Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Assessing Software Product Line Potential: An Exploratory Industrial Case Study

[Industrial Experience Report, Special Issue on Empirical Evidence in Software Product Line Engineering]

Heiko Koziolek · Thomas Goldschmidt · Thijmen de Gooijer · Dominik Domis · Stephan Sehestedt · Thomas Gamer · Markus Aleksy

November 25, 2014

Abstract Corporate organizations sometimes offer similar software products in certain domains due to former company mergers or due to the complexity of the organization. The functional overlap of such products is an opportunity for future systematic reuse to reduce software development and maintenance costs. Therefore, we have tailored existing domain analysis methods to our organization to identify commonalities and variabilities among such products and to assess the potential for software product line (SPL) approaches. As an exploratory case study, we report on our experiences and lessons learned from conducting the domain analysis in four application cases with large-scale software products. We learned that the outcome of a domain analysis was often a smaller integration scenario instead of an SPL and that business case calculations were less relevant for the stakeholders and managers from the business units. We also learned that architecture reconstruction using a simple block diagram notation aids domain analysis and that large parts of our approach were reusable across application cases.

Keywords Software Product Lines, Domain Analysis, Business case

1 Introduction

ABB is one of the largest engineering companies in the world mainly operating in the areas of power and automation technology. In general, systematic software reuse is difficult to achieve in large corporate companies with several divisions and dozens of business units [27]. Company mergers add software products with functional overlap to the existing portfolio. The benefit of merging such products using a software product line approach is often hard to assess.

Although functional overlap between different software products in a company's portfolio is often perceived informally [8,35], it is not sufficient to start a

ABB Corporate Research
Industrial Software Systems Programm
68526 Ladenburg
E-mail: heiko.koziolek@de.abb.com

costly SPL initiative. Features might be similar on a high abstraction level, but often there are subtle differences that require a deeper technical analysis involving software architecture evaluation. Besides the technical analysis, there is the need for proper scoping, analysis of organizational constraints, and assessment of the development organization's readiness for SPL engineering. The value for SPL engineering needs to be communicated to various involved stakeholders taking into account the technical and organizational analysis.

Analysts use numerous methods in industry to assess product line potential given a set of software products. For example, the SEI applies its 'Product Line Technical Probe' method [14,39] to examine readiness for product line adoption. The Family Evaluation Framework (FEF) [37] provides a four-dimensional evaluation profile to assess organizational readiness. PulSE Eco [46] provides another approach for domain potential analysis. We do not introduce a new method in this paper, but have tailored the existing methods into our own approach that respects our company's specifics and lays special emphasis on software architecture evaluation [34].

The contribution of this paper are experiences (Section 5.1, Appendix) and lessons learned (Section 5.3) from executing a domain analysis method in four cases on more than 20 industrial software systems. In our study the result of the domain analyses were often smaller integration scenarios due to high migration costs in the industrial domain. We understood that business flexibility is often an argument against an SPL approach in the analyzed domains. The business cases could only be executed in two cases, where they showed a positive return on investment after three and seven years, respectively. The goal of the paper is to support similar domain analysis approaches in other companies by pointing out valuable experiences and avoidable pitfalls.

The remainder of this article is structured as follows: Section 2 summarizes foundational work on domain analysis, SPL potential analysis, and architecture reconstruction, before Section 3 introduces our seven-step approach to assess SPL potential . Section 4 states research question and working hypotheses for our empirical study. Section 5 discusses the answers to the research questions, threads to validity, and lessons learned. Related studies are discussed in Section 6, before Section 7 concludes the paper and discusses future work. The application cases are described in a condensed manner in the appendix, more detail is availabe in other articles [34,16].

2 Foundations

A general overview of SPL engineering can be found in several publications [14,41]. Linden et al. [37] provide an industrial perspective and the Software Engineering Institute (SEI) has cataloged numerous reports of industrial case studies in the Product Line Hall of Fame [11]. Recently, Ahnassay et al. [1] reviewed 79 empirical evaluations for software product line approaches, but found that the studies often lacked proper design and reporting.

2.1 Domain Analysis

To assess the potential for implementing an SPL, a domain analysis can be instrumental. In this context, a domain is "an area of business/technology processes or knowledge, which is characterized by a set of concepts and terminology understood by stakeholders in that area" [2]. Domain analysis can then be defined as "the process by which information used in developing software systems within the domain is identified, captured, and organized with the purpose of making it reusable (to create assets) when building new products" [2]. In this paper the domain analysis process is used to identify commonalities and variabilities among a given set of software products in order to facilitate future common software development.

Domain analysis was first proposed by Neighbors [38] and later integrated into modern SPL techniques. Prieto-Diaz and Arango [43] provide additional seminal work on domain analysis for software systems. Simos [48] presents Organization Domain Modeling (ODM), which is a systematic domain analysis method structured along a core domain modeling life cycle.

Khurum and Gorschek offer a comprehensive overview of domains analyses for software product lines [31] citing 89 studies from the literature, 25 of which were authored by practitioners. However, according to Khurum and Gorschek many of the domain analysis studies lack rigorous empirical validation. While our study is an exploratory case study, we try to improve rigor by formulating distinctive research questions and discussion the study's threats to validity.

2.2 Product Line Potential

There are also several recent approaches for analyzing the potential for software product line engineering. The SEI Product Line Technical Probe [12] (PLTP) examines the readiness of a company to succeed with a software product line approach. The PLTP consists of three phases: Preliminary Phase, Technical Probe Phase, Follow-on Phase. In the Preliminary Phase, the goals and preconditions of the Probe are elaborated and documentation is collected in a one day workshop together with the (customer) organization to prepare the Technical Probe Phase. The Technical Probe Phase is an interactive process of data gathering (mainly done by interviews), data analysis, result consolidation, and reporting. Each iteration usually focuses on one of the 29 SPL Practice Areas defined by the SEI and the interviews are performed with the corresponding stakeholders for each Practice Area. The aim of the Follow-on Phase is the development of an action plan for addressing the issues identified in the Technical Probe Phase and proceeding with the software product line effort. The Practice Areas are divided into Software Engineering Practice Areas such as Architecture Definition, Technical Management Practice Areas such as Scoping, and Organizational Management Practice Areas such as Building a Business Case. Northrop et al. have reported experiences with this method [39].

The Family Evaluation Framework [37] has been influenced by the SEI PLTP and CMMI [13] for assessing an organization's maturity for software product line engineering along four different dimensions: Business, Architecture, Processes, and Organization. Each dimension is divided into several aspects such as Vision and Business Objectives and Organizational Structure similar to the SPL Practice

Areas used by the SEI Technical Probe. For each of the four dimensions, the Family Evaluation Framework defines five levels similar to the capability levels of CMMI to assess the readiness of an organization to succeed with a software product line and to identify improvement potentials.

The Reuse Capability Model (RCM) [15] includes a model for assessing an organization's strength and improvement opportunities for reuse. Critical factors, such as management, application development, and process factors, are evaluated to implement reuse initiatives.

As a part of the PulSE method [3] for SPL engineering, PulSE-Eco from Schmid [46] intends to analyze functional overlaps and consists of three steps: product line mapping, domain potential assessment, and reuse infrastructure scoping. Product line mapping summarizes relevant products and their corresponding features, which are grouped into domains, in a product or feature map. In the domain potential assessment, the benefits and risks of reusing or sharing features are assessed and used for selecting domains that could be included in a software product line. The final decision which features become part of the product line is done in the reuse infrastructure scoping, which performs a quantitative evaluation of the features w.r.t required effort and business goals. John et al. [28] describe how PulSE-Eco is customized for different contexts such as a light-weight scoping for limited efforts.

2.3 Architecture Reconstruction

Software architecture reconstruction is a reverse engineering approach to extract architecture views of a sofware application and can help to better assess the technical feasibility of an SPL approach for existing products. For architecture reconstruction, there are many automated and semi-automated approaches [17]. For example, Guo et al. [24] propose a method for architecture reconstruction based based on the recognition of patterns. Kazman et al. [30] extract information from software system implementations and use this for architectural reasoning. They also provide guidelines for architecture reconstruction.

To identify reusable components and features [42], the MAP [49] approach allows semi-automatically extracting architecture information from code, identifying patterns / styles, and evaluating the potential for software product lines. An experience report [10] applies MAP for recovering the architecture of a web system. Pinzger et al. [40] apply architecture recovery in the context of product families, and perform a semi-automated, pattern-supported reconstruction.

Frenzel et al. [21] use reflection and clone detection for identifying ad-hoc copied code between different products. Harhurin and Hartmann [25] apply a commonality analysis on formal design models for extracting product lines from existing systems. Ganesan and Knodel [22] use object oriented metrics for usefulness, readability, and testability to identify reusable components. Eisenbarth and Simon [18] propose the usage of the Bauhaus suite, a tool for source code analysis, to identify features appropriate for a software product line. Hariri at al. [26] mined numerous online product descriptions and used various clustering mechanisms to identify common features across products and their relationships.

3 Software Product Line Potential Analysis

3.1 Methodology

The goal of our SPL potential analysis approach is to provide technical and economic arguments for the feasibility of establishing a software product line for a set of separated products.

The analysis comprises a domain analysis and and an economical analysis (Fig. 1). The domain analysis includes feature modeling of the domain, comparing software architectures, mining candidate assets. The economic analysis includes calculating the return on investment for an SPL based on the outputs of the domain analysis. The overall output is rather a technical risk assessment and a ROI calculation, not a reference architecture or migration strategy. After executing our analysis, a business decision is needed on further steps towards an SPL or systematic reuse.

Our SPL potential analysis approach is mainly based on the SEI PLTP [12] and has been iteratively updated during the different application cases. The major differences are our more limited resources and that our main goal is to assess the appropriateness of a set of independent products for the transition towards a Software Product Line.

Therefore, we aim at an 1-2 days interview (steps 5 and 6) only with an architect and product manager for each product instead of a one day preparation phase meeting plus the iterative set of interviews, data analysis, consolidation and reporting for each Practice Area of the PLTP. Instead of the PLTP's one day preparation phase meeting, we perform a video conference or phone meeting with our sponsors in order to agree on the list of products to be analyzed, corresponding information sources (step 1), and reuse potential criteria (practice areas) (step 2) to be considered in the analysis. In order to get an overview and compare the different products we focus on the PLTP's Practice Areas Architecture Definition, COTS Utilization, Mining Assets, Understanding the Relevant Domain and Scoping. These are usually complemented by Technology Forecasting, Process Definition, Organizational Structure, Market Analysis, Funding, and Business Case. However, aspects can be added or removed based on the goals and requirements of the sponsors. Note, Practice Areas that are not covered by our SPL potential analysis need to be considered in follow-up activities, if the result of our analysis is to go further into the direction of a software product line.

Before the interviews, we already collect and analyze the available documentation (step 3) in order to prepare a questionnaire, an initial feature list, and an architecture sketch (step 4) for focusing the interviews (step 5) on the important open issues. For Product Line Scoping and creating the feature list we follow a lightweight variant of PulSE-Eco from Schmid [46] as descibed by John et al. [28].

In contrast to the PLTP, we focus with our analysis more on identifying the products, feature (sub-)domains and components that have the potential for being merged into a software product line (step 6) and try to create a business case calculation (step 7) for these identified reuse potential scenarios.

The following subsection describe the steps of our domain and economic analysis in more detail.

