Experiences from Identifying Software Reuse Opportunities by Domain Analysis

[Industrial Experience Report]

Heiko Koziolek, Thomas Goldschmidt, Thijmen de Gooijer,
Dominik Domis, Stephan Sehestedt
ABB Corporate Research
Industrial Software Systems
Ladenburg, Germany
heiko.koziolek@de.abb.com

ABSTRACT

In a large corporate organization there are sometimes similar software products in certain subdomains with a perceived functional overlap. This promises to be an opportunity for systematic reuse to reduce software development and maintenance costs. In such situations companies have used different domain analysis approaches (e.g., SEI Technical Probe) that helped to assess technical and organizational potential for a software product line approach. We applied existing domain analysis approaches for software product line engineering and tailored them to include a feature analysis as well as architecture evaluation. In this paper, we report our experiences from applying the approach in two subdomains of industrial automation.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*Domain engineering*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; D.2.9 [**Software Engineering**]: Management—*Life cycle*

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. 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. Additionally, due to the size and global distribution of a large corporate company, it is challenging to identify all functional overlaps that provide a high potential for a successful software product line (SPL).

Although functional overlap between different software products in a company's portfolio is often perceived informally [2, 16], this is not sufficient to start a costly SPL initiative. Features might be similar on a high abstraction level, but often there are subtle difference 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 maturity 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 [6] to examine readiness for product line adoption. The Family Evaluation Framework (FEF) [22] provides a four-dimensional evaluation profile to assess organizational readiness. PulSE Eco [20] provide 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.

This paper presents a seven step domain analysis approach to assess SPL potential, which we have applied on two different sets of software products within ABB. The approach includes product line scoping through identifying and analyzing a multitude of information sources, interviewing key stakeholders for product feature support and architecture analysis, as well as reuse potential assessment and the creation of SPL business cases. The approach is intended to support the decision for starting a SPL initiative. It does not result in a reference architecture, which would require an additional approach.

We applied the approach in two subdomains of industrial automation software, where ABB provides numerous software products. Depending on the complexity and size of the product the assessment becomes harder. In our cases we experienced different results in both cases and report our findings and lessons learned in the paper. The goal of the paper is to support similar analysis approaches in other companies by pointing out valuable experiences and avoid-

able pitfalls.

The remainder of this paper is structured as follows: Section 2 briefly summarizes related work, before Section 3 introduces our seven step domain analysis approach. Section 4 explains the special means of architectural reconstruction we included in the approach. Section 5 and 6 present our two application cases of the approach and the achieved results. Section 7 provides lessons learned and Section 8 concludes.

2. RELATED WORK

A general overview of software product line engineering can be found in several publications [6, 19]. Linden et al. [22] 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 [4].

Our paper focuses on domain analysis for identifying candidates for software product lines in a large and distributed corporate organization. Khurum and Gorschek offer a comprehensive overview of domains analyses for software product lines [14]. Our domain analysis is based on different (mainly manual) approaches from literature.

The SEI Product Line Technical Probe [5] examines the readiness of a company to succeed with a software product line approach. The probe consists of a series of structured interviews followed by data analysis. Our approach follows similar steps, but additionally includes a high-level architecture assessment.

The Family Evaluation Framework [22] follows the CMMI philosophy and assesses an organization's maturity for software product line engineering along different dimensions (e.g., business, architecture, process). In each dimension, an organization is ranked within five levels, so that improvement potential can be identified. Our approach has a technical focus and aims at supporting the decision for an SPL approach.

The Reuse Capability Model (RCM) [7] 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.

PulSE Eco from Schmid [20] intents 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. [13] describe how PulSE is customized for different contexts such as a light-weight scoping for limited efforts, which we have applied.

For architecture reconstruction, we used a manual, interview-based approach. There are many automated and semi-automated approaches for identifying reusable components and features: MAP [21] is an approach for semi-automatically extracting architecture information from code, identifying patterns / styles, and evaluating the potential for software product lines. An experience report [3] applies MAP for recovering the architecture of a web system.

