The Journal of Systems and Software 84 (2011) 285–300

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Key activities for product derivation in software product lines

Rick Rabiser[a,1], Pádraig O'Leary[b,*], Ita Richardson[c,2]

[a] Christian Doppler Laboratory for Automated Software Engineering, Johannes Kepler University, Altenberger Str. 69, 4040 Linz, Austria
[b] RiSE – Reuse in Software Engineering and Computer Science Department, Federal University of Bahia, Salvador, BA, Brazil
[c] Lero – The Irish Software Engineering Research Centre, University of Limerick, Ireland

ARTICLE INFO

Article history:
Received 1 February 2010
Received in revised form 17 June 2010
Accepted 26 September 2010
Available online 7 October 2010

Keywords:
Software product lines
Product derivation, Process

ABSTRACT

More and more organizations adopt software product lines to leverage extensive reuse and deliver a multitude of benefits such as increased quality and productivity and a decrease in cost and time-to-market of their software development. When compared to the vast amount of research on developing product lines, relatively little work has been dedicated to the actual use of product lines to derive individual products, i.e., the process of product derivation. Existing approaches to product derivation have been developed independently for different aims and purposes. While the definition of a general approach applicable to every domain may not be possible, it would be interesting for researchers and practitioners to know which activities are common in existing approaches, i.e., what are the key activities in product derivation. In this paper we report on how we compared two product derivation approaches developed by the authors in two different, independent research projects. Both approaches independently sought to identify product derivation activities, one through a process reference model and the other through a tool-supported derivation approach. Both approaches have been developed and validated in research industry collaborations with different companies. Through the comparison of the approaches we identify key product derivation activities. We illustrate the activities' importance with examples from industry collaborations. To further validate the activities, we analyze three existing product derivation approaches for their support for these activities. The validation provides evidence that the identified activities are relevant to product derivation and we thus conclude that they should be considered (e.g., as a checklist) when developing or evaluating a product derivation approach.

1. Introduction and motivation

There is a clear trend away from single systems to product lines in software engineering (Clements and Northrop, 2001; Pohl et al., 2005; van der Linden et al., 2007b). Software product lines (SPL) aim to leverage extensive reuse in software development to address many of the challenges in software development such as increasing quality requirements and competition in a global market. Software product line engineering (SPLE) involves domain engineering (building the product line) and application engineering (building products based on the product line). In domain engineering, reusable assets (e.g., requirements, components, documentation, test cases) are developed and their commonalities and variability are explicitly defined, typically using variability models. A significant body of research is available on approaches and notations for variability modelling and management, for example (Czarnecki and Kim, 2005; Gomaa, 2004; Pohl et al., 2005; Schmid and John, 2004). In application engineering, concrete products are built based on these reusable assets. Product derivation is a key process in application engineering and addresses the selection and customization of assets from the product line (utilizing the provided variability) to satisfy customer or market requirements (Deelstra et al., 2005). It is important to work on minimizing product-specific development in application engineering and maximize reuse.

In practice, a number of publications have shown that product derivation must not be underestimated. For example, (Griss, 2000) identifies the inherent complexity and the required coordination in the derivation process by stating that "*. . .as a product is defined by selecting a group of features, a carefully coordinated and complicated mixture of parts of different components are involved*". As (Deelstra et al., 2005) point out: the derivation of individual products from shared software assets is still a time-consuming and expensive activity in many organizations. Both publications base their statements on experiences made with product derivation in industry. Our own experiences in research industry collaborations also confirm that product derivation is often underestimated. A strong focus in SPLE has to be on domain engineering, i.e., building up the product line. However, product derivation brings the return of investment required for setting up the product line in the first

* Corresponding author.
E-mail addresses: rabiser@ase.jku.at (R. Rabiser), padraig.oleary@rise.com.br (P. O'Leary), ita.richardson@lero.ie (I. Richardson).
[1] Tel.: +43 0 732 2468 8873; fax: +43 0 732 2468 8878.
[2] Tel.: +353 0 61233799; fax: +353 0 61213036.

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.09.042

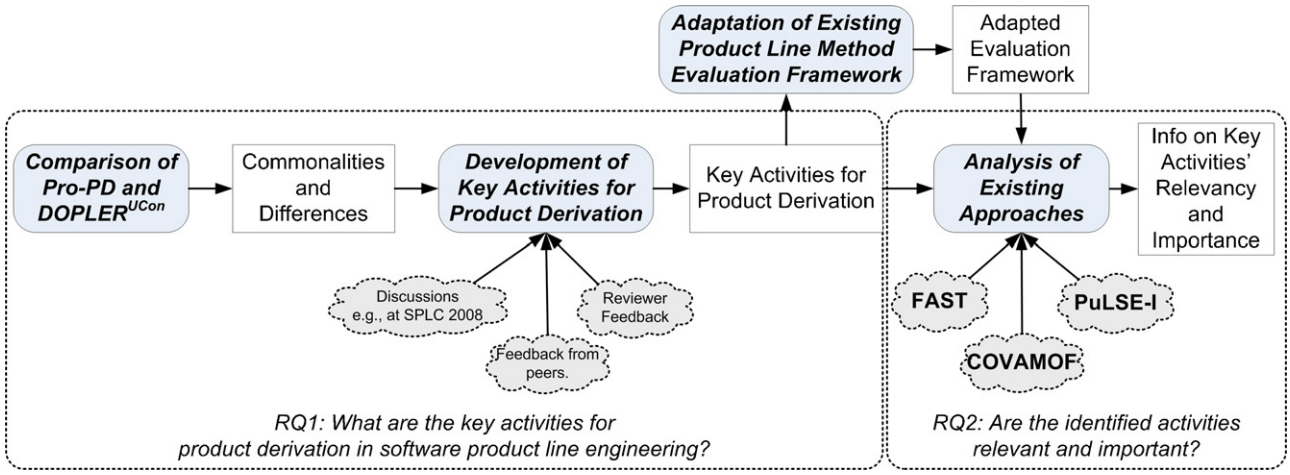

Fig. 1. Research method and research questions.

place by allowing to derive customized products quickly and in an automated way.

Research in SPL has, in the past, focused more on how to scope, define, and develop product lines rather than on how to effectively utilize them in product derivation. A recent systematic literature review (Rabiser et al., 2010) however shows an increasing number of publications, conference tracks, and workshops over the last decade demonstrating the general interest in product derivation. While the requirements for product derivation tool support have been outlined (Rabiser et al., 2010), there is still no clear picture regarding the activities to be supported. Available product derivation approaches and tools have been developed independently to address requirements in different contexts or domains.

Two such approaches are:

(i) Pro-PD (*Pro*cess reference model for *P*roduct *D*erivation) was developed at Lero (the Irish Software Engineering Research Centre) with the goal of defining a process reference model for product derivation as a foundation for situation-specific process approaches to product derivation. Pro-PD focuses on the activities, roles and work artefacts used to derive products from a software product line. Pro-PD uses process patterns that capture solutions to product derivation process challenges (e.g., co-ordinating product-platform synchronisation) as building blocks for creating a product derivation process instance. Pro-PD, its development, and its validation are also described in O'Leary (2010).

(ii) DOPLER^{UCon} (*D*ecision-*O*riented *P*roduct *L*ine *E*ngineering for effective *R*euse: *U*ser-centered *Con*figuration) was developed at the Christian Doppler Laboratory for Automated Software Engineering (Johannes Kepler University (JKU) Linz, Austria) driven by industry needs with the goal to define a user-centred, tool-supported product derivation approach. DOPLER^{UCon} is one of two parts in a decision-oriented product line engineering approach called DOPLER. The other part – DOPLERP^{VM} (Dhungana et al., 2010) – supports variability modelling and management. DOPLERP^{UCon} aims to support both domain experts like sales staff or managers as well as engineers in product derivation based on DOPLER variability models. DOPLERP^{UCon}, its development, and its validation are also described in (Rabiser, 2009).

Both approaches independently sought to identify product derivation activities, Pro-PD through its process reference model and DOPLERP^{UCon} through its tool-supported product derivation approach. Neither approach was designed exclusively for a particular organization or domain but the development of both approaches was driven by industry needs and experiences. The two approaches have already been applied in different cases (cf. Section 2).

In a research collaboration between Lero and JKU we have compared our approaches in detail and identified key activities for product derivation common to both approaches. While the two approaches have been developed in independent projects, with different goals and for different purposes, we still found many interesting parallels. In a previous publication (O'Leary et al., 2009) we presented an overview of our first results, i.e., we described key activities, important issues and lessons learnt for product derivation. In this paper we present details about the comparison and focus on the identification and validation of product derivation key activities. We illustrate the key activities with examples from industry collaborations at both Lero and JKU, and provide evidence for their relevance by systematically analyzing three often-cited and well-known product derivation approaches for their support for these activities.

2. Research method

The goal of this research is to define key activities for product derivation through comparing two product derivation approaches developed by the authors in two different, independent research projects. While a general approach to product derivation might not be possible, we envision that a list of activities that are common in existing approaches will help researchers and practitioners when developing, adapting or evaluating a product derivation approach.

More specifically we are investigating two research questions:

- *What are the key activities for product derivation in software product line engineering?* We elicit the activities by comparing our two approaches in detail and motivate the activities using examples from industry collaborations.
- *Are the identified activities relevant and important?* We systematically analyze existing product derivation approaches regarding their support for the activities using a validation framework.

Fig. 1 depicts an overview of our research method. We being by comparing our two product derivation approaches to elicit commonalities and differences. Based on these, we developed the key activities for product derivation which we refined based on discussions (remote and at conferences) and feedback from peers. Based on an adapted existing product line method evaluation framework, we finally analyzed the key activities to be able to provide evidence

for their relevancy and importance. We present details on how we performed the research in the remainder of this section.