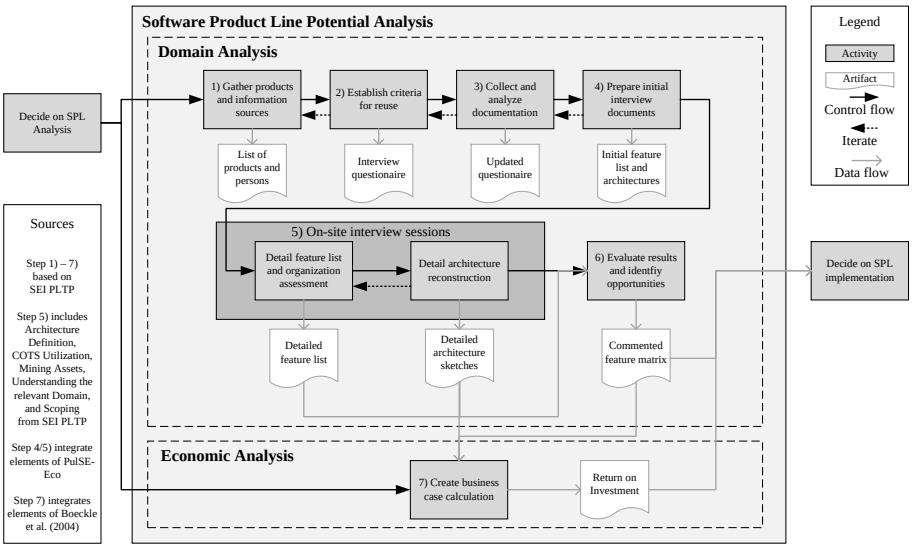

Fig. 1 Domain analysis process

3.2 Steps of the Domain Analysis

This section introduces in more detail the different steps of the domain analysis part of our Software Potential Analysis as sketched in Fig. 1. The order and separation of steps is not necessarily fix and complete, but provides a guideline to be utilized and adapted for future application cases.

1) List products and information sources We assume that a number of similar products is already available for a specific application domain. For ABB, this is a common situation in many different application areas, for example due to company mergers or local differences [35].

The first step is thus to gather a list of the available products and their respective information sources (e.g., product managers, software architects, documentation archives, competitor products). The initial list of products is given by product stakeholders and project sponsors such as product or technology managers and architects, who are interested in analyzing an intuitively perceived functional overlap. Often they can also provide pointers to similar products and we also ask domain experts from other units. In addition, the intranet can be searched for potentially similar products to get a more comprehensive list. In the end, we decide together with the involved product stakeholders and project sponsors about the final list of products to be analyzed.

In practice, the scoping analysis however can sometimes not aim at completeness, but is instead restricted by stakeholder interest and the availability of required contact persons. The list of products can of course be extended over the course of the analysis as more information becomes available.

For a large corporate company with many globally distributed business units developing software for different market segments, gathering the list of products

and information sources can be complicated and time-consuming. Some products may lack suitable documentation to decide whether to include them into the scope. Other products may be out of active development, therefore complicating reusing parts of them.

Therefore, this scoping step is often a best effort activity driven by organizational and budget constraints. The output of this step is an initial list of products and information sources.

2) Establish criteria for reuse potential In a second step, we first establish the criteria for evaluating the reuse potential of the given products. We draw these criteria from a generic list that we have assembled. The Reuse Capability Model [15] and the SEI Technical Probe [11] inspired our list of criteria and can also help other practitioners in defining their own criteria. Our criteria are software-related, process-related, and organization related and include the following elements:

- High-level features: the major functionalities of a product are a main source and required ingredient to identify reuse potential.
- Architecture and technologies: if the architecture of different products are aligned there is usually a higher potential for reuse. The architecture consists of components, interfaces, and their interaction. The existing component and connector structure of a product can, but does not need to be, important for the reuse potential. The technologies include programming languages and third-party components, such as operating systems, graphic libraries, communication middleware, encryption components, etc. They also include the implemented communication protocols, whose compatibility is often a critical factor for distributed systems. The use of industry standards (e.g., for protocols, interfaces, components) can facilitate reuse in certain domain, but is not applicable in any domain.
- Development process readiness: To participate in a software product line development, the development processes of the participating units can be a critical factor influencing the reuse potential. Thus, it can be useful to evaluate each unit's release management strategies and change management processes and rate them according to their suitability for an SPL approach. For example, it is beneficial, if certain fixed release cycles are established that could be synchronized across units. Additionally, well-established requirements management and change tracking processes can facilitate an SPL approach.
- Organizational fit: this can be assessed by analyzing organizational structures, development roles, and funding models. If a unit's organizational structure already allows a seamless development of reusable components, it contributes to a higher SPL potential. Likewise, if their are established developer roles, such as software architects and reuse coordinators this is a again beneficial. Finally, appropriate funding model for the development teams can be a contributing factor, e.g., if there are already models in place that allow a shared development between different units.
- Market fit: if the business market of the products under scope is expected to increase or decrease significantly in the near future, this can be a crucial factor for an SPL approach. For industrial, long-living software systems controlling critical infrastructures, migration costs to a SPL can be prohibitively high,

because the installed base of products cannot be easily updated. Therefore, this can also be an important factor.
- Reuse culture: a development unit's reuse culture comprises their knowledge, commitment, and past experiences on software reuse. If a unit has already established reuse processes both for providing and consuming external components, it is easier migrate to an SPL approach.
- Future outlook: an important criteria for future common software development between units is the alignment of the business roadmaps. If there are overlapping elements in the plans for different products it positively contributes to reuse potential. This may include upcoming trends or domain-specific technologies that expected to be used within many products in the future.

Many criteria are generic and can be directly used across different domain analysis executions. However, some criteria depend on the scope of the analysis. For example, if the software of a single business unit is analyzed, collaborations with other units are less relevant. We generally do not use quantitative scores for the criteria but rather rely on qualitative assessments.

3) Collect and analyze documentation We gather existing documentation on the products and development processes. User manuals help to get an overview of the feature support. Architecture documentation provides hints on the technology compatibility of the products as well as the support for standard interfaces.

Besides product-specific documentation, it is also helpful to collect domain-specific reports, recent technology surveys, and industry standards. If possible, documentation on competitor products can complement this step. Besides the analysis team, additional domain experts participating in this step are often researchers from ABB's corporate research labs, who are not necessarily from the IT domain. Because they usually have worked on multiple products within an industrial domain, they can provide valuable product-independent information for a domain analysis. Since they monitor market and technology trends, they can further point to analytical reports that can be added to the documentation under analysis.

A deep analysis of the documentation is not needed; rather the documentation is screened for answers to the questionnaire concerning feature support and development process properties. In our experience, the documentation is often not in a condition to answer all questions, which is why we conduct stakeholder interviews. A deeper analysis of the documentation might be warranted after the stakeholder interviews for specific issues.

4) Prepare initial interview documents Based on the reuse potential criteria defined in step 2 and the documentation analyzed in section 3, we create a questionnaire for the interviews in step 5. In order to accelerate the interview process and using the time for addressing the most important open issues, we already include answers from the document analysis in the questionnaire and prepare an initial feature list and architecture sketch.

To start feature modeling, in this step the analysis team creates a list of features in a given domain. The analysts carry out this step based on the available documentation and their own domain knowledge. Architects or developers of the

products under analysis are not yet involved to save time. Additional persons, such as ABB researchers or external consultants, may be contacted if needed.

This step is specific for each domain analysis execution. Initially, a flat list of high-level features is gathered and a short description of each feature is provided. We usually create the initial list based on our domain knowledge and experience with individual products. User manuals are often a good starting point for the list as they usually include sections on individual features. Industry standards and current technology trends also serve as input for the feature list. This step can already include an initial alignment of terms for certain features between products, as different user manuals may describe the same feature under a different term.

Before further analysis, domain experts from within our company (e.g., product managers, researchers, external experts) review the feature list. The number of reviewers depends on the domain, the availability of domain experts, and may vary from at least one up to three or four, because we need to keep the effort of the entire analysis low.

We then integrate questions on each feature into the questionnaire created in step 2. The questions ask whether the feature is supported at all, in which sense is it supported, and if there are any sub-features which are subsumed under this feature. We already try to adjust the feature terminology according to our findings from step 3. We also integrate some example answers to guide the interview partners and speed up the process. In our experience, the feature list usually needs to be updated after each interview as new information becomes available.

If we received a sufficient documentation of a product architecture in step 3, we start drawing a first architectural sketch as starting point for the architecture reconstruction in the interviews. Such documentation does not necessarily include UML models, but most often power point presentation with own proprietary notations. The goal of the architecture reconstruction in our domain analysis is to identify candidate assets for a SPL from existing software products. The reconstruction shall point out the major subsystems and interfaces of a product. It shall focus on logical structures and functionality and abstract from the binary representation of the code. Additionally, the communication patterns (e.g., request/response, publish/subscribe) and the data storage facilities shall be illustrated to evaluate technological compatibility. Finally, selected important extra-functional properties or constraints of an architecture and its components (e.g., performance constraints, security considerations) shall be analyzed.

To allow for a comparison of the architectures in the domain analysis, we aim at a common documentation notation. We ruled out using UML for several reasons: Despite the availability of component diagrams, UML is still tied to programming level concepts, such as classes and methods. There is no default representation of data storages. It is difficult to integrate multiple levels of abstraction into a single diagram, which is sometimes useful for a domain analysis. Integrating more information into UML with stereotypes often generates visual clutter, because UML tools usually do not support non-standard graphical shapes for stereotypes. Similar consideration ruled out SysML block definition and internal block diagrams.

Finally, we decided to use simple block diagrams for a common architectural documentation in our domain analysis according to the Fundamental Modeling Concepts (FMC) notation reference [33]. FMC block diagrams are highly ab-

stracted and do not use class-level elements. Thus, they are hardly suited to implement a design, but useful to communicate an architectural overview. They condense the information needed for a domain analysis and allow for a common documentation according to a notation reference. It is also possible to hierarchically refine such block diagrams or have different refinement levels for different subsystems in the same diagram. The notation has been used in other industrial contexts, e.g., at SAP [23].

While FMC allows to use Petri nets for dynamic structure, we wanted to show the data flow within the architecture overview, i.e. using the block diagram notation. We used coloured, directed lines to represent a usage scenario flow similar to the use case maps notation [9]. These flow paths are additionally annotated with extra-functional information, such as latencies or data volumes, where available. They can allow assessing the technical compatibility between products.

5) Conduct Interviews In this step, we travel to the development site of the software product and conduct a one-day interview with the software architect and product manager. We structure the interview with the questionnaire prepared in steps 2-4 and try to answer all the included questions. During the interview we note down the raw answers and re-formulate the included questions or provide examples for understanding if needed. Besides information on the features and process, we also ask for future plans and roadmaps. Roadmaps are instrumental in defining easy to implement reuse or SPL scenarios across business units. Our interviews are executed per product and we do not conduct workshops with multiple stakeholders at this stage to be time- and cost-efficient.

During the interview, we include a 1-2 hour slot where we let the architects sketch and explain the architecture on a whiteboard. We use this session to ask about architectural features that relate to our analysis, but cannot be derived from the available documentation. In our experience this is the quickest way to get to an up-to-date bird's eye view of the architecture that can serve as input for our domain analysis.

6) Evaluate results and identify opportunities The results of step 3, 4, and 5 are for each product a sketch of its architecture with components, interfaces, and used technologies, a list of the features supported by the product, and a questionnaire covering the answers to the other reuse potential criteria defined in step 2. In the first step, we clean up all documents, e.g., layouting architecture sketches, correcting grammar and wording, and removing duplicated features. Then, we compare the products with each other.