Frenzel et al. [9] use reflexion and clone detection for identifying ad-hoc copied code between different products. Harhurin and Hartmann [12] apply a commonality analysis on formal design models for extracting product lines from existing systems. Ganesan and Knodel [10] use object oriented metrics for usefulness, readability, and testability to identify reusable components. Eisenbarth and Simon [8] propose the usage of the Bauhaus suite, a tool for different architecture analysis, to identify features appropriate for a software product line.

However, automated and semi-automated approaches are hard to apply on the heterogeneous landscape of ABB's legacy systems considering diverse technologies and programming languages. For successful systematic reuse, not only the current architectures are important, but also future plans. Thus, we decided to reconstruct product architectures manually based on interviews with system architects.

3. DOMAIN ANALYSIS

The goal of our domain analysis approach is to provide technical and economical arguments for the feasibility of establishing a software product line. The analysis involves feature modeling of the domain, comparing software architectures, mining candidate assets, and creating a business case. The output is rather a technical risk assessment and a ROI calculation, not a reference architecture or migration strategy. The output can also be a set of smaller suggested reuse scenarios, not a full-blown SPL design for a given domain. A business decision on further steps towards an SPL or systematic reuse is needed after executing our analysis approach. Our approach is intended to support this decision with a technical and economical assessment.

Our domain analysis approach comprises the following activities, which are based on existing methods and is tailored for use in our company and to focus on software architecture evaluation. The activities do not have to be executed linearly.

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 [16]. The first step is thus to gather of a list of the available products and record the respective information sources (e.g., product managers, software architects, documentation archives, competitor products). For a smaller company this step may be trivial. However, for a large corporate company with many globally distributed business units developing software for different market segments, this step can be complicated and consume a lot of calendar time. Some products might lack suitable documentation to decide whether to include them into the scope. Other products might be out of active development and only in a maintenance mode. The output of this step is an initial list of products to analyze. From this list, the most promising candidates are chosen for further analysis based on experience and expected analysis support.

2) Establish criteria for reuse potential.

In a second step, we first establish the criteria for evalu-

ating the reuse potential of the given products. The criteria are software-related (e.g., feature support, architecture, technologies, standards), process-related (e.g., release management, development process maturity, change management), and organization-related (e.g., reuse knowledge, reuse motivation, former reuse successes). 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. The list of criteria is then used to create a questionnaire that is used to interview the product managers and architects of each product.

3) Create initial feature list.

As a first step of feature modeling, this step creates a list of features in a given domain. 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. Industry standards and current technology trends also serve as input for the feature list. ABB's industrial domains are complex and can thus lead to complex feature models. For our domain analysis approach we aim at limited number of approx. 20 high-level features. Before further analysis, the feature list is reviewed by several domain experts. Questions on each feature are then integrated into the questionnaire created in step 2. Some example answers are integrated into the questionnaire as well to guide the interview partners and speed up the process. In our experience, the feature list is usually updated after each interview.

4) Collect and analyze documentation.

Besides interviewing product stakeholders, existing documentation on the products and development processes are gathered. 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. 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. This activity needs to be time-boxed to avoid wasting efforts. 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.

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, architecture, and process, we also sketch the high-level architectures (Section 4) and 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. Such workshops are executed after the initial domain analysis was completed successfully.

6) Identify opportunities.

With the information from the questionnaires gathered in step 4 and 5, we analyze the reuse potential between the products. For each product, we summarize the answers from the questionnaire in a report. We create a matrix illustrating the feature support per product. This matrix is complemented with technical comments about the extent of the feature support in a particular product. The matrix serves to find commonalities and variabilities among the products. We then compare the architecture sketches from step 5 pairwise to identify technology compatibilities, standard interfaces, reusable assets, and matching non-functional requirements. Furthermore, we classify the SPL maturity of each business unit with a list of criteria. The outputs of this step are the identification of SPL assets and/or scenarios for merging features or products.

