A Case Study of Two Configurable Software Product Families

Mikko Raatikainen, Timo Soininen, Tomi Männistö, and Antti Mattila

Software Business and Engineering Institute (SoberIT)
Helsinki University of Technology
P.O. Box 9600, FIN-02015 HUT, Finland
{Mikko.Raatikainen, Timo.Soininen, Tomi.Mannisto, Antti.Mattila}@hut.fi

Abstract. A configurable software product family allows the deployment of individual products without customer-specific design or programming effort. Despite the fact that such software product families have recently gained research interest, there are only few empirical studies on them. This paper presents some results of a descriptive case study undertaken in two companies that develop and deploy configurable software product families. The similarities found in comparisons between characteristics of the configurable software product families were remarkable, although the companies, products, and application domains were different. The study shows that the configurable software product family approach is already applied in the industry. Furthermore, the approach seems to be a feasible and even efficient way to systematically develop a family of products and manage the variability within it.

1 Introduction

The *software product family* approach aims at producing a set of products that are developed on a basis of a common *software product family architecture* and a set of *shared assets* [1]. The shared assets and the software product family architecture are developed for reuse in a *development process* and reused to build products in a prescribed way in a *deployment process* [2], [3]. One class of such software product families is the configurable product base [4], for which we use the term *configurable software product family* (CSPF), meaning that at least a significant part of deployment is performedby *configuring*, for example, by setting parameter values or by selecting components or modules without programming.

The CSPF approach is considered in [4] the most systematic software product family approach and is appealing because it seems to promise an efficient and systematic deployment of products. However, there are only a few reported empirical research results on the nature and benefits of the CSPF approach, the established methods and practices for such an approach, and the problems encountered when applying the approach.

We have carried out a descriptive case study in five Finnish companies in order to characterize the state of the software product family approach in practice [5]. In the study, we found two companies that apply the CSPF approach. In this paper, we describe how the two companies apply the CSPF approach from the points of view of the characteristics of, the adoption of, the variability within, the deployment process of, and the configuration-tool support for a CSPF, as well as the after-sales and maintenance of products deployed from the CSPF. The results indicate that the CSPF approach is applicable in the industry. There were remarkable similarities between the companies, although the CSPFs were developed independently from each other, and for different application domains.

The rest of the paper is organized as follows. In Section 2, we present the research method. In Section 3, the characteristics of the companies and their CSPFs are described, while in Section 4, they are compared with each other. In Section 5, the results are discussed particularly in the light of the characteristics of the CSPF approach and the factors influencing its feasibility. Section 6 provides a comparison of the results to related work, while, in Section 7, the validity and reliability of the results are discussed. Finally, Section 8 draws conclusions and presents some ideas for future work

2 Research Method

The research design was a descriptive case study [5] undertaken in a number of companies. The objectives of the study were to characterize the state of the software product family approach in Finnish industry in practice. For the study, we looked for companies that potentially had a software product family.

To form an initial theory for the study, i.e., to gain a preliminary understanding of the software product family approach, we conducted a literature study of previous research results. Before starting the interviews, we developed a set of mostly open-ended and qualitative interview questions. The structure of the interviews was broken down into business, artifact, process, and organizational (BAPO) concerns, similar to the one described in [6], but we replaced the word 'architecture' with the word 'artifact' in order to emphasize that the term referred to several kinds of assets in addition to only software architecture. The BAPO analysis framework was used for triangulation purposes in order to find out different points of views on the software product family approach.

Data was collected in an interview and additional documentation analysis in each company at the turn of the year 2002, in a validation session a few months later, and through enabling the responders to read and comment on the reports we had written, including this paper.

In Company A, we interviewed a product manager, who was present when interviewing on business and process issues for about two hours, the manager of the deployment process, who was present for about half an hour when interviewing on deployment process issues, and a software engineer responsible for developing the CSPF, who was present for about two hours during the interview on artifact-specific issues. In addition, the chief architect was present all the time.

In Company B, we conducted the interview in two sessions. In the first session, we interviewed the architect about the business issues for a half an hour, the quality manager about the primary development process and organizational issues for one hour, and finally the architect, again together with a software engineer, about artifact-specific issues for two and a half hours. In the second session, we interviewed a business area manager about business issues and the deployment process for two hours.

In the interviews, we skipped some questions that seemed unimportant for the particular company, changed wordings and the order of the questions, and asked additional clarificatory questions, questions that deepened the topic under discussion and summarized comments into question format to generalize responses and enable the responders to correct our misinterpretations and misunderstandings. We tape-recorded the interviews and took notes, and later transcribed the interviews.

A few months later, we held an about 3-hour validation session in each company. In these sessions, we presented our findings to the company and asked questions to clarify issues that had remained unclear. We tape-recorded validation sessions.

We analyzed data from each company, first separately, and then across cases [7]. The analysis was based on a discussion of the impressions of researchers who were present at the interviews, as well as on the recordings, received documentation, written notes and transcripts of the interviews. The notes from the validation session were added afterwards. In the analysis, we used ATLAS.ti [8], which is an application designed for qualitative data analysis. The results of the analysis were validated in the validation sessions and by enabling the responders to read this report. For further details on the research method, see [9].

3 Companies

In this section, the characteristics of the companies are described first in Section 3.1 and their CSPF are illustrated in more depth in terms of the characteristics of the CSPF - variability, deployment, configurator, and after-sales and maintenance - in the following two sections.