For the features, we build up a feature map according to PulSE-ECO [46]. The feature map is a table which shows for the analyzed products in the columns, the supported features in the rows by an x or black circle. Partial support can be specified by an (x) or half filled circle. When the feature lists of the different products have been created independently of each other, the same features can be named differently for each product or features with the same name can have different meanings for different products, for example. These cases need to be identified and also cleaned up.

Low level features are grouped into feature subdomains and feature domains according to their logical relation in order to get a higher abstraction level and a

better overview for comparing the different systems. For example, all features of a PC tool for importing or exporting to XML, PDF, or Microsoft Office documents are grouped under the feature subdomain *Data Im-/Export* and under the feature domain *Interfaces*. The creation and cleaning up of the feature map can already be started in step 4 and step 5 for more easily identifying equivalent features with different names, for example. After cleaning up the feature map, we go through each feature domain and sub-domain, rate the feature overlap (commonalities and variabilities) between the different products with low, medium, and high, and provide a rational for the rating, which also considers the results of the comparison of the architectures and questionnaires.

For the architectures, we arrange the architecture sketches in a similar way, which is usually possible, if the products use similar architectural patterns and styles. Then we manually compare the architectures in order to identify commonalities and variability and to identify larger parts of the architecture that could be shared or reused between products or that could become part of a common core asset base. Automating the architecture comparison is most often not an option for us, because we have only informal FMC sketches and the products are developed completely independently of each other, in different technologies, and without any copied code. In this investigation, we also compare the interfaces and technologies used by the products in the different parts of their architecture and summarize this in tables. We check for the feature (sub-)domains rated with a high reuse potential in the feature analysis, where these are implemented and whether they could be shared between the different products. The outcome is a list of potential reuse scenarios, which are lager system parts that could be shared in the future between the products.

In parallel to the feature and architecture comparison, we also compare the other reuse potential criteria based on the questionnaires with each other such as: development process readiness, organizational fit, market fit, reuse culture, and future outlook. This analysis and comparison is usually done only informally. For each reuse potential criteria, we check whether a product would fit into a future common software product line, identify potential issues, and provide recommendations how to address these issues and go forwards towards a common product line. We do not apply a formal maturity level rating as proposed by the Family Evaluation Framework [37], but for visualization purposes, we summarize the results in tables with a rating of high, medium, and low for each pair of product and reuse potential criteria.

3.3 Economic Analysis

To provide a financial justification, we create a business case for the establishment of a SPL in the given application domain.

7) Create business case The calculation for the return of investment (ROI) of a future SPL follows the proposal by Boeckle et al. [5] and needs the following inputs:

- C_{evo}: the costs for maintaining each of the former products without reuse

- C_{org}: the costs for changing the organization to adopt SPL enginneering and for training people in SPL engineering. These costs need to be estimated based on the organizational analysis in step 7.
- C_{cab}: the costs for building the core asset base or platform including product scoping and implementation.
- F_{cab}: the fraction of the core asset base that needs to be updated per release or year.
- C_{unique}: the costs of building the unique part of a product of a SPL. These are calculated as the costs of an individual product multiplied with its unique part in percent multiplied with the change rate of the product. For example, if the costs of a product are 10 million dollars, 30% of the product are unique, and 20% of the product change per release, C_{unique} is the product: 600.000 dollars.
- C_{reuse} the costs of reusing the core asset base for building product of the SPL. These are calculated as the costs of an individual product multiplied with its common part, which is based on the core asset base, in percent, multiplied with a factor that reflects the costs of building the equivalent code based on the core asset base and not from scratch. For the last factor, Boeckle et al. [5] assume 10 percent to be usually a high conservative estimation. For example, if the costs of a product are 10 million dollars, 70% of the product are in common with the core asset base, and we assume the cost of building the equivalent code based on the core asset base to be 10% of building the code from scratch, C_{reuse} is the product: 700.000 dollars.

The ROI for merging n products into a new product line is calculated with equation 1:

$$ROI = \frac{\sum_{i=1}^{n} C_{evo_i} - \left(C_{org} + C_{cab} + F_{cab} \times C_{cab} + \sum_{i=1}^{n} (C_{reuse_i} + C_{unique_i})\right)}{C_{org} + C_{cab}} \tag{1}$$

The ROI is the ratio of the cost savings to the cost of investment. The investment costs in the denominator are $C_{org} + C_{cab}$. Boeckle et al. [5] estimate C_{cab} by using the development costs of a single product and multiplying it by the ratio of the product that becomes part of the core asset base and by a factor that represents the higher development costs for making the assets reusable.

Product management can most often provide the development costs of a product, but they are sometimes outdated or no longer available due to the age and history of some products. In such cases, cost models such as COCOMO II [6] can be used. The higher cost for building reusable assets with more generic interfaces and functionality are assumed with 150% as rule of thumb.

In the numerator of equation 1, $\sum_{i=1}^{n} C_{evo_i}$ are the costs for maintaining the products individually, which can be easily acquired. The second part of the numerator, the subtrahend, are again the investment costs for the core asset base, the cost for maintaining it ($F_{cab} \times C_{cab}$), and the cost for building all n products based on the core asset base ($\sum_{i=1}^{n} (C_{reuse_i} + C_{unique_i})$).

F_{cab} can be estimated by measuring the change rate of the old products in the past. For developing a product i, core assets are (re-)used from the core asset base (C_{reuse_i}) and extended by product specific functionality (C_{unique_i}). For each

product it must be estimated how many percent can be reused from the core asset base and how many percent are product-specific. Boeckle et al. [5] estimate the factor of building the same functionality as in the former products with the core asset base as 10% compared to implementing it from scratch. They mention that this is actually on the high side, but there is no further rational for this number.

The return on investment can be calculated over years or releases. All recurring numbers need to be normalized accordingly. The formula describes the integration of n existing products into a new product line. For some other scenarios such as adding one product to an existing product line, variances of the formula are described in [5].

In summary, the business case calculation relies on the estimation of the reusable fraction in each product, the costs for developing a generic core asset base, as well as the costs for adapting reusable core assets to realize a specific product. These values are difficult to estimate precisely. However, we can coarsely estimate the reusable fraction based on our feature maps, as well as the higher cost for the core asset base based on the feature map and the architecture maps. Thus, the estimates are backed up by our domain analysis, which makes them easier to defend.

3.4 Wrap-Up

At the end of our approach we document the results and hand them over to the stakeholders of the analysis and products. An overview of the analysis and all findings are documented in a report. The questionnaire, feature map, architecture map, and artifacts generated during the analysis are cleaned up and attached to the report. In addition, the analysis and most important results are summarized in a presentation and discussed in a one or two hours meeting with the stakeholders.

At this stage our SPL potential analysis is complete. The next steps are getting a decision for a particular reuse scenario or a full systematic software product line development. Stakeholder workshops across the available products need to be held and champions for the SPL need to be found in the business units. The subsequent steps of domain engineering including the design of a reference architecture and the sketching of migration roadmaps can be executed. These steps are out of scope for this paper.

4 Case Study Design

The underlying objective of applying our domain analysis approach was to decrease ABB software development and maintenance costs in the future by identifying areas for software development shared among business units. As the SPL Potential analysis can be applied in several application domains within ABB, we also expected it to increase our understanding about software sharing and the applicability of software product lines within our organization in general. Multiple cases can support identifying patterns and provide general findings beyond the specific cases.

4.1 Research Questions

Therefore, we considered the application of the domain analysis as an exploratory case study [51, 45, 19]. We formulated five research questions, which we tried to answer after executing the domain analysis method in the different cases. For each research question we formulated an initial working hypothesis on the expected outcome. Due to the low number of data points, these hypotheses cannot be statistically tested. In this case, they serve to make the learning effect from the case studies more explicit.

Q1: What are the results of the domain analyses for the analyzed products? With this question we were interested in the outcome of the domain analysis and the actions derived from its results. Our working hypothesis H1 was: *"the domain analysis supports the decision for starting an SPL initiative if it finds a high feature overlap and good organizational compatibility in the analyzed products and business units"*. In turn the analysis would lead to a rejection of an SPL approach in case of minor feature overlap or organizational challenges. If our working hypothesis would be invalidated in our application cases, it would be a hint for improving the method or for considering additional factors driving the decision for an SPL approach.

Q2: Which factors facilitate or complicate SPL development among business units? This question aimed at finding patterns in the reuse criteria and understanding the stakeholders' priorities for them. Our working hypothesis H2 was *"the stakeholders favor future SPL development if it provides them short-term to mid-term benefits and if a positive return on investment can be calculated"*. The question has been often addressed in literature, but we wanted to find out whether there are different factors relevant for our organization. Especially in a large corporate company such as ABB, organizational hurdles may be a major factor limiting shared software development beyond financial factors. But we also anticipated that technology incompatibilities in the different products could be a roadblock despite high feature overlap.

Q3: What granularity is required for the domain analysis in order to reach the decision for implementing an SPL or not? Feature modeling and technical analyses can be tackled on different abstraction levels. Features could relate to coarse-grained subsystem or to fine-grained single functions. They can be modeled as flat lists, hierarchical models, or even domain-specific languages [50]. Technical analysis can be carried out on an broad technology level or on a fine-granular code level. The question is how much detail is necessary to reasonably support a design decision. Based on findings in the literature [14], our initial working hypothesis H3 was *"the required granularity for the domain analysis - in order to reach a decision - is correlated with the analyzed products' size and complexity"*.

Q4: What is the projected return on investment for SPLs in our cases? The seventh step in our method is to create a business case calculation to map the technical analyses back to financial terms. Literature suggests different assumed parameters for such a calculation, which are supposed to be based on real products. For example, Boeckle et al. [5] assumed a 70 percent reuse level for different products and a 50 percent higher costs for building a generic platform instead of a single platform. We anticipated different reuse levels and platform costs for

the different application cases. Our working hypothesis H4 was thus formulated in more abstract terms: *"for cases with high feature overlap an SPL approach would pay off within five years"*. The working hypothesis is based on former experiences with integrating similar sized industrial systems at ABB and may not transfer to other domains with smaller products or shorter release cycles.

Q5: How does the domain analysis method need to be adapted for different cases? We described our domain analysis method in an idealized manner in Section 3. In practice there is a need to adapt the method to each application case. For example, some inputs, such as architecture documentation may not be given, or the stakeholder may not be interested in a business case calculation. This question is especially interesting for practitioners who want to conduct similar investigations. Our working hypothesis H5 was: *"more than 80 percent of the method's steps, templates and reuse criteria can be reused across all application cases"*.

Given the exploratory nature of our case study and high effort for carrying out single cases, we cannot support our hypotheses H1-H5 with any statistical significance. The following application cases merely serve to collect empirical evidence in favor or against our hypotheses. Threads to validity will be discussed in more detail in Section 5.2.

4.2 Case Overview

We applied our SPL potential analysis method in four different application cases in different industrial domains of ABB (cf. Table 1). The first case concerned a set of large-scale, distributed industrial control systems, the second case two families of stand-alone PC (Personal Computer) tools. Furthermore we applied the method on three device engineering PC tools in application case 3 and several Enterprise Information Systems in case 4. Appendix 7 provides a detailed description on how we executed our approach for each case.

Application Case	**1: Industrial Control Systems**	**2: Automation PC Tools**	**3: Device Engineering Tools**	**4: Enterprise Information Systems**
Type of systems	Client-server	Stand-alone PC tools	Desktop applications	Desktop applications
Number of products	6	4 + 2 families of 10	3	2 + family of 5
Number of business units	6	2	1	2
Number of developers	<10 to >40 per product	10	<5	2-3 per product
System size (LOC)	1 - 6.5M	300K - 2M	300 - 500K	10 - 100K

Table 1 Overview of the four case studies.