7) Create business case.

In a final step, we create a business case for the establishment of a SPL in the given application domain with the existing products. Our calculation is based on development and maintenance costs provided by the business units and roughly follows the calculation described by Boeckle et al. [1]. We calculate the net present value and return on investment, which requires a number of assumptions, e.g., for the higher development effort for the reusable assets, or the ratio of generic software parts in the products. These assumptions can be criticized but are at least substantiated with the findings of our formerly conducted domain analysis in step 6. This eases defending the calculation.

At this stage our domain 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. ARCHITECTURE RECONSTRUCTION

4.1 Goals and Constraints

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.

The architecture reconstruction is complicated because there is no standard notation for architecture documentation. While UML is the de-facto standard for the documentation of software systems, architectural UML diagrams are hardly available for legacy systems. Even if UML is used, the diagrams for different systems might be incomparable because the language is usually applied differently for each product. From our experience some business units have no adequate architecture documentation or use semantically ambiguous PowerPoint presentations which are hard to understand on their own.

Our architecture reconstruction thus uses additional information sources besides the existing architecture documentation. During the domain analysis 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.

4.2 Selection of a Notation

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.

Architecture description languages (ADLs) are mainly used in the academic community and often lack robust tool support. Their semantical soundness is useful, but the added overhead for specification is usually not compensated with valuable analysis methods [17]. Woods and Bashroush [23] successfully used their own proprietary shapes for architecture documentation in a similar context as our domain analysis. However, these shapes lack a formal reference specification and maturity.

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 [15]. FMC block diagrams are highly abstracted 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 [11]

4.3 Fundamental Modeling Concepts (FMC)

FMC provides block diagrams for compositional structures, Petri nets for dynamic structures, and entity relationship diagrams for value range structures. For our domain analysis, we only used block diagrams for simplicity.

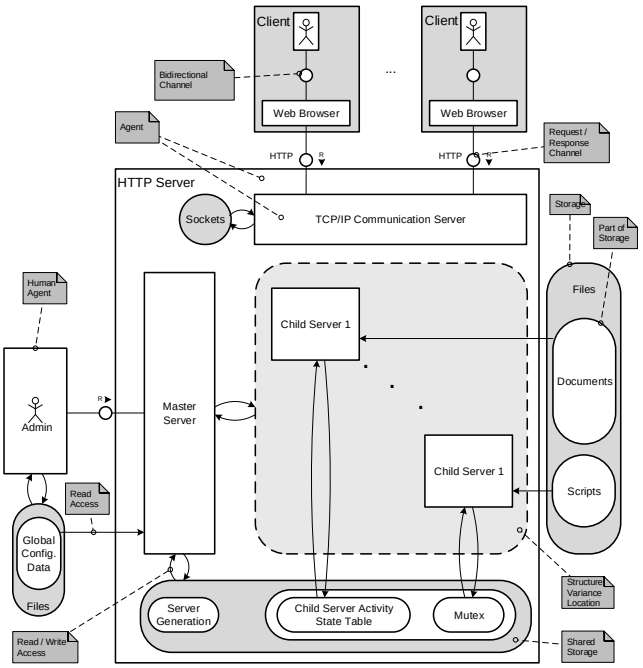


Figure 1: Fundamental Modeling Concepts (FMC) notation example[18]

Fig. 1 shows a notation overview. Squared rectangles represent active components (also called 'agents'). They comprise software components as well as human actors. Rounded rectangles represent passive components, i.e., data storages or communication channels. Directed and undirected edges represent read or write operations between active and passive components. To draw these diagrams, we used available stencils for the drawing tool MS Visio [1].

In order to express data flows, we added our own path overlays to the block diagram notation. 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. These flow paths are additionally annotated with extra-functional information, such as latencies or data volumes, where available. The path overlays need to be added with care to avoid visual clutter. We used MS Visio layers to be able to activate/deactivate multiple levels of path overlays.

