Scoping Software Process Lines

Ove Armbrust[1], Masafumi Katahira[2], Yuko Miyamoto[3], Jürgen Münch[1], Haruka Nakao[4], Alexis Ocampo[1]
[1]Fraunhofer Institute for Experimental Software Engineering (IESE), Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
[2,3]Japanese Aerospace Exploration Agency, 2-1-1, Sengen, Tsukuba, Ibaraki, 305-8505, Japan
[4]Japan Manned Space Systems Corporation, 1-1-26, Kawaguchi, Tsuchiura, Ibaraki, 300-0033, Japan
[1]{armbrust, ocampo, muench}@iese.fraunhofer.de
[2]katahira@computer.org
[3]miyamoto.yuko@jaxa.jp
[4] haruka@jamss.co.jp

Abstract. Defining organization-specific process standards by integrating, harmonizing, and standardizing heterogeneous and often implicit processes is an important task, especially for large development organizations. On the one hand, such a standard must be generic enough to cover all of the organization's development activities; on the other hand, it must be as detailed and precise as possible to support employees' daily work. Today, organizations typically maintain and advance a plethora of individual processes, each addressing specific problems. This requires enormous effort, which could be spent more efficiently. This article introduces an approach to developing a Software Process Line that, similar to a Software Product Line, promises to reduce the complexity and thus, the effort required for managing the processes of a software organization. We propose as majors steps Scoping, Modeling, and Architecting the Software Process Line, and describe in detail the Scoping approach we recommend, based on an analysis of the potential products to be produced in the future, the projects expected for the future, and the respective process capabilities needed. In addition, the article sketches experience from determining the scope of space process standards for satellite software development. Finally, it discusses the approach, and related work, conclusions, and an outlook on future work are presented.

Keywords: software process line, software product line, scoping, process selection, process analysis

1 Introduction

Many facets of process technology and standards are available in industry and academia, but in practice, significant problems with processes and process management remain. Rombach [1] reports a variety of reasons for this: Some approaches are too generic, while some are too specific and address only a small part of daily life. Many approaches are hard to tailor to an organization's needs. In addition, some approaches impose rather strict rules upon an organization – but since not everything can be foreseen, there must be room for flexibility. Yet it remains unclear what must be regulated, and what should be left open. In general, support for process problems is plentiful, but very scattered, without a systematic concept that addresses problems in a comprehensive way. This unsatisfactory support leads to a number of problems, including:
(1) Within one organization, processes are often redundant, i.e., different processes address similar problems. For example, an organization may utilize five different processes for requirements engineering – all addressing the same problem of capturing the system requirements, and all coming with their own templates, examples, tools, etc.
(2) In addition to process redundancy, there are often areas of activity where process support is incomplete. For example, a mandatory product certification may require certain information

This is the peer reviewed version of the following article: Armbrust, O., Katahira, M., Miyamoto, Y., Münch, J., Nakao, H. and Ocampo, A. (2009), Scoping software process lines. Softw. Process: Improve. Pract., 14: 181–197. doi: 10.1002/spip.412, which has been published in final form at http://onlinelibrary.wiley.com/doi/10.1002/spip.412/abstract. This article may be used for non-commercial purposes in accordance With Wiley Terms and Conditions for self-archiving http://olabout.wiley.com/WileyCDA/Section/id-817011.html'.

to be provided, but there is no activity defined that collects and maintains the respective information.
(3) Both problems stated above lead to unnecessary effort being spent on unproductive tasks: problem (1) leads to parallel maintenance of all templates, examples, tools, etc., and problem (2) is likely to lead to increased effort for collecting and maintaining the requested information due to the missing guidance.
(4) Additionally, if problems (1) and/or (2) are recognized (usually as root causes for problem (3)), efforts to align an organization's processes are usually retrospective, i.e., mostly consider only past experience and do not actively analyze the (anticipated) future.

A traditional countermeasure taken to overcome problems (1) and partially (2) (and thus, problem (3)) is to define fixed process reference standards like the German V-Modell® XT [2], which aim at fulfilling the requirements of maturity models such as CMMI [3] or ISO/IEC 15504 [4] while reducing the variance in an organization's processes at the same time. While standardization like this potentially reduces the number of processes and process variants, it often also results in very generic processes that are no great help in dealing with daily problems, and that do not provide the necessary variability for coping with changing contexts. Furthermore, generic standards must always be integrated with an organization's specific characteristics (for example, the product certification mentioned) in order to fully address problem (2) – a fact that is often neglected and therefore leads to suboptimal processes.

Nevertheless, problem (4) remains. It can be addressed by providing fully tailored processes for every project, but this quickly leads to problem (1) again – an enormous number of (partially) redundant processes. Per-project process tailoring also consumes quite some effort, which is generally not desirable.

The problems lead to two main research questions:
(a) How can the number of variations within an organization's processes that are necessary to support all of its activities be identified, modeled, and managed so that the effort required for maintaining all is minimized?
(b) Can the anticipated future of an organization be considered in such activities, so that when a need arises, the respective process is available, and not just identified as missing? How can this be achieved?

With respect to question (a), one way to decrease maintenance effort in general is to decrease the number of objects that need maintenance. For example, in today's cars, a lot more parts than ten or 20 years ago are designed to last throughout the car's entire life cycle without any maintenance at all – thus reducing maintenance effort, but also incurring higher costs at production time.

For software, the concept of reducing the number of objects that need maintenance was introduced through Software Product Lines [5], [6]. While tailoring often results in an increase in the number of objects to handle, Software Product Lines aim at reversing this effect, i.e., at reducing this number. However, it has also been shown that converting a collection of software products into a software product line requires additional effort up-front, which generally pays off after about three products have been constructed from the product line core [7], [8]. Thus, to limit up-front effort, one key aspect is to sensibly select the products that will be integrated into the software product line, and to explicitly state which products will not be part of it – thus defining the scope of the product line.

Following Osterweil's statement that "Software processes are software, too" [9], we have transferred the product line concept to software processes, and extended it in order to provide a possible answer to research questions (a) and (b) stated above. In this article, we provide an overview of what a Software Process Line concept might look like and a detailed approach for the first step: Scoping for Software Process Lines. Our experience shows that the Software Process Line concept in general and the Scoping approach we developed in particular address problems (1) to (4) well and help to answer research questions (a) and (b). Our approach identifies redundant and missing processes, based on past, present, and anticipated future projects and products of an organization, and assists software process engineers in selecting the right processes for an organization.

The article is structured as follows: Section 2 presents related work. Section 3.1 briefly introduces Software Product Lines in order to provide the grounds for our approach, Section 3.2 gives an overview of our concept of Software Process Lines, and Section 3.3 details the first step of this approach, namely Scoping Software Process Lines. Section 4 presents a case study performed at the Japanese Space Exploration Agency (JAXA), where we evaluated an initial version of our scoping approach. Section 5 discusses the approach, and Section 6 draws some conclusions and gives an outlook on future work.

2 Related Work

In this section, we connect some related work to the issue of Software Process Scoping. As a basis for all scoping activities, descriptive process modeling [10] is necessary for identifying essential process entities. Becker describes an 8-step approach to descriptive process modeling. During the first step, the objectives and the scope of the modeling effort are determined. This narrows the extent of the model, but the approach considers only solitary process instances on the project level, not a set of processes with variabilities. Nevertheless, descriptive process modeling can be used to determine isolated, real processes that can be used as input for a variant analysis.

In [11], we presented some initial work leading to the approach described in this article. In that paper, we laid out the general problem of process management and process variability, and formulated some requirements that a solution should satisfy.