We did not choose these cases systematically, but the selection was instead driven by stakeholder requests and intuitively perceived good opportunities for future reuse cases. As seen in Table 1, the analyzed product vary in size and the cases have different number of involved parties, so that bias from a single case can be avoided to some extent. However, for example none of the application cases

comes from the area of embedded systems, so that the coverage of our study is far from being complete.

For data collection to answer our research questions, we simply used meeting minutes from the interview sessions (step 5 in our method, Section 3) as well as informal notes taken during document analysis. In each case we interviewed multiple stakeholders with different roles, such as product manager or software architect.

5 Case Study Results

5.1 Answers to Research Questions

Q1: What are the results of the domain analyses for the analyzed products? Our working hypothesis H1 for this question was: *"the domain analysis supports the decision for starting an SPL initiative if it finds a high feature overlap and good organizational compatibility in the analyzed products and business units"*. To investigate this working hypothesis, Table 2 summarizes the high level findings and outcomes in our application cases.

Application Case	**1: Industrial Control Systems**	**2: Automation PC Tools**	**3: Device Engineering Tools**	**4: Enterprise Information Systems**
Feature overlap	high	high	medium/high	medium/high
Opportunities for feature overlap	Reuse of entire subsystems across products	Development of common components, common platform	Development of a common platform	(Data) Integration, common platform, common tool
Technology compatibility	low/medium	high	low/medium	medium/high
Migration effort	very high	medium	medium	low
Organizational maturity / compatibility	low/medium	high	high	medium
Decision made	Not yet	Yes, both short-term and long-term integration	Yes, SPL prestudy	Yes, short-term integration

Table 2 Results of the domain analysis for each application case

We found a high (approx. 70 percent) or at least medium to high (approx. 40-70 percent) feature overlap in all application cases. However, the types of possible integration scenarios or shared developments varied from case to case. In case 1, an almost black-box reuse of entire subsystems was possible pending the creation of proper adapters for the different products. For case 2, 3, and 4 the development of a shared platform with common components was deemed feasible. For case 4, a simple data integration scenario leaving the existing products largely intact was an alternative.

Despite the high feature overlap, the technology compatibility was limited in most cases and the migration effort medium to very high. The existing products in each case were built on different technology stacks or had been developed at different points in time so that multiple incompatible versions of a technology stack were used. Not surprisingly, the organizational complexity for shared software development was higher for cases with more business units.

Consequently, the decision to try to pursue a SPL approach was only made for application case 3. In the other cases the stakeholders either favored short-term integration scenarios or deferred a decision for more shared development. The stakeholders were afraid of substantial changes to their existing products, which could have incurred high and unpredictable migration costs to evolve to a shared platform. Especially in the domain of industrial systems, these costs can be prohibitively high because of the involved hardware resources and required user expertise. The risk of a common platform was thus seen as too high to outweigh its benefits.

In general, it was challenging to sell the SPL idea to a number of stakeholders, which maybe one reason why the decision to pursue an SPL approach was made only in one case. While future saved development and maintenance costs for shared components were deemed plausible, some stakeholders were afraid of sacrificing their business flexibility when tightly collaborating with other units, which may serve other markets. For example, the coupling of release roadmaps could delay serving customers with new features in time. The trade-off between independence and efficient software development due to a common platform was seen by some as a critical issue in addition to technological differences.

Besides these findings, our investigations had some additional, secondary effects. For the analysis, we re-documented several product architectures with a common notation, which improved the communication between and within the stakeholders' units. The knowledge about the capabilities and technical decisions of the different products increased among the stakeholders, which might lead to less formal collaborations in the future. Our architecture sketches are now used by several units as an up-to-date architecture documentation, which aids discussing design changes and educating new employees about the product. We also aligned different domain-specific terms among the stakeholders, which enabled more efficient communication.

In conclusion, the result of our approach is often a less risky reuse or integration scenario between different products, which can be seen as the first step towards a software product line. Getting the commitment for a full-blown SPL approach based only on the feature overlap and organizational analysis of our approach was hard in the analyzed cases. Therefore, our working hypothesis has failed. As a new working hypothesis, more emphasis could be put on aligning business goals and development roadmaps during the investigation to motivate the benefits of an SPL even better.

Q2: Which factors facilitate or complicate SPL development among business units? Working hypothesis H2 for Q2 was *"the stakeholders favor future SPL development if it provides them short-term to mid-term benefits and if a positive return on investment can be calculated"*. The stakeholders preference for more manageable, short-term benefits was partially confirmed by the findings discussed for question Q1. For existing products, many stakeholders did not want to take the risks for a complicated SPL project.

For the positive return on investment, we surprisingly found only limited interest. Most stakeholders (including sponsors, managers, and architects) were hardly interested in the business case calculation, sometimes because it was intuitively clear for them, sometimes because they would not trust the assumptions going

into these calculations. These assumptions for example concerned the expected migration costs, the costs for creating generic components, the amount of shared software between components, as well as the overall development and maintenance costs.

During our interviews with the stakeholders, we received considerable input which factors would facilitate or complicate reuse and joint software development. Several stakeholders pointed out that strong management support is a prerequisite for such initiatives. If managers from different units can agree on some common business goals there would usually be a way to technically build a common platform. Several former experiences at ABB supported this statement.

Some stakeholders saw higher potential for joint software development if entirely new products were developed or new technologies should be utilized. Refactoring existing products towards shared components was seen as difficult to justify financially. If the old products were to be abandoned, e.g., due to an outdated technology, the motivation for a future common platform would be higher. For new products most interviewed business units had a process in place which explicitly asked for the consideration of existing components to start the implementation.

However, the stakeholders mentioned a number of factors against more shared software development. Most units were not aware of explicit incentives for software reuse from other ABB units, as they were assessed only by the revenues from their own products. Multiple interviewees noted a lack of an overview of the software in their domain stemming from a strong focus on their own product.

A suitable organizational structure can also facilitate more systematic reuse. Bosch [7] for example distinguishes four organizational structures:

1. A single *Development Department* with up to 30 developers
2. Multiple *Business Units* for 30 to 100 developers, each responsible for a particular product with shared development of common assets
3. A single *Domain Engineering Unit* responsible for common assets and several product engineering units (in case of more than 100 developers)
4. *Hierarchical Domain Engineering Units* in case the common assets require more than 30 developers.

Other factors contributing to the organizational structure are geographical distribution, project management maturity, organizational culture, and type of systems. In the context of this paper, the term "business units" has a broader meaning as in Bosch's paper, because the ABB business units do not only include software developers but also many other engineering-related employees.

Nevertheless, the application cases analyzed in this paper best match with the category of "Business Units" as defined by Bosch, as we analyzed collaborating business units without a dedicated domain engineering unit. But only in application case 1 the number of developers was larger than 30, thus the categorization by Bosch does not fully match our application cases. For other common software products within ABB that have not been analyzed in this paper, there are several examples for both the categories of "Development Department" and "Domain Engineering Unit". The latter category can facilitate SPL development involving multiple ABB business units as experienced within ABB, but these cases are beyond of the scope of this paper.

While organizational structures can facilitate SPL development, several stakeholders cited trust issues as a roadblock for more shared development, here. Former poor experiences with reused software of low quality and mismatching functionality limited their motivation for more reuse. Also some units did not provide proper maintenance and support processes for their shared software, because their organization was not tailored to support software reuse. Another issue was that software not designed for reuse lacked stable interfaces and generic functionality for multiple contexts.

Especially for dynamic business areas with changing business priorities it was difficult to envision the success of an SPL development project, which required a long-term commitment and did not provide immediate business benefits.

In summary, mostly organizational factors besides technical factors limited more reuse in the cases we analyzed. Our working hypothesis has partially failed, because the business case calculations were not as important to the stakeholders as we had expected.

Q3: What granularity is required for the domain analysis in order to reach the decision for implementing an SPL or not? Working hypothesis H3 was *"the required granularity for the domain analysis - in order to reach a decision - is correlated with the analyzed products' size and complexity".*

Tab. 3 shows the number of feature domains, feature subdomains, features, and subfeatures analyzed in each application case as well as the time spent on feature analysis alone. For case 1, we analyzed 26 subdomains, but did not break them down into more fine-granular features. Still we needed about 2 days per product to collect the answers for feature-related questions from our questionnaires. Due to the high complexity of the products as well as the emphasis on architecture reconstruction and organization we were not able to break down the subdomain list into more fine-granular features. However, in this case the stakeholders were satisfied with the achieved detail level, since they were not interested in a more fine-granular break-down, which would have consumed significantly more time.

Application Case	**1: Industrial Control Systems**	**2: Automation PC Tools**	**3: Device Engineering Tools**	**4: Enterprise Information Systems**
Number of Feature Domains	6	19	11	8
Number of Feature Subdomains	26	53	70	42
Number of Features	n/a	289	196	172
Number of Subfeatures	n/a	39	57	170
Analysis time spent for features	2 days / product	1-2 days / product	1 day / product	1 day / product

Table 3 Results of the feature granularity analysis

For the cases 2-4, we conducted an analysis on a more fine-granular level and created feature maps containing 200 to 300 features each. In this case, the features were sometimes simple functions, such as the ability to color elements in the graphical user interface. Because the features were more simple, it was possible to fill the feature lists within 1-2 days per product.

Considering these findings our working hypothesis is supported, as the more complex products needed more time for analysis. To get to a feature and subfeature level in the analysis of case 1, we would have needed much more time than in case 2-4. In our cases, the invested efforts were to a certain extent controlled by the stakeholders, who only dedicated a specific amount of time to the domain analysis. It is an open question on how this allotted time can be best spent in a domain analysis, and how much time to spent on feature elaboration or architecture reconstruction. In our cases, the already described effort distribution worked reasonably well, but this may not transfer to other cases.

An option for time-boxing the efforts for the domain analysis is starting with an initial mock-up business calculation for SPLs based on rough development and maintenance costs data. Then it could be identified what information is required to support the assumptions going into the calculation. This may include assumptions for the percentage of functional overlap or the overhead for creating reusable components. The goal is to be able to defend these assumptions with corresponding feature maps and architecture illustrations. All remaining activities, such as the feature analysis and architecture reconstruction, should be sized accordingly. However, in all our application cases it consumed too much calendar time to retrieve the necessary development and maintenance costs data, so that we started the technical analysis in parallel.

The technical analysis re-documenting the high-level structure of the architectures and the used technologies can often be executed within 1-2 hours per product if the product's architect is available for an interview. This of course does not allow to avoid any technological issues when merging or integrating products. Instead, it roughly serves as a indicator of the general technical feasibility of an integration.

Q4: What is the projected return on investment for SPLs in our cases? Working hypothesis H4 was: *"for cases with high feature overlap an SPL approach would pay off within five years"*. We were only able to calculate the SPL business case in two of our four application cases. For case 2, it was computed for one reuse scenario that the shared approach would pay off within three years, whereas for case 4, it was computed for one reuse scenario that the shared approach would pay off within seven years. For the other cases, it was not possible to come up with a business case calculation because of missing development cost numbers or lacking stakeholder interest.

Given our result, we have to *reject H4*, because for case 4 the SPL approach would need more than five years to pay off. Especially for a low number of similar products and moderate technological incompatibility, it may be financially beneficial to decide against developing a shared SPL platform and to integrate the products through less expensive means.