3.1 The Companies

In Table 1, the major characteristics of each company and its products are given. The application domain refers to the application domain for which the CSPF is developed. In fact, Company A also operates and develops products for other closely related domains, although the focus is on the CSPF, whereas Company B operates only in the medical information management domain and does not develop products other than the CSPF. The number of employees and software engineers is the total number of people working within the companies. In Company A, several of the employees also work with other products, but most of the time they work with the CSPF studied; this is especially true of the software engineers. The type of CSPF refers to the kind of product the CSPF is. The operating environment of the CSPF is the software platform that a delivered product based on the CSPF installs and operates when in use. Total size of code in the CSPF means the total number of lines of code that is associated with shared assets in the CSPF; each delivery contains all or part of the assets and, in addition, a possible small amount of product specific code. The programming language refers to the programming languages that the companies have used in developing the CSPF.

Table 1. Characteristics of the companies

	Company A	**Company B**
Application domain	Factory automation	Medical information management
Employees	200	130
Software engineers	25	35
Type of CSPF	Software augments electronics and mechanics	Software product
Operating environment	PC, MS Windows	PC, MS Windows
Total size of code	0.5 MLOC	Over 1 MLOC
Programming language	Visual basic, C++	C++

3.2 Company A

General Characteristics Company A produces factory automation *products*. The company delivers about 50 products a year and assumes the number of annual deliveries stays relatively stable in the long term. The product is an investment good that is used for years or even decades. The company has decades of experience in the application domain as it has developed products for the same domain.

Before the CSPF approach, similar products were delivered in projects for each customer separately and each project used the source code and other deliverables of earlier projects non-systematically copy-pasting as a basis for a new delivery. In the mid 1990's, Company A realized the opportunity to improve the effectiveness of the deliveries by systemizing the reuse of common parts between the projects; following this through, they ended up developing the CSPF. Another objective during the initiation of the CSPF approach was to systemize variability to a generic configurable product that enabled efficient deployment. At the time of the initiation, Company A had some experience of developing a software product family because a similar approach had been applied to other products in the application domain as well.

A delivered product based on the CSPF is integrated to work with existing systems in the customer-environment, intermediating between, managing, and cooperating with the systems. Depending on the customer's order, a product includes a varying amount of features so that some tasks in the factory can be performed automatically using the system. A delivery includes mechanical and electronic systems that are controlled by software derived from the CSPF. The application domain of the CSPF is well understood and relatively stable. The interfaces in the application domain are relatively stable and to some extent standardized.

The employees in the software development department (Figure 1) have roles according to the domain engineering model [1] in which the work is divided between those developing the software product family and those doing the deployment projects. According to the responders, a few employees work in both roles. Both developers and deployers are all located close to each other in the same office premises.

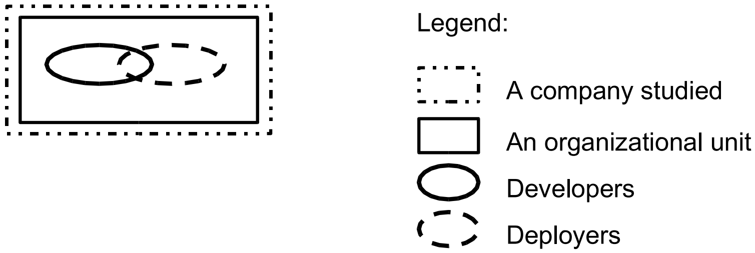

Fig. 1. Software development organization in Company A

The responders estimated that Company A has been able to double the number of product deliveries in a year with roughly the same number of employees as a result of adopting the CSPF approach. Furthermore, the CSPF approach was seen as the only reasonable way to develop products. They even went as far as to state that the CSPF approach is the only reasonable way to do the business, despite the initial investment that was needed for building the CPSF. However, the company had not done a more precise analysis of, for example, the return on the investment of the CSPF.

Variability Variability in the CSPF is achieved by *configuring* and *tailoring.* For Company A, configuring means setting parameter values without developing or modifying the source code. Tailoring, on the other hand, involves modifications that require developing new source code or modifying the existing source code.

The company divided configurable variability into *vertical* and *horizontal* variability. Vertical variability concerns the existence of *major functions*, which are large entities adding significant functionality to a product. In total, a product can contain about a dozen major functions that can enable some tasks in the factory to be performed automatically using the system. Horizontal variability typically involves adapting products to the customer-environment and preferences ranging from system-level features, such as the type of alarm when something goes wrong in the factory, to detailed issues, such as the numeric values specifying the length and width dimensions of the physical environment.

Vertical variability (Figure 2) is organized into five stacks, each containing a few major functions and a *base* that contains major functions common to all products. A customer can select the major functions needed for her purposes from the stack, while the base is included in all products. The order of the major functions in the stacks specifies how a selection can be done: The functions are selected such that all functions from the lowest to the desired level in the same stack have to be selected, although there may not be holes in the stack, resulting in one function not being selected, while a function above it is selected.

