a domain analysis of the web-shop domain, including customer requirements. The developers modeled both variability (optional features) and commonality (mandatory features) of the product line. In this manner, following advice and processes from SPLE literature, the developers scoped the product line. New features were introduced either when requested by customers or when the team saw added value for future customers:

A: *The question in our case was rather: What can we sell to the customer? What would be the added value a customer might want to have [...]? We always looked at it from the perspective of what we can sell.*

A core part of a model is the feature hierarchy, which was developed top-down, based on domain-specific ontological relationships (*part-of*):

A: *We tried to come up with logical relationships between the features [...] we had a feature that was called "Catalog System". That was the basis, since a shop always has a catalog. If you cannot display articles, then you just don't have a shop. Underneath, we put features such as "Shopping Cart", since only when you have a catalog, it makes sense to take the shopping cart as a feature.*

However, decomposing the initial product might have influenced the creation of the feature model, and optional features mapped to artifacts might stem from a bottom-up approach. Thus, the commonality (domain modeling) was likely created top-down, while the actual variability was created bottom-up.

The resulting feature model had around 40 Boolean features and was relatively balanced. Our interviewee estimated around 2–6 children per non-leaf feature on

average, and a maximum depth of 5 or 6 levels. The model was under-constrained. Although constraints among features existed, only hierarchy constraints and feature groups (OR and XOR, Fig. 1) were modeled and could be used to support the configuration process.

The model evolved rarely and only 2–3 features were added per new customer. The overall structure of the feature model was also rarely changed and feature were never removed. Feature additions almost never affected existing functionality.

Benefits (RQ2). Our interviewee sees the main benefit of variability modeling in organizing the information needed to maintain an overview understanding of the system. He emphasizes that the tool and the model provide management facilities that are useful to summarize product capabilities, to understand relationships between features, and to see the assignment of features to customers.

The company also sees benefit in a feature-model configurator. However, shortcomings in it can negatively impact the configuration process. Yet, the company experienced no significant impact given the limited scale of the system:
A: *The tool wasn't really that optimal [...] we had no real support where we could see that feature X conflicts with feature Y [...] We might have sometimes reached a point where we didn't know what happens why, or when the nesting was too deep. But that wasn't anything dramatic.*

In the literature, a common argument for SPLE is the reduced time-to-market. When asked about this benefit, our interviewee responded:
A: *I'd answer with a clear "depends on." It reduces time-to-market when I can rely on a basis and only have to make small changes for a client. On the other hand, I cannot do it rashly or without care, because otherwise I break something in my product family, which is not planned either. Where it also helps is when the customer wants exactly what we already implemented, then the time-to-market really converges to zero.*

In summary, the company considered its prototype successful and reused most of its infrastructure in a subsequent system: a jewelry-ring configurator developed for a ring manufacturer. The company developed a DSL used to describe properties of rings, in order to generate 3D models of them. This DSL used feature-model concepts, but introduced domain-specific terminology and language elements to facilitate a fine-grained configuration of the rings. Interestingly, it also introduced feature cardinalities [15], which allow multiple instantiations (cloning) of features, since a prototyped feature model became too wide and shallow.

Challenges (RQ3). Despite rare evolution, the co-evolution of the variability model and the product-line infrastructure is considered as a major challenge:
A: *I think the biggest problem we faced at that time and also today, and which is not really solved yet, is the evolution: To exactly know how to evolve features, on which implementation components they depend, so that you don't break anything when you work in the generator. I mean, to keep the complete overview: what is there and how does it all play together?*

Interestingly, even though one of the main purposes of feature models is to allow non-experts (customers) to configure a product, this turned out to be difficult, as the customers did not have the right prerequisites:

A: *Currently, we use it just internally. When we started in 2003, the underlying idea was also to build a frontend from the feature model where the customer can freely configure—exactly like the paradigm. But we abandoned this idea relatively quickly, because it is still very difficult for the customer [...] In the end, you need a consultant who tells the customer what he needs, because that is the first problem. And then [you need] to understand what that means in our configuration.*

Another challenge lies in the organization of teams. Since the company is small, the same developers were building the platform infrastructure and target products simultaneously. Developers would get confused working in both worlds:
A: *We tried to develop the generator and target code in parallel. That was rather driven by the theory. But we noticed that it doesn't really make sense, I mean it slows us down [...] When you work in both worlds and you come to a spot in the target code where variability is addressed, you always automatically ask yourself whether it's something that you resolve in the target code or in the generator. And then you start pondering what makes most sense, and you loose time, although it's not your task to think about that as a target code developer.*