Q5: How does the domain analysis method need to be adapted for different cases? Working hypothesis H5 was stated as: *"more than 80 percent of the method's steps, templates and reuse criteria can be reused across all application cases"*. Table 4 summarizes how we executed each of the method's steps for each application case. Each application case had different characteristics caused by different goals, contexts, and stakeholders.

#	Activity	1: Industrial Control Systems	2: Automation PC Tools	3: Device Engineering Tools	4: Enterprise Information Systems
1.	List products & information sources	Selected products in active development, gathered documentation and contacts, agreed scope with sponsors of the study.	Received list of tools to analyze from sponsors of the study.	Received list of available tools from sponsors of the study, reduced focus to particular subdomain.	Received list of products and information sources from sponsors of the study.
2.	Establish criteria for reuse potential	Used all criteria.	Used all criteria, except organizational fit due to only two units particpating, focused on features and architecture.	Used all criteria, focused on future roadmaps, lessons learned, and features.	Used all criteria, focused on features, architecture, and added business processes and data model.
3.	Collect and analyze documentation	Analyzed user and configuration manuals, power point presentations, and few architecture documentations.	Analyzed user manuals and presentations about the products.	Analyzed manuals and architecture documenation.	Analyzed product presentations, manuals, database schemas, UML use case descriptions.
4.	Prepare initial interview documents	Created initial feature list based on analysts' domain knowledge. Constructed initial architecture models based on documenation.	Only the questionaire has been prepared before the interview.	Created initial feature list based on documentation, drafted architecture overview maps per product.	Created initial feature list, created data mapping table.
5.	Conduct interviews	Interviewed lead architects for each product as well as several product managers.	Interviewed three project managers and developers.	Interviewed lead architects, product managers, and team managers per product.	Interviewed architects and products manages, split interview sessions by business processes.
6.	Evaluate results and identify opportunities	Created feature support matrix, highlighted potential providers and consumers for reuse.	Identified 10 of 50 feature subdomains with high reuse potential as well as three short-term reuse scenarios.	Aligned technology roadmaps, started project for designing a common platform.	Found overlap in business processes, but differences in techn.; created 4 reuse scenarios.
7.	Create business case	Deferred business case calculation due to complexity.	Created business cases for three short-term reuse scenarios.	No business case calculation due to partly missing business numbers and a lack of analysis time.	Projected possible return on investement for a common platform after seven years.

Table 4 Execution of the steps per application case

Nevertheless, there were significant *similarities* between the application cases. We were able to reuse most of the identified reuse criteria between application cases, although we adjusted priorities and depth depending on stakeholder inputs. For example, in application case 2 the stakeholders did not want to discuss organizational issues. In application case 3, the focus was on future developments, features, and lessons learned w.r.t. reuse within the participating units. Thus, we investigated the product roadmaps more deeply and asked the architects and product managers about the knowledge of software reuse and the accompanying processes. In application case 4, two new reuse criteria have been added to the analysis: business process and data model.

Our questionnaires had some common reusable content, such as questions about the organization, development process, and technologies. Questions about the features had to be completely replaced per application case. In some cases the mere existence of a feature was asked for, while in other cases (e.g., for complex features), we created several questions about a single feature to let the stakeholders better explain the extent of the feature support in their product. In our experience, it was difficult to generalize the questions about features across the analyzed domains.

In each case we successfully used the FMC notation for architecture redocumentations. We also interviewed stakeholders with similar roles across application cases, namely architects and product managers. Feature tables were iteratively refined in each cases and the final presentation format was similar.

Differences occurred in other tasks. The scoping activity for determining the products under analysis was sometimes driven by the contracting stakeholders, but sometimes also from taking a broader, independent look into the company's product portfolio. As stated for Q3, the granularity of analysis varied from case to case. This also impacted the phase "identify opportunities", which in some cases relied heavily on the feature tables and in other cases relied more on the architecture maps. The business case calculation was based on the same template in each case but needed to be adjusted per case to account for peculiarities, such as high migration costs or limited feature overlap.

In conclusion, our working hypothesis H5 was supported as we were able to reuse almost all method steps as well as notations and most of the templates across application cases.

5.2 Threats to validity

This section discusses the trustworthiness of our results by describing internal, construct, and external threats to validity.

Internal Validity Internal validity is affected when a researcher has not properly controlled interfering variables or was not aware of them at all. This can influence the outcomes of the study, which can lead to false cause/effect relationships. Being an exploratory case study, our investigation had a low level of controlling interfering factors, such as bias from the selection of interviewed stakeholders, inappropriate use of analysis methods, or unrevealed political factors influencing reuse decisions.

Our findings for Q1-Q5 were based on the documents provided by the stakeholders and their answers in interview sessions. We cross-checked both documents and interview recordings for contradictions, to reduce the amount of false or incomplete information. Still our findings may be incomplete and missing important factors due to the limited time for analysis.

The required efforts for each application case were partially time-boxed by our product stakeholders, who did not want to invest more time and money to get the analysis results. The stakeholders influenced the planned efforts based on their knowledge of the respective system and expected/required deliverables.

To a certain extent we controlled the factor of the analysts influencing the results and having learning effects as different analysts were involved in each case. Still learning effects were visible, as we made minor changes and improvements to the domain analysis method after each application case.

Construct Validity Construct validity is affected, if the studied issues do not really represent what the research wanted to investigate, i.e., how representative the case setup was for learning about domain analyses. Our application cases were all drawn from practical situations, i.e., software products being sold to ABB customers, and thus can be considered realistic cases for a domain analysis.

Our researchers executing the analysis can be seen as consultants for the affected ABB units and thus are typical users of a domain analysis method. However,

our analysis was carried out in a large, corporate, and decentralized organization, so that the findings may only be valid for similar circumstances. In smaller companies or more centralized organizations the outcomes to our research questions Q1-Q5 may be different.

A threat to construct validity is that the measured results are not sufficiently adequate and accurate. For most research questions, we did not use quantitative measures, but instead reported qualitative findings, such as the different kinds of actions resulting from the domain analysis or how the domain analysis was adapted for the different cases. Due to confidentiality reasons, we cannot disclose exact percentages of feature overlap or precise return on investment calculations. We also did not create a quantitative measure for feature granularity, which we deem as future work.

The FMC notation is not as well-known as the UML notation, but has been used in other large companies, such as SAP for similar purposes. The business case calculation formulas were initially taken from literature, where they had be reported being derived from similar investigations. Our whole domain analysis method can be considered typical for the goal of identifying reusable components and working towards an SPL as it is similar to other methods, such as SEI Product Line Technical Probe and PulSE-Eco, cf. Section 6.

External Validity External validity is affected if the studied cases cannot be generalized, so that the findings are not valid for similar cases. We argue that our findings are generalizable to similar software tools and reuse situations. We analyzed real, non-trivial software products. Our investigation spanned four different cases so that the bias from a single application case can be reduced to some extent, although no statistical evidence is given.

The application cases included software products of different sizes ranging from 10 KLOC up to several million lines of code. This implies that our results are applicable for range of software systems, especially considering the slightly different outcomes per case discussed in Section 5.1.

However, all our application cases originated from a single corporate organization and may thus not be easily transferable to other organizations. We did not consider software for embedded systems in any of the application cases, but only analyzed PC-based software. We only applied a single domain analysis method and were not able to compare our method to other similar methods.

5.3 Lessons learned

In the following we report on several lessons learned from the four application cases of our domain analysis approach. The lessons are intended to support similar, future investigations.

Required efforts Table 5 provides an overview of the efforts required in each case study. In case 1, the efforts for Step 6-7 are very high compared to the other cases. This has mainly two reasons. First, the *Industrial Control Systems* in application

case 1 were the most complex products we analyzed. Second, we applied the domain analysis the first time in this application case. We extensively compared the products with regard to all reuse potential criteria and created a detailed documentation of the high-level architecture of each product. In case 2, we had only a very small budget and, therefore, prioritized the reuse potential criteria and were very efficient during the interviews, which resulted in the low efforts for Step 6-7. In case 3 and 4, we customized and prioritized the reuse potential criteria, and invested more time in the preparation of the interviews (Step 1-4). In case 4, one product was added during the analysis based on the intermediate results, which increased the efforts of Step 6-7 compared to case 2 and 4.

There are many factors that influence the time required for a domain analysis and Table 5 can only give some orientation. However, the table shows that one can estimate per product up to one week for Step 1-4 and up to two days for the interviews (Step 5). Evaluation and business case (Step 6-7) should be planned with two to three weeks for similar cases as 2, 3, and 4.

Application case	**1: Industrial Control systems**	**2: Automation PC Tools**	**3: Device Engineering Tools**	**4: Enterprise Information Systems**
Preperation (Step 1- 4) per product	1-3 days	1-2 days	1-5 days	1-5 days
On-site interview session (Step 5) per product	1-2 days	1-2 days	1-2 days	2 days
Evaluation and business case (Step 6-7)	15 weeks	1 week	3 weeks	2 weeks

Table 5 Overview of required effort per case study.

Consider migration costs in SPL business cases Existing business case calculations that we found in the literature often did not incorporate the costs for migrating existing installations to new reusable components. While these costs might be marginal in some domains where simple software updates are required, this factor is especially crucial in the domain of industrial automation. The software products are integrated into complex hardware environments and require significant engineering work that is sometime expensive to adapt to new software versions. Nevertheless we learned that a business case calculation can in some cases be a valuable argument to facilitate the discussions on SPL adoption with higher management.

Follow a reactive SPL approach for established products Krueger et al. [36] distinguish between (i) a proactive SPL approach, where a top-down design for a domain is performed, (ii) an extractive SPL approach, where commonalities are factored out of existing products (bottom-up), and (iii) a reactive SPL approach, where core assets are created upon new requirements for a new product. In all of our four application cases, we targeted a combination of an extractive and a reactive approach. As for each case a number of existing products were analyzed, we intended to extract existing common functionality, where possible. However, the domain of industrial automation systems is rather stable and established compared

to consumer-oriented domains. To protect the investment for installed products, a reactive approach creating common core assets for products upon novel common requirements is economically most viable. This finding was supported by all of our analyzed application cases, which favored smaller, short-term integration scenarios.

Do not rely on code analysis for reconstruction When we started the domain analysis, we expected that a static or dynamic code analysis could be helpful for architecture reconstruction. Besides a picture of the dependencies among the high-level subsystems, it would also give an up-to-date view and overcome outdated architecture documents. However, several factors counted against this in our case. First, there is a different level of abstraction needed for domain analysis, i.e., a logical, feature-oriented view, compared to an implementation-oriented view. Second, industrial software systems are implemented with diverse programming languages, technologies, and third-party components thus complicating tool-supported code analysis. Third, in our case there are no code clones between the products since the development units operate rather independently. Fourth, it is more efficient to interview the architects about the architecture than setting up a working build environment, configuring a static analysis tool, and interpreting the results. For future domain analyses, we would thus abstain from automated code analysis if large-scale systems are under analysis. An interesting pointer for future work are feature location techniques [44] that could be assessed for their industrial maturity.

Use FMC for architecture reconstruction We learned in our domain analysis that FMC is a valuable notation for high-level architecture illustration suitable for initial technical reuse assessments. The unified notation forces architects to reformulate their architecture descriptions in common terms, which is by itself already a benefit. The visualization is more streamlined than the UML, thus more suitable to compare multiple complex systems visually. The notation resonated well with the participating architects who liked to think in predefined templates. The documentation gave a fresh view on the products and is useful for communication purposes even beyond the domain analysis and SPL assessment. For some suggested reuse case, the diagrams proved to be helpful to analyze the technical challenges and required adaptations.