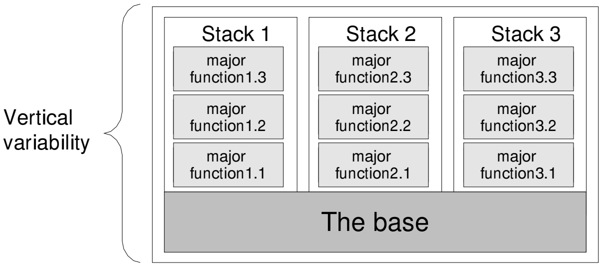

Fig. 2. The vertical variability stacks in Company A. The actual number of major functions is not exactly the same in the figure as in the CSPF studied

Tailoring typically involves introducing new interfaces to exchange data with, or report data to, another system; it rarely adds new features or modifies the existing features. Typically, tailoring is accomplished by adding new modules to the CSPF. Therefore, even if a product is tailored, the product is, and has to be, configured, as the configuring affects the parts of the product that are not changed in tailoring.

All deployed products are, in practice, based on the same source code. Parameter values specify what parts of the code are executed and which of the major functions or features are enabled.

The responders pointed out that there is a risk that customers might require more tailoring, beyond the scope achievable by configuring, and that this might lead to large scale changes to the CSPF that may, in turn, lead to further problems in the maintenance of the CSPF. However, such a problem had not occurred so far.

Deployment In a deployment process (Figure 3), a product is first tailored if necessary, after that, auxiliary software and the product are installed on a PC. After the product is installed, it is configured by changing parameter values using a specific configuration tool. The product is tested with its mechanical and electronic systems integrated in order to ensure that it works as required; finally, it is shipped to the customer.

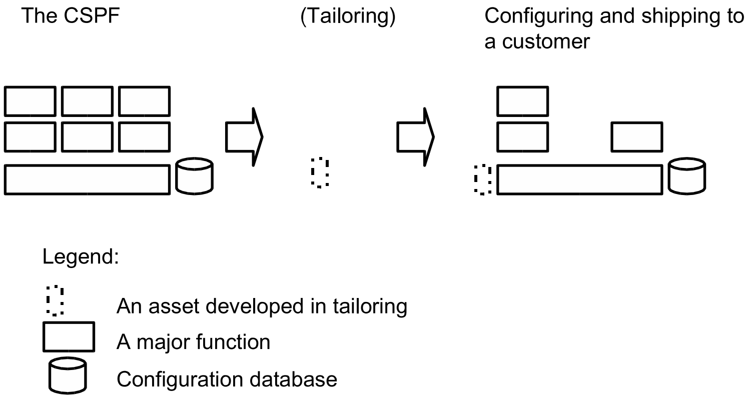
The CSPF
(Tailoring)
Configuring and shipping to a customer
Legend:
An asset developed in tailoring
A major function
Configuration database

Fig. 3. Deployment process in Company A. The number of major functions in the figure is not exactly the same as in the actual CSPF

The parameter values are initially set to default values. The responders estimated that the number of parameter values set in a deployment is typically about 200, although, as a minimum, the number may be closer to 100. A checklist that contains guidelines as to which parameter values have to be checked in configuring is used to ensure that all necessary parameter values are checked and set.

Tailoring is performed in addition to configuring in the case of only approximately 30% of the delivered products. Tailoring is usually very simple and routine and takes from a few days to a few weeks for two software engineers.

The responders stated that an employee who has experience on deploying products based on the CSPF sets most parameter values correctly in deployment and the dependencies between parameter values, for example, are clear to her from the context. Configuring needs primarily deep application domain knowledge, but hardly any software engineering knowledge. In fact, the employees who deploy products have their background in automation, not in software engineering. The responders estimated that a new employee needs at least three months of experience in the application domain and with the CSPF before she has a good enough understanding to be able to configure the system correctly by herself.

Deployment takes roughly one person-day, half of which is spent on configuring. Even with tailoring, the software part of the delivery typically takes such a short time that software development is not a bottleneck in the total process, which also includes mechanical and electronic manufacturing processes. The total process takes from weeks to a few months.

Configurator The *configurator* is a software application that Company A has developed for configuring. The configurator is used to resolve horizontal and vertical variability by setting and storing the parameter values in a configuration database. The configurator has a graphical user interface.

The configurator is developed for a particular CSPF and is not usable in any other CSPF without reprogramming; this is because it includes parts that were specific to the CSPF such a configuration database scheme and checks to ensure that parameter values were set correctly according to rules that were embedded into the configurator. The configurator checked the correctness of some parameter values and their combinations but only for a small subset. Thus, the checks that the configurator carries out do cover the correctness of some, but not all, parameter values and therefore it is possible to specify a non-working product using the configurator. A configuration explanation, which is a textual description in natural language, is attached to each parameter to describe, for example, what kind of effect the parameter has on the product, what the possible parameter values are and what the parameter values mean. The user interface for setting parameter values is developed using typical graphical user interface components such as text fields, combo-boxes and radio buttons.

Configuring in the configurator is done by setting optional values, selecting alternatives from a set of alternatives, or specifying numeric values. To enable an optional parameter value means that, for example, some feature exists in a product. For alternative parameter values, there are typically about five possible values; selecting an alternative means, for example, selecting one of a set of similar features for use. For numbers, there is a field where a value, such as an integer, has to be entered and an allowed range for the number. A numeric parameter can, for example, specify the physical length and width dimensions of the physical environment of the facility where the product is located.

After-sales and maintenance The typical customer buys a product and then uses it without changes for years. Updates to newer main versions of the CSPF are made in some cases, but it is relatively rare. In fact, Company A does not see selling new features to old customers as a viable business model. A more typical product update for customers is made if a serious bug is noticed in the CSPF and the update includes a fix for the bug. Updating, either to a newer version or as a bug fix, is easy if a product is not tailored, because in that case all customers have the same code. Company A does not guarantee that it is possible to update a tailored product, at least not with as low a price as non-tailored ones. However, tailoring typically does not change a product to such an extent that it could not be easily updated if necessary. Nevertheless, by requiring tailoring, a customer may complicate updating her software.