The actual FMC block diagrams for the candidate assets in the scope of a domain analysis are aligned to a similar layout per product so that they are easily comparable when shown side-by-side. For example the components realizing a specific feature are placed at the same position in each diagram.

5. APPLICATION CASE 1: INDUSTRIAL CONTROL SYSTEMS

We applied our approach in two cases in different domains at ABB. Section 5 describes the application to industrial control systems and Section 6 describes the application to conditioning and monitoring PC tools.

[1]http://www.fmc-modeling.org/fmc_stencils

5.1 Domain and Challenges

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.

Industrial control systems can include millions of lines of code and with a plethora of different features and subsystems. Many features are slightly different from one industrial sector to another one. However, the basic functionality is still comparable. Currently mainly opportunistic reuse is realized among ABB's products. Therefore, the goal of this application case was to apply our approach to identify opportunities for systematically reusing subsystems between products within ABB. We analyzed several different control systems with our domain analysis method.

5.2 Domain Analysis and Architecture Reconstruction

1) List products and information sources.

As a first step we surveyed existing products in the control system domain within ABB. The initial survey resulted in a list of over 20 products. We excluded products that are no longer in active development. We gathered readily available documentation and identified contact persons for each product.

2) Establish criteria for reuse potential.

The criteria for assessing the reuse potential were based on the Reuse Capability Model [7] and the SEI Technical Probe [4]. However, as these approaches did not entirely fit our purpose and domain we adapted them accordingly. The following list gives an overview on the selected criteria.

- High-level features
- Architecture and technologies
- Use of industry standards (e.g., protocols, interfaces, components)
- Development process readiness (e.g., release management, change management)
- Organizational fit (e.g., subunits, development roles, funding models)
- Market fit (e.g., market size, amount of installed products)
- Reuse culture (e.g., knowledge on reuse, commitment for Reuse, etc.)
- Future outlook (e.g., roadmaps, future trends, technologies)

Some criteria are specific to the domain of industrial control systems. For example, as these kinds of systems have a long life-cycle it is useful to know how many installations there are (installed base), because many existing installations might prevent exchanging certain components. The *high-level features* are specific to the domain and as well the granularity of the targeted reuse. 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.

3) Create initial feature list.

We based the selection of the features to analyze on our own domain knowledge. Then, after review by experts and experiences during the interview process, we extended it to take into account features missing or needing refinement. For example, we included features, such as data monitoring, alarm handling, and operational historians. We finally ended up with 30 high-level features. Figure 3 provides an excerpt of the feature map.

4) Collect and analyze documentation.

For each of the products we gathered user documentation and architecture documentations from the architects. For some of the systems an extensive architectural documentation was available. We used this information to come up with a sketch of the architecture in our common, high-level notation (cf. Section 4). These basic sketches then helped in the interviews as they reduced the time needed for the architecture reconstruction and served as a cross check whether we understood the architecture correctly.

5) Conduct interviews.

We scheduled interviews with the lead architects of the products to get answers for the prepared questionnaires and learn the architectures. The interviews ranged from half a day to three days depending on the size of the system, our previous knowledge on the system, as well as the availability of the interviewees.

During the interview we filled out the questionnaire, went through the feature list, and created and extended the architecture 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.

Figure 2 gives an overview on the architecture illustrations of some of the products under analysis. For confidentiality reasons, the figure lacks numerous details, such as the component and connector names as well as quality attribute and technology annotations. 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.

6) Identify opportunities.