Bella et al. [12] describe their approach to defining software processes for a new domain. Based on a reference process model, they used descriptive process modeling to document the as-is processes and utilized this model as a basis for deriving suitable processes for engineering wireless Internet services. Through a number of iterations, they collected qualitative and quantitative experience and adapted the processes where necessary. Their focus thus was on the past; they evaluated only past events and processes. Software Process Scoping also considers the future in terms of expected products and projects.

The idea of systematically combining software product lines with matching processes was described by Rombach [1]. We consider Software Process Scoping as one potential building block of such a combined approach.

Characterization and customization approaches exist for a number of software engineering concepts, for example for inspections [13], [14], [15]. However, they are constrained to characterizing a limited number of methods from a class of methods (in the above case, the class of inspection methods). This comprises only a fraction of a Software Process Scoping approach, namely, that when scoping determines the need for certain characteristic features in an inspection approach, the above characterization can be used to determine which inspection approach should be used.

Denger [16] broadens the scope to quality assurance activities in general and provides a framework for customizing generic approaches to the specific needs of a company. The goal of the framework, however, is to optimize only a single factor (software quality), whereas Software Process Scoping as proposed in this article aims at optimizing multiple factors, which can be chosen freely through the product and project characterization vectors.

Avison and Wood-Harper [17] describe an approach to supplying an organization with a number of methods from which a suitable one can be selected for different purposes. The authors admit that the necessary method competence for a multitude of methods is hard to achieve in reality, and therefore suggest that alternatives should be included within a single method already. Based on our experience, we support this assumption and consider this for Software Process Scoping by representing variability on different levels of abstraction.

Fitzgerald et al. [18] describe an approach taken at Motorola, which involves tailoring up-front to encompass expected deviations from the organization standard, and dynamic tailoring during project runtime to encompass unanticipated circumstances.

The idea of a software product line was first described by Parnas [19] under the term of program families. Within software product lines, Scoping has been considered in a number of publications. Clements and Northrop [6] describe three essential activities for software product line development, with scoping being a part of one of them. The authors give a detailed description of what scoping is for and what it should accomplish, but do not provide practical guidance on how to actually do it in a project. This has been done by Schmid[20]. He developed a product- and benefit-based product line scoping approach called PuLSE-Eco 2.0, which defines the scope of a software product line depending on the economical benefit of the products to be produced. The latest version of the approach is described in [21], integrating 21 customization factors that can be used to adapt the generic approach to a company's specific needs. These works were used as a basis for the Software Process Scoping approach and terminology; however, product line scoping focuses on products only and does not consider process or other context factors. Bayer et al. developed a product line based on scoping a number of business processes [22]. Their product line reflects business processes, and by determining the scope of the business processes to be implemented in software, they determined the scope of the product line. However, no further information on how scoping was done is disclosed.

Under the name of Quality Function Deployment [23], Cohen published a method for clearly specifying and ranking customer needs and then evaluating each proposed product or service capability systematically in terms of its impact on meeting those needs. This corresponds to the Software Process Scoping concepts of product/project mapping and process mapping, respectively, but is also strictly limited to products and services.

There is currently only little research going on that tries to provide a similarly systematic approach for software processes, although the idea of reusing not only software products, but also software processes has been around for some time, e.g., in the form of the experience factory described by Basili and Rombach [24]. So far, adapting processes (also known as "process tailoring") is done either generally for an organization, resulting in a single process standard, or individually for every project, resulting in a large number of process variants. Most available tailoring instructions are very generic, e.g., in international standards such as ISO/IEC 12207:1995 [25] or the German V-Modell® XT [2]. However, due to their general applicability, they rarely provide more than phrases like "pick the activities and work products necessary for the purpose", and thus provide only little help in actually tailoring a process.

Considering the three steps described for Software Process Line Engineering, Simidchieva et al. have presented, under the name of Process Families, an approach to identifying and modeling commonalities and variabilities within processes. They also created a reference process architecture based on the identified entities, thus representing the second and the third step described in Section 3.2 [26].

3 Software Process Line Engineering

3.1 Software Product Lines

The Software Product Line concept takes advantage of the fact that the software a specific organization produces typically has some common aspect to it – although usually only implicitly. This aspect may be that it uses similar calculation procedures, is designed for a specific platform, or shares a number of features among different products. Using a conventional development approach, every product is developed by one team, with little or no systematic reuse occurring (mostly on the code/library level). “A Software Product Line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” [6]. This way, the (common) core of all systems only needs to be maintained once, reducing development and maintenance effort.

Of course, transforming existing products into a product line, or developing new products following the product line approach, requires additional effort compared to conventional development. Therefore, a product line typically selects those products for which it is economically sensible to invest the effort to integrate them into the product line (Product Line Scoping) [27]. Once this selection has been completed, the to-be members of the product line are analyzed for common and variable parts (Product Line Modeling) and transformed into a product line architecture (Product Line Architecting).

One of the most comprehensive Software Product Line approaches is the PuLSE approach [5], which we will use to explain the Software Product Line concept. After an initial baselining and customization step (PuLSE-BC), where an instance of the PuLSE method is created that is tailored to the specific enterprise context, the construction of a PuLSE Software Product Line consists of the three main steps depicted in Figure 1. The first step (PuLSE-Eco (Economic Scoping), [20]) represents **Product Line Scoping** and defines the scope of the product line by identifying the range of characteristics that systems within the product line are to cover. The result is basically a list of features that should be packaged as reusable assets within the product line, and is based on the economic benefit of each feature for the organization. The key question for Eco is: Does it pay off to implement this feature in the product line, or should it better be implemented conventionally, outside the product line?

The second step, representing **Product Line Modeling**, analyzes the features/characteristics within the product line for commonalities and variabilities. This

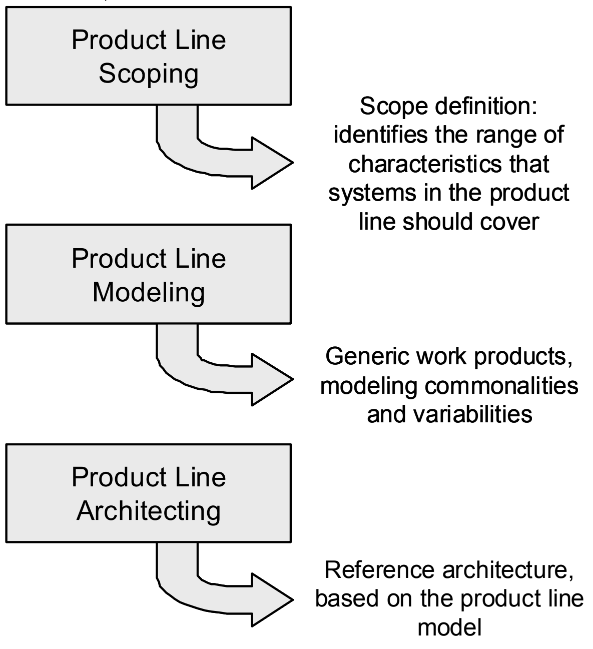

Product Line Scoping
Scope definition: identifies the range of characteristics that systems in the product line should cover
Product Line Modeling
Generic work products, modeling commonalities and variabilities
Product Line Architecting
Reference architecture, based on the product line model

Figure 1: Software Product Line Construction

is realized in PuLSE-CDA (Customizable Domain Analysis). During this step, “the product line concepts and their interrelationships are elicited, structured, and documented” [5]. CDA’s outputs are generic workproducts and a decision model in the form of a decision table corresponding to the variabilities in the generic work products. These decisions are resolved during product line instantiation to create a specific product from the product line.