A product can also be *reconfigured* later. Reconfiguring means that a parameter value is set to a different value, when, for example, something in the customer-environment changes. However, reconfiguring is only rarely carried out. Customers could, in principle, reconfigure the product by themselves, but the configurator is password-protected. Company A, does not give access to the customer, probably for both business and technical reasons, such as wishing to earn money from reconfiguring and wishing to prevent customers configuring a product improperly.

3.3 Company B

General Characteristics A *system* delivered by Company B consists of several *products* for the medical domain. Dozens of systems are sold each year and each customer is typically sold several similar systems. The customers use the systems for years. Company B was founded in the early 1990's for emerging markets, and expects these markets to continue to grow.

The CSPF was developed from scratch when the company was founded in the early 1990's. From the very beginning, Company B had as an objective the development of a generic configurable system that is deployable efficiently and without source-code modification. Since its initial development, the CSPF has been further developed and new features have been added.

A delivered system based on the CSPF presents, monitors, produces, analyzes, and stores data regarding customer business processes that are safety-critical. The functionality of each product that consists of a system is different and focuses on different aspects of the application domain. The products complement each other such that one, for example, performs analysis for data, one combines data from multiple sources, and one stores data. The products in a system share data, data models, architectural principles, and some other minor assets. The system has interfaces to other systems in the customer-environment, enabling it to exchange data with them. It is also possible to customize the kind of data handled and how is it presented in the system. The application domain is relatively young, but nonetheless quite stable, standardized, and even governmentally regulated.

The organizational structure in Company B (Figure 4) follows the domain engineering model [1]. The CSPF is developed in a development organizational unit, while separate deployment organizational units and partner companies deploy systems. Furthermore, the CSPF development organization is in a single location, whereas the deployment organizations are located in several locations and countries and separated geographically from each other and from the development unit.

Company B has never applied another approach to develop software, and therefore there is no analysis of, for example, the benefits of the CSPF approach compared with some other software development approach. Nevertheless, the responders reported that they saw the CSPF approach as the best way to develop the products in Company B.

Variability A system delivered on the basis of the CSPF is only varied by *configuring*, that is, without developing or modifying the source code. The configurable variability in the CSPF can be broken down into *high-level* and *low-level* variability. In addition, separate products or components that complement the system, but which are separate and not included in the system, might be developed by a deployment organization in a deployment project.

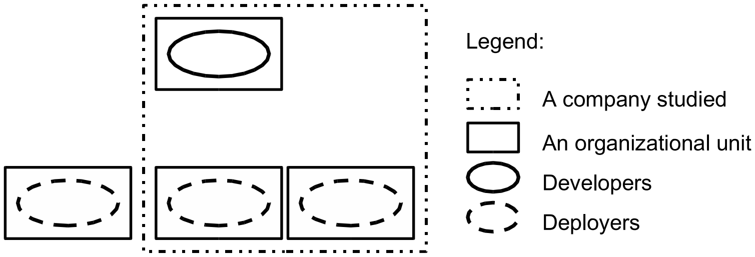

Fig. 4. Software development organization in Company B

At the high level, variability pertains to selecting a particular set of products for a system. In principle, a customer can select any subset of about ten products, but, in practice, combinations are not arbitrary because one of the products forms a basis for the others and is always included in a system. Only the selected products of a system are delivered and installed for a customer.

At the low level, variability pertains to adapting a system to the customer-environment by configuring. Configuring sets parameter values that influence, for example, the kind of external devices a system interacts with, as well as the kind of data and the form in which the data is presented.

The source code in the CSPF is practically never changed for a single customer. If assets in the CSPF have to be changed in such a way that the source code is modified according to a single customer's requirements, then the modifications are immediately put into use for all other customers, as well as for part of the CSPF. Thus there is no customer-specific code, but the CSPF may be further developed to meet customer requirements. However, deployment organizations may develop additional components or products on their own to complement the CSPF. The degree of separation between the development and deployment organizations is such that the development organization does not even necessarily know about the additional components or products implemented by the deployment organization and the deployment organization does not have access to the source code. These components or products, for example, export data, perform some special data handling or wrap an interface in order to be compatible with the existing customer-environment. Therefore, development that resembles tailoring takes place occasionally in deployment projects, even though the CSPF itself is not tailored.

The responders estimated that 30% of CSPF development effort is put into developing configurability. If the CSPF were not to meet customer requirements, the possible effort wasted on developing configurability and the fixed and laboriously changeable scope of the CSPF due to not allowing source code modification, and thus being inflexible compared to competitors, were perceived by the responders as the greatest threats for the success of the CSPF.

Deployment Deployment (Figure 5) begins with selecting a set of products for a system. Then the system is shipped to a customer and installed with auxiliary software. Finally, the system is configured using a configurator tool. A deployment project typically deploys several similar systems for the same customer.