4.2 Component Producer

Our next subject is a large (≤25,000 employees) vendor of electronic and mechanical components for end-user and industrial applications. The company has a large portfolio of products, many of which are derived from ad-hoc product lines, often using a clone-and-own approach. We interviewed two software architects responsible for variability management in a division that develops a product line of software controllers for power electronics. The product line has twelve products, which are fully integrated into the platform, and over 30 optional add-ons for sub-products maintained outside the platform. The product line has been in production since 2005. In 2009, variability modeling was introduced using the tool pure::variants [8], which also provides variability mechanisms: a "family model" representing the source files, and a build system. The C/C++ preprocessor handles fine-grained variations. The binding of variability is mainly static, but the shipped products include a large number of runtime parameters, which can be configured by customers in a semi-static manner (they are normally not changed during normal operation of a component, only in the configuration phase). The codebase has 1.5M lines of C++ (98%) and C code, distributed over 10,000 files with around 14,000 conditional compilation directives. The development can be characterized as follows:

- *Research-driven*: SPLE and variability modeling practices were adopted in interaction with consultants and researchers.
- *Medium-scale*: The feature model has slightly more than 1100 features. The product-line-development involves around 60 software engineers.
- *Mostly platform-oriented*: Core parts of the product line are integrated into one platform. Customer-specific artifacts (sub-products) exist outside.
- *Extractive*: The product line and the feature model are results of a migration of existing individual products originating from a clone-and-own approach.

Modeling Practices (RQ1). Variability is modeled using a single centralized feature model. Features are mapped to code using a family model. Building, maintaining, and evolving the feature model is under the control of one expert:
B: *We have a colleague who [...] really has the domain knowledge, because he took care of all the development [...]. He consults with the other development teams. [...] So we try to have one place, or one person that is responsible. But then it's not the case that he decides all the things. So, whenever we have an issue, we try to organize a workshop or a meeting [...], it's actually his responsibility to make [sure] [...] that it's correct.*

The modelers focus on building the hierarchy (child-parent) relationship between features and try to avoid cross-tree constraints; few exist in the model. However, they begun adding custom relationships, such as "recommended". The latter often indicates bug fixes, which are actually modeled as features, since not every customer has an interest in enabling them. Some exploit the "invalid" behaviors in their applications and prefer to keep them without fixing.

The variability models are under-constrained. Dependencies among source files are not modeled in the family model, which could be used by pure::variants to verify configurations. Instead, the company finds it easier to maintain tested configurations of its twelve main products instead of exhaustively modeling all constraints. The few dependencies used are primarily binary "requires" and "excludes" relations. There are no numeric or string constraints in the feature model. Instead, complex constraints are put into the feature-to-code mapping as presence conditions of source files. This strategy is interesting, as it reduces constraints both in the variability model and the family model, which contains no dependencies at all. The team strives to keep all models simple.

The hierarchy of the feature model is reasonably well balanced. The engineers avoid deep trees and consider a maximum depth of three or four levels reasonable. Yet, problems with finding the optimal grouping of features occur in some cases:
B: *Then actually it becomes too flat somehow. So, it's a question of how to group them. We're still working on the optimal way. But I think four, that's really the maximum. We don't have really like huge trees over there.*

The resulting model has around 1100 features. Evolution of it is mostly limited to adding features. Feature removal occurs within rare, but important, clean-up tasks. The hierarchy is also relatively stable without any major refactoring. The overall growth rate of the model is estimated at around 5–10% per year, with up to 50 new features per release, 3–4 times a year. Versioning is considered orthogonal to modeling, so models are versioned but not features (i.e., no multiple temporal versions of features in the model).

Benefits (RQ2). Our interviewees emphasize the organization of knowledge and the visualization of variability as the main benefits. Naming and organizing features makes them visible and accessible to developers, encouraging reuse:
B: *The first one is that it's visible, you see the features that you had in the code, before, and actually you see the features of the whole product line. Before, they saw features of the specific products. And then there was a process to make sure that the new features were propagated to the rest of the product.*

Q: So you know what's common?