The next step was to identify reusable features and subsystems based on the information from system documentation and the interviews transcripts. We used a product feature support matrix (Figure 3). The matrix shows for each product whether a feature from the feature list is supported. A feature can either be fully supported according to our definition, partially supported, or not 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 poten-

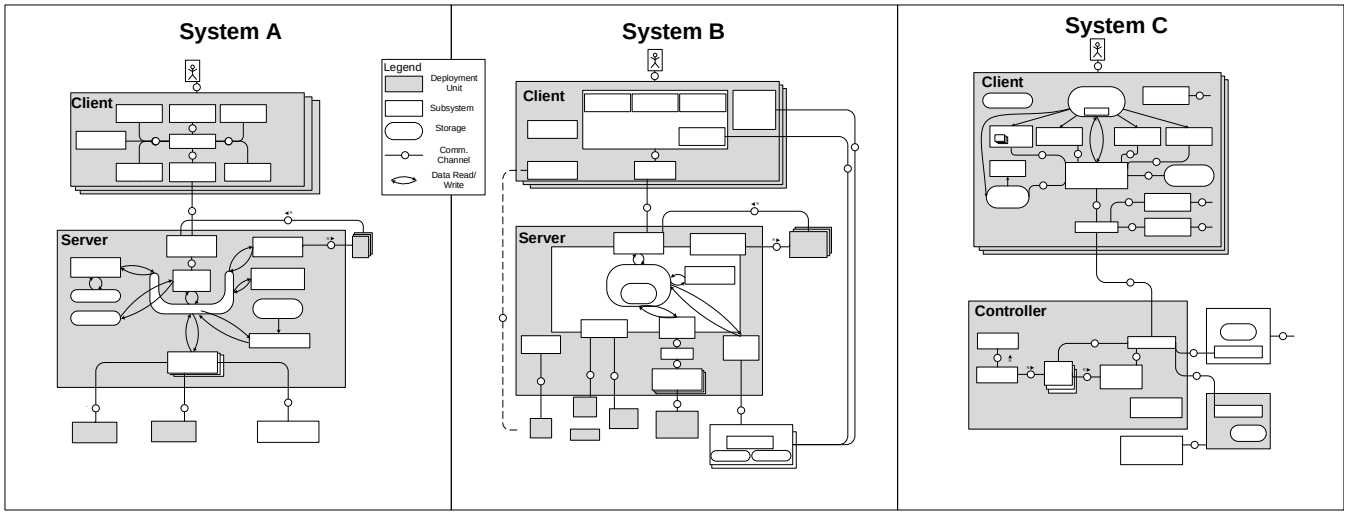


Figure 2: Architecture map for three of the analyzed systems in application case 1 (simplified)

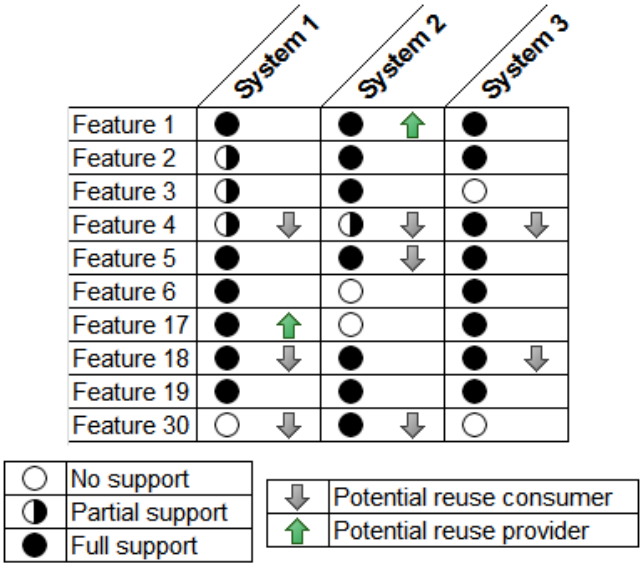


Figure 3: Anonymized excerpt of the feature support matrix for the control systems case study

tial reuse consumer for that feature. Products that have a high-quality and generic implementation of a feature are highlighted as potential reuse providers. The feature support matrix is augmented by explanatory notes based on the interviews and document analysis.

To identify the reuse potential of certain feature implementations we not only rely on the interviewee judgment, but also on the architecture illustrations. For example, we analyzed the cohesion and coupling of the subsystems that implement a feature. We also colored subsystems in different products that implement similar features.