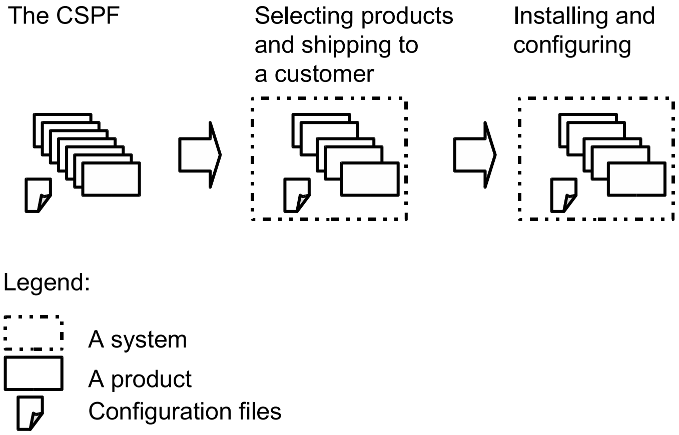
The CSPF
Selecting products and shipping to a customer
Installing and configuring
Legend:
A system
A product
Configuration files

Fig. 5. Deployment process in Company B. The number of products in the figure does not correspond to the actual CSPF.

The company has developed a default configuration in which the parameter values are set to default values in order to avoid the need to set all parameter values from scratch. Despite the default configuration, in order to specify the correct configuration for each customer, hundreds or even thousands parameter values still need to be set.

Developing additional components or products in a way that resembles tailoring as noted earlier is not, strictly speaking, included in the deployment project, but they are developed in a separate project.

Finding a correct configuration relies on application domain knowledge and testing the configuration. Deployment projects include application domain experts and experts with computers, such as system administrators, but not necessarily software engineers who develop software.

A deployment project takes from a few days to a few weeks for two experts and often includes not only installing and configuring a system but also setting up, or at least making modifications to, customer infrastructure, including, for example, PCs and network. Deployment is completed at the customer site. System deployment is straightforward, whereas finding out the customer requirements takes more time. In fact, finding out customer requirements, specifying the environment, and training users may extend the total project duration, including deployment, by up to several months.

Configurator One of the products in the CSPF is an in-house developed *configurator* that is a tool with a graphical user interface developed for configuring a system. The configurator is used to configure the low-level variability by setting and storing the parameter values in configuration files.

There are a few checks to ensure that parameter values are set correctly, but the configurator does not check that all parameter values are set correctly. The configurator is not usable without modifications in other CSPFs because it is developed for a particular CSPF, including its CSPF-specific configuration checks, for example. A more in-depth configurator is described in a manual of over 300 pages.

After-sales and maintenance Company B considers updates as a business opportunity and, in practice, requires a maintenance contract from every customer. Whenever a new version of the CSPF is released, it is delivered to customers who have the update contract. Because the CSPF itself is never tailored, the customers have the same source code, so updating is relatively easy.

A system can be reconfigured, which means setting parameter values to different values in order to readapt a system to a changed environment, for example. The set of products included in the system cannot be changed by reconfiguring. Reconfiguring primarily requires an understanding of the application domain and of how the system works, not software engineering skills. Even end users can use the configurator to reconfigure the system; reconfiguring is therefore common.

4 Comparison

In this section, we compare the characteristics of the CSPFs studied. A summary of the comparison is presented in Table 2.

4.1 Similarities

Similarities between the CSPF approaches studied were striking, despite the differences between the companies, products, and application domains. Both companies saw that their business was to deliver adaptable software rapidly to their customers, and according to customer requirements.

Both companies have applied the CSPF approach for about a decade and the success of the CSPF approach have forced them to make compromises that were also perceived as threats; these included, for example, fixing the scope of the CSPF, relaxing the demand for new features and fulfilling every customer-specific requirement that needs code modifications. However, the responders in both companies saw that the conscious decision to develop a CSPF was advantageous, despite this meaning that some of the flexibility in satisfying customer requirements compared with modifying the source code would be lost. Nevertheless, neither of the companies had any quantitative data to support the benefits of the CSPF approach.

Table 2. Comparison of the characteristics of the CSPFs

	Company A	**Company B**
Adopting CSPF	Evolutionary, old products	Revolutionary, new SPF
Organization	Roles according to domain engineering model in the same premises	Units according to domain engineering model and separated geographically
Deliveries	50	Dozens
Deployment process	Tailoring, installing, configuring, shipping	Selecting products, shipping, installing, configuring
Configuring	About 10 major functions, hundreds parameter values in adapting to environment	About 10 products, thousands parameter values in adapting to environment
Tailoring	30% of deployments	Not allowed, additional products
Delivered code	Everything	Only code for the selected products
Configurator	In-house developed, GUI, modify parameter values, not general purpose. Configuration rules embedded in the configurator, don't check entire validity of a configuration	In-house developed, GUI, modify parameter values, not general purpose. Configuration rules embedded in the configurator, don't check entire validity of a configuration
Parameter values	In a database	In configuration files
Configuring	Company A itself	Deployment organizations of Company B, partners, customers
Knowledge needed for configuring	Application domain knowledge, no software engineering skills required	Application domain knowledge, no software engineering skills needed
Reconfiguring	Rare, Company A reconfigures	Typical, customers can reconfigure
Updates	Rare	Update contract for new versions nearly always

Both companies were roughly the same size and had a similar organizational structure, which followed the domain engineering model. The operating environments of both the CSPFs were PCs running the Microsoft Windows operating system. The number of annual deliveries was roughly the same, i.e., a few dozens.