B: *This, and actually now you can see them. I think the best is you can see relationships, to actually know what configurations are allowed and what are not allowed. That was also not so easy to express in the past [...] This is from the developer's point of view. But it's also, we can see that from the, say project development, it's also important, because before we noticed that the same functionality was implemented twice within the same project, basically they haven't realized that. They implemented the same features.*

Q: Because it was not visible?

B: *Yes, exactly. So it's not only from the development point of view; now you can somehow understand the code easily; you can see the dependencies between the features; you know actually how this code works. And there's also documentation that is attached to the modeling. So you can generate documentation automatically.*

Although our interviewees could not estimate increased productivity quantitatively, they claimed substantial quality improvements by employing SPLE and feature modeling, that it reduced the number of critical bugs significantly. They also claimed that time-to-market was significantly reduced due to automated product derivation. Interestingly, the organization presently strives for further automation by linking features to portfolio and requirements models.

Challenges (RQ3). The interviewees expressed three challenges related, respectively, to organization, modeling, and development. First, it is difficult to convince all stakeholders to invest in core assets when the organization has a matrix structure. Such an organization has two opposing forces: those who optimize for short-term revenue by resorting to clone-and-own approaches with less (short-term) development effort, and those who insist on proper SPLE activities to assure revenue and less maintenance effort in the long run, but with higher (short-term) effort:

B: *In a big, big, really big company that has this [inertia], it's easier to enforce things also, because the management can actually push the things. [Our company] is in the middle, it's not very big, it's not a small one. So there's some kind of let's say, maybe not fight, but some kind of*

Q: pushback?

B: *Yes, between the development and the management. And that's actually a challenge, because introduction of a product line requires that there's some kind of organizational structure introduced [...] So people will have to start thinking in terms of developing assets that can be reused and this can be achieved only if you have a group that takes care of [...] domain engineering. And I'd say in the really big companies, maybe they have somehow the will to invest in actually organizing the whole undertaking. Whereas in the companies that are in the middle, it's some kind of a strained situation when we have product development that is really looking at the business and economy point of view, and then we have the technology people, or the part that are developing the product, that are pushing really for doing things the right way. And then the management is somehow in between. [...] Because we have to earn money, so business tells them we have to earn money. But we cannot do it the way we do it.*

The second challenge concerns modularization of the model. It was difficult to find a good structure when trying to separate product-specific features. Both common and product-specific features exist, as well as commonalities between these two groups. It is also possible to have many different combinations between the groups. Thus, the result would have been an intricate model.

The third expressed challenge concerns the high amount of conditional compilation directives in the code and the additional variability model layer that developers need to take into account:
B: *I think the biggest problem is that the developers are used to working for a long time on the same abstraction level, basically text. Now somehow we introduce a concept, a new way of working, because they cannot just, for example, merge everything at the source code level. They also have to think about models [...] So whenever they add a feature, they have to add the feature to the model. So later whenever they merge back the integration branches, they have to merge all the artifacts. They just have to learn about the modeling part. But I think the modeling in pure::variants, I think the way they realize that is a big advantage. Because in the past, they did it all in source code. So you had a huge header file with features enabled and disabled. The dependency was hardly expressed.*

4.3 Car Manufacturer

The third subject is a very large ($\leq 150,000$ employees) car manufacturer producing over 400,000 cars per year. From three main platforms, an estimated number of three million different car models can be derived. Our interviewee is a software architect who was involved in modeling and managing variability. The product-line engineering comprises two major activities: *development* of car components and *manufacturing* of cars. Our focus is on the development, which uses features to capture variability. Features are mapped to hardware and software article numbers to facilitate the manufacturing. The software is mainly written in C, with a few exceptions, such as the infotainment system relying on C++ and Java. Variability mechanisms comprise component composition, conditional compilation with the C preprocessor, and dynamic adaptation at car startup using configuration options. The latter allow finer-grained variations, while features are generally coarser-grained. The development can be characterized as follows:

- *Practice-driven*: Software variability-management strategies are an adaptation of the mechanical manufacturing processes, which have evolved over decades.
- *Large-scale*: There are three main vehicle software platforms. 50–100 teams constantly interact with one another on individual subsystems of the platform.
- *Multi-level modeling*: Three levels of feature models are maintained in the company, each level facing different dynamics and governance.
- *Heterogeneous modeling*: The company uses diverse modeling approaches, including behavior modeling (Simulink) and structural modeling using a specific subset of UML (Sparx Systems Enterprise Architect).
- *Pro-active*: Product-line engineering was adopted from the beginning. Single-system development was infeasible due to the huge diversity. The current platforms are the result of a slow evolution over 15 years.