Create a shared understanding as important result In general, when analyzing a number of similar software products which have been developed rather independently, it is a value by itself to create a shared understanding about the products between the stakeholders. This is facilitated in a domain analysis through the uniform treatment of each product and the alignment of domain-specific terminology. Certain terms might be used different in different products although they represent similar concepts. During our interviews we had numerous discussions about the feature terminology, which proved to be valuable to create a shared understanding and see similarities that were unclear before.

Keep in mind the human factor Our approach is heavily depended on the input provided from the interviews. This might distort the analysis results, if the interview partners do not share relevant information. This might be intentional out of

fear of the impact of the domain analysis on the current development. But it also might be unintentional, because the interviewees have become blind for certain relevant issues that they take for granted. Thus, the interviews require experienced moderators who can challenge given answers and call for clarifications. In general the output from the interviews must be reflected critically given other information sources, such as documentation, source code, or third parties.

6 Related Work

There are a number of case studies on SPL adoption reported in literature [11, 37, 31]. However, they often focus on lessons learned from setting up an SPL in a given organization, but do not necessarily detail the domain analysis step and how the potential for an SPL was assessed. We summarize the findings of several studies with focus on SPL potential assessment similar to ours in the following.

Northrop et al. report on experiences from applying the SEI Product Line Technical Probe [39, 29] (PLTP) in nine organizations. Four of the nine investigated organizations made "aggressive progress in their product line efforts" after executing the PLTP, while the others abandoned, deferred, or just initiated an SPL adoption approach. From their experience, most organizations underestimate the required management commitment and involvement. Especially engineering organizations tend to focus on technical and implementation issues for SPLs, while investing too little effort in proper scoping and business case development. These results may correspond to our finding that a business case calculation was in most of our cases less interesting for the stakeholders. Additionally, stakeholder interviewed during the PLTP showed a higher buy-in to the product line effort, and the sponsors of the PLTP were grateful to receive a product line readiness assessment. Similarly, our domain analysis execution were appreciated by our stakeholders.

Knauber et al. [32] applied product line concepts according to the PuLSE method in six small organizations with 2 to 11 developers each. In particular, they applied the PuLSE-Eco method for domain potential assessment. They learned that it was crucial in the investigated cases to convince key decision makers of the product line benefits to be successful. Similar to our study they analyzed the granularity of feature analysis and reported that PuLSE-Eco is flexible for this task and can be easily adapted for different granularities. They found a lack of architecture documentation in most organizations, and also experienced resistance to changing the existing products due to effort concerns. The method per-se was perceived time-consuming but appreciated by the stakeholders.

PuLSE-Eco was also applied at TESTO AG, a company creating portable electronic measurement instruments, by Schmid et al. [47]. In this case, PuLSE-Eco was used as a product line feasibility studies, thus in a similar manner as in our application cases. In addition to the described method, the authors analyzed future products together with product management, which we covered partially by including questions about the product roadmap into our interview questionnaires. They had to re-document the existing system architecture as in our approach and facilitated the domain analysis through an architecture workshop. The approach also included creating a domain model in which novel requirements were integrated. Lessons learned included the usefulness of an architecture workshop, the

developer disinterest in domain modeling and architecting, and the importance of training as part of an SPL adoption approach.

Birk et al. [4] analyzed product line adoption approaches in five organizations among them Hewlett-Packard and Bosch. Although they do not focus on SPL potential assessment, they recommend to perform a through domain analysis before starting an SPL approach. Three of the analyzed organizations explicitly modeled product line requirements, which can be output of a domain analysis, while the others relied on experienced architects and domain experts to develop an SPL core asset base. They also recommended to create architecture documentations using well-established notations to facilitate the SPL adoption approach.

The Family Evaluation Framework [37] (FEF) was developed over six years in several European projects involving academic and industrial partners. It provides four dimensions (business, architecture, process, organization) each of which are divided into five maturity levels. The evaluation profile resulting from an SPL potential assessment can be used to support the decision for an SPL. While the framework is based on industrial experience, we are not aware of any replicated case studies applying and analyzing the framework. The FEF description uses an artificial application case for explanation, but not for investigation.

The present article is an extended analysis of the application cases 1 and 2 (detailed in Koziolek et al. [34]) and application case 3 and 4 (detailed in Domis et al. [16]). The former papers focused more on the description of the application cases, which are here summarized in the appendix for self-containment. In this paper, we have formulated five research questions and according hypotheses, whose answers we subsequently discussed. This paper thus analyzes and compares all cases more deeply. The main contributions are the answers to the research questions and the lessons learned. We have also refined the steps of our SPL potential assessment approach compared to the former papers and discussed threats to validity as well as related case studies in much greater detail.

7 Conclusions

We applied a domain analysis approach featuring architecture evaluation in four large-scale application cases from the domain of industrial automation. The domain analysis resulted in an SPL approach only in one of the cases, whereas in the other cases the stakeholders favored smaller integration scenarios. We learned that the business flexibility is a factor complicating development approaches across BUs and that business case calculation were either difficult to make or less interesting for the stakeholders. Using the FMC notation for architecture reconstruction proved to be successful for the given purpose.

Our lessons learned are intended to improve future domain analysis investigations in similar complex domains at other companies. We encourage to use similar tools and notations as in our analyses to support a decision for SPL engineering efficiently. Our results can also stimulate researchers to improve existing domain analysis methods. Such methods should recognize strict cost constraints in industry as well as technical constraints such as legacy systems complicating automated code analyses.

In future work, we will extend and refine our domain analysis approach and apply it on additional cases. We intend to create templates and models that can be reused to speed up future applications. The approach should be extended to design an SPL reference architecture for the analyzed domain to enable implementation of the found reuse potential. To further increase the understanding and usefulness of SPL potential analysis, a number of research questions remain to be answered:

- Is it possible to define stopping criteria for a domain analysis with the stakeholders to potentially shorten the analysis time?
- Which arguments can convince decision makers best for making an SPL decision?
- Can the complexity of an expected SPL adoption be quantified in a more detailed manner?
- How can roadmaps of loosely coupled development units be aligned to support an SPL approach?
- Can (semi-)automatic feature mining approaches shorten the time to create a feature list for a complex industrial software system?
- Can (semi-)automated approaches compare software architectures in the context of a domain analysis to support SPL potential analysis?

References

1. Ahnassay, A., Bagheri, E., Gasevic, D.: Empirical evaluation in software product line engineering. Tech. rep., Laboratory for Systems, Software and Semantics, Ryerson University (2013). URL http://ls3.rnet.ryerson.ca/wp-content/uploads/2013/08/TR-LS3-130084R4T.pdf
2. America, P., Thiel, S., Ferber, S., Mergel, M.: Introduction to domain analysis. http://www.ibrarian.net/navon/paper/Introduction_to_Domain_Analysis_Pierre_America__e.pdf?paperid=15855122 (2001)
3. Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., DeBaud, J.M.: Pulse: A methodology to develop software product lines. In: Proceedings of the 1999 Symposium on Software Reusability, SSR '99, pp. 122–131. ACM, New York, NY, USA (1999). DOI 10.1145/303008.303063. URL http://doi.acm.org/10.1145/303008.303063
4. Birk, A., Heller, G., John, I., Schmid, K., von der Massen, T., Muller, K.: Product line engineering, the state of the practice. Software, IEEE **20**(6), 52–60 (2003). DOI 10.1109/MS.2003.1241367
5. Boeckle, G., Clements, P., McGregor, J., Muthig, D., Schmid, K.: Calculating roi for software product lines. Software, IEEE **21**(3), 23 – 31 (2004). DOI 10.1109/MS.2004.1293069
6. Boehm, B.W., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R., Steece, B.: Software Cost Estimation with Cocomo II with Cdrom, 1st edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (2000)
7. Bosch, J.: Software product lines: Organizational alternatives. In: Proceedings of the 23rd International Conference on Software Engineering, ICSE '01, pp. 91–100. IEEE Computer Society, Washington, DC, USA (2001). URL http://dl.acm.org/citation.cfm?id=381473.381482
8. Breivold, H.P., Larsson, S., Land, R.: Migrating industrial systems towards software product lines: Experiences and observations through case studies. In: Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications, SEAA '08, pp. 232–239. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/SEAA.2008.13. URL http://dx.doi.org/10.1109/SEAA.2008.13
9. Buhr, R.J.A.: Use case maps as architectural entities for complex systems. IEEE Trans. Softw. Eng. **24**(12), 1131–1155 (1998). DOI 10.1109/32.738343. URL http://dx.doi.org/10.1109/32.738343

10. Capilla, R.: Using map for recovering the architecture of web systems of a spanish insurance company. Software Technology and Engineering Practice, International Workshop on **0**, 92–101 (2005). DOI http://doi.ieeecomputersociety.org/10.1109/STEP.2005.33
11. Carnegie Mellon University - Software Engineering Institute: Product Line Hall of Fame. http://splc.net/fame.html (2013). Last visited 2013-01-21
12. Carnegie Mellon University - Software Engineering Institute: Software Product Lines. http://www.sei.cmu.edu/productlines/ (2013). Last visited 2013-01-21
13. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI: Guidelines for Process Integration and Product Improvement. Addison-Wesley (2003)
14. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2001)
15. Davis, T.: The reuse capability model: a basis for improving an organization's reuse capability. In: Software Reusability, 1993. Proceedings Advances in Software Reuse., Selected Papers from the Second International Workshop on, pp. 126 –133 (1993). DOI 10.1109/ASR.1993.291710
16. Domis, D., Sehestedt, S., Gamer, T., Aleksy, M., Koziolek, H.: Customizing domain analysis for assessing the reuse potential of industrial software systems. In: Proc. 18th Internal Software Product Line Conference (SPLC2014), Industry Track. ACM (2014)
17. Ducasse, S., Pollet, D.: Software architecture reconstruction: A process-oriented taxonomy. Software Engineering, IEEE Transactions on **35**(4), 573–591 (2009)
18. Eisenbarth, T., Simon, D.: Guiding feature asset mining for software product line development. In: Proceedings of the International Workshop on Product Line Engineering: The Early Steps: Planning, Modeling, and Managing, Erfurt, Germany, Fraunhofer IESE, pp. 1–4 (2001)
19. Eisenhardt, K.M.: Building theories from case study research. Academy of management review **14**(4), 532–550 (1989)
20. Fairbanks, G.: Just Enough Software Architecture: A Risk-Driven Approach, 1st edn. Marshall & Brainerd (2010)
21. Frenzel, P., Koschke, R., Breu, A.P.J., Angstmann, K.: Extending the reflexion method for consolidating software variants into product lines. In: Proceedings of the 14th Working Conference on Reverse Engineering, WCRE '07, pp. 160–169. IEEE Computer Society, Washington, DC, USA (2007). DOI 10.1109/WCRE.2007.28. URL `http://dx.doi.org/10.1109/WCRE.2007.28`
22. Ganesan D.; Knodel, J.: Identifying domain-specific reusable components from existing oo systems to support product line migration. In: Proceedings First International Workshop on Reengineering towards Product Lines, R2PL 2005, Pittsburgh, Pennsylvania, USA, pp. 16–20 (2005)
23. Groene, B.: Introducing architecture modeling at a big software product company. In: Proceedings Praxisforum Modellierung 2012 (2012). URL `http://qfam.gi.de/fileadmin/user_upload/PraxiforumModellierung2012/Introducing-architecture-modeling-at-a-big-software-product-company_Groene.pdf`
24. Guo, G.Y., Atlee, J.M., Kazman, R.: A software architecture reconstruction method. In: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1), WICSA1, pp. 15–34. Kluwer, B.V., Deventer, The Netherlands, The Netherlands (1999). URL `http://dl.acm.org/citation.cfm?id=646545.696370`
25. Harhurin, A., Hartmann, J.: Service-oriented commonality analysis across existing systems. In: Software Product Line Conference, 2008. SPLC '08. 12th International, pp. 255 –264 (2008). DOI 10.1109/SPLC.2008.19
26. Hariri, N., Castro-Herrera, C., Mirakhorli, M., Cleland-Huang, J., Mobasher, B.: Supporting domain analysis through mining and recommending features from online product listings. IEEE Trans. Softw. Eng. **39**(12), 1736–1752 (2013). DOI 10.1109/TSE.2013.39. URL `http://dx.doi.org/10.1109/TSE.2013.39`
27. Holmes, R., Walker, R.J.: Systematizing pragmatic software reuse. ACM Trans. Softw. Eng. Methodol. **21**(4), 20:1–20:44 (2013). DOI 10.1145/2377656.2377657. URL `http://doi.acm.org/10.1145/2377656.2377657`
28. John, I., Knodel, J., Lehner, T., Muthig, D.: A practical guide to product line scoping. In: Software Product Line Conference, 2006 10th International, pp. 3 –12 (2006). DOI 10.1109/SPLINE.2006.1691572
29. Jones, L.G., Northrop, L.M.: Clearing the way for software product line success. IEEE Softw. **27**(3), 22–28 (2010). DOI 10.1109/MS.2010.71. URL `http://dx.doi.org/10.1109/MS.2010.71`