The decision to use Pro-PD and DOPLER^{UCon} as a basis for our research was influenced by a number of factors. The approaches were both looking at product derivation activities. Pro-PD identified activities for product derivation for its process reference model. DOPLER^{UCon} was designed to provide tool-support for product derivation activities. The approaches were developed by the authors, this reduced risk of misinterpreting the documentation on each approach. Both Pro-PD and DOPLER^{UCon} were independently developed and validated. Both approaches were developed in different domains making use of extensive case study research in their development. A case study is especially helpful in situations where researchers are seeking to develop understandings of the dynamics of a phenomenon in its natural context (Yin, 2003). It is considered to be the optimal approach for researching practice-based problems, where the aim is to represent the case authentically "in its own terms" (Hammersley et al., 2000). Both approaches applied the guidelines of (Runeson and Höst, 2009), who present guidelines for conducting and reporting case study research in software engineering. A more general set of guidelines on empirical research in software engineering that both approaches aimed to follow is presented by Kitchenham et al. (2002). Pro-PD was developed through case study research with Bosch. DOPLER^{UCon} was developed through case study research with Siemens VAI Metals Technologies. Pro-PD used sources in the literature such as an inter-model evaluation with the SEI Product Line Practice Framework (Clements and Northrop, 2001) to prevent bias from the Bosch case study. Similarly for DOPLER^{UCon}, a systematic literature review (Rabiser et al., 2010) was conducted to help generalize the findings from the Siemens VAI case study. Finally, the two approaches have already been applied in different cases, e.g. (Grünbacher et al., 2009; O'Leary, 2010; O'Leary et al., in press, 2008a; Rabiser et al., 2007, 2009).

In the following sections, we first briefly discuss how Pro-PD and DOPLER^{UCon} have been developed and validated, as this is the basis for the research presented in this paper (cf. Section 2.1). We then discuss how we performed the comparison and identified the key activities (Section 2.2). We present the evaluation framework we adapted from (Matinlassi, 2004) and describe how we performed the analysis of existing product derivation approaches to provide evidence for the relevancy and importance of the key activities (Section 2.3).

2.1. Starting point: development and validation of Pro-PD and DOPLER^{UCon}

The goal of this research is to define key activities for product derivation through comparing two product derivation approaches which have been developed by the authors in two different, independent research projects. The development and validation of Pro-PD and DOPLER^{UCon} thus is the basis for the research presented in this paper.

2.1.1. Development and validation of Pro-PD

The preparatory stage of developing Pro-PD was an extensive literature review that revealed a lack of methodological support for product derivation. A preliminary version of Pro-PD was developed based on this review. This preliminary version was iteratively developed and assessed through a series of workshops with academic and industry product line experts. The output of this 4-month iterative development stage was version one of Pro-PD (O'Leary et al., 2007). This version was extended through case study research with Robert Bosch GmbH.[3] The case study investigated product derivation practices within an automotive systems sub-unit. The systems produced comprise both hardware (such as processors, sensors, connectors, and housing) and software. Data collection involved studying internal company documentation, an onsite visit to their headquarters and a 2-day workshop with key employees. By generalizing and discussing the case study observations version one of Pro-PD was revised and version two was developed (O'Leary et al., 2008b). Pro-PD was further developed through a 6-month visit to LASSY lab;[4] where Pro-PD and FIDJI (Perrouin et al., 2008) were mapped. We performed further validation of Pro-PD through an inter-model evaluation with the SEI Product Line Practice Framework (Clements and Northrop, 2001).

2.1.2. Development and validation of DOPLER^{UCon}

In research-industry collaboration with Siemens VAI Metals Technologies,[5] the world leader in engineering and building steel plants, the DOPLER product line approach has been developed. The goal of the collaboration was to support modelling and utilizing the variability of Siemens VAI's software system for the automation of continuous casting in steel plants. The concepts of existing decision-oriented approaches, i.e., by Schmid and John (2004), were preferred by Siemens VAI staff. The decision-oriented DOPLER SPL approach and supporting tools were developed iteratively over a period of 4 years based on existing work and constant feedback and close collaboration with Siemens VAI. Workshops with project managers, software architects, and developers were frequently conducted. DOPLER^{UCon} – the product derivation part of DOPLER – was developed as an integrated, tool-supported approach. At Siemens VAI it is used in pilot projects for software product lines in the metals domain, i.e., product lines for automating process optimization and tracking of particular machines or lines of machines in steel plants such as continuous casters. DOPLER^{UCon} also has been applied in other domains, e.g., in the enterprise resource planning domain (Rabiser et al., 2009) or for Eclipse-based software engineering tools (Grünbacher et al., 2009). Approach, tools, and concepts of DOPLER and DOPLER^{UCon} have frequently been presented at scientific workshops and conferences (Dhungana et al., 2010; Rabiser and Dhungana, 2007; Rabiser et al., 2007, 2009). A systematic literature review (Rabiser et al., 2010) helped to define requirements for the DOPLER^{UCon} approach and tools that go beyond the industry partner's requirements.

2.2. Comparison of our approaches and development of key activities

The idea of comparing Pro-PD with DOPLERP^{UCon} emerged during a meeting of JKU and Lero researchers in February 2008. While Pro-PD was mainly influenced by Deelstra et al. (2005) and a case study with Robert Bosch GmbH, DOPLERP^{UCon} was mainly influenced by the research-industry collaboration with Siemens VAI. While the first approach was developed as a process reference model, the latter was developed focused on adaptable tool support usable in practical settings. These differences motivated our efforts to compare the two approaches. The main motivation at first was to learn from each other and try to improve both approaches. However, we quickly realized that we could also use the results of the comparison to define key product derivation activities.

Based on initial discussions and existing documentation of our two approaches, we created a first high-level mapping in a distributed manner using spreadsheets to visualize commonalities

[3] http://www.bosch.com.
[4] Laboratory of Advanced Software Systems, University of Luxembourg.
[5] http://www.industry.siemens.com/metals/en/.

Table 1
Evaluation framework for analyzing product derivation approaches regarding their support for key activities (adapted from (Matinlassi, 2004)).

Category	Elements	Questions
Context	Specific goal	What is the specific goal of the approach regarding product derivation?
	Product derivation aspect(s)	What aspects of product derivation does the approach cover?
	Application domain(s)	What is/are the application domain(s) the approach is focused on?
	Inputs	What is the starting point for the approach?
	Outputs	What are the (desired) results of the approach?
User	Target group	Which stakeholders are addressed by the approach and how?
Contents	Activities	What activities/steps/sub-processes does the approach define to accomplish product derivation?
	Artefacts	What artefacts are created and managed by the approach?
	Support for key activity 1	To which extent does the approach support key activity 1 (fully: all sub-activities of key activity 1 are supported; partly: some sub-activities of key activity 1 are supported; not supported)?
	...	...
	Support for key activity *N*	To which extent does the approach support key activity *N* (fully: all sub-activities of key activity *N* are supported; partly: some sub-activities of key activity *N* are supported; not supported)?
	Not covered by key activities	What activities/sub-activities does the approach include that are not covered by the defined key activities/sub-activities?
Validation	Maturity	Has the approach been validated in practical industrial settings?

and differences between the two approaches. More specifically, the spreadsheet contained columns listing each Pro-PD activity, the activity's purpose, a statement whether DOPLER^{UCon} supports the Pro-PD activity fully/partly/not at all, the involved DOPLER^{UCon} activities and an explanation of the mapping, i.e., how the Pro-PD activity is supported/partly supported/not supported by DOPLER^{UCon}.

Using such a high-level mapping, the authors of this paper met at the International Software Product Line Conference (SPLC) 2008 (http://www.splc.net) to analyse the first results, discuss open issues, and detail the comparison. After this meeting we regularly conducted telephone conferences over a 6-month period to work on the details.

During this period, we were able to define key activities for and important issues of product derivation (O'Leary et al., 2009). Feedback from reviewers and discussions during and after SPLC 2009 then allowed us to further develop our ideas.

2.3. Validation of key activities

We refined the initial activities defined in our earlier paper (O'Leary et al., 2009) and collected illustrative examples from our industry projects (cf. Section 4). To validate the activities (with the aim to provide evidence for their relevancy and importance) we systematically analyzed the support for these activities in often cited and well-known existing approaches (cf. Section 5.1). We selected COVAMOF (Sinnema et al., 2006a), FAST (Weiss and Lai, 1999), and PuLSE-I (Bayer et al., 2000) because in our literature surveys we both independently identified that these three approaches were influential through their frequent citations. Furthermore, due to the multitude of publications on each approach, clearly defined descriptions existed. This does not necessarily mean they are ideal for our validation but a good starting point is to look at prominent existing approaches for parallel findings. Also, the three approaches were developed independently for different purposes and in different contexts.

Analyzing existing approaches for their support for the key activities allows finding out whether the key activities we defined are not only relevant for Pro-PD and DOPLER^{UCon} but are in fact also supported/implemented by other approaches. This contributes to the validation of our research as it provides evidence that the activities we defined are relevant and important; especially as the three approaches analyzed also are (or have been) used in practical settings. With regard to generalization and external validity, five cases (our two approaches and the three others we analyzed) are not enough to prove that the key activities will be relevant and important for every context, domain, or organization. However, they are evidence that it makes sense to consider the defined activities when developing or evaluating a product derivation approach.

To enable systematic analysis, we needed a suitable evaluation framework. While a framework specifically for evaluating product derivation approaches does not exist, we found a framework developed for the purpose of evaluating software product line architecture design methods (Matinlassi, 2004) which we adapted for our purpose. We used this framework as a basis for our validation for two reasons. Firstly, it provided a simple tabular evaluation structure. Secondly, it had previously been published at the ICSE Conference which ensures it has been peer-reviewed sufficiently. As our goal was to validate the key activities we defined by studying how they are supported by often cited and well-known existing approaches, we found a simple, tabular framework sufficient. It allows us to compare the approaches systematically but on a level high enough to also enable the presentation of our results.