Modeling Practices (RQ1). Feature models are used on different organizational levels to describe the variability of the in-car software. All models are stored in a database, the TeamCenter product lifecycle management tool [2]. Before that, Excel was used. Each of the three platforms has a top-level model describing the "complete vehicle level" with around 300–500 features. Most of these are customer-visible features with a few exceptions, such as "remote diagnostics". The top-level model is built and maintained by a central group in the company. Features are refined into lower-level models to a maximum of three levels. For instance, the infotainment system has an "intermediate" level with 700–800 features and a low-level model with up to 3000 features. For other subsystems, fewer levels suffice, such as the chassis system with two levels.

Often, just a superset of the actual variability is modeled: finer-grained variability is realized by configuration options or via dynamic adaptation. Thus, the feature modeling concepts used are very simple, without any strong formalization. Structural grouping of features according to functionalities exists, such as for chassis, powertrain, or comfort features in the top-level model. Features are tagged as optional or mandatory, together with information about their relevance for marketing purposes. Only Boolean features exist; more types, such as enumerations (up to ten values) and integers, occur in the dynamic configuration options managed separately from features. Neither feature groups (OR/XOR, Fig. 1) nor cross-tree constraints are modeled. Although many exist, they are only documented informally or contained in the manufacturing database. Likewise, the mapping between features and software components or other models is only informally documented. Checking constraints does not play a role in development:

C: *We do that check in the manufacturing though, because we have a lot of constraints. I mean, two physical things can't occupy the same place physically. So, for example, if you have an engine of this size, there are things that you cannot have because it's so big. And that type of constraints we have, or checks, we have in the manufacturing. And we also do that, the same thing to software, for example, this software article is not compatible with that one. But, we don't do that a lot in development, but in manufacturing.*

The different model levels face different governance and evolution. The top-level is very stable, with updates only occurring at specific "update" events twice a year. There, features are primarily added. Old features are removed, but usually directly replaced with new ones. The low-level models are highly volatile. For instance, the infotainment subsystem changes almost weekly.

Given all of these large-scale practices, the company never aimed for a configurator-based approach to facilitate more automated derivation processes. The latter is partly handled by a home-grown manufacturing tool, which combines hardware and software article numbers during manufacturing, while adhering to dependencies. Thus, the prime reason for variability modeling is the management of variability, which does not require more formal modeling techniques.

Benefits (RQ2). Our interviewee sees the largest benefit of variability modeling at the requirements level: in scoping products, understanding configuration spaces,

maintaining development overviews, and fostering communication across teams. Further benefit lies in marketing and coordination of new model releases:
C: *I would say the most important purpose is to agree between the R&D organization and with the product planning organization over the content of each product. And based on that, you, what I'd say, you break down, or derive the requirements on each subsystem to realize these features. In most cases, these features are realized by several subsystems co-operating.*

Our interviewee expressed a neutral opinion about the value of variability modeling in our previous survey. According to him, feature modeling in its simplest form, as practiced in the company, provides the mentioned benefits, but provides little assistance with product configuration and derivation activities.

Challenges (RQ3). Our interviewee sees the biggest challenge in organizational and cultural issues among heterogeneous teams. While he expressed some issues with code, his focus as a software architect was primarily on the organizational level, where he is concerned about interaction and efficient use of modeling:
C: *We have a lot of dependencies between subsystems and between teams, so it's quite difficult for the teams to work autonomously [...] I think there is an inherent complexity, because the number of interfaces is also great [...] If we look at the present processes at [...] when it comes to modeling, it seems like we're aiming to [...] keeping practices, which means that we're trying to align the modeling efforts between different domains, we try to align the design artifacts that we are using, and so on. And [we focus] on keeping the traceability between the different kinds of artifacts that we use—the feature models, the software in itself, the different [...] component models, AUTOSAR component models, our design model, our architecture model, and so on. So we put a lot of effort in maintaining all these design artifacts in a consistent way [...] My personal opinion is that I don't think that's the right way to go, because since the complexity of our systems is exponentially increasing [...] we actually need to identify ways of working such that different development teams can work more autonomously, that they can use the tools they need for their specific problems [...]*

In summary, our interviewee is concerned with handling the many dependencies between subsystems and establishing a harmonized collaboration between teams. While he observes that processes are heading towards textbook practices that strive to unify the current diversity in modeling approaches and introduce coherent traceability, he would prioritize autonomous teams over a unified architecture, which is increasingly complex due to a high amount of dependencies. The whole development might become even harder to manage with increasing effort spent on maintaining traceability and explicitly modeled dependencies.