Given the potential reuse providers and consumers in the matrix we identified several realistic new reuse opportunities among the products. For some features a reuse consumer can be directly mapped to a provider. For other cases, there are several consumers that could collaborate on a new SPL-based implementation of a feature.

The technical feasibility for each reuse opportunity was documented based on the information from the interviews and the available documentation. The organizational feasibility of each reuse opportunity was also analyzed similarly.

7) Create business case.

Creating a business case calculation (e.g., as in [1] for the reuse potential in this application domain) is not trivial. 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 system 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 yet, because the inputs for such a calculation are difficult to gather. At this point, the focus of the analysis shifted to novel or planned features for the future. High potential lies in the collaborative development of modern features for control systems, such as interfaces tailored for mobile devices.

5.3 Results

Our approach enabled us to create a thorough, albeit high-

level, assessment of reuse opportunities at reasonable cost. It showed to scale to complex systems and still provides useful results despite the relatively short feature list.

In one case, 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 hypothesis that there is functional overlap between the various system was confirmed by our analysis. However, an actual re-engineering of the systems towards reusing subsystems still seems difficult. Therefore, the focus of reuse opportunities in this domain lies more in guiding current and future development towards a better alignment and reuse of subsystems.

6. APPLICATION CASE 2: AUTOMATION PC TOOLS

6.1 Domain and Challenges

In the second application case, we analyzed a set of PC-based commissioning and monitoring tools for industrial automation systems. Commissioning tools are used to load software applications onto a device and to set application parameters. Monitoring tools are used to observe an automation system, check and predict its operating conditions, and raise alarm events.

The analysis comprised two tool families, each consisting of around ten tools, and four individual tools. Each tool family is based on a platform and the products are derived by using different components and libraries. Both tool families have not been planned as software product lines. One originated from a single product that was extended several times. The other family resulted from integrating different tools, which had similar functionality.

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. Thus, there was interest in analyzing the SPL potential in more detail and calculating the probable business case for reusing components or merging the tools into a single product line. For this purpose, we instantiated our domain analysis approach.

6.2 Domain Analysis and Architecture Reconstruction

The time frame for this domain 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 wrap-up (step 6-7) and documentation of the results.

1) List products and information sources.

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. The identification of information sources was trivial in this step as for each tool we could contact the responsible developer.

2) Establish criteria for reuse potential.

We used similar reuse potential criteria as in application case 1 (Section 5). In this application case the number of considered products was larger, but they were of smaller size and complexity. As the tools are owned by only two business units, there was less emphasis on organizational criteria during the assessment.

3) Collect and analyze documentation.

During preparation, we gathered documentation about the tools. This mainly includes user manuals, coarse-grained architectural documentation, as well as internal power point presentations about the architecture and functionality of the tools. The power point presentations helped to gain insight into the current status of the tool development and future plans, which are important for planning reuse or a software product line.

4) Create initial feature list.

Before the interviews, we had limited knowledge about the tools and their application domain. Because of this, we based the initial feature list on the user manuals. However, due to our limited domain knowledge at the beginning, the value of this initial feature list was limited and we made several changes during the interviews.

5) Conduct interviews.

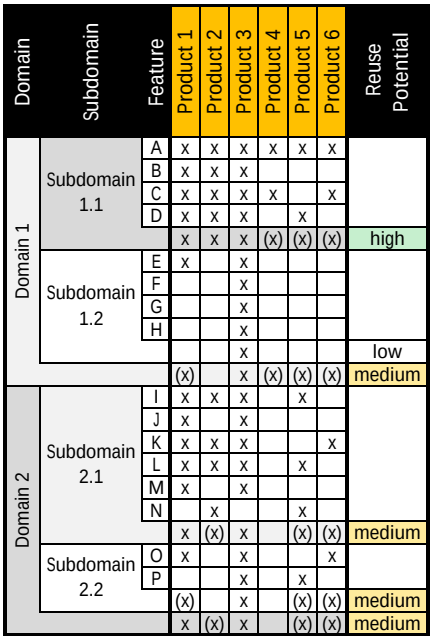
During the week on site, we interviewed three project managers and developers. With each interviewee, we filled out the questionnaire and created the architecture illustrations. We used a feature map [20] immediately in the interviews (Figure 4). A feature map groups low-level features such as *store data* into subdomains and domains, which are high level features such as *data management*.