We adapted the questions regarding the category *context* proposed by Matinlassi from "product line architecture design method" to "product derivation approach" (cf. Table 1). We adopted only one element for the category *user* (target group) as our focus is on evaluating the contents (support for key activities) and not the user support. For the category *contents*, we adopted the first two elements activities and artefacts. Instead of focussing on product line architecture (elements defined by Matinlassi: architectural viewpoints, language, variability, tool support), we defined one element for each key activity to evaluate (cf. Section 4). For the *validation* category, we adopted the element maturity and not quality because we are not interested in the approaches' procedures to validate the results of product derivation but more in whether the approaches themselves have been validated. Table 1 depicts the adapted evaluation framework. We present and discuss the results of the analysis we conducted based on the framework in Section 5.

3. Comparison of two product derivation approaches

We provide a brief overview of Pro-PD and DOPLER^{UCon} (for more details on the two approaches refer to (O'Leary, 2010; Rabiser, 2009)) and summarize the results of comparing our two approaches. In Tables 2–4 we list all the activities contained within Pro-PD and compare them with the activities in DOPLER^{UCon}, i.e., for each Pro-PD activity, we analyzed whether DOPLER^{UCon} supports the activity fully, partly or not at all and how it provides this support (cf. Section 2.2). We discuss interesting parallels and important differences we found.

Table 2
Overview of mapping Pro-PD pre-derivation activity to DOPLER^{UCon}.

Pro-PD Pre-derivation activity	Purpose	Supported by DOPLER^{UCon}?	DOPLER^{UCon} (sub-)activities involved
Translate customer requirements	"Translate" customer requirements to domain language	*Not Supported* (customer requirements are assumed to be available in domain language)	–
Coverage analysis	Determine requirements satisfied through base configuration and document mapped/unmapped requirements	*Supported* (possible to start with an existing configuration which contains requirements for the required configuration. Mapping to new requirements and documentation therefore can be achieved)	Review variability model; elicit and capture requirements
Customer negotiation	Negotiate unmapped customer requirements and check their feasibility	*Supported* (by relating new requirements with variability. Based on this information, effort and risk level for realization can be defined - Requirements are negotiated with the customer)	Relate product-specific requirements to the available variability; negotiate product-specific requirements
Create the product-specific requirements	Involves merging mapped and negotiated requirements	*Supported* (negotiations lead to changes in captured requirements)	Capture product-specific requirements; negotiate product-specific requirements
Scope requirements implementation	The functional and non-functional requirements for the system are specified and scoped. Requirements are designated for platform implementation or product-specific	*Partly supported* (Captured requirements can be assigned arbitrary types. This can also be used to define whether they are platform-specific or product-specific)	Capture product-specific requirements
Create the product-specific test cases	Create test-cases using the product-specific requirements	*Partly supported* (assumed to happen in additional development phase but not defined how)	Additional development
Allocate requirements	Allocate requirements to relevant disciplines, e.g., hardware discipline, algorithms. Prioritise implementation iteration of particular product requirements	*Partly supported* (related decisions grouped in tasks can represent disciplines. Requirements related with decisions in a task are then also allocated to a discipline. Prioritization of iterations is not supported.)	Define roles and tasks
Create guidance for decision makers	Guidance is linked into the product-specific requirements. Remaining variability must be explained to deal with complexity issues in representing product line variability	*Supported* (arbitrary guidance (e.g., multimedia) can be created for open decisions. These can be related with requirements)	Create/manage guidance

3.1. Overview of Pro-PD

From a high-level point of view Pro-PD comprises the following activities which need to be conducted in an iterative manner (see Fig. 2).

In *Pre-Derivation* the preparatory steps required before actual derivation can begin are performed. Pre-Derivation is aimed at forming the product-specific requirements based on customer requirements and negotiation with the platform team. Requirements are prioritized and assigned to development iterations. Sub-activities can be seen in Table 2.

In *Product Configuration* the goal is to build the product by reusing as much as possible the platform artefacts and minimizing the amount of product-specific development effort. Requirements

Table 3
Overview of mapping Pro-PD product configuration activity to DOPLER^{UCon}.

Pro-PD product configuration activity	Purpose	Supported by DOPLER^{UCon}?	DOPLER^{UCon} (sub-)activities involved
Derive new configuration	Derive a new product configuration from the platform architecture	*Supported* (deriving a new product by making decisions goes hand in hand with selecting assets due to the explicit linkage of assets with decisions in DOPLER)	Make decisions and customize assets
Select closest matching configuration	Select a base configuration from existing/previous configurations	*Supported* (possible to start with an existing configuration, i.e., an existing derivation model. Also possible to "import" decisions from earlier configurations)	Adapt variability model
Select platform components	Components are selected from the collection of platform components for addition to or replacement of components in the base configuration	*Supported* (goes hand in hand with deriving a new configuration. Explicit linkage of assets with decisions allows selecting platform components by making decisions in the base configuration (derivation model)	Make decisions and customize assets
Integrate and create product build	The base product configuration and the selected platform components are integrated and integration testing is performed	*Partly Supported* (the selected base configuration is represented by a derivation model. Platform components are selected by making decisions. No manual integration is necessary. Integration testing relies on correctness and completeness of the model	Make decisions and customize assets; generate configurations
Integration testing	Validates the platform assets for this particular configuration. The integration tests should reuse platform test artefacts	*Partly Supported* (assumed to happen in additional development phase but not defined how)	Additional development

Table 4
Overview of mapping Pro-PD product development and testing activity to DOPLER^{UCon}.

Pro-PD product development and testing activity	Purpose	Supported by DOPLER^{UCon}?	DOPLER^{UCon} (sub-)activities involved
Identify required product development	Satisfy requirements which could not be satisfied through reuse of platform assets	*Supported* (in application requirements engineering additionally required development is identified through mapping with the available variability and negotiation with the customer)	Relate product-specific requirements to the available variability; negotiate product-specific requirements
Develop/adapt components	The source code to implement new functionality or to adapt an existing platform component at product level is developed	*Supported* (main purpose of additional development phase)	Additional development
Component unit testing	When a component is built or adapted, initial or tailored versions of a component will need to be tested rigorously through unit testing	*Partly Supported* (can happen in additional development phase but not defined how to do this)	Additional development
Integrate and create product build	The developed or adapted components are integrated into the integrated product configuration	*Supported* (main purpose of the product integration and deployment phase)	Product integration and deployment
Run system tests	The product has to be checked for compliance with the product-specific requirements (Deelstra et al., 2005). Tests used at platform level can be reused	*Partly Supported* (assumed to happen during product integration and deployment but not defined how to do this)	Product integration and deployment
Assess results	The success or failure of the product delivery process is determined. Improvements that can be made to the delivery process are discussed	*Partly Supported* (assumed to happen during product line evolution after or in parallel to product derivation)	Product line evolution (analyze new assets; analyze new requirements)
Provide feedback to platform team	Feedback is provided to the platform team on core asset usage during the project. Also, the product team identifies product-specific components that the platform could potentially benefit from through adoption	*Supported* (main purpose of product line evolution phase in DOPLER^{UCon})	Product line evolution (analyze new assets; analyze new requirements)

are developed iteratively based on their priority given in the previous step, iterations continue until all customer requirements have been fulfilled. Sub-activities can be seen in Table 3.

During *Product Development and Testing*, product-specific development is undertaken. Both the changes and the final product are tested to ensure it satisfies customer expectations. Sub-activities can be seen in Table 4.

Pro-PD is defined at a high level, but in order to create a working company-specific model these processes need to be specialized and a lower level of model abstraction needs to be constructed. Pro-PD provides a link to automated approaches by providing product derivation context and facilitating tool support for the overall process.

3.2. Overview of DOPLER^{UCon}

From a high-level point of view DOPLERP^{UCon} comprises the following activities which need to be conducted in an iterative manner (see Fig. 3 and cf. (Rabiser, 2009)).

In *Configuration Preparation* (1) project managers prepare DOPLER variability models for a concrete project/customer. They capture customer and project information and, based on high-level requirements known early on, resolve variability. They further define the roles and tasks of the people involved in product derivation. Additionally, domain experts model guidance to provide additional rationale or recommendations for decision-making. The sub-activities of configuration preparation are: define project, review variability model, create and manage guidance, adapt variability model, and define roles and tasks. Configuration preparation is supported by the tool ProjectKing (Rabiser et al., 2007). The output is a project-specific version of the original variability model called the derivation model.

Product Configuration (2) starts with presenting decisions to users according to their roles and tasks defined in the derivation model. Typically, sales people communicate with customers to elicit their detailed requirements and make decisions accordingly. Engineers typically perform more technical configuration based on sales decisions. Sub-activities of product configuration are: review available variability, communicate variability, make decisions and customize assets, and generate configurations. Product configuration is supported by the ConfigurationWizard (Rabiser and Dhungana, 2007) tool. The outputs are selected and customized assets.

Application Requirements Engineering (3) aims at capturing, negotiating, and managing the requirements that cannot be fulfilled by the product line. The sub-activities are elicit and capture product-specific requirements, relate product-specific requirements to the available variability, and negotiate product-specific requirements. ConfigurationWizard supports capturing such requirements and relating them to existing assets and decisions (Rabiser et al., 2007).

During *Additional Development* (4) product-specific requirements are addressed. Developers have to take into account the already existing assets and their relationships. New developments need to be tested. DOPLER^{UCon} does not define concrete sub-activities because it assumes this activity to be too domain-specific.