The application domain and interfaces with the environment were relatively stable in both CSPFs. Fundamental properties of the products did not change rapidly and application domain experts employed with the companies were able to predict variability reasonably well and communicate it to software engineers.

The deployment project in both companies took a relatively short time for the software, when considering the size of the delivered code and the fact that it always included as an essential part the configuring that was carried out without modifying the existing source code, or developing a new one. In fact, the time taken by configuring was quite short compared with, for example, hardware manufacturing, training, the sales process, and requirements specification.

Deployment was conducted in both companies by first selecting about ten large functional entities: In Company A, the entities were the configurable major functions in the stacks for a product, and in Company B, the entities were the products for a system. Second, in both companies, a product was configured to adapt to the customer-environment by setting numerous parameters using a configurator. The parameter values were separated from the source code and were initially set to default values in both CSPFs.

Neither of the companies typically changed the source code during deployment, for example, in programming, compiling or linking time, but configuring was performed on the executable code by setting parameter values.

For configuring, both companies had, in principle, a very similar configurator. The configurators were developed in-house and had a graphical user interface. The configurators were developed for the particular CSPF and could not be used in any other CSPF without modifications.

The configurators were easy to use and provided guidance for the user, who primarily needed an understanding of the target environment and application domain, but no more than basic software engineering skills. In configuring, the configurators modified the parameter values, but did not guarantee that all parameter values were set correctly; configuration rules to check the correctness of parameter values were embedded in the configurators in both CSPFs.

In both companies, their use of the configurators during deployment relied heavily on the user's experience in the application domain and on testing the configuration, as it was possible that parameter values were set incorrectly.

Finally, in both companies, deployed products or systems could be reconfigured in the customer's environment using the configurator.

4.2 Differences

The main differences between the CSPFs studied seemed in many cases to lie in the different decisions made with regard to equally feasible choices. In fact, when we explicitly asked about these differences, the responders told us that they could have decided in a way similar to the other company.

The CSPF adoption strategy was probably the most notable difference. Company A followed the evolutionary strategy for old products [1], whereas Company B followed the revolutionary strategy for a new product family [1].

Company A expected its markets to be relatively stable, whereas Company B expected the markets to have a large growth potential.

Although the organization model in both companies was similar, in Company A, deployers and developers were separated by role level, whereas, in Company B, they were separated into different organizational units and geographically.

Company A had explicitly defined how major functions for a product can be selected. Company B allowed, in principle, an arbitrary selection of products for a system, but, in practice, one of the products was always selected in a way similar to the base in Company A. In Company A, all customers received all the code of the CSPF, while the parameter values determined whether a piece of code was to be executed, whereas, in Company B, only the code of the installed products was shipped to a customer.

In Company A, a database was used, while, in Company B, configuration files were used to store the parameter values.

Unlike Company B, Company A allowed tailoring. However, even in the case of Company A, tailoring took place typically on the interfaces, while the basic functionality of a product remained untouched. Furthermore, most deployment projects in Company A were completed without tailoring. In the case of Company B, new products or components, in addition to a deployed system, could be developed in addition to the system. However, in fact, customers typically got the same code and parameter values specified by the configuration in both cases.

Configuring in Company A took only half a day, whereas in Company B it typically took from a few days to a few weeks longer. Nevertheless, in both companies, understanding the application domain and the customer-environment was more challenging than configuring.

Company A configured the products itself, whereas Company B or its partners did an initial configuration and customers were able to reconfigure systems later. The reason for this difference was a business decision based on the characteristics of the application domain and business model, not on technical grounds. For similar practical reasons, Company A configured the products at its own facilities, whereas, in Company B, systems were configured at the customer's site.

After-sales and maintenance strategies were completely different. Company A did not allow a customer to reconfigure a product and did not see updates or reconfiguring as a core business. Company B allowed reconfiguring and saw updating and maintaining the systems as a business strategy.

5 Discussion

5.1 Characteristics of the CSPF Approach

The results in this study show that the CSPF approach is, in practice, applied in the industry. To the best of our knowledge, the companies had developed their CSPFs independently of each other. Since the companies operate in very different application domains, their successful use of the CSPF approach is probably not tied to a specific application domain but a similar approach can be applied in other application domains as well.

The responders in both companies indicated that the CSPF approach provided a competitive advantage compared with competitors who modify and develop the source code in deployment projects. For both companies, the CSPF approach seemed an efficient way to systemize the software development and enable an efficient control of versions and variants in a set of systems. They even went as far as to state that the CSPF approach is the only reasonable way to do business, regardless of the initial investment needed to build the CPSF.

However, neither of the companies had estimates of investment payback times or other economic justifications when compared with, for example, project-based software development.

An intuitively appealing opportunity in the CSPF approach is the kind of reconfiguring that allows variability-binding operation time, in which binding is not fixed in the sense that the selected parameter value can be changed, as is noted to be a tendency in [10]. The CSPF approach enabled the companies to delay variability binding to installation and even operation time. Nevertheless, reconfiguring was effectively taken advantage of only in Company B.

What clearly differentiated the CSPF approach from the other software product family approaches in the three other companies in the case study was that the configurators were one of the key enablers for very efficient deployment. The fact that two companies had developed a tool for configuration on their own exemplifies the feasibility of developing and using configurators. Furthermore, in the companies, configurators seemed to be essential enablers for a large number of different parameters values that customize software for customers. These parameter values were set correctly even though the configurators did not check all aspects of the correctness of configuration, because the people using them were experts in the application domain and tested the system after configuring. In fact, by using the configurators, the companies were able to deploy products in such a way that, in practice, there is no software engineering knowledge needed.