30. Kazman, R., Carrière, S.J.: Playing detective: Reconstructing software architecture from available evidence. Automated Software Engineering **6**(2), 107–138 (1999)
31. Khurum, M., Gorschek, T.: A systematic review of domain analysis solutions for product lines. J. Syst. Softw. **82**(12), 1982–2003 (2009). DOI 10.1016/j.jss.2009.06.048. URL http://dx.doi.org/10.1016/j.jss.2009.06.048
32. Knauber, P., Muthig, D., Schmid, K., Widen, T.: Applying product line concepts in small and medium-sized companies. IEEE Softw. **17**(5), 88–95 (2000). DOI 10.1109/52.877873. URL http://dx.doi.org/10.1109/52.877873
33. Knoepfel, A., Groene, B., Tabeling, P.: Fundamental Modeling Concepts: Effective Communication of IT Systems. Wiley (2006)
34. Koziolek, H., Goldschmidt, T., de Gooijer, T., Domis, D., Sehestedt, S.: Experiences from identifying software reuse opportunities by domain analysis. In: Proc. 17th Internal Software Product Line Conference (SPLC2013), Industry Track. ACM (2013)
35. Koziolek, H., Weiss, R., Doppelhamer, J.: Evolving Industrial Software Architectures into a Software Product Line: A Case Study. In: Proc. 5th Int. Conf. on the Quality of Software Architecture (QoSA'09), *LNCS*, vol. 5581, pp. 177–193. Springer (2009). DOI http://dx.doi.org/10.1007/978-3-642-02351-4
36. Krueger, C.W.: Easing the transition to software mass customization. In: Revised Papers from the 4th International Workshop on Software Product-Family Engineering, PFE '01, pp. 282–293. Springer-Verlag, London, UK, UK (2002). URL http://dl.acm.org/citation.cfm?id=648114.748909
37. van der Linden, F.J., Schmid, K., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer (2007)
38. Neighbors, J.M.: The draco approach to constructing software from reusable components. Software Engineering, IEEE Transactions on (5), 564–574 (1984)
39. Northrop, L., Jones, L., Donohoe, P.: Examining product line readiness: Experiences with the sei product line technical probe. https://resources.sei.cmu.edu/asset_files/Presentation/2005_017_001_23904.pdf (2005)
40. Pinzger, M., Gall, H., Girard, J.F., Knodel, J., Riva, C., Pasman, W., Broerse, C., Wijnstra, J.: Architecture recovery for product families. In: F. van der Linden (ed.) Software Product-Family Engineering, *Lecture Notes in Computer Science*, vol. 3014, pp. 332–351. Springer Berlin Heidelberg (2004). DOI 10.1007/978-3-540-24667-1_26. URL http://dx.doi.org/10.1007/978-3-540-24667-1_26
41. Pohl, K., Bckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer (2005)
42. Poulin, J.S.: Measuring Software Reuse: Principles, Practices, and Economic Models, 1st edn. Addison-Wesley Professional (1996)
43. Prieto-Diaz, R., Arango, G.: Domain Analysis and Software Systems Modeling. IEEE Computer Society Press (1991)
44. Rubin, J., Chechik, M.: A survey of feature location techniques. In: Domain Engineering, pp. 29–58. Springer (2013)
45. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Engg. **14**(2), 131–164 (2009). DOI 10.1007/s10664-008-9102-8. URL http://dx.doi.org/10.1007/s10664-008-9102-8
46. Schmid, K.: A comprehensive product line scoping approach and its validation. In: Proceedings of the 24th International Conference on Software Engineering, ICSE '02, pp. 593–603. ACM, New York, NY, USA (2002). DOI 10.1145/581339.581415. URL http://doi.acm.org/10.1145/581339.581415
47. Schmid, K., John, I., Kolb, R., Meier, G.: Introducing the pulse approach to an embedded system population at testo ag. In: Proceedings of the 27th International Conference on Software Engineering, ICSE '05, pp. 544–552. ACM, New York, NY, USA (2005). DOI 10.1145/1062455.1062552. URL http://doi.acm.org/10.1145/1062455.1062552
48. Simos, M., Creps, D., Klingler, C., Levine, L., Allemang, D.: Organization domain modeling (odm) guidebook, version 2.0. Tech. Rep. STARS-VC-A025/001/00, Lockheed Martin Tactical Defence Systems, United States of America (1996)
49. Stoermer, C., O'Brien, L.: Map - mining architectures for product line evaluations. In: Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on, pp. 35 –44 (2001). DOI 10.1109/WICSA.2001.948405
50. Van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. Sigplan Notices **35**(6), 26–36 (2000)
51. Yin, R.K.: Case Study Research: Design and Methods, 5th edn. Sage Publications Ltd. (2013)

A Application Cases

This section provides a detailed description of the application of our Software Product Line potential assessment approach and is intended to facilitate executing similar approaches.

A.1 Application Case 1: Industrial Control Systems

Industrial control systems monitor and control automation processes in a variety of different industrial sectors (e.g., power generation, chemical plants, or substations). ABB has several such systems in its portfolio, which address specific industrial subdomains and have been incorporated in former company mergers.

For **step 1** of our approach we conducted an internal survey of existing ABB prodcuts, which resulted in a list of products, from which we removed such products that are no longer in active development. In **step 2** we decided to use almost all of the generic reuse criteria formerly defined. We also decided to focus on high-level features only. Since these are large-scale systems, there is no fine-granular break-down of the features as the analysis rather targets the systematic reuse of whole subsystems. According to the selected reuse criteria, we started to create the questionnaire for the later interviews.

In **step 3**, we found that for some systems extensive architectural documentation was available. We used this information to come up with a sketch of the architecture in the FMC notation. We created an initial list of 30 features in **step 4**, which we based on our own domain knowledge from working with similar systems (Fig. 2). The list was reviewed by multiple ABB-internal domain experts, who did not work on the products themselves.

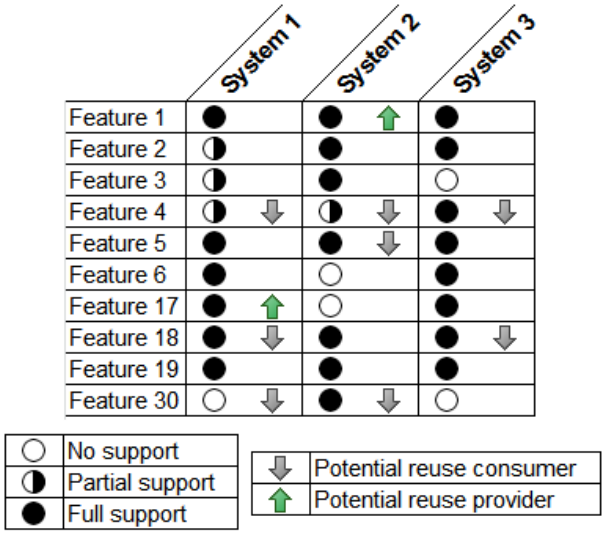

Fig. 2 Anonymized excerpt of the feature support matrix for the control systems case study

We conducted on-site interviews for each product ranging from half a day to three days (**step 5**) depending on the product's size. As the last part of each product interview, we created and extended our architectural sketches. Due to the different sizes of the systems and the varying depth of the interviews, the architecture illustrations created during the interviews had different granularity. In subsequent document analyses and clarifications via phone the architecture illustrations of all systems were brought to a comparable level (example in Fig. 3. Based on the architecture illustrations we were capable of visualizing the data flows through

the components, which are omitted here for confidentiality. For example, one data flow included a sensor value read from an industrial device and propagated through the system to be shown on the operator screen.

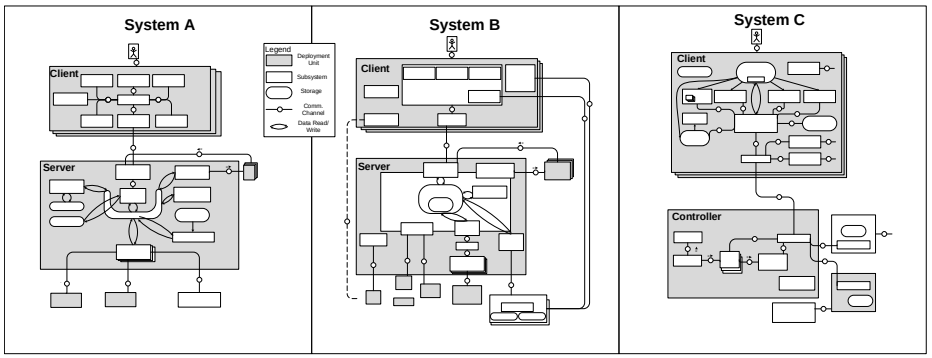

Fig. 3 Architecture map for two of the analyzed systems in application case 1 (simplified)

The next **step 6** was to identify reusable features and subsystems based on the information from system documentation and the interviews transcripts. We executed this step offsite after all interviews had been finished. We used a product feature support matrix (Figure 2). The matrix shows for each product whether a feature from the feature list is supported. We also indicated whether a new implementation of a feature or a replacement of an existing subsystem is currently realistic. In this case, we marked the product as a potential reuse consumer for that feature. Products that have a high-quality and generic implementation of a feature are highlighted as potential reuse providers.

Furthermore, we analyzed the cohesion and coupling of the subsystems that implement a feature to find out whether subsystems could be easily extracted from the implementations to serve as core assets. We also colored subsystems in different products that implement similar features. Using the matrix we identified several realistic new reuse opportunities among the products. We documented the technical feasibility for each reuse opportunity based on the information from the interviews and the available documentation. The organizational feasibility of each reuse opportunity was also analyzed similarly.

In parallel, we attempted to create a economic analysis (**step 7**), which proved to be too complicated in this case. Besides the development and maintenance cost of each product, also the migration costs for existing installations need to be considered if a subsystem is replaced. This is especially critical in the industrial control system domain, since these systems include numerous hardware devices and engineering work to customize the system. We were thus not able to create a reasonable business calculation in this domain, because the inputs for such a calculation were difficult to gather.