Product Integration and Deployment (5) means integrating derived assets with new developments and preparing them for deployment. Again, the concrete steps involved differ from company to company. Configuration Wizard can however be extended with domain-specific tools, e.g., to enable generating build files or settings files.

In *Product Line Evolution* (6) domain and application engineers collaborate to find out which of the additionally developed and/or changed assets should become part of the product line. Sub-activities are: analyze new assets and analyze new requirements.

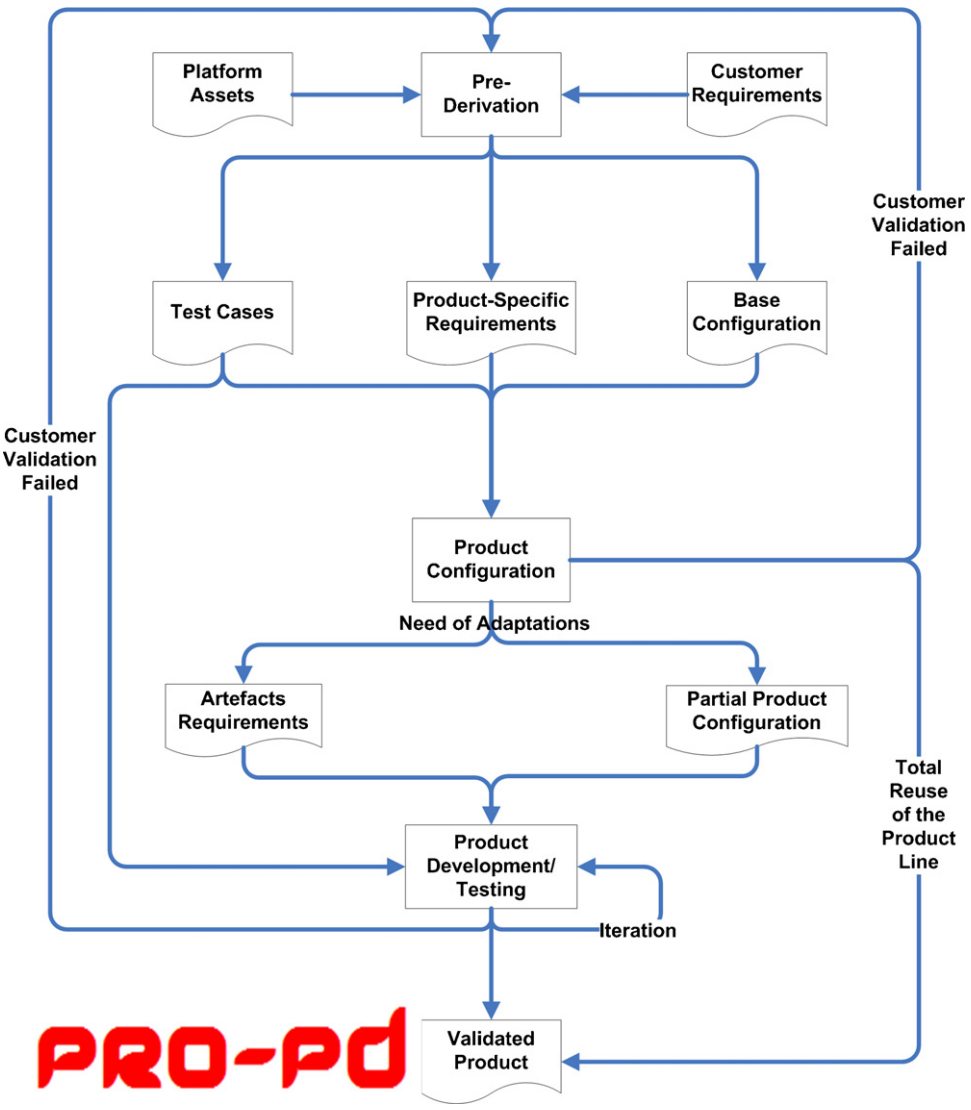

Fig. 2. Pro-PD (O'Leary, 2010).

3.3. Comparison of Pro-PD Pre-Derivation with DOPLER^{UCon}

Pre-derivation in Pro-PD is an activity where derivation is prepared. In DOPLER^{UCon} the activity configuration preparation has a similar purpose. This is a problematic area of product derivation because all further activities depend on these early steps. Some of the Pro-PD pre-derivation sub-activities are supported by DOPLER^{UCon} as part of its application requirements engineering activity. In Table 2, we summarize which sub-activities of the pre-derivation activity in Pro-PD (cf. Section 3.1) are supported by DOPLERP^{UCon} (cf. Section 3.2) and how they are supported.

From the eight sub-activities defined within the pre-derivation activity of Pro-PD, seven sub-activities are fully or partly supported by DOPLERP^{UCon}. Only one activity i.e. "translate customer requirements" is not supported. This missing activity in DOPLERP^{UCon} can be explained with the differences in approaches to customer management. In a collaborative environment, as assumed by DOPLERP^{UCon}, customer requirements are typically delivered in domain language.

3.4. Comparison of Pro-PD product configuration with DOPLER^{UCon}

Product configuration is focused on the derivation of a product configuration from the product line, i.e., selecting, customizing, and integrating reusable assets. In both approaches this activity is called product configuration. In Table 3, we summarize which sub-activities of the product configuration activity in Pro-PD (cf. Section 3.1) are supported by DOPLERP^{UCon} (cf. Section 3.2) and how they are supported.

From the five sub-activities identified within the product configuration activity of Pro-PD, we can see that all five sub-activities are fully or partly supported by DOPLERP^{UCon}. DOPLER^{UCon} assumes domain-specific plug-ins to be developed for the partly supported activities. One major difference between the two approaches is the assumption of DOPLER^{UCon} that testing will be performed in the additional development phase and that the approach does not define how because it assumes this to be too domain-specific. Furthermore, due to the model-based approach the base configuration is represented by a dedicated model in DOPLER^{UCon}, the derivation

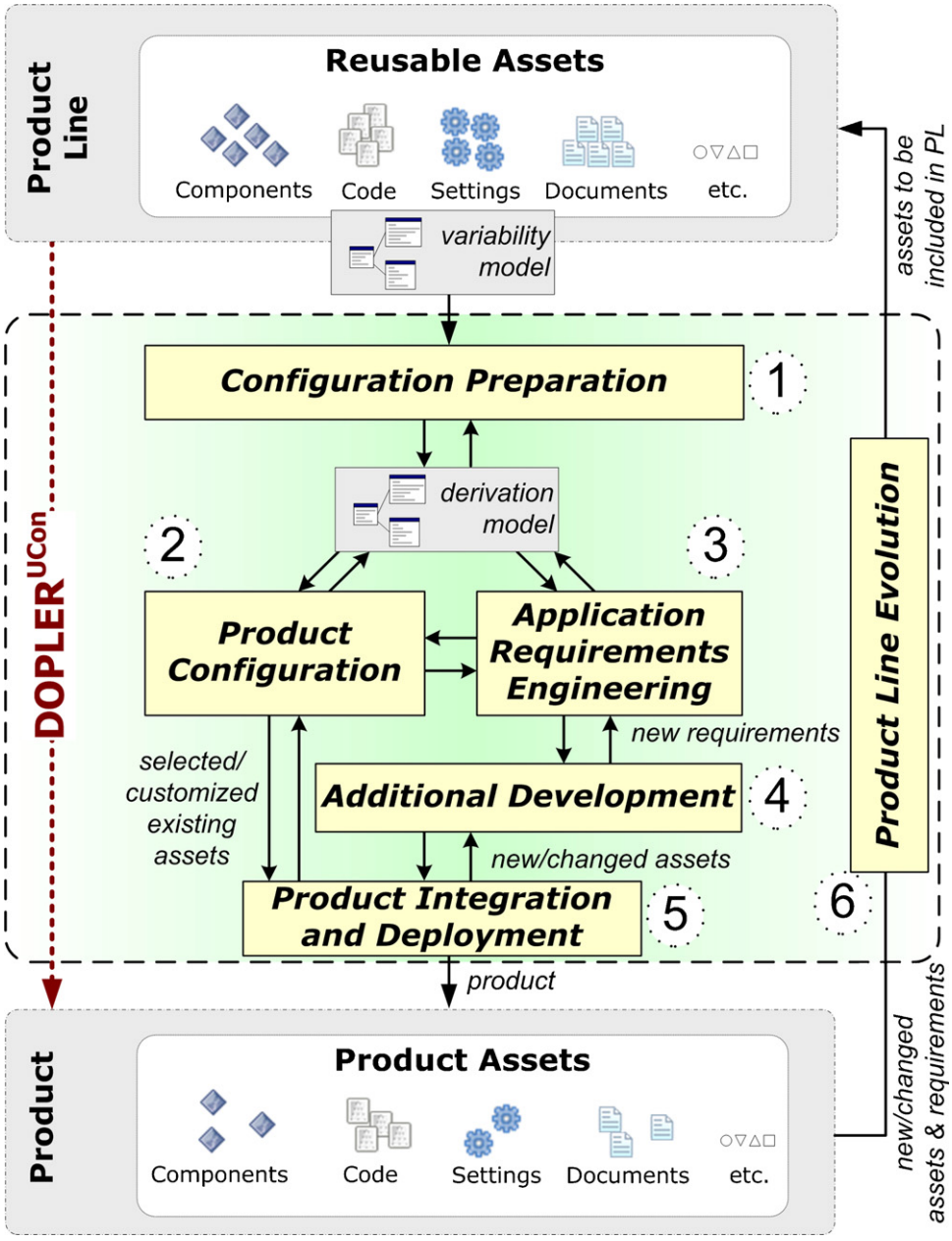

Fig. 3. DOPLER^{UCon} (Rabiser, 2009).

model. In this model a base configuration can be selected or managed by making decisions. Making these decisions directly lead to the inclusion and/or adaptation of (platform) components which makes integration of the base configuration and selected platform components unnecessary. However, the correct "integration" in this case depends on the correctness and completeness of the model on which it is based. Currently, DOPLER provides only basic support for validating models in this regard (syntactical correctness and consistency with the architecture (Vierhauser et al., 2010)).