5 Cross-Case Analysis

We now conduct a cross-case analysis and discuss commonalities and differences across cases. After summarizing the context in which variability modeling is performed, we compare practices (RQ1), benefits (RQ2), and challenges (RQ3).

Our cases applied variability modeling in very different contexts. The *consulting company* used feature modeling and SPLE as known from the literature, using the original graphical notation, a configurator, and a generator that resolves variability in an automated process. Although these web shops generated revenue, the project was a means to experiment and to gain expertise in variability modeling. For the *component producer*, feature modeling and SPLE was a core strategy to conquer complexity and maintenance issues stemming from a previous clone-and-own approach. Their practices originated from a close collaboration with researchers and the vendor of their modeling tool. The company was open to adopt solutions from research and saw the value of the solutions in lower time-to-market and increased code quality, but faced friction between demands for short-term revenue and the necessity of systematic variability management for long-term advantages. The *car manufacturer* applied feature modeling at a much larger scale and with simpler modeling concepts than both previous subjects. Practices originated from an engineering culture that had evolved over decades. While the other subjects strive for higher automation, unification, and integration of modeling, the reported experience suggests that textbook approaches might not work, or their effort might outweigh any potential benefit to this organization.

Practices (RQ1). *Limited constraint modeling:* Our most surprising finding is that all subjects avoid modeling constraints. This observation is in contrast to our previous observations in systems software, where detailed constraints are formally defined in rich languages. However, while configuration in our subject companies is performed by only a few knowledgeable domain experts, the systems software projects are configured by a large number of third-party users in ways that are not closely controlled by the platform developers. The latter setting requires constraint modeling, choice propagation, and conflict resolution facilities in order to guide users to correct configurations and prevent incorrect ones. These facilities do not seem to be essential in the context of our subjects.

Centralized model governance: Variability models need to be controlled centrally. While in the consulting company, the total development team was too small to draw any conclusion, the other subjects apply strict governance of either the whole model (component producer) or the top-level model (car manufacturer). Either one expert or a central team control the evolution and maintenance of the model. Interestingly, this observation confirms Hypothesis 1 in our study [4] on variability mechanisms in software ecosystems.

Furthermore, larger organizations, such as the car manufacturer, even require clear responsibilities per feature: it has to be defined, specified (e.g., by writing use cases) and developed by a dedicated entity in the organization.

Pragmatic versioning: We have not found any sophisticated support for versioning. One could, for example, imagine specific modeling elements for deprecated or experimental features, version annotations, or constraints over versions of features. Instead, the component producer uses an ordinary version control system for the whole model, and the car company applies a pragmatic solution: features have a unique identifier capturing lifecycle information. This approach, however, would lead to highly redundant constraints when modeled among features.

Domain knowledge in feature hierarchies: Our first subject built the feature hierarchy using domain knowledge. This indicates that hierarchical relations between features in fact represent domain-specific, ontological relationships. This observation supports insights from our previous work on reverse-engineering feature models [36,30]. Thus, feature models contain unique ontological information, and building a feature hierarchy can hardly be automated and will have to be done by domain experts in a largely manual effort.

Top-down and bottom-up creation of models: All our subjects obviously needed some amount of top-down knowledge and analysis. The first two subjects also used the code of existing products to identify features. Thus, we believe that the creation of models that are used to configure products will be often created by a mixture of top-down and bottom-up approaches.

Benefits (RQ2). *Organization of knowledge:* The most important benefit of variability modeling, emphasized by all interviewees, is the organization of knowledge. This benefit resembles perceived benefits of MDD [24]—companies appreciate the potential of MDD to ease communication and to overview the development.

Visualization and scoping. All interviewees appreciate the visualization and product-scoping capabilities of feature models. Most of them were able to better understand product functionalities. The consulting company found it beneficial to see which customers have which features. The component producer was even able to identify duplicate implementations of certain features.