For each product the feature map shows all supported ("x") or partially supported ("(x)") features and (sub-)domains. The last column of our feature map gives an assessment of the reuse potential. This is based on the coverage of the features of a domain by the product under consideration for reuse as well as other factors such as technical challenges and customer acceptance, which are denoted as notes. Compared to Schmid's approach [20], we only performed *product line mapping* and a short domain potential assessment, but no quantitative reuse infrastructure scoping, due to effort limitations.

After the individual interviews, we consolidated the list and made a first assessment of the reuse and SPL potential for each domain or sub-domain. We discussed this preliminary assessment with the interview partners in a 30 minute meeting on site and noted down several comments in the feature map.

The last step of each interview, was the architecture reconstruction as described in section 4. 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 quite well and supported an active discussion about the architecture and its concepts, such as used patterns and styles.

After completing the architecture illustration for all tools, we gained a good overview showing many parallels as well as



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

Figure 4: Example Feature Map

variations and potentially new interfaces. As for the feature map, the architecture illustration was discussed in a short workshop-style session with the interviewees. All agreed on the results and ranked the value of the architecture illustrations high for discussions, communication, and documentation. They were very interested in maintaining and extending them in their future work.

6) Identify opportunities.

After completing the interviews, we finalized the assessment of the reuse potential for each domain and subdomain in the feature map. We added a rational 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 reuse potential 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. However, some of the subdomains would require deeper technical investigations before deciding on reuse. In the end, we identified three short-term reuse scenarios that can be implemented in the next releases. Additionally, six domains and subdomains are long-term reuse candidates, but require more technical investigations.

7) Create business case.

For the three identified short-term reuse scenarios, we created a business case calculation. In parallel to the preceding steps of the analysis, we collected numbers for the current development and maintenance costs of the tools, which required some calendar time. Then we used the formulas similar to the ones provided by Boeckle et al. [1] to calculate the return on investment for each reuse scenario.

As originally defined, the formulas [1] were only applicable to one reuse scenario without changes. In one scenario, the calculation was simpler, because a new service (component) is implemented and used by many tools. In this case, the development and maintenance costs for the new component are lower than the maintenance costs of the existing solutions. In another scenario, some individual products are integrated into the existing platform of a tool family.

6.3 Results

The results of the domain analysis were documented and discussed in an additional meeting with the responsible stakeholders from the business unit. The six subdomains identified as potential long-term reuse candidates were agreed upon and planned to be followed-up on. The developers were particularly interested in the three short-term reuse scenarios and their business cases. Not the exact numbers were important, but the trend that is expected for implementing a scenario. Because this was positive for all three scenarios, the developers planned to discuss them with the other relevant stakeholders of the tools for implementation.

7. LESSONS LEARNED

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

Start with business case and size feature mining.

Considering the size and complexity of the systems under analysis, one important question for a domain analysis is: how much effort shall be spent for feature mining and technical investigation? This relates to how much detail is necessary to support decisions for further SPL investigation. We learned during the application cases that it is most efficient to start with an initial mock-up business case calculation and to identify which information is needed to support its assumptions. 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.

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 defendable business case calculation is a valuable argument to facilitate the discussions on SPL adoption with higher management.

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 doc-

uments. 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.

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 re-formulate 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 domain analysis 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.

Consider cost and benefits.