3.5. Comparison of Pro-PD product development and testing with DOPLER^{UCon}

The Pro-PD product development and testing phase is where customer requirements are addressed which could not be fulfilled by the product line and the provided variability. In DOPLER^{UCon} the activities of the product development and testing phase occur in the application requirements engineering, the additional development, the product integration and deployment, and the product line evolution phases. In Table 4, we summarize which sub-activities of the product development and testing activity in Pro-PD (cf. Section 3.1) are supported by DOPLERP^{UCon} (cf. Section 3.2) and how they are supported.

From the seven sub-activities identified within the product development and testing activity of Pro-PD, all seven are fully or partly supported by DOPLERP^{UCon}. DOPLER^{UCon} assumes domain-specific plug-ins to be developed for the partly supported activities.

3.6. Conclusions

Even though the two approaches were developed separately and with different aims and purposes in mind, we discovered many interesting parallels (cf. Tables 2–4) and found comparably few differences. Both approaches independently sought to identify

product derivation activities, Pro-PD through its process reference model and DOPLERP^{UCon} through its tool-supported product derivation approach.

Requirements management is one area where Pro-PD and DOPLERP^{UCon} have different approaches. In DOPLERP^{UCon} customer requirements that cannot be satisfied by the product line are captured and documented together with relations to existing variability. Pro-PD takes unsatisfied customer requirements and performs customer negotiation where the feasibility of implementing customer requirements is investigated and discussed with the customer extensively. DOPLERP^{UCon} does not clearly define how to handle/negotiate unsatisfied customer requirements, it is "only" possible to capture these requirements and mark them as product-specific implementations.

The definition of *iterative development* cycles for additional development is only partly supported in DOPLERP^{UCon}. Additional attributes can be defined for requirements and these can be used to allocate requirements to specific iterations. Pro-PD is designed with iterative development cycles in mind. The specification of product-specific requirements goes hand in hand with allocation of these requirements to specific iterations based on prioritization and customer negotiation.

Customer involvement in product derivation is often portrayed as a combative relationship involving negotiation between separate parties with contrasting motivations. This is how customer involvement is portrayed in Pro-PD. By comparison, the DOPLER^{UCon} approach is a more collaborative approach that assumes both the product team and the customer are making decisions which will serve both their interests.

Pro-PD is applicable to organizations seeking to achieve *regulatory compliance* such as Auto-SPICE (Automotive Special Interest Group, 2008) due to specific practices dedicated to the formation of requirements specifications. DOPLERP^{UCon} would require additional requirements specification practices to make it applicable in regulated environments.

DOPLERP^{UCon} is focused on providing user-centred *tool support* for product derivation. For example, different views on existing variability are provided for different users to allow them making decisions. Pro-PD does not define which activities should be supported by tools and how they can be supported.

Product derivation *user management* is also not directly supported in Pro-PD. While DOPLERP^{UCon} requires defining the people involved in product derivation and their roles and responsibilities (who decides what and when), Pro-PD does not explicitly enforce such a user management.

4. Key activities for product derivation

Based on the mapping of our two approaches we defined key activities (cf. Fig. 4) for product derivation to address our first research question (cf. Section 1): *What are the key activities for product derivation in software product line engineering?* We illustrate the activity descriptions with examples from case studies we conducted with Robert Bosch GmbH (Pro-PD) (O'Leary, 2010) and Siemens VAI (DOPLER^{UCon}) (Rabiser, 2009). The case studies were conducted in two different domains, one considered product derivation practices within automotive systems while the other investigated product line engineering support for a software system for the automation of continuous casting in steel plants.

4.1. Key activity 1: preparing for derivation

In both our approaches as well as in existing work it is noted that derivation does not start "from scratch", i.e., by just selecting features or making decisions, for example, as defined in a variability model.

When developing Pro-PD, the need for a more sophisticated requirements management process when dealing with large distributed teams was observed. For instance, when communicating information across large distributed teams, such organizations tend to be overly reliant on documentation. An organization's documentation often becomes bloated as teams attempt to capture too much. Such overly detailed documentation decreases traceability of relevant information and results in failure to correctly identify artefacts for reuse especially in team sizes where the transfer of tacit knowledge is prohibitive. In Pro-PD these case study observations were captured. During the early phases (preparing for derivation), the customer requirements were translated into a set of internal company documents. These documents were processed and augmented through various tasks where requirements are analyzed for reuse potential and then assigned to responsible disciplines.

During the development of DOPLER^{UCon}, the industry partner Siemens VAI's typical projects motivated the need for preparing derivation (Rabiser et al., 2007). Customers often wish to upgrade existing steel plants in order to improve steel quality by deploying Siemens VAI's most recent casting technologies. In this case the software needs to interoperate with diverse legacy software and hardware systems of the customer. Requirements regarding existing hardware and software have to be captured and mapped with the existing variability of the product line. Other customers require a complete plant solution. In such cases, it is often possible to reuse a base configuration from past projects as a starting point. The duration of typical customer projects is between a few months up to more than a year and involves numerous meetings between sales people and customers as well as sales staff and engineers. The roles and tasks of the involved people therefore have to be defined to address these stakeholders' needs and responsibilities in derivation. Also, guidance is essential, especially for domain experts, who are confronted with many – often technical – decisions.

From both research projects, we thus observed that before actual derivation can start several preparatory activities need to be conducted as listed:

- Specify and translate customer requirements.
- Define base configuration.
- Map customer requirements.
- Define role and task structures.
- Create derivation guidance.

4.1.1. Specify and translate customer requirements

Customer requirements need to be clearly specified as a starting point for product derivation. If necessary, they need to be translated into the internal organizational language. The goal is to prevent terminology confusion and customer-specific description of assets. This has to be done in close collaboration with the customer.

4.1.2. Define base configuration

A "base configuration" may be chosen as a starting point for derivation, i.e., from a set of existing platform configurations. Experiences made in past projects are of great use as similar customers often have comparable requirements. If no (at least partly) matching base configuration can be found, a new one has to be created.

4.1.3. Map customer requirements

Customer requirements are mapped to the base configuration. Requirements which cannot be satisfied by existing assets have to be negotiated with the customer. Effort estimation issues (the estimation of effort required to satisfy unmapped customer requirements through the adaptation of platform assets or additional development) can make customer negotiation difficult. The trade-off here is to meet as many of the customer's needs as pos-

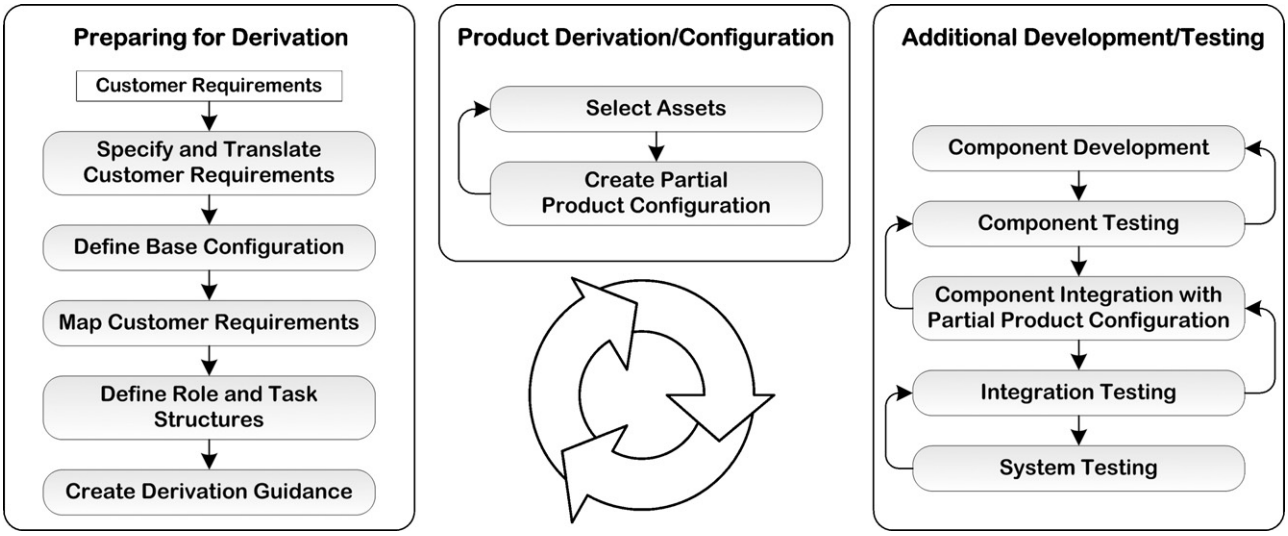

Fig. 4. Key activities for product derivation.

sible while retaining the profitability of the platform assets for the whole product line.

4.1.4. Define role and task structures

The role and task structures for the product derivation project have to be defined. For example, discipline mapping can be performed where product requirements are allocated to relevant disciplines. The goal is to define who is responsible for resolving what remaining variability in product derivation to fulfil the product requirements. This is very helpful to provide different views on variability for different people involved in product derivation and helps to lower the complexity of large decision spaces. Also, as the duration of product derivation projects can be quite long, it is important to know who decided what and when.