Configurator support: The insignificance of configuration in the three industrial cases is surprising, which is in contrast to our previous study of systems software (Sect. 2). We conjecture cost/benefit considerations. Configurators require formally declared constraints and, according to our previous experience, also proper usage of different constraint types (e.g., configuration constraints, visibility constraints, default constraints) to leverage a configurator. Such effort would not pay off for our subjects, and even with intelligent configurators, users can spend substantial time configuring products when intricate constraints exist [23].

Challenges (RQ3). *Mindset changes:* Introducing variability modeling requires mindset changes of all actors. As can be seen from the component producer, developers commonly think at one abstraction level and struggle with trying to maintain features (at another abstraction level). Even in the consulting company, where the developers had bought into modeling, both struggled with developing application code and infrastructure code in parallel. The situation for the car manufacturer was different, however. SPLE was an adaptation of mechanical-engineering practices that have a long tradition; thus, developers always understood that single-systems or cloning-based development is infeasible.

Short-term versus long-term benefit: SPLE requires discipline from all actors—specifically, to consistently co-evolve models and code. In a matrix organization, there is a higher risk of conflict between proponents and opponents of systematic variability management. Actors who strive for short-term revenue might fall back to clone-and-own for product derivation, leading to high maintenance in the future, as the clone requires maintenance. Establishing a culture for systematic management and modeling is a core challenge.

Evolution: All subjects primarily add features and seldom remove features or restructure the hierarchy. The consulting company mentioned that evolution was challenging as it requires understanding the feature-to-code mapping and the impact of feature changes, to avoid breaking the system. The car manufacturer expressed concerns about exploding complexity of their development, but not specifically about the (simple) models. Although the company strives for increased commonality of the software for all car models, whether this effort will affect features or only the finer-grained configuration options remains an open question.

6 Threats to Validity

External validity. Our findings originate from only three cases. However, we do not attempt to reach any statistical generalizations from the data, but describe substantial cases in their full richness. In fact, case-study research does not aim at representativeness, which is impossible to assess since the whole population of cases is usually unknown. Our selection of cases is based on theoretical sampling [17]. We chose them according to three criteria (Sect. 3) among all of our subjects. A limitation of our study is that all subjects successfully applied variability modeling. Studying failed attempts would be valuable future work.
Internal validity. Our findings rely on interview data, since no other datasources (e.g., artifacts) were available. We interviewed actors centrally involved with variability modeling. Still, triangulating our results with data gathered using other methods, such as action research or ethnographic field studies, would be valuable future work. Interestingly, the practices of the car manufacturer correlate with our experiences with another car manufacturer of similar size, improving our confidence in the results. Last, the interview data could be biased due to leading or misphrased questions. We did pre-tests and carefully analyzed the transcripts, omitting responses that indicated uncertainty. The consulting company interview was done in German. We carefully, almost literally, translated it.

7 Conclusion

We have provided empirical data on variability modeling in successful industrial applications. The reported experiences show that feature models are perceived as intuitive and simple notations that organize unique domain knowledge and foster understanding and collaboration among developers. Many practices are pragmatic, such as versioning, the mix of top-down and bottom-up modeling, central model governance, or the very limited constraint modeling. Interestingly, instead of declaring and maintaining constraints, our subjects prefer to manage a set of configurations or to let experts configure products. Thus, the primary benefit of variability modeling lies in variability management—organizing, visualizing, and scoping features—less in configuration and automation for our subjects. Yet, the benefits require acceptance of an additional abstraction level and discipline in maintaining models. Otherwise, long-term advantages can be compromised for quick revenue, which we found is especially a problem in matrix organizations.