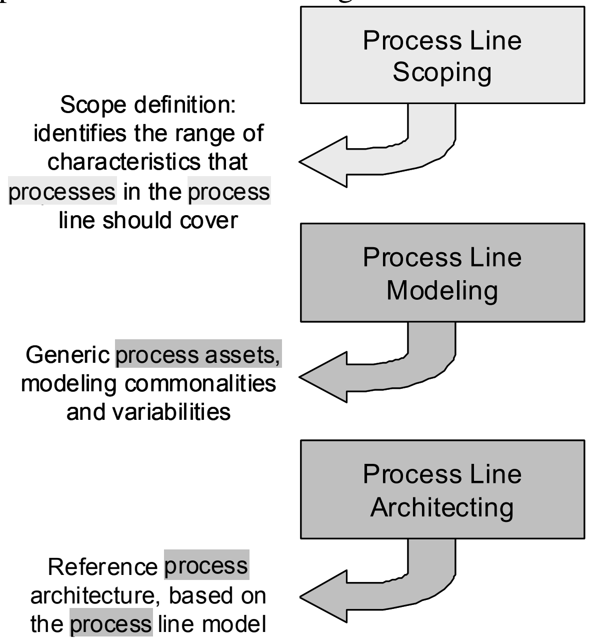
The third step, representing **Product Line Architecting**, is the creation of a reference software architecture that corresponds to the results of the first two steps. This is covered by PuLSE-DSSA (Domain Specific Software Architecture). The reference architecture maps the generic work products from CDA to architecture elements, so that software instances can be created during product line usage.

Once construction is complete, the Software Product Line can then be used to derive specific products from the product line (**Product Line Instantiation**, PuLSE-I). Simplified, one must resolve all decisions from the decision table created during the second step, which leads to a product that should cover most of the required features. The still missing features are added using “glue code”, to account for functionality not foreseen in the product line. During instantiation, neither the reference architecture nor the generic work products are altered. If some “glue feature” turns out to be suitable for integration into the product line, the construction steps are repeated, modifying the scope and all subsequent product line elements to encompass the additional feature.

3.2 Software Process Lines

The concepts of software product line engineering can be transferred to software processes. In fact, the three major steps of Product Line construction map fairly well to software processes, which we will describe briefly. Figure 2 displays our approach, with Scoping (light gray) being the focus of this article and Modeling and Architecting (dark gray) introduced briefly.

The first step realizes the **Process Line Scoping** activity. During this step, the organization’s current and future products and projects are analyzed in order to elicit the process needs of the organization. These needs form the scope of the Process Line. Additionally, the current processes of the organization are analyzed with respect to fulfilling the needs, in order to determine their capabilities. The analysis results provide information on where investments in further Process Line activities are most likely to pay off, namely those processes that are actually needed currently and in the future. This constitutes a difference to Product Line Scoping, which identifies features along the lines of one or several product domains, independent of whether these features are currently realized in the company’s products or not. For Process Line Scoping, this analysis actively reveals gaps in the organization’s current process landscape, where needs are not covered by existing processes. We will detail this in Section 3.3.


Process Line Scoping
Scope definition: identifies the range of characteristics that processes in the process line should cover
Process Line Modeling
Generic process assets, modeling commonalities and variabilities
Process Line Architecting
Reference process architecture, based on the process line model
Figure 2: Software Process Line Construction

The second step (**Process Line Modeling**) then takes the processes deemed worthy of further attention and analyzes them in terms of commonalities and variabilities. Based on the analysis results, a collection of generic process assets (the "building blocks" from which the processes are built, for example, single activities or document fragments) capturing the identified commonalities and variabilities is created. A decision model accounts for all variation points, so that a specific process model can be constructed from the generic process assets.

The third step (**Process Line Architecting**), finally, creates a process reference architecture (i.e., a process model containing all generic process assets and a decision model governing which assets to put together under which circumstances, forming a specific process instance) following the results of the first two steps. The resulting process model contains the common core and all variabilities and is instantiated into individual process models, using the decision model from the second step, during process line usage.

Once construction is complete, the Software Process Line can be used to derive process models for specific projects with unique characteristics (**Process Line Instantiation**), for example, for creating safety-critical software, or to support rapid development. This is supported by the characteristics determined in the Scoping step. As with Software Product Lines, during Process Line usage, neither the scope nor the process reference architecture is altered. A customization step may introduce process capabilities not provided by the Process Line (for example, the production of an additional artifact for auditing purposes); should these capabilities be integrated into the Process Line, however, the construction steps (Scoping, Modeling, and Architecting) need to be (partially) repeated, resulting in a modified Software Process Line Scope and subsequent elements.

Corresponding to [6], we define a Software Process Line as a *set of software processes with a managed set of characteristics that satisfy the specific needs of a particular organization and that are developed from a common set of core processes in a prescribed way*. Corresponding to [20], Software Process Line Scoping is defined as the *identification of the range of characteristics that processes in the process line should cover*. The following Section describes our Software Process Scoping approach in detail.

3.3 Process Line Scoping

Our scoping approach consists of five main steps:
(1) **Product Analysis,** in order to identify product-imposed process needs,
(2) **Project Analysis,** in order to identify project-imposed process needs,
(3) **Process Analysis,** using the same attributes as for products and projects in order to identify process capabilities,
(4) **Attribute Prioritization,** and
(5) **Scope Determination** using a mathematical model.
The result of these five steps is an objective analysis of the organization's process needs (steps (1) and (2)), the capabilities of the current processes and potential (external, not yet utilized) processes (step (3)), and a recommendation for the scope of the organization's processes (steps (4) and (5)). Depending on the kind of organization, product and project characterization may have a different weight for assessing process needs: An organization developing five products, following a fixed-release cycle, might put more emphasis on product characterization, whereas a project organization that develops customer-individual software in projects might concentrate more on project characterization.

3.3.1 Product Analysis

During this step, existing and future software products are identified for which development and maintenance processes must be supplied. Each identified product is then analyzed using specific attributes. For existing software products, potential sources of information are all work products, e.g., product requirements, design documentation, roadmaps, test documentation, etc. For planned software products (i.e., products for which development has been decided and is firmly planned), artifacts such as market analyses, business strategies, product portfolio plans, product roadmaps, release plans, or even standard roadmaps may be utilized. For potential products (i.e., products that have been thought of, but whose realization is not sure yet), information sources such as organization strategy or organization roadmaps can be used to predict the near-to-midterm product future.

Product identification is simplified if a software product line approach has been implemented in the organization, in which case it is sufficient to extract the required information from the software product line artifacts (e.g., product map and asset scope when using PuLSE). However, if such an approach has not been taken, the required information must be elicited by other means.

For each identified product, its probability of being implemented or maintained (and thus, of requiring a process to support its development or maintenance) is determined. This can be done individually for each product, or in groups for existing, planned, and potential products, for example, rating existing products at 100% (meaning that every existing product is going to be maintained as well), planned products at 75%, and potential products at 50%.

Next, product characterization attributes are identified. These attributes represent product characteristics that are known or assumed to have an influence on the development process. Naturally, the list of product characterization attributes depends largely on the kind of product, therefore there is no generic list that can be applied to all products. Still, some suggestions can be found in Table 1. What is required is a description of product characteristics with respect to processes – any characteristic that requires special process attention should be taken into account. For example, a product that is safety critical and needs to be certified according to SIL [28] requires certain mandatory processes to be followed. Thus, "safety criticality" would be considered a process-related product characterization attribute.