4.1.5. Create derivation guidance

Preparing for derivation also means to create guidance for decision-makers. Guidance is essential, especially for domain experts like customers and sales people, who are confronted with many – often technical – decisions or features. Remaining variability must be explained to deal with complexity issues in representing product line variability. Furthermore, different people need to understand different aspects of the provided variability. While sales people interacting with customers need to understand variability from a rather high level, engineers need more details to perform technical configuration. Depending on the roles and tasks of the people involved, different representations of variability are required.

4.2. Key activity 2: product derivation/configuration

The goal of product derivation is to build the product by reusing as much as possible the platform artefacts and minimizing the amount of product-specific development required.

When developing Pro-PD, it was observed how the platform manager informs the platform integrator what configurations should be build. The product architect identifies product components required by the customer and identifies extensions required by the platform architecture to facilitate new requirements. The platform manager will either accept or reject these requests depending on whether they fall under the scope of the platform. The product team identifies the partial configuration to use, a selection of the platform components to reuse, and the setting of parameters for each selected component.

During the development of DOPLER^{UCon}, product derivation has been perceived as a project that can run over a comparably long period (up to more than a year). Based on the defined role and task structures, derivation stakeholders are presented with the variability they are responsible for and have the rights to resolve. Sales people or project managers are assumed to communicate high-level decisions to customers and elicit their requirements. A first high-level customization is performed based on these decisions early in the project. Engineers perform technical configuration afterwards. Simulation based on partial configurations was important in the project. Existing simulator applications of Siemens VAI were used (Rabiser and Dhungana, 2007).

In both research projects we saw that product derivation/configuration has to be an iterative process starting with selecting/customizing a set of assets from the product line, determining possible additionally required developments and testing. Iterations are required until all customer requirements have been fulfilled. The activities that need to be conducted are listed below:

- Select assets.
- Create partial product configuration.

Select assets: based on the role and task structures defined before, assets have to be selected (and customized) from the product line, e.g., by making decisions or selecting features. Tool support is inevitable for this activity. Dependencies and constraints in the variability description and among assets have to be evaluated by this tool support during the decision-making process to ensure the correctness of the selected and customized set of assets.

Create partial product configuration: a partial configuration is created step-by-step in an iterative manner. A partial configuration partially implements a software product in the sense that not all variability has been resolved (Deelstra et al., 2005). Theoretically, at this stage a partial configuration could satisfy customer requirements. However, this is the ideal case and assumes all the customer requirements are covered by the platform. In most industrial projects some additional development will be required. Required development activities have to be defined and prioritized based on customer requirements. Simulation based on partial configurations might be used to support further negotiations (on requirements) with customers.

4.3. Key activity 3: additional development/testing

It is the responsibility of the product development team to implement the required changes at the product level.

In the development of Pro-PD, component development and adaptation typically occurred at product level. First, requirements which cannot be satisfied through the reuse of existing assets are identified. Then, a strategy for component development or adaptation is decided to satisfy these requirements. Any platform changes are applied retrospectively when considered against the scope of the product line. Required architectural changes for new components are negotiated with the platform team. The product team integrates the developed/adapted components with the partial configuration through writing 'glue' code to interface with components. At this stage, product testing can begin.

During the development of DOPLER^{UCon}, additional development and testing also mainly occurred at product level. Changes however also can occur at product line level. Newly developed or adapted components are tested, integrated, and then deployed to the customer. After the end of the product derivation project, domain and application engineers collaboratively decide whether product-specific developments are to be integrated in the product line.

From both research projects we learned that the following activities need to be conducted in additional development/testing:

- Component development.
- Component testing.
- Component integration with partial product configuration.
- Integration testing.
- System testing.

Component development: the source code to implement new functionality or to adapt an existing platform component at product level is developed. New components should be developed with the possibility in mind that they might later be promoted to a platform asset. If a platform component is considerably adapted and considered to have reuse value, it should be termed a new version of the same platform component and added to the platform with an associated definition of its parent (Ahmed et al., 2009).

Component testing: When a component is built or adapted, initial or tailored versions of a component need to be tested rigorously, for example through unit testing. As confirmed by Kauppinen (2003), conventional unit test methods must be utilized as no product line specific methods have been developed so far.

Component integration with partial product configuration: newly developed and adapted assets need to be integrated with the partial product configuration. This can for example require writing sufficient "glue" code to interfaces (Chastek et al., 2002) or even implementing architectural changes to facilitate integration of the developed and adapted assets.

Integration testing: integration testing is essential to find out whether the newly developed and adapted assets interact correctly with the existing architecture: the product has to be checked for consistency and correctness.

System testing: in system testing the product has to be checked for compliance with the product-specific requirements (Deelstra et al., 2005). If the customer requirements for this iteration have been satisfied, the product is delivered. Otherwise, further iterations are required.

4.4. Adaptability

The defined activities require a degree of variability to enable striking the right balance for a particular situation; this variability involves the selection, tailoring and or removal of activities from a process. Activities can be adapted and customized for a particular context, domain or organization. This has the potential for making the defined activities as applicable to a small software development team working on a mobile application as for a large aerospace and defence contractor building a system of systems.

For example, in Bosch it was observed that the embedded software development was a cross-discipline activity. In this context, when defining role and task structures (sub-activity of key activity *preparing for derivation*), "discipline mapping", where requirements are allocated to software, hardware or mechanical disciplines, is a relevant task. However this may not be necessary when developing a product line in another domain with only one type of discipline. For example, in the case study with Siemens VAI, where a software product line for the automation of continuous casting was the focus, discipline mapping was not necessary as the focus of the study was on the software level only. Another sub-activity that is domain-specific is "specify and translate customer requirements". While in case of Bosch, translating requirements was considered necessary; in case of Siemens VAI it was assumed that the customer requirements are already available and represented in the internal organizational language when product derivation starts.

These examples demonstrate that for each domain, context or organization the key activities have to be analyzed and a decision has to be made as to which sub-activities make sense. We consider all sub-activities as optional, except for "select assets" and "create (partial) product configuration", i.e., the product derivation/configuration key activity. An organization might just decide not to prepare for derivation (and just start with deriving and configuring right away) and not to test derived products at all, whatever the consequences may be. Especially, when a high degree of automation is possible, this might make sense. Additional development is optional in the sense that if all customer requirements can be fulfilled by reusing the product line assets, no additional development is required. However, this will seldom be the case in practice.

5. Validation: analyzing existing approaches regarding their support for the key activities

Existing product derivation approaches have been developed with different goals, for different purposes, and in different domains. Some are focused on the early phases of derivation, some are intended to provide a (process) framework for product derivation, and others mainly focus on tool-support. To validate the key activities for product derivation that we defined, we systematically analyzed prominent existing approaches for their support for the defined activities. This addresses our second research question (cf. Section 2): *Are the identified activities relevant and important?* We first provide a brief overview of these approaches and then show the results of our analysis based on the adapted evaluation framework introduced in Section 2.3. Finally, we discuss the results.

5.1. Selected existing approaches

5.1.1. COVAMOF

The product derivation approach for *COVAMOF* (*CO*nfiguration in Industrial Product Families *VA*riability *MO*deling *F*ramework) (Deelstra et al., 2005) consists of two phases: an initial and an iteration phase. During the initial phase, a first product configuration is derived from a product line's assets. This initial configuration is modified in a number of subsequent iterations during the iteration phase until the product sufficiently implements the requirements imposed. Requirements that cannot be accommodated by existing assets are handled by product-specific adaptation or reactive evolution. Parts of COVAMOF have been implemented in the research

tool COVAMOF-VS (Sinnema et al., 2006b). The work by Deelstra et al. provides a framework of terminology and concepts for product derivation. The framework focuses on product configuration and is a high-level attempt at providing the methodological support that (Deelstra et al., 2005) and others (Halmans and Pohl, 2003; Kang et al., 1990; McGregor, 2005; Sinnema et al., 2006b) described to be required for product derivation. COVAMOF has been validated in several industrial case studies including (Deelstra et al., 2005; Sinnema and Deelstra, 2008).

5.1.2. PuLSE-I

The application engineering part of the PuLSE (*Prod*uct *L*ine *S*oftware *E*ngineering) method developed at the Fraunhofer IESE (Institute for experimental software engineering) is called *PuLSE-I* (I stands for instantiation) (Bayer et al., 2000). PuLSE-I focuses on the instantiation of the product line infrastructure created in the domain engineering part of PuLSE. It describes in detail the activities, products and roles involved in application engineering. PuLSE-I activities cover planning product derivation, instantiating a product architecture from the product line reference architecture using decision models, and additional designing, implementation, and testing activities. Delivery and maintenance processes are also addressed. Several process steps are defined based on other PuLSE artefacts, e.g., reference architecture, domain decision model and scope definition.

5.1.3. FAST

FAST (*F*amily-oriented *A*bstraction, *S*pecification, and *T*ranslation) (Weiss and Lai, 1999) is an approach that considers most of the facets of product line development. It defines roles for diverse team members of a product line organization and links these roles to product line artefacts and corresponding activities. The FAST approach is very practice-focused which may be accounted for by its industry origin. The FAST process begins by identifying variabilities and commonalities among potential product line members and then creating a language for specifying the individual products within that domain. This language is then used as the basis for building a generator to support semi-automatic product derivation. Product derivation is greatly simplified through describing the products in the language and generating the products. If the product line requirements are properly identified, FAST can develop individual products very quickly.

5.2. Analysis

We summarize the results of our analysis of COVAMOF, PuLSE-I, and FAST regarding their support for the defined product derivation key activities (cf. Section 4) using the evaluation framework first introduced in Section 2.4 (Table 5).

5.3. Discussion of results

The analysis of the three approaches (COVAMOF, PuLSE-I and FAST) shows that the key activities we defined are at least partly supported by existing approaches. There is no activity we defined which has no support. Of course, how the activities are supported differs from approach to approach and depends on both the focus and the scope of the approach. For example, the COVAMOF approach is tool-supported and concentrates primarily on product configuration. FAST has a larger scope but mainly concentrates on automated derivation, i.e., describing products in an application modelling language and then using generators based on that language to create products. PuLSE-I has the largest scope of the three approaches but does not focus on tool support.