References

1. Product Line Hall of Fame. http://www.splc.net/fame.html, accessed 03/2014
2. TeamCenter. http://www.plm.automation.siemens.com/en_us/products/teamcenter/, accessed 07/2014
3. Bednasch, T.: Konzept und Implementierung eines konfigurierbaren Metamodells für die Merkmalmodellierung. Master's thesis, Fachhochschule Kaiserslautern (Oct 2002)
4. Berger, T., Pfeiffer, R.H., Tartler, R., Dienst, S., Czarnecki, K., Wasowski, A., She, S.: Variability mechanisms in software ecosystems. Information and Software Technology (2014)
5. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wąsowski, A.: A survey of variability modeling in industrial practice. In: VaMoS (2013)
6. Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K.: A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering 39(12) (2013)
7. Berger, T., She, S., Lotufo, R., Wąsowski, A., Czarnecki, K.: Variability modeling in the real: A perspective from the operating systems domain. In: ASE'10 (2010)
8. Beuche, D.: pure::variants Eclipse Plugin (2004), user Guide. pure-systems GmbH. Available at http://web.pure-systems.com/fileadmin/downloads/pv_userguide.pdf
9. Chen, L., Ali Babar, M.: A survey of scalability aspects of variability modeling approaches. In: SCALE (2009)
10. Chen, L., Ali Babar, M., Ali, N.: Variability management in software product lines: a systematic review. In: SPLC (2009)
11. Chen, L., Ali Babar, M., Cawley, C.: A status report on the evaluation of variability management approaches. In: EASE (2009)
12. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2001)
13. Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wąsowski, A.: Cool features and tough decisions: A comparison of variability modeling approaches. In: VAMOS (2012)
14. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston, MA (2000)
15. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software Process Improvement and Practice 10(1) (2005)
16. Dhungana, D., Grünbacher, P.: Understanding decision-oriented variability modelling. In: ASPL (2008)
17. Eisenhardt, K.M., Graebner, M.E.: Theory building from cases: Opportunities and challenges. Academy of management journal 50(1), 25–32 (2007)
18. Gheyi, R., Massoni, T., Borba, P.: Automatically checking feature model refactorings. The Journal of Universal Computer Science 17(5), 684–711 (2011)
19. Gillan, C., Kilpatrick, P., Spence, I., Brown, T., Bashroush, R., Gawley, R., et al.: Challenges in the application of feature modelling in fixed line telecommunications. In: VaMoS (2007)
20. Glaser, B., Strauss, A.: The discovery of grounded theory: Strategies for qualitative research. Aldine de Gruyter (1967)
21. Grünbacher, P., Rabiser, R., Dhungana, D., Lehofer, M.: Model-based customization and deployment of Eclipse-based tools: Industrial experiences. In: ASE'09 (2009)

22. Hubaux, A., Classen, A., Mendonça, M., Heymans, P.: A preliminary review on the application of feature diagrams in practice. In: VaMoS'10 (2010)
23. Hubaux, A., Xiong, Y., Czarnecki, K.: A user survey of configuration challenges in linux and ecos. In: VaMoS (2012)
24. Hutchinson, J., Rouncefield, M., Whittle, J.: Model-driven engineering practices in industry. In: ICSE (2011)
25. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of mde in industry. In: ICSE (2011)
26. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (Nov 1990)
27. Krueger, C.W.: Easing the transition to software mass customization. In: PFE (2001)
28. van der Linden, F.J., Schmid, K., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer (2007)
29. Loesch, F., Ploedereder, E.: Optimization of variability in software product lines. In: SPLC (2007)
30. Nadi, S., Berger, T., Kästner, C., Czarnecki, K.: Mining configuration constraints: Static analyses and empirical results. In: ICSE (2014)
31. Passos, L., Novakovic, M., Xiong, Y., Berger, T., Czarnecki, K., Wasowski, A.: A study of non-boolean constraints in variability models of an embedded operating system. In: FOSD (2011)
32. Pohl, K., Böckle, G., Van Der Linden, F.: Software product line engineering: foundations, principles, and techniques. Springer-Verlag New York Inc (2005)
33. Reiser, M., Tavakoli, R., Weber, M.: Unified feature modeling as a basis for managing complex system families. In: VaMoS (2007)
34. Riebisch, M., Streitferdt, D., Pashov, I.: Modeling variability for object-oriented product lines. In: ECOOP'03 Workshop Reader (2004)
35. Schmid, K., Rabiser, R., Grünbacher, P.: A comparison of decision modeling approaches in product lines. In: VaMoS (2011)
36. She, S., Lotufo, R., Berger, T., Wąsowski, A., Czarnecki, K.: Reverse engineering feature models. In: ICSE (2011)
37. Software Engineering Institute: Catalog of software product lines. `http://www.sei.cmu.edu/productlines/casestudies/catalog/index.cfm`
38. Steger, M., Tischer, C., Boss, B., Müller, A., Pertler, O., Stolz, W., Ferber, S.: Introducing PLA at bosch gasoline systems: Experiences and practices. In: SPLC (2004)
39. Sugumaran, V., Park, S., Kang, K.C.: Software product line engineering. Communications of the ACM 49(12), 29–32 (Dec 2006)