Table 1: Product and Project Characterization Attributes

Product	Project
Size	Degree of Distribution
Complexity	Schedule Pressure
Criticality	Available Personnel
Requirements Stability	Cooperation with Customer
Safety Criticality	Temporal Distribution
Developer Experience	Developer Experience

Once the attribute list is complete, products can be rated using this list. We use a 3-item scale with the value "3" standing for a high rating (for example, a "3" in "size" means that the product is big) and "1" for a low one (small product), leaving "2" for a medium rating. We will use this scale throughout this article.

To reflect the lower probability of planned and potential products becoming relevant for the organization and thus for its processes, the product rating must then be discounted. Discounting is done using the following formula:

(1) $PR_d(i,j) = P(i) \cdot PR_o(i,j)$ with

$PR_d(i,j)$ = discounted product rating for product i, attribute j

$P(i)$ = probability of product i

$PR_o(i,j)$ = original rating of product i, attribute j.

Example. A supplier for a car manufacturer currently builds three products: a steering angle sensor (SAS), power windows, and a rain sensor. It has plans to build two new products: a light sensor (LS) and a power trunk. A long-term strategy is to move up the value chain and produce larger, more complex (and more expensive) components: a lane assistant and a distance sensor (DS). Current products are rated at 100% probability (meaning that these products with their unique features are going to be maintained), planned products at 75%, and potential products at 50%. The product characterization attributes that were identified are:

- Safety criticality (SC), e.g., in terms of SILs [28],
- Requirements fuzziness (RF), e.g., in terms of number of use cases defined,
- Complexity, e.g., in terms of cyclomatic complexity, and
- Size, e.g., in terms of function points.

Table 2 shows the results of the product analysis activity. The first column contains the products, the second column their probabilities. Columns 3, 5, 7, 9 contain the original ratings of the respective attributes, columns 4, 6, 8, 10 (gray background) the discounted ratings, based on the probability displayed in column 2. The values in the gray columns will be used for further steps.

Table 2: Product Analysis Results

		Probability	Safety criticality	SC discounted	Requ. fuzziness	RF discounted	Complexity	Complexity discounted	Size	Size discounted
Product 1	Steering angle sensor (SAS)	100%	1	1	1	1	2	2	2	2
Product 2	Power windows	100%	2	2	1	1	1	1	1	1
Product 3	Rain sensor	100%	3	3	1	1	1	1	2	2
Planned 4	Light sensor (LS)	75%	1	0.75	2	1.5	1	0.75	2	1.5
Planned 5	Power trunk	75%	2	1.5	3	2.25	2	1.5	2	1.5
Potential 6	Lane assistant	50%	3	1.5	3	1.5	3	1.5	3	1.5
Potential 7	Distance sensor (DS)	50%	3	1.5	2	1	3	1.5	3	1.5

3.3.2 Project Analysis

Project analysis is done exactly like product analysis is, just using projects as the basis. This means that first, existing, planned, and future projects need to be identified for which development and maintenance processes must be supplied. Existing projects can be extracted from project plans, project traces, project documentation packages, (tailored) process documentation, and the like. Planned projects can also be extracted from project plans, a project portfolio, tailored process documentation, or other context information. Potential projects, however, rely heavily on context information such as expert estimates as to "what might come", aside from the organization strategy/roadmap information already mentioned in Section 3.3.1.

For each identified project, its probability is determined, for example 100% for existing projects that need continued support, 75% for planned projects, and 50% for potential projects. Next, project characterization attributes are identified. As with product characterization attributes, there is no generic list, some suggestions, however, can be found in Table 1. Once the attribute list is complete; the projects can be rated using the list and the discounted rating, using the following formula (in analogy to formula (1)):

(2) $PJ_d(i,j) = P(i) \cdot PJ_o(i,j)$ with

$PJ_d(i,j)$ = discounted project rating for project i, attribute j

$P(i)$ = probability of project i

$PJ_o(i,j)$ = original rating of project i, attribute j.

Example. The automotive supplier from Section 3.3.1 sells its products to multiple customers. Therefore, they currently have three projects running, namely, preparing the steering angle sensor (SAS) for Daimler, BMW, and Volkswagen (VW). They have already planned two more projects, providing a light sensor (LS) for Daimler and Toyota, and are trying to acquire two more projects: adapting the light sensor for BMW, and developing a distance sensor (DS) for VW. The rating attributes identified were:

– Degree of Distribution (DoD),
– Cooperation with Customer (CwC),
– Developer Experience (DE), and
– (anticipated) Schedule Pressure (SP).

Table 3 shows the results of the project analysis activity. Column 1 contains the identified projects, column 2 their probabilities. Columns 3, 5, 7, 9 contain the original ratings for the attributes, columns 4, 6, 8, 10 (gray background) the discounted values. The values in the gray columns will be used for further steps.

Table 3: Project Analysis Results

		Probability	Degree of Distribution	DoD discounted	Cooperation w/ Customer	CwC discounted	Developer Experience	DE discounted	Schedule Pressure	SP discounted
Project 1	**SAS-Daimler**	100%	1	1	3	3	1	1	3	3
Project 2	**SAS-BMW**	100%	1	1	1	1	2	2	2	2
Project 3	**SAS-VW**	100%	3	3	1	1	3	3	1	1
Planned 4	**LS-Daimler**	75%	1	0.75	3	2.25	1	0.75	2	1.5
Planned 5	**LS-Toyota**	75%	3	2.25	1	0.75	2	1.5	2	1.5
Potential 6	**LS-BMW**	50%	2	1	1	0.5	1	0.5	2	1
Potential 7	**DS-VW**	50%	2	1	1	0.5	1	0.5	2	1

3.3.3 Process Analysis

Process analysis determines the capabilities of processes. First, currently used and potentially usable processes are identified. Then these processes are analyzed, using the attributes defined during product and project analysis.

Process identification for currently used processes is supported by available process documentation and the implicit process knowledge of employees. Identification of potentially usable processes can be supported by empirical knowledge published in the literature, by consulting institutions, or by process standards. Rating the processes can be supported by assessment results and project evaluations.

Once the list of processes to be analyzed is complete, the list of characterization attributes must be compiled. This means merging the characterization attributes from product and project analysis into one list. Using this list, all processes that were identified in the first activity of this step are rated. The result is a preliminary process characterization, which will be refined in one more step in order to provide a suggested scope for the organization.

Example. The exemplary automotive supplier has determined the following processes to be characterized:

– Requirements Processes: Formal Specification [29], Brainstorming [30], Use Cases [31], Storyboards [32], and the Delphi method [33].
– Design Processes: Cleanroom [34], Object-Oriented Design (OO) [35], Leonardo [36], Structured Design (SD) [37].

Table 4 shows the results of the process analysis. The first column contains the processes that were analyzed, columns 2-9 the values for the characterization attributes.

Table 4: Process Analysis Results

		Safety	Distribution	Fuzzy Reqs.	Cooperat.	Complexity	Dev. Exp.	Size	Schedule P.
Req.	**Formal Sp.**	3	3	1	1	1	2	1	3
	Brainstorm	1	1	1	3	1	3	2	1
	Use Cases	2	2	2	2	2	2	2	2
	Storyboards	1	1	1	1	2	3	2	1
	Delphi	3	3	3	3	3	2	2	3
Design	**Cleanroom**	3	2	2	1	3	2	2	1
	OO	2	2	3	2	3	2	2	2
	Leonardo	1	1	1	2	3	1	3	2
	SD	1	1	1	2	1	2	3	1

3.3.4 Attribute Prioritization

So far, the analysis activities considered all attributes equally important. In reality, this will most likely not be the case. Thus, we propose an additional step in order to prioritize the attributes and to be able to better distinguish processes for different needs.