The *preparing for derivation* key activity is only partly supported by all three approaches. Our research has demonstrated that *preparing for derivation* is an important activity and *has to be at least closely related to product derivation*. We have experienced that a lack of support for preparing derivation is one of the main reasons that product derivation often fails in practice (O'Leary et al., 2009; Rabiser et al., 2007). A special focus has to be the definition of *roles and tasks* for product derivation stakeholders as well as the creation of *guidance for domain experts*.

The product derivation/configuration key activity is fully supported by all three approaches in different ways. The focus is clearly on *automating the derivation of products as far as possible* to ensure return on investment for adopting a product line approach and to make efficient and effective product derivation possible.

All three approaches perceive derivation as an iterative process. COVAMOF and FAST include explicit activities (product testing, verify integrated application) for deciding whether to deliver or perform additional iterations. The key activities we defined also *strongly focus on testing and on the iterative nature of product derivation*.

PuLSE-I is not an isolated description of product derivation but has many dependencies to other parts of the overall PuLSE product line methodology. It would also make sense to relate our key activities to domain engineering activities and ensure there is a "fast feedback loop" (Heider and Rabiser, 2010).

We have seen that, in addition to our product derivation approaches (Pro-PD and DOPLER^{UCon}), three others support most of the sub-activities of the key activities we defined. We therefore claim that the key activities should be considered when developing or evaluating a product derivation approach. However, how the activities are implemented in an approach strongly depends on the domain and context. In some cases it might be best to define a domain-specific derivation approach. Some sub-activities may simply not make sense in particular contexts (cf. Section 4.4). The activities we defined can be used as a checklist when defining, adapting, or evaluating a product derivation approach for a certain domain, context, or problem.

6. Related work

Every existing product derivation approach describing product derivation activities can be considered related work. We have analyzed three such approaches in detail in Section 5. Pro-PD and DOPLER^{UCon} have been developed based on different existing approaches. These are discussed in detail in the authors' earlier publications. Here, we focus our discussion of related work on two frameworks, i.e., the SEI Product Line Practice Framework (PLPF) (Clements and Northrop, 2001) and the Family Evaluation Framework (FEF) (van der Linden et al., 2007a). We decided to discuss these frameworks as they also define key activities (i.e., as part of practices and patterns and as part of evaluating product lines) in SPLE which can be compared with our work on defining key activities for product derivation.

6.1. PLPF

The PLPF is built around what its authors term the "three essential activities" of SPLE, namely Core Asset Development, Product Development, and Management. The PLPF is the result of an investigation performed by the SEI at "leading-edge" software development organizations and based on years of experiences with industry and academia co-operation projects. The development of our key activities was also driven by our experiences with industry and academia co-operation projects, i.e., at Siemens VAI and Bosch. The PLPF identifies foundational concepts underlying soft-

Table 5
Analysis of COVAMOF, PuLSE-I, and FAST regarding their support for the defined product derivation key activities.

Context	Questions
What is the specific goal of the approach regarding product derivation?	
COVAMOF	To support the construction of a software product by selecting and configuring product family artefacts in an iterative process
PuLSE-I	To support using a product line to create and maintain one member of the product line
FAST	To support rapidly generating products from the product line using generation tools
What aspects of product derivation does the approach cover?	
COVAMOF	Main focus is on product configuration; only partly covers preparing for derivation and additional development and testing
PuLSE-I	Covers preparing for derivation, product configuration, as well as additional development and testing
FAST	Covers requirements elicitation and analysis, product configuration, and additional development and testing
What is/are the application domain(s) the approach is focused on?	
COVAMOF	Generic enough to be usable in arbitrary domains. However, adaptations to the tool support (COVAMOF-VS) will be required depending on the usage context
PuLSE-I	Generic and not focused on a specific application domain
FAST	Sufficient flexibility to allow its use with different methods in different domains
What is the starting point for the approach?	
COVAMOF	Creating a "product entity" based on customer requirements
PuLSE-I	Customer or management has a product request that falls under the scope of the product line
FAST	Final product requirements are established by contract or informal discussion of customer requirements. An application engineer then tries to understand and validate customer requirements and their relation to product line models
What are the (desired) results of the approach?	
COVAMOF	A product derived from the product line that meets the customer requirements
PuLSE-I	A product instantiated from the product line comprising specification, architecture, and code – partly reused from the product line and partly developed during instantiation (in an iterative manner)
FAST	An application that satisfies customer expectations, created from generated code and also by using traditional development methods
User	
Which stakeholders are addressed by the approach and how?	
COVAMOF	*Engineers* are the target group of the approach. They are (tool) supported to enable iterative derivation of a product based on *customer* requirements
PuLSE-I	*Customers and management* are explicitly considered as providing input in form of product requests. *Project management* is also addressed with a project plan artefact. Derivation activities are performed by dedicated *application engineers*
FAST	*Customers* are involved in defining the requirements and in validating the derived product. *Application engineers* and so-called *"producers"* define models from which the application is then generated
Contents	
What activities/steps/sub-processes does the approach define to accomplish product derivation?	
COVAMOF	*Product definition*: Defining customer and product name *Product configuration*: Binding of variation points based on customer requirements *Product realization*: Tool-based translation of the configuration of the variability model to a configuration of an executable product, e.g., by setting compiler flags or creating make files *Product testing*: Determining whether the product meets the customer requirements and deciding whether an additional iteration (product configuration/realization/testing) is required
PuLSE-I	*Plan for the product line instance* (the product): Determine whether all characteristics of the required product are covered by the product line *Create project plan*: Define what is product-specific and what can be fulfilled by the product line *Instantiate and validate product line model*: Incrementally resolve decisions defined in the product line model (representing variation points) *Instantiate and validate reference architecture*: Instantiate variability to derive an "intermediate architecture" from the product line, validate, and then modify if necessary *Product construction*: Lower level design, implementation, and testing based on reusable assets
FAST	*Determine requirements*: The customer identifies or refines the requirements *Create application model*: The application engineer represents the product requirements as an "application model" *Analyse model*: The application model is analyzed to determine whether it satisfies the product requirements *Generate application*: Generation tool(s) are created and used to generate code and documentation based on the application model *Develop product*: Engineers develop any custom parts that cannot be generated manually and integrate them with the application *Verify integrated application*: The customer either accepts the application or the process returns to start
What artefacts are created and managed by the approach?	
COVAMOF	*Product entity*: Created in product definition with selected variants
PuLSE-I	*Detailed project plan*: Output of "plan for product line instance" activity *Requirements specification*: Output of "instantiate and validate product line model" *Product architecture*: Output of "instantiate and validate reference architecture" *Product ready for delivery*: Output of "product construction"; comprising specification, architecture, and code *Product configuration*: Output of all aforementioned activities; comprising domain decision model instance, architecture decision model instance, and low level configuration
FAST	*Application model*: Created by application engineers based on product requirements *Product*: Deliverable code for the application which is typically generated from the application model using generation tools *Customer documentation*: Might be generated from the application model
How is Preparing for Derivation supported by the approach?	
COVAMOF	*Partly supported*: customer requirements are assumed to be available and phrased so that engineers can perform product configuration and testing based on these requirements (no explicit specification and translation of customer requirements). Mapping of customer requirements to the base configuration is not part of preparing for derivation but rather assumed to be done manually by engineers during product configuration. COVAMOF provides partial support for creating the product-specific requirements: a list of characteristics that the final system will have is created or reused if the requested product is fully within the scope of the product line. COVAMOF assumes engineers to do the work supported by COVAMOF-VS. It does not consider defining role and task structures. Creating derivation guidance is not considered part of product derivation but may be done in variability modelling by creating variability views

Table 5 (*Continued*)