In summary, in this application case, our approach enabled us to create a thorough, albeit high-level, assessment of reuse opportunities. It showed to scale to complex systems and still provides useful results despite the relatively short feature list. In one product, the high-level architecture overviews provided by our architecture illustrations made the respective business unit rethink architectural trade-offs. The architecture illustrations also were instrumental in stimulating discussions among the product architects and visualize the functional overlap.

Our initial intuition about high functional overlap between the various systems was confirmed by our analysis. However, an actual re-engineering of the systems towards reusing subsystems still seems difficult due to the expected high migration costs. These costs are a result of the high configurability of the products, which would require high efforts to update customer configurations. Therefore, the focus of reuse opportunities in this domain lies more in guiding future development towards a better alignment and systematic reuse of subsystems.

A.2 Application Case 2: Automation PC Tools

In the second application case, we analyzed a set of PC-based commissioning and monitoring tools for industrial automation systems. Commissioning tools load software applications onto a device and set application parameters. Monitoring tools observe an automation system, check and predict its operating conditions, and raise alarm events. During their development history, the developers of the tools have changed. The product management was aware of the tools in the families as well as of the potential functional overlaps, which could provide a potential for a software product line and for reducing future development and maintenance efforts.

The time frame for this SPL potential analysis was three working weeks. We used one week for preparation (step 1-4), one week on site for interviews with the developers (step 5), and one week for potential identification and economic analysis (step 6-7). For **step 1** the developers provided an initial list of all tools. We excluded tools, for which a low reuse potential could be assumed or which are no longer maintained. From the generic reuse criteria we excluded organizational criteria, since the tools were owned by only two units, which were already collaborating successfully **step 2**.

We gathered user manuals and architecture documents in **step 3**, which were instrumental to build up an initial feature list (**step 4**). Nevertheless, due to our limited former domain knowledge, we had to significantly adapt the list in **step 5**. During this step, we interviewed three project managers and developers, who provided a business and technical perspective. There was no explicit architecture role defined for these products. We used a feature map [46] immediately in the interviews (Figure 4). Compared to Schmid's approach [46], we only performed *product line mapping* and a short domain potential assessment, but no quantitative reuse infrastructure scoping, due to effort limitations.

The last session of each interview, was the architecture reconstruction. We started from a black box view of a tool with its interfaces, refined this into a view showing clients and servers, for example, and modeled the next levels of components and internal data flow. The data-flow oriented view of FMC was a new notation for the developers. After a short learning phase, modeling worked well and supported an active discussion about the architecture and its concepts, such as used patterns and styles.

After completing the interviews, we finalized the assessment of the reuse and SPL potential for each domain and subdomain in the feature map (**step 6**). We added rationale for each potential rating of low, medium, and high. In the end, the feature map consisted of more than 300 features grouped into 50 subdomains and 19 domains. The potential for systematic reuse of ten subdomains, e.g., data management, are ranked high. The architecture illustration was then used to cross-check the technical feasibility of creating shared components for the identified subdomains. In the end, we identified three short-term reuse scenarios involving smaller subsets of features that can be implemented in the next releases. Additionally, six feature domains and subdomains are long-term candidates for being integrated into a common platform, but require deeper technical investigations.

To round up our investigation with an economic analysis, we performed a business case calculation **step 7**. In parallel to the preceding steps of the analysis, we collected numbers for the current development and maintenance costs of the tools, which required several weeks of calendar time, as the information had to be gathered from different sources. Then we calculated the return on investment for each reuse scenario. As originally defined, the formulas [5] were only applicable to one reuse scenario without changes. In this scenario, two components are merged for being reused in both tool families. In this case, we estimated C_{cab} as 50 percent higher than building one of the individual products. F_{cab} was 10 percent and C_{reuse} was approx. 12 percent of the costs of building the whole products from scratch. We have omitted the other costs for confidentiality reasons. The calculation showed a positive ROI after three years.

The stakeholders were particularly interested in the three short-term reuse scenarios and their business cases. They did not require the exact numbers, but were satisfied by the positive trend expected for implementing the scenarios. Thus, they planned to discuss the scenarios with the other relevant stakeholders of the tools for implementation.

Domain	Subdomain	Feature	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6	Reuse Potential
Domain 1	Subdomain 1.1	A	x	x	x	x	x	x	
		B	x	x	x				
		C	x	x	x	x		x	
		D	x	x	x		x		
			x	x	x	(x)	(x)	(x)	high
	Subdomain 1.2	E	x		x				
		F			x				
		G			x				
		H			x				
					x				low
			(x)		x	(x)	(x)	(x)	medium
Domain 2	Subdomain 2.1	I	x	x	x		x		
		J	x		x				
		K	x	x	x			x	
		L	x	x	x		x		
		M	x		x				
		N		x			x		
			x	(x)	x		(x)	(x)	medium
	Subdomain 2.2	O	x		x			x	
		P			x		x		
			(x)		x		(x)	(x)	medium
			x	(x)	x		(x)	(x)	medium

Fig. 4 Example Feature Map

A.3 Application Case 3: Device Engineering Tools

In the following application case, we focus on three different ABB device engineering tools for configuration, commissioning, and maintenance within a single industrial sector. In **step 1** of our approach these tools had been identified by management and a contact person had been named for each tool. Specific for this case we considered a new product with only a technical and market requirement specification available at the time of analysis. This is to ensure that this new product can directly benefit from and support the systematic software reuse opportunities that are identified during our analysis.

While establishing the reuse criteria in **step 2** management asked us to put specific emphasis on lessons learned from the existing tools and future developments. Finally, as management appeared to be open minded towards reuse for future developments, reuse culture was also integrated into the questionnaire. Processes and market fit were excluded in this case.

During **step 3** we collected documentation on the different engineering tools as well as the devices to be engineered, e.g., product manuals, specification sheets, and getting started guides. For the most powerful and complex tool, we also received detailed internal technical documentation, such as an architecture review of a previous tool version, from which we created a first architectural sketch. In **step 4**, we created an extensive feature list with roughly 200 entries categorized in 10 domains (Figure 4).

For **step 5**, we scheduled one and a half days for the interviews with lead architect, product manager, and team manager for each tool. We started each interview with a short tool and device demonstration, which proved to provide a good common understanding. During the interviews we spent most time for discussing and refining the feature list. To reconstruct the

architecture in the interviews we additionally relied on the CrossModel Architecture Discovery Pattern Language [20] for guidance, which was not yet known to us in the other application cases. In addition, the FMC notation was used to document the architecture.

After the interviews, we identified reuse opportunities on the feature and system level (**step 6**). The regarded tools partially have similar functionality as well as similar technology roadmaps for future releases. This finding resulted in the creation of a new project to design and implement a common underlying platform. Especially, collected lessons learned and identified high level requirements covering all tools will be useful. The lessons learned were categorized into organizational, technical, and known challenges with reuse. This allows for avoiding known issues but also to benefit from known opportunities and good decisions.

We did not create a business case (**step 7**) in this application case, as the management's priority was on future reuse opportunities. Nevertheless the involved stakeholders confirmed their perception of an expected positive return on investment by starting a new platform design and development project. Based on the positive analysis results, one additional ABB device engineering tool was added to the potential scope of a future common platform and will be analyzed soon.

A.4 Application Case 4: Enterprise Information Systems

In this application case, we analyzed a set of ABB enterprise information systems, which are used to store large data sets and to provide them to different kinds of users via corresponding views, filters, and search functions. The analysis comprised a family of five products and two individual products, which have grown independently of each other. Today, the two individual products (A and B) and one product from the family (C) are used to support similar business processes, i.e., some users require all three systems to manage related sets of data for particular tasks. The four remaining products of the family have different users and handle independent data.

To better understand the relationships of the considered enterprise information systems, we extended the domain analysis: we used a comparison of the database schemata to analyze the relation and overlap of the data managed by the products. Additionally, we conducted an evaluation of the underlying business processes to assess the tool overlap on this level as well as the impact of merging the tools on the business processes.

In **step 1**, the product managers provided the list of products to analyze as well as the contact persons. To establish the reuse criteria in **step 2** we reused the criteria prioritization from application case 3, which was agreed upon by the contact persons. The provided documentation for **step 3** included product overviews as power point presentations, user manuals, database schemata, use case descriptions in UML, and high-level architectural descriptions.

From the collected documentation, we were able to prepare around 70 percent of the final feature table in **step 4** and major parts of the architecture maps before the interviews. Similar to the feature table, we prepared a table that mapped the data model entities (in the rows) to products (in the columns). We mainly used the product overviews and user manuals of the products as input for the data entity map, because the database schemata were too detailed and complex for such an high level mapping. Based on a use case documentation, we were able to create a draft version of a business process description of one of the products.

The on-site interviews in **step 5** required one and a half day for each product. Showing the feature map for all products to the developers of a single tool created discussions. Some interviewees disputed the features supported by other products or argued that despite missing support in their own product, a work around would be possible. These discussions were valuable, as they allowed to cross check the feature table and achieve a better understanding of the differences of the products. The final feature map contained more than 250 features grouped into subdomains and domains.

Afterwards, we reviewed the prepared FMC architecture maps and completed them together with the interviewees. The original architecture documentations used proprietary notations and described the architecture on different levels of abstraction for each product. Redrawing the architectures in FMC unified notation and abstraction level for better comparability.

All interviewees understood the notation immediately and were able to provide corrections and additions.

In **step 6**, we compared the business process, features, architectures, and data schemata. For comparing the business processes, we mapped them onto a related reference business process specification. As a result, we identified a large overlap between the individual product A and product C from the family, although they might be instantiated differently. This fact was determined in three business processes which are supported by both systems but handled in a different way. For the feature map, we assessed the overlap with a subjective of rating high, medium, and low and added a rationale statement. As for the business processes, the individual product A and product C from the family showed a large feature overlap. The functional overlap inside product B and the product family was lower and mainly covers basic tool functionality such as open, edit, and store data sets.

From the architecture maps, we compared logical structuring and used technologies. The two individual products and the tool family had been implemented in different technologies, which hinders direct reuse of components between the products. Also the logical structures of the products were different, but the cores of the individual product A and the product family were similar and provided potential for a common platform. The data schemata included different names and types, but semantically similar elements. We observed the largest overlap between product A's and C's data model, while product B had a low data model overlap.

Based on these results, we elaborated several scenarios for integrating the products: 1) data integration of A and C via corresponding interfaces, 2) merging A and C into a single product, 3) building A and C based on a common platform, and 4) building a software product line for implementing product A, product B, and the family (including C). For all scenarios, we listed the most important advantages and disadvantages such as usability, complexity, maintainability, time to market, and investment cost. We also sketched migration roadmaps between the scenarios.

In **step 7**, we only performed a return on investment calculation for reuse scenarios 3) and 4). The business units provided the former development and maintenance costs for the calculations, and we estimated a 50 percent higher effort for building generic, shared parts instead of specific, separated parts (C_{cab}). Although the feature map indicated a high functional overlap, we conservatively assumed the overlap to be 50 percent after consulting the architects. With these and other assumptions, we expected a positive return on investment for scenario 3) after seven years, latest. Due to the larger number of products, the return on investment scenario 4) will likely pay off earlier. However, in both cases not the exact numbers, but the positive trend even under conservative assumptions was important for the stakeholders.

The business unit will first implement scenario 1) (data integration) for better usability of products A and C in common business processes and will align their business processes for preparing a more deep integration scenario. After the alignment of the business processes, the proposed integration scenarios well be reassessed for selecting the final solution.