There are a number of ways to prioritize arbitrary entities. The most straightforward one is to just assign numbers to each attribute; the higher the number, the higher the respective attribute's value. However, this may lead to imprecise priorities, especially with a larger number of attributes, because when using this technique, any attribute is only compared to very few other attributes based on which its priority is assigned.

Therefore, we propose a somewhat more elaborate, but therefore more objective method for prioritizing the characterization attributes: pair-wise comparison [38]. Using this method, every characterization attribute is compared to all other characterization attributes and a value is assigned, depending on whether it is considered more important than its counterpart, less important, or equally important. By summing up all comparison values for each attribute, their importance relative to each other is determined. This importance can then be scaled to meet arbitrary needs, e.g., to fit between predefined boundaries for highest and lowest relative value.

Example. Attribute prioritization for the example company leads to the results displayed in Table 5. The first column denotes the attributes. Comparison results are noted above the black diagonal, the values below the diagonal (gray background) are calculated by negating their counterparts above the diagonal. If the attribute denoted in the line is more important than the one denoted in the column, the cell where both cross each other is marked with a "+". If it is the other way around, the cell is marked with a "-". An "o" denotes equal importance.
Column 10 contains the calculated priority, assigning a "+" 2 points, an "o" one point, and a "-" zero points. Column 11 contains the relative importance of the attributes, projected on a scale of 50% … 100%. This means that the attribute that is considered least important is assumed to be half as important as the most important one.

Table 5: Attribute Prioritization Results

Attribute	Safety Criticality	Requ. Fuzziness	Complexity	Size	Degree of Distribution	Cooperation w/ Customer	Developer Experience	Schedule Pressure	Priority	Relative Importance
Safety criticality	■	+	+	+	+	+	+	+	14	100%
Requ. fuzziness	-	■	-	-	-	+	-	-	2	54%
Complexity	-	+	■	+	+	+	+	+	12	92%
Size	-	+	-	■	+	+	+	+	10	85%
Degree of distribution	-	+	-	-	■	+	+	+	8	77%
Cooperation w/ customer	-	-	-	-	-	■	-	o	1	50%
Developer experience	-	+	-	-	-	+	■	o	5	65%
Schedule pressure	-	+	-	-	-	o	o	■	4	62%

3.3.5 Scope Determination

With the attributes prioritized, the scope of the Software Process Line can finally be determined. First, the process analysis described in Section 3.3.3 is modified, based on the attribute prioritization. Then, the values in each line are summed up. The sums describe the degree to which each process covers the needs of the existing, planned, and potential products and projects, considering the uncertainty in events that have not happened yet and characterization attribute priorities.

Updating the process analysis from Section 3.3.3 is done by calculating the weighted process capabilities based on attribute importance using the following formula:

(3) $C_w(P_i, A_j) = C_u(P_i, A_j) \cdot P(A_j)$ with

$C_w(P_i, A_j) =$ weighted capability of process i for attribute j,

$C_u(P_i, A_j) =$ unweighted capability of process i for attribute j, and

$P(A_j) =$ priority of attribute j.

(Weighted capability of a process i for attribute j equals unweighted capability of process i for attribute j * priority of attribute j.)

Example. Applying the above formula to the process analysis results (Section 3.3.3) and the attribute prioritizations (Section 3.3.4) of our example company leads to the results displayed in Table 6. The first line of the table contains the attributes used for characterization with their respective priorities on a scale of 50% … 100%. The first column contains the processes that were analyzed. Columns 2-9 contain the weighted characterization values that are the output of formula (3). Column 10 sums up the lines, with the gray lines as the suggested scope of the Software Process Line: Use Cases and Delphi for Requirements Engineering, and Cleanroom and Object-Oriented Design for Software Design Engineering. The cutoff criterion in this case was to select the 2 top rated processes from each category; however, looking at the 11.3 rating of Formal Specification, which puts it close to Use Cases (11.7) and clearly away from Brainstorming (9.0) and Storyboards (8.9), a different cutoff criterion may make sense.

Table 6: Weighted Process Analysis Results

		Safety 100%	Distribution 77%	Fuzzy Reqs. 54%	Cooperat. 50%	Complexity 92%	Dev. Exp. 65%	Size 85%	Schedule P. 62%	SUM
Req.	**Formal Sp.**	3.0	2.3	0.5	0.5	0.9	1.3	0.8	1.8	**11.3**
	Brainstorm	1.0	0.8	0.5	1.5	0.9	2.0	1.7	0.6	**9.0**
	Use Cases	2.0	1.5	1.1	1.0	1.8	1.3	1.7	1.2	**11.7**
	Storyboards	1.0	0.8	0.5	0.5	1.8	2.0	1.7	0.6	**8.9**
	Delphi	3.0	2.3	1.6	1.5	2.8	1.3	1.7	1.8	**16.0**
Design	**Cleanroom**	3.0	1.5	1.1	0.5	2.8	1.3	1.7	0.6	**12.5**
	OO	2.0	1.5	1.6	1.0	2.8	1.3	1.7	1.2	**13.2**
	Leonardo	1.0	0.8	0.5	1.0	2.8	0.7	2.5	1.2	**10.5**
	SD	1.0	0.8	0.5	1.0	0.9	1.3	2.5	0.6	**8.7**

4 Case Study

We applied an abridged version of the scoping approach described above at the Japanese Space Exploration Agency (JAXA). Unfortunately, practical constraints (mostly project schedule issues and personnel availability) forced us to cut some corners:

– Product, project, and process analyses were carried out, but without discounting planned and potential products and projects (thus considering all as equal).
– Since only few relevant characterization attributes were identified, prioritization was straightforward, so the pair-wise comparison as described in Section 3.3.4 was not needed.

The scope determined was used as input for the commonality and variability analysis, i.e., the Process Line Modeling step described in Section 3.2. However, this activity did not immediately lead to elements being removed from the processes used due to a variety of reasons. Instead, those processes and process parts that were not required constantly (thus having a low scope rating) were considered optional, with precise criteria defined as to when to apply them. These criteria were derived from the analysis results.

The experiences we collected during the JAXA application of the approach were integrated in the version described in Section 3.3. In this section, we will briefly describe our test run and our experiences from an ongoing effort within JAXA to provide a process line for their space software development. Our focus in doing so lies on scoping for satellite software development (see Figure 3). We describe the project context and results, but due to confidentiality reasons, we cannot disclose all the details, such as the complete characterization profiles, cost information, or the detailed scoping results; however, we will share our experiences.