Both companies had parameterized variability and separated parameter values from the source code to the configuration files in Company A and to the configuration database in Company B. Another approach could be to change the source code, for example, or compile time parameters, but if configuring is performed to non-executable code it would, in fact, probably not allow reconfiguration as flexibly as in the case of Company B. Furthermore, in both companies, because separate storage for parameter values is used, configuration is independent from implementation in the sense that software can be changed to another improved version, which does not include many new features, and the configuration and software remain operational even after the change.

5.2 Applicability of the CSPF approach

In the two companies studied, there seemed to be several factors that had a positive influence on the success of the CSPF approach. However, due to the relatively small sample size and lack of more detailed data obtained by, for example, observing the operations in the companies in more depth, rigorous analysis of the influence they had is not possible. The following discussion should therefore be treated as speculative in nature and the issues therein as hypotheses to be tested.

The number of deliveries that was large enough to make the CSPF approach feasible for these companies in these application domains is probably a necessary but not sufficient prerequisite for the CSPF approach to be economically feasible; this is because of the investment required for developing the configurator and configurability.

Application domain understanding was emphasized often in both companies: Developing configurability and configuring seem to be especially reliant on knowledge of the application domain.

It also seems that the application domain and interfaces to the environment should be as near stable as possible so there are no frequent changes to the fundamentals and so the new variability can be predicted reasonably well.

A clear separation of development and deployment organizationally, at least in roles, seemed to be beneficial, even in small organizations. In both companies, parameter values separated from the source code seemed to be a feasible way to achieve variability.

We did not find any special relationship between special software engineering skills, such as those specific to product family development and very advanced process models, methods, modelling and implementation tools, the CSPF approach adopting strategy, or application domain or company maturity at the time of the initiation of the CSPF approach and the success of the CSPF approach. In fact, it seemed that most of the software developers had their work and educational background in the application domain rather than in software engineering. Neither of the companies had developed a large repository or library of assets from which an applicable asset could be selected, but all assets that existed were included in the CSPF, and parameters were used to achieve variability. To develop the CSPFs, the companies did not use any special product family oriented tools or methods but, rather, state-of-the-practice tools and methods that probably could be found in any other company such as Microsoft Visio, Word, and IDE, and UML for modeling.

The possible drawbacks of the CSPF approach seemed to pertain mainly to the evolution of the CSPF. Evolution was seen as especially unpromising as far as the ability to meet customer requirements by configuring was concerned. If the configuring was not enough to meet customer requirements, the need for tailoring would increase. Alternatively, in the worst case, the scope of the CSPF would not be right and configuring would be useless, meaning that the considerable effort put into developing the CSPF would be wasted. The respondents felt that evolving software seems to be hard to manage and that keeping software configurable even increased the burden.

6 Related Work

There are some empirical studies on companies that exhibit a great deal of variability and are at least on the threshold of developing a CSPF, for example, CelsiusTech [11] and MarketMaker [3]. These companies have not systemized their SPF approach or developed configurators to the extent that the companies in this study have. However, because changing dozens of lines in configuration files is "burdensome, error-prone, and annoying", a need for a tool that sets parameter values was identified in MarketMaker. Apparently, a similar configurator as that developed by the companies in this study would be the kind of tool MarketMaker is hoping to have, although does not necessarily need. In addition, Securitas had developed a configuration tool Win512 [1], but there are not enough details available for detailed comparison.

Component-based software engineering [12] focuses on components, their composition and integration, while in the generative approach [13], products are constructed automatically on the basis of a precise specification; both these approaches are comparable to the CSPF approach. In fact, it seems both these could be used to develop a CSPF. However, the companies studied used neither the component based nor generative approach. When comparing the approaches, a drawback of the generative approach may be that it does not allow as much flexibly as the reconfiguring in the companies studied; this is because it requires the generation of the code again, and updating software, because not all customers have the same code. In the case of the component-based approach, it seems that components need to be parameterized in order to achieve similar customization, as in the companies studied in setting parameter values; the result of this might be a need for a configurator similar to that used in the companies in this study.

Linux Familiar, which is a Linux distribution for PDA devices, has been successfully modeled using a logic-based product configurator [14]. Futhermore, configuring in Linux is based on selecting components, while the companies in this study used almost only parameter values for this purpose.

BigLever [15] uses an approach that is similar to the CSPF approach and has developed a tool for deploying products. However, the lack of details of the aspects described in this paper makes detailed comparison difficult.

Configurable product families, or configurable products have also been studied in the field of mechanical and electronic products [16] and there are some initiatives to combine the results with software engineering [17], [18], [19].

7 Validity and Reliability

Case study as a qualitative method forces us to delve into the complexity of the problem as a whole, rather than to abstract it away [7]. Therefore, the quality of qualitative research pertains to how well the phenomenon under study has been understood. [20]. However, Yin [5] suggests the use of *construct*, *internal* and *external validity* and *reliability* to judge the quality of a case study.