For conducting a domain analysis towards a SPL product it is useful to have reference numbers for the required efforts to plan similar projects. We learned that the domain analysis effort varies heavily depending on the size and complexity of the products under analysis. In application case 1, we needed about 1 person month (PM) for preparation, 1 PM for the actual interviews, 3 PM for the analysis, and 2 PM for documentation. In application case 2, we invested 1 person week (PW) for preparation, 1 PW for the interviews, and 1 PW for analysis and documentation. Due to the heavy involvement of architects and developers, calendar time is an issue for our domain analysis approach, as it is sometime difficult to arrange interviews with these persons given other development duties. Nevertheless, considering the potential impact of migrating towards a SPL approach with saved development and maintenance costs, the business case for the domain analysis itself is comparably easy to create.

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.

8. CONCLUSIONS

We have created a domain analysis approach based on existing methods and applied it in two application cases on numerous ABB software products from the industrial automation domain. This paper summarizes the experiences from applying the approach and reports on several lessons learned. We found that a domain analysis should be driven by a business case calculation that needs to consider migration costs. Furthermore, we experienced that code analyses were not helpful in our cases, but that high-level architecture illustrations based on architect interviews using the FMC notation proved to be a valuable tool. In general domain analyses can be sped up by involving stakeholders with the relevant expertise early.

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 in different subdomains of industrial automation. We intent 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.

9. REFERENCES

[1] G. Boeckle, P. Clements, J. McGregor, D. Muthig, and K. Schmid. Calculating roi for software product lines. *Software, IEEE*, 21(3):23 – 31, may-june 2004.

[2] H. P. Breivold, S. Larsson, and R. Land. 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, pages 232–239, Washington, DC, USA, 2008. IEEE Computer Society.

[3] R. Capilla. 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.

[4] Carnegie Mellon University - Software Engineering Institute. Product Line Hall of Fame. http://splc.net/fame.html, 2013. last visited 2013-01-21.

[5] Carnegie Mellon University - Software Engineering Institute. Software Product Lines. http://www.sei.cmu.edu/productlines/, 2013. last visited 2013-01-21.

[6] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[7] T. Davis. 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*, pages 126 –133, mar 1993.

[8] T. Eisenbarth and D. Simon. 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*, pages 1–4, 2001.

[9] P. Frenzel, R. Koschke, A. P. J. Breu, and K. Angstmann. Extending the reflexion method for consolidating software variants into product lines. In *Proceedings of the 14th Working Conference on Reverse Engineering*, WCRE '07, pages 160–169, Washington, DC, USA, 2007. IEEE Computer Society.

[10] J. Ganesan, D.; Knodel. 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*, pages 16–20, 2005.

[11] B. Groene. Introducing architecture modeling at a big software product company. In *Proceedings Praxisforum Modellierung 2012*, 2012.

[12] A. Harhurin and J. Hartmann. Service-oriented commonality analysis across existing systems. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 255 –264, sept. 2008.

[13] I. John, J. Knodel, T. Lehner, and D. Muthig. A practical guide to product line scoping. In *Software Product Line Conference, 2006 10th International*, pages 3 –12, 0-0 2006.

[14] M. Khurum and T. Gorschek. A systematic review of domain analysis solutions for product lines. *J. Syst. Softw.*, 82(12):1982–2003, Dec. 2009.

[15] A. Knoepfel, B. Groene, and P. Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Wiley, 2006.

[16] H. Koziolek, R. Weiss, and J. Doppelhamer. 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)*, volume 5581 of *LNCS*, pages 177–193. Springer, July 2009.

[17] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang. What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, TBD, 2013.

[18] Peter Tabeling. Home of Fundamental Modeling Concept. http://www.fmc-modeling.org/home, 2013. last visited 2013-01-21.

[19] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[20] K. Schmid. A comprehensive product line scoping approach and its validation. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 593–603, New York, NY, USA, 2002. ACM.

[21] C. Stoermer and L. O'Brien. Map - mining architectures for product line evaluations. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, pages 35 –44, 2001.

[22] F. J. van der Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[23] E. Woods and R. Bashroush. Using an architecture description language to model a large-scale information system–an industrial experience report. In *Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture, Helsinki Finland*. IEEE Computer Society, 2012.