4.1 Process Scoping in the Aerospace Domain

The ultimate goal of the ongoing project we are reporting on is to provide a software process line for JAXA's space software development. This includes satellite software, launch vehicle software, and ground segment software (see Figure 3). So far, we have completed the first (Scoping) and the second (Modeling) steps of the Software Process Line Engineering approach described in Section 3.2. Scoping provided analyses of two products (satellites) developed in two projects. In the space domain, there is a very strong correlation between

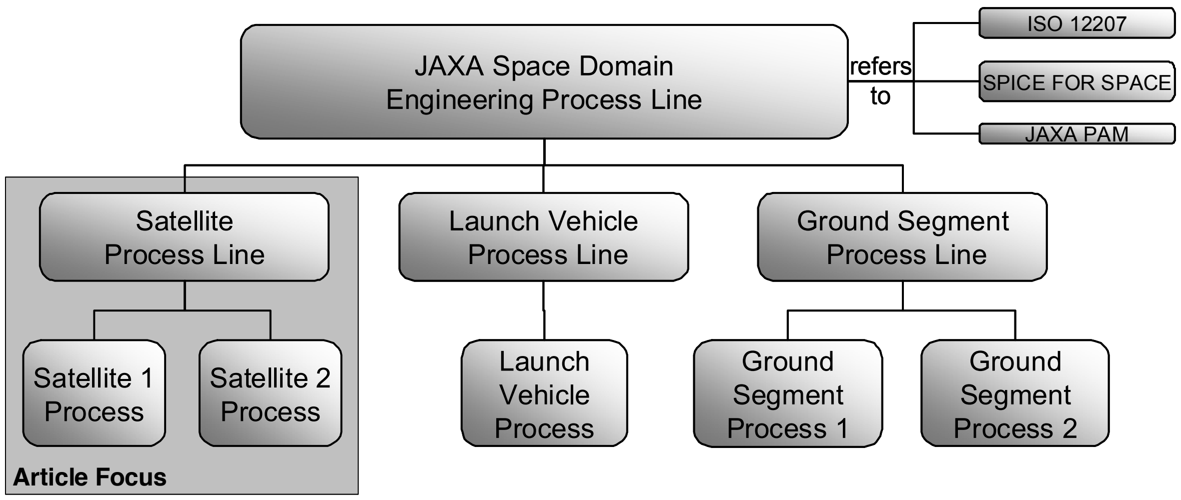


Figure 3: JAXA Process Line Overview

product and project, since each product is unique. Nevertheless, a meaningful project and product analysis is not trivial. In our case, it became apparent very soon that while attributes for project analysis often had only two possible values (e.g., "National" and "International" for the "Collaboration type" attribute), this was not the case for product analysis. For example, complexity, criticality, and size were determined on a 3-item scale by experts.

Table 7 and Table 8 show an extract of the analysis results of the projects and products, respectively. So far, only satellite products and projects have been analyzed: however, similar work for the launch vehicle and ground segments is currently going on (gray background in the tables). Due to confidentiality reasons, subsystems and suppliers are represented by numbers.

Table 7: Excerpt from JAXA Project Analysis Results

	Collaboration Type	Mission Type	Subsystem	Supplier	...
Satellite 1	National	Engineering	1, 2, 3	1, 2	
Satellite 2	International	Science	3	1	
Launch Vehicle 1					
Launch Vehicle 2					
Ground Segment 1					
Ground Segment 2					

Table 8: Excerpt from JAXA Product Analysis Results

		Complexity	Criticality	Size	Stable Requirements	...
Satellite 1	**Subsystem 1**	3	2	3	yes	
	Subsystem 2	2	3	3	yes	
	Subsystem 3	1	1	2	yes	
Satellite 2	**Subsystem 3**	1	1	2	no	
	Launch Vehicle 1					
	Launch Vehicle 2					
	Ground Segment 1					
	Ground Segment 2					

There are a number of interdependencies between project and product analysis data that are not apparent at first sight, but that surfaced during scoping efforts. For example, the unstable requirements for Satellite 2, Subsystem 3 require an iterative development approach – this led to the fact that for each potential supplier, it had to be checked whether such a process could be supported. In our case, Supplier 1 was chosen and had to adapt (for Satellite 2) its processes to the international collaboration type. Other interdependencies led to conflicts, e.g., the collaboration type "international" demanded that documentation had to be made available in English upon request, suggesting one set of potential suppliers, but the mission type suggested a different set – this was solved by prioritizing characterization attributes.

4.2 Experience

Translating the project and product analysis results into requirements for the process proved not to be an easy task. Most "soft" product characteristics such as complexity, size, or criticality could not be used to directly derive new or changed processes. In fact, these factors mostly did not lead to qualitative process changes (i.e., new or changed activities or work products), but influenced project planning in such a way that, for instance, the number of reviews of a particular work product was increased, or that the amount of independent Verification and Validation was increased. This was not modeled in detail in the software process line due to JAXA-internal standards for process models; instead, only high-level directives and quality requirements were given, which have to be implemented individually by the suppliers.

Project analysis, on the other hand, led to a number of variation points within the process itself. A variation point means that a decision has to be made in terms of choosing among several alternatives: While some findings did not change the process itself (e.g., the requirement that for international projects, documentation was to be produced in English upon request), others did. For example, for international cooperation projects with the European Space Agency (ESA), a new activity was introduced for analyzing hardware/software interaction, producing the new work product FMECA (Failure Mode, Effects, and Criticality Analysis). Especially for exploratory science projects, the usual process standard was perceived as being too heavy. As a consequence, the number of quality assurance activities was reduced, and the requirements and design rationales were waived. Also, source code quality assurance measures were decreased for this type of project.

The variations were modeled using the graphical software process modeling tool SPEARMINT™ [39]. Process parts that were optional were marked accordingly, with a detailed description of when to consider the respective part. Figure 4 displays an excerpt of the result of the Process Line Modeling activity.

The resulting model contained a number of variation points, with the work products FMECA and Rationale for Design being shown as Opt1 and Opt2. The characterization results from the Process Line Scoping activity govern the decisions on variation points. Opt1 concerns the creation of the FMECA document, which is mandatory for international collaboration type projects, and does not need to be created for other

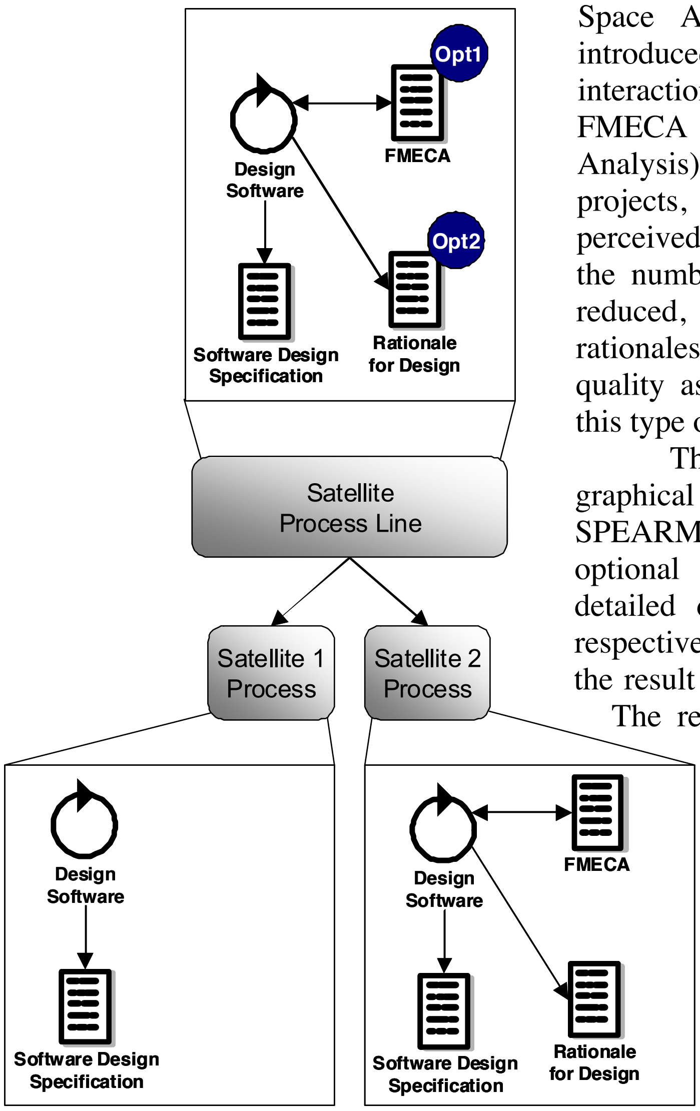


Figure 4: JAXA Satellite Process Line Architecture Excerpt

collaboration types. Opt2 concerns the Rationale for Design document, which is mandatory for engineering mission types, and not necessary otherwise.