Construct validity pertains to establishing measures correct for the concept being studied. To ensure construct validity, we based our initial understanding of the software product family approach on the results presented in the literature, used multiple responders in the interviews and documentation, carefully stored and used as much original data, such as transcripts, as possible, held validation sessions in the companies and allowed the companies to see and comment on the report before publishing. However, a possible weakness in the construct validity is that the study lasted a relatively short time, did not include observing the operations in the companies in more depth, and therefore could not cover all the details, and because a part of the data was based on only one source, our triangulation did not cover everything. However, we felt that the atmosphere in the interviews was open and trustworthy, while responders did not hesitate to answer our questions honestly.

The internal validity relates to causal relationships. We do not consider these relevant to this study because we have concentrated on describing the CSPF approach and tried not to make claims for causal relationships.

External validity means establishing the correct domain to which results can be generalized. Since only five software product families not randomly chosen were studied, two of these being CSPFs, we cannot make statistical generalizations as to the commonality of the CSPF approach in the industry. However, we believe that some of the commonalities found between the two CSPFs described are generalizable and would be found in other possible CSPFs as well.

Reliability means that operations of the study can be repeated with similar results. In order to ensure reliability, we used tools designed for qualitative data analysis and developed rigorous procedures, including interview questions, analysis framework, and data storage that could be followed to conduct similar research again.

8 Conclusions and Future Work

We described some results of a descriptive case study of the configurable software product family approach in two companies that had developed the approach independently. The study shows that the configurable software product family approach can be a feasible and even efficient way to systematically develop a family of products and to manage the variability within it. When compared, the characteristics of the configurable software product families turned out to be strikingly similar. The application domain and interfaces with the environment were relatively stable and the fundamentals of the product did not change; variability could be predicted reasonably well by application domain experts and could be bound by configuring.

Similarities were also found in the deployment and variability of the software product family, and in organizing software development. An application domain expert carried out deployment using a configurator tool. Variability included selecting a set of about ten major functional entities and adapting according to hundreds of parameter values. Organizationally, developers and deployers were clearly separated.

The clearest differences between the companies were to be found in the product family adoption approach and the after-sales strategy. The first company, which used the evolutionary approach to replace existing products, did not consider after-sales as a business strategy, and did not allow reconfiguring. The second company, which used the revolutionary approach for new products, allowed reconfiguring, and wanted customers to buy after-sales services.

On the basis of the results in this study, it is suggested that the configurable software product family approach may well be an important phenomenon for further study. A more detailed study and comparison of the similarities and differences between companies that have, and do not have, a configurable software product family should be carried out to elicit more indications of the potential applicability, success factors, and benefits of the configurable software product family approach. In addition, in both configurable software product families, there were hundreds of parameters with many possible values; the two companies studied had developed in-house configurators to assist in setting the parameter values and to do some error-checking of them. Despite the fact that the two companies had developed their configurators by themselves, it would seem to be advantageous to develop generic variability management and configuration methods and tools to model the variability evidenced by the parameters, and to assist in setting them correctly in deployment, in order to help companies that aim at developing a configurable software product family.

Acknowledgements

We gratefully acknowledge the financial support of Technology Development Centre of Finland and Academy of Finland (grant number 51394).

References

1. Bosch, J.: Design and Use of Software Architecture. Addison-Wesley (2000)
2. Weiss, D., Lai, C.T.R.: Software product-line engineering: a family based software development process. Addison Wesley (1999)
3. Clements, P., Northrop, L.M.: Software Product Lines: Practices and Patterns. Addison-Wesley (2001)
4. Bosch, J.: Maturity and evolution in software product line: Approaches, artefacts and organization. Lecture Notes in Computer Science (SPLC2) **2379** (2002) 257–271
5. Yin, R.K.: Case study Research. 2nd edn. Sage (1994)
6. van der Linden, F.: Software product families in europe: The esaps and cafe projects. IEEE Software **19** (2002) 41–49
7. Seaman, C.B.: Qualitative methods in empirical studies of software engineering. IEEE Transactions on software engineering **25** (1999) 557–572
8. Scientific Software Development, ATLAS.ti User's Manual and Reference, version 4., http://www.atlasti.de/.
9. Raatikainen, M.: A research instrument for an empirical study of software product families. Master's thesis, Helsinki University of Technology (2003)
10. van Ommering, R., Bosch, J.: Widening the scope of software product lines - from variation to composition. Lecture Notes in Computer Science (SPLC2) **2379** (2002) 328–346
11. Bass, L., Clements, P., Klein, D.V.: Software architecture in practice. Addison-Wesley (1998)
12. Szyperski, C.: Component Software. ACM Press (1999)
13. Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison-Wesley (2000)
14. Kojo, T., Soininen, T., Männistö, T.: Towards intelligent support for managing evolution of configurable software product families. Lecture Notes in Computer Science (SCM-11) **2649** (2003) 86–101
15. Biglever Software inc, http://www.biglever.com.

16. Faltings, B., Freuder, E.C.: Special issue on configuration. IEEE intelligent systems & their applications (1998) 29–85
17. Männistö, T., Soininen, T., Sulonen, R.: Product configuration view to software product families. In: Proceedings of Software Configuration Management Workshop (SCM-10) of ICSE01. (2001)
18. Hein, A., MacGregor, J.: Managing variability with configuration techniques. In: International Conference on Software Engineering, International Workshop on Software Variability Management. (2003)
19. Hotz, L., Krebs, T.: Supporting the product derivation process with a knowledge-based approach. In: International Conference on Software Engineering, International Workshop on Software Variability Management. (2003)
20. Maxwell, J.A.: Understanding and validity in qualitative research. Harvard Educational Review **62** (1992) 279–300