Similarly, decisions for the other variation points (not shown in Figure 4) can be derived from the analysis results. Thus, the satellite process line could then be instantiated into two specific satellite processes. From one of these, the formerly optional parts were erased (lower left part of Figure 4), whereas in the other one, these parts were now made mandatory (lower right). The resulting Satellite 1 process supports a national science-type project, the Satellite 2 process an international engineering type project.

The resulting process model contains 76 modeled activities, 54 artifacts, 18 graphical views depicting product flow, and another 18 graphical views depicting control flow. Transferring the satellite process line into daily practice, however, has proved to be no simple task. The modification of standards in the aerospace domain cannot be done on-the-fly because many stakeholders are involved and the consequences of software failures (possibly stemming from a faulty standard) are potentially grave. So far, the software process line we have developed has been published as an appendix to the official JAXA software standard. It has therefore not yet replaced the current standard, but JAXA engineers and their suppliers are encouraged to examine the process line and provide comments and feedback.

Both our and JAXA's experience with the scoping approach taken were positive. The feedback collected from JAXA engineers supports the expectation that the approach is feasible for application in an industry setting, which was our main concern at the beginning. For the future, we expect a significant decrease in maintenance effort for the satellite processes: The classic approach would have developed two independent processes for satellite development. The process line approach enables engineers to consider the variable parts separately, while for most of the process line, there is only one process to be maintained instead of two.

5 Discussion

In this article, we have transferred the concepts of Software Product Lines to the process world – with success, we believe. Our experience during the development of the Scoping approach and its application within JAXA supports this impression. However, this is only one application, and while it indicates the feasibility of the approach, it does not prove its general applicability. This will need to be proven through empirical work, just as for product lines.

In our opinion, the greatest advantage of the Scoping approach we presented is that it makes explicit a number of facts and decisions that are implicit at best otherwise. This way, they can be discussed and evaluated, something that is not possible for implicit knowledge. Another advantage is that the approach makes an organization very flexible within its scope. Setting up a new (or modified) process, based on the process repository, can be completed very quickly, as opposed to fully tailoring a standard. For products or projects outside the domain, this is obviously not the case. However, from our experience, this kind of flexibility on a global scale ("we're great at everything") is an illusion anyway. Therefore, our approach assists organizations in determining their scope and then achieving process excellence for this scope.

Product Analysis forces an organization's process engineers to think about the product future of the organization. This makes the approach vulnerable, of course. If there is no somewhat defined product strategy within an organization, it will be hard to derive process demands from this. On the other hand, without the Product Analysis of our Scoping approach, this information would also not be available, even worse: It may not even be noticed that the product strategy is incomplete. Thus, we do not consider this a problem of the approach, but rather a necessary precondition for it to work with full effectiveness. The same applies to Project Analysis.

Attribute selection is vital for the selection recommendation made by the approach. If the wrong attributes are selected, the approach may determine a completely wrong scope for an organization, resulting in a faulty process repository. This is clearly a threat that lies within the approach itself. We have tried to mitigate it by providing some generally applicable attributes, but due to the highly different domains that exist in the software world, domain-specific attributes will almost always be needed and thus will need to be specified. In the future, we will direct some effort towards attribute selection strategies in order to strengthen this part of the approach.

The results of the Process Analysis, when not based on empirical evidence, depend on the people executing the analysis. If their evaluation of a certain process and attribute is wrong, so will be the Process Analysis results. However, we do not consider this to be a flaw of the approach, since the same people would probably also make this (wrong) decision without the approach. On the contrary, by making the analysis explicit and transparent, we believe that the approach actually helps to identify potentially problematic evaluations and correct them before any decisions are made

Attribute Prioritization may influence the scope in different ways. If attributes are not prioritized, they are all considered equal. This may lead to a suboptimal scope definition, which puts too much emphasis on issues that are too unimportant. Even worse, a wrong prioritization may turn the scope around, providing just the wrong processes for an organization. While this is an inherent threat to the approach, we also believe in this case that the transparency created by the explicit prioritization will help to avoid such situations.

Scope Determination, finally, obviously has a great impact on the final scope. Depending on the selection approach, the scope may be defined very differently for different approaches. Since we do not yet have empirical evidence on different selection approaches, we can only speculate on better and worse ones. However, the problem at hand looks very much like a multi-objective optimization problem, for which a number of possible approaches are available. We will investigate this further in the future.

In general, the Process Scoping approach seems to benefit more process-mature organizations most. Obviously, processes must be defined and followed – a precondition that is unfortunately not met by many organizations. Further, there must be experience available for past products and projects as well as their respective processes. A clear plan for the short- to mid-term future is finally required to fully exploit the approach. This indicates that a certain process maturity is required to apply the approach – this corresponds to the Product Line approach, which also requires a certain organization maturity in order to be applied successfully. As for specific domains or processes, at this point we do not see any domain or process that is particularly well-suited (or ill-suited) for our approach. Considering problems (1) to (4) and research questions (a) and (b) stated in Section 1, the experience we have collected suggests that our approach can provide significant help in solving the problems and answering the research questions.

6 Conclusions and Outlook

In this article, we presented an approach to building a Software Process Line, in analogy to Software Product Lines, and detailed the first step of such an approach, which is Software Process Line Scoping. The approach, when developed and implemented successfully, promises significant effort savings for process management, since an organization's processes do not need to be maintained in parallel, isolated from each other, but rather their common elements can be maintained only once, with specific extensions for each variation within otherwise similar processes. We do not yet have empirical proof of the potential savings, but expect these based on the savings proven for software product lines [7], [8]. We will investigate this further in the future.

As a first step towards a comprehensive Software Process Line Engineering methodology, we have developed an approach for systematically scoping processes for a Software Process Line. The scope determines which processes to include in a Software Process Line, and which ones not to, thus directing the additional effort for setting up a Software Process Line to areas where this effort is expected to pay off most. This directing is done by analyzing existing, planned, and potential products and projects for their process needs using a number of characterization attributes, and then rating the available process options in terms of fulfillment of these process needs, using a mathematical model. This way, an organization's current situation and its anticipated future development are accounted for, allowing for better process selections than when using entirely retrospective analyses.

Even after this first step towards Software Process Lines, significant savings are already to be expected, by streamlining and harmonizing an organization's processes to what is really needed now and in the short- to mid-term future, and by discarding those processes that are determined to be less relevant. This gives process groups within organizations a tool for objectively deciding in which direction the organization's processes should be developed and for helping them to become more independent of individual experience.

Our experiences with the approach, especially those collected during its pilot application at JAXA, encourage us to continue on this path, and to expand the JAXA process line from satellite software development both horizontally to other branches (launch vehicle, ground segment) and vertically (JAXA-wide). The experience we have collected so far supports the requirements we set up in [11]. However, since process scoping research is yet in its infancy, a number of open questions remain. For one, there are a number of ways to define the scope from the scope determination results as described in Section 3.3.5: Choose the n processes with the highest rating, choose those processes that are rated significantly higher than the others, etc. – it is unclear whether there is a "best" way to define the scope, and if so, which one this might be. A meaningful limitation of characterization attribute values (e.g., for attributes such as "complexity" or "criticality") and their objective assessment is another open issue. Furthermore, thorough investigation is needed on the subjects of how to handle different levels of abstraction in processes and characterizations (especially when talking about variability on these levels of abstraction, introduced, for example, by a vertically growing process line), how to describe interdependencies among variable parts and characterization attributes, and how to sensibly limit the number of characteristics and variation points.

Following up on what we have learned so far, our next steps will be the application of the full approach in industry and the horizontal expansion and the inclusion of more satellite projects and products into the base of the JAXA process line. From a scientific viewpoint, we will address the open questions, e.g., the identification of sensible ways to define the scope from the scope determination results, or the collection of empirical evidence for the expected savings.

Acknowledgments

We would like to thank Ms. Sonnhild Namingha from Fraunhofer IESE for reviewing the first version of this article. We also thank the anonymous reviewers for their valuable comments on the first version of this article.

References

1. Rombach HD. Integrated Software Process and Product Lines. *Proceedings of the International Software Process Workshop (SPW 2005), Beijing, China*, 2005, pp. 83-90.
2. Das V-Modell XT 1.2.1.1, http://www.vmodellxt.de/, last visited 2009-01-15.
3. Capability Maturity Model Integration, http://www.sei.cmu.edu/cmmi/, last visited 2009-01-19.
4. International Organization for Standardization. *ISO/IEC 15504*. ISO/IEC: Geneva, Switzerland, 2006.
5. Bayer J, Flege O, Knauber P, Laqua R, Muthig D, Schmid K, Widen T, DeBaud J. PuLSE: A Methodology to Develop Software Product Lines. *Proceedings of the 5th ACM SIGSOFT Symposium on Software Reusability (SSR99), Los Angeles, California, United States*, 1999, pp. 122-131.
6. Clements P, Northrop L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.
7. Ganesa D, Muthig D, Yoshimura K. Predicting return-on-investment for product line generations. *Proceedings of the 10th International Software Product Line Conference (SPLC'06), 21-24 August 2006, Baltimore, Maryland, USA*, 2006, pp. 13-24.
8. Böckle G, Clements P, McGregor JD, Muthig D, Schmid K. Calculating ROI for Software Product Lines. *IEEE Software* **2004**; **21**(**3**), pp. 23-31.
9. Osterweil LJ. Software Processes are Software Too. *Proceedings of the 9th International Conference on Software Engineering (ICSE 1987), Monterey, California, USA, 1987*, 1987, pp. 2-13.
10. Becker U, Hamann D, Verlage M. 1997. Descriptive Modeling of Software Processes. ISERN Report 97-10, Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany.
11. Armbrust O, Katahira M, Miyamoto Y, Münch J, Nakao H, Ocampo A. Scoping Software Process Models - Initial Concepts and Experience from Defining Space Standards. *Proceedings of the International Conference in Software Process (ICSP2008), Leipzig, Germany, 2008*, 2008, pp. 160-172.
12. Bella F, Münch J, Ocampo A. Observation-based Development of Software Process Baselines: An Experience Report. *Proceedings of the Conference on Quality Engineering in Software Technology (CONQUEST), September 22-24 2004, Nuremberg, Germany*, 2004.
13. Biffl S, Halling M. Managing Software Inspection Knowledge for Decision Support of Inspection Planning. In: Aurum, A., Jeffery, R., Wohlin, C., Handzic, M. (eds.): Managing Software Engineering Knowledge. Springer Verlag: Berlin; 2003.
14. Schweikhard T. Identification of inspection-variation-factors for a decision-support-tool. Diplomarbeit, Fachbereich Informatik, Technische Universität Kaiserslautern, 2006.
15. Vegas S, Basili VR. A Characterization Schema for Software Testing Techniques. *Empirical Software Engineering* **2005**; **10**(**4**), pp. 437-466.
16. Denger C, Elberzhager F. 2006. A Comprehensive Framework for Customizing Quality Assurance Techniques. IESE-Report No. 118.06/E, Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany.
17. Avison DE, Wood-Harper AT. Information Systems Development Research: An Exploration of Ideas in Practice. *The Computer Journal* **1991**; **34**(**2**), pp. 98-112.
18. Fitzgerald B, Russo NL, O'Kane T. Software Development Method Tailoring at Motorola. *Communications of the ACM* **2003**; **46**(**4**), pp. 65-70.
19. Parnas DL. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering* **1976**; **SE-2**(**1**), pp. 1-9.
20. Schmid K. Planning Software Reuse - A Disciplined Scoping Approach for Software Product Lines. PhD

Thesis. Fachbereich Informatik, Universität Kaiserslautern, 2003.

21. John I, Knodel J, Lehner T, Muthig D. A Practical Guide to Product Line Scoping. *Proceedings of the 10th International Software Product Line Conference (SPLC 2006), 21-24 August 2006, Baltimore, Maryland, USA*, 2006.
22. Bayer J, Kose M, Ocampo A. Improving the Development of e-Business Systems by Introducing Process-Based Software Product Lines. *Proceedings of the 7th International Conference on Product-Focused Software Process Improvement (PROFES'2006), Amsterdam, The Netherlands*, 2006, pp. 348-361.
23. Cohen L. *Quality Function Deployment: How to Make QFD Work for You*. Addison Wesley Longman, 1995.
24. Basili VR, Rombach HD. Support for Comprehensive Reuse. *IEEE Software Engineering Journal* **1991**; **6**(**5**), pp. 303-316.
25. International Organization for Standardization. *ISO/IEC 12207:1995*. ISO/IEC: Geneva, Switzerland, 1995.
26. Simidchieva BI, Clarke LA, Osterweil LJ. Representing Process Variation with a Process Family. *Proceedings of the International Conference on Software Process (ICSP 2007), May 19-20, 2007, Minneapolis, MN, USA*, 2007, pp. 109-120.
27. Schmid K. A Comprehensive Product Line Scoping Approach and Its Validation. *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), May 19-25, 2002, Orlando, Florida, USA*, 2002, pp. 593-603.
28. International Electrotechnical Commission. *IEC 61508*. IEC: Geneva, Switzerland, 2005.
29. Formal Methods Europe, http://www.fmeurope.org/, retrieved 2008-08-12.
30. Osborn AF. *Applied Imagination*. Charles Scribner's Sons: New York, USA, 1957.
31. Bittner K, Spence I. *Use Case Modeling*. Addison-Wesley Longman: Amsterdam, 2002.
32. Form feeds function: The role of storyboards in requirements elicitation, http://www.ibm.com/developerworks/rational/library/dec05/krebs/index.html, retrieved 2008-08-12.
33. Linstone HA, Turoff M. *The Delphi Method: Techniques and Applications*. Addison-Wesley, 1975.
34. Mills HD, Dyer M, Linger RC. Cleanroom Software Engineering. *IEEE Software* **1987**; **4**(**5**), pp. 19-25.
35. Baudoin C, Hollowell G. *Realizing the Object-Oriented Lifecycle*. Prentice Hall, 1996.
36. Belady LA. Leonardo: the MCC software research project. In: Ng, P. A., Yeh, R. T. (eds.): Modern software engineering, foundations and current perspectives. Van Nostrand Reinhold Co.: New York, NY, USA; 1989.
37. Yourdon E, Constantine LL. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall: Englewood Cliffs, NJ, USA, 1979.
38. David HA. *The Method of Paired Comparisons*. Lubrecht & Cramer, Limited, 1988.
39. Spearmint, http://www.iese.fhg.de/fhg/iese/research/quality/pam/index.jsp.