Context	Questions
PuLSE-I	*Partly supported*: During the "plan for product line instance" activity a detailed project plan is created as preparation for derivation. Customer requirements (product request) are assumed to be available and phrased so that they can be used to determine whether the requested product is inside the scope of the product line. Overlaps are evaluated and required system-specific developments are defined The output in PuLSE-I is "a set of characteristics upon which the customer (or the marketing) and the developers have agreed". Defining a base configuration is also supported: during "plan for product line instance", a "list of characteristics that the final system will have" is created or reused if the requested product is fully within the scope of the product line. PuLSE-I as such defines the involved stakeholders and their roles and tasks, however, on a rather high-level. Creating derivation guidance is assumed to be provided by the product line decision model and no explicit creation of additional guidance is part of the approach
FAST	*Partly supported*: During activity "determine requirements" the customer identifies the product requirements. The product requirements are the basis for the created application model. The application model is then analyzed to determine whether it satisfies the product requirements. This supports the activities specify (and translate) customer requirements, define base configuration, and map customer requirements. FAST provides no explicit support for activities define role and task structures and create derivation guidance
How is product derivation/configuration supported by the approach?	
COVAMOF	*Fully supported*: In the task "Derive new configuration", a new product entity is created in the COVAMOF variability model. Engineers select variants by specifying values at variation points. COVAMOF-VS supports this with it's configure mode where additional configuration information about the product at hand is shown in variability views. An inference and a validation engine automate this process. A partial product configuration is iteratively created, by selecting more and more variants for the product entity. Each selected variant can have "effectuation actions" that can be executed to realize the product (product realization activity of COVAMOF), e.g., by creating make files or settings files
PuLSE-I	*Fully supported*: PulSE-I supports selecting a subset of existing components as part of the PuLSE-I activity instantiate and validate product line model where decisions are resolved through the customer. Creating a partial product configuration is part of PuLSE-I activities instantiate and validate reference architecture (instantiate variabilities to create an "intermediate architecture" from the product line) and product construction (low-level configuration based on reusable product line assets)
FAST	*Fully supported*: In the "generate application" activity, generation tools are used to generate application code and documentation based on the application model. This is defined support for the select assets and create partial product configuration activities
How is additional development/testing supported by the approach?	
COVAMOF	*Partly supported*: System testing is fully supported through the COVAMOF product testing activity. This determines whether the product meets both the functional and the non-functional requirements. COVAMOF however defines no explicit support for component development, component testing, component integration with partial product configurations, or integration testing but assumes this to happen, just like DOPLER^{UCon} does
PuLSE-I	*Fully supported*: Part of the PuLSE-I activity product construction is the implementation of non-existing product line assets and of product-specifics. This includes testing (unit testing, integration testing, and acceptance testing). All this is conducted in several iterations under consideration of existing reusable product line assets. This supports component development and component testing, component integration and integration testing, as well as system testing
FAST	*Fully supported*: FAST provides full support for additional development and testing. In the "develop product" activity, any custom parts of the application that cannot be generated are developed and integrated with the generated product. In the "verify integrated application" activity, the customer either accepts the application or the process returns to start
What activities/sub-activities does the approach include that are not covered by the defined key activities/sub-activities?	
COVAMOF	Our key activities include all activities defined by COVAMOF
PuLSE-I	Apart from activities that are considered as application engineering and not product derivation (i.e., system delivery and maintenance), PuLSE-I also includes several feedback loops to other PuLSE phases (e.g., PuLSE-Eco with its scoping activities) belonging to domain engineering. Such feedback loops are currently not considered by our key activities
FAST	Apart from activities that are considered as application engineering activities and not product derivation activities (i.e., product delivery and support), our key activities include all activities defined by FAST
Validation	
Has the approach been validated in practical industrial case studies?	
COVAMOF	COVAMOF has been validated in three industrial product lines (Sinnema et al., 2004); two of them are reported in more detail in (Deelstra et al., 2005). (Sinnema and Deelstra, 2008) report on an industrial validation of the COVAMOF framework. They focus on showing how the use of COVAMOF (supported by COVAMOF-VS) reduced the number of iterations required to derive products from a product line of their industry partner. They also compare results of the use of their framework and tool by "non-experts" vs. the use by "experts"
PuLSE-I	The PuLSE approach has been applied in case studies, for example (Schmid et al., 2005). (Atkinson et al., 2000) claim the approach to have been used in various contexts
FAST	Several application examples are presented in (Weiss and Lai, 1999). The authors claim that FAST has been applied for several real-world systems

ware product lines and activities to be considered when creating a product line. We defined activities to be considered when deriving a product from a product line.

The PLPF is interesting and robust, involving important technical and non-technical aspects grouped in SPL practical areas. In the PLPF, a practice area is defined as a body of work or a collection of activities that an organization must master to successfully carry out the essential work of a product line (Clements and Northrop, 2001). Practice patterns define practice areas to be used for particular processes in SPLE. Our research does not provide a grouping of activities in practice areas or patterns. However, for each key activity, we defined several sub-activities and we also discussed how the key activities can be tailored for different domains or contexts. The PLPF does not provide this level of detail for product derivation as it focuses on SPLE as a whole.

We analyzed the PLPF practice patterns to identify which practices are relevant to product derivation. There are 12 practice patterns defined by the SEI but only two patterns are concerned with application engineering, i.e., the "Product Builder" and the "Essentials Coverage" pattern.

The Product Builder pattern consists of practice areas that should be used whenever any product in the product line is being developed. The practice areas in this pattern are: Requirements Engineering, Architecture Definition, Architecture Evaluation, Component Development, Testing, and Software System Integration. The Essentials Coverage pattern assigns each practice area to the three essential product line activities core asset development, product development, and management. Our key activities cover the PLPF practice areas for product derivation except for the architecture evaluation area which we do not explicitly address. Requirements engineering is part of the preparing for product derivation key activity. Architecture definition is part of both the preparing for product derivation and the product derivation/configuration key activities. Component development, testing

and software system integration are part of the additional development/testing key activity.

6.2. Family evaluation framework

The family evaluation framework (van der Linden et al., 2007a) has been developed as a consolidated result of the European industry and academia co-operation projects ESAPS, CAFÉ and FAMILIES. This can again be compared to the development of our key activities which was also mainly based on the experiences made in co-operation projects with industry, i.e., Siemens VAI and Bosch. However, the focus of the FEF was not on defining key activities but on creating the foundation of a systematic and comprehensive strategy for software product line process evaluation. Its purpose is to support evaluating the performance of SPLE in organizations.

The framework highlights four dimensions: business, architecture, process, and organization (BAPO) as a set of four variables to describe the maturity of the software product line process. *Business* deals with the business relationship between domain engineering and application engineering. *Architecture* deals with the use of architecture in domain and application engineering and how the architecture enables variability. *Organization* measures the effectiveness of domain and application engineering activities in the organization. *Process* measures the SPL processes. These are subdivided into domain, application, collaboration and coordinate processes. Each of these can be evaluated using a maturity model such as CMMI.

The result of an FEF evaluation is an evaluation profile consisting of four values, one for each BAPO dimension. The profile defines various maturity scales for individual dimensions. The FEF can be regarded as an attempt to develop a comprehensive strategy for SPL process maturity assessment. The framework however does not assess SPLE for the occurrence of individual activities but rather deals with their results and with process areas. For instance, in the process dimension, application engineering is defined as an aspect "playing a role" and defined as comprising "processes that guide the application engineering work". For the process dimension and, more specifically, the application engineering aspect, the FEF describes the necessary amplifications for the respective level, e.g., on Level 2 (Managed) the use of common assets by application engineering activities needs to be measured. These amplification descriptions can be viewed as a definition of what is key in application engineering, just like we define what activities are key in product derivation. Our research focuses on a different level of interest than the FEF and does not consider process evaluation aspects.

7. Conclusions

The definition of a general product derivation approach applicable to every domain will not be possible. However, comparing two product derivation approaches developed by the authors in two different, independent research projects (where both sought to identify product derivation activities) allowed us to define key activities for product derivation. The comparison of the two approaches was very beneficial, exposing the researchers to alternative viewpoints. Researchers gained a better sense of the strengths and weaknesses of their particular approach through discussion and debate. We validated the identified activities by systematically analyzing existing product derivation approaches regarding their support for the activities. This provides evidence that the identified activities are relevant and important.

We observed how preparing for derivation is only partly supported by existing approaches. Our research confirms that this is one of the main reasons that product derivation often fails in practice. It is important to put a special focus on the definition of roles and tasks for product derivation stakeholders as well as the creation of guidance for domain experts. All three approaches we analyzed perceive product derivation as an iterative process. In our key activities, we also strongly focus on the iterative nature of product derivation. Overall, we observed how the key activities we defined are at least partly supported by existing approaches. We therefore claim that these activities are important and should be considered by both researchers and industry practitioners when developing or evaluating a product derivation approach.

Despite the growing adoption of software product line approaches, according to our research product derivation remains an expensive and error-prone activity which is still hard to automate and support by tools. The goal should be to avoid product-specific development in application engineering as far as possible. However, in practice this is often not possible. We see the contribution of the identification of key activities to the automation of product derivation as twofold: (i) it allows us to put automated approaches, which tackle one particular task, into a bigger context and (ii) it lays the foundation for tools which support the overall process.

We do not claim that the key activities we present are complete. In some situations domain-specific activities are required. Some sub-activities may simply not make sense in some contexts as we have described based on our industry experiences. Further work is required with regard to defining when and how to tailor the key activities to specific contexts, domains or organization. Also, validation is necessary with regard to the usefulness of the key activities in practice.

Nevertheless, we hope that other researchers can use our work as a starting point for presenting their experiences with product derivation or as a basis for defining, adapting or evaluating their product derivation approaches.

Acknowledgements

This work is supported by IRCSET under grant number RS/06/167 and by Science Foundation Ireland through Lero–the Irish Software Engineering Research Centre under grant number 03/CE2/I303_1. This work has also been supported by the Christian Doppler Forschungsgesellschaft, Austria and Siemens VAI Metals Technologies.

References

Automotive Special Interest Group, 2008. Automotive SPICE™ Process Reference Model (PRM), available: http://www.automotivespice.com.

Ahmed, F., Capretz, L.F., Campbell, P., 2009. Software Product Lines: A Process Assessment Methodology. A Practitioner's Approach, 1st ed. VDM-Verlag, Berlin, Germany.

Atkinson, C., Bayer, J., Muthig, D., 2000. Component-based product line development: the KobrA approach. In: Proceedings of the First Conference on Software Product Lines: Experience and Research Directions, Kluwer Academic Publishers, Denver, Colorado, United States.

Bayer, J., Gacek, C., Muthig, D., Widen, T., 2000. PuLSE-I: deriving instances from a product line infrastructure. In: 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems, Edinburgh, UK, pp. 237–245.

Chastek, G., Donohoe, P., McGregor, J.D., 2002. *Product Line Production Planning for the Home Integration System Example*. In Technical Report CMU/SEI-2002-TN-029. Carnegie Mellon Software Engineering Institute, Pittsburgh.

Clements, P., Northrop, L., 2001. Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Czarnecki, K., Kim, C.H.P., 2005. Cardinality-based feature modeling and constraints: a progress report. In: Proceedings of the International Workshop on Software Factories at OOPSLA'05, ACM Press, San Diego, USA, pp. 1–9.

Deelstra, S., Sinnema, M., Bosch, J., 2005. Product derivation in software product families: a case study. J. Syst. Software 74 (2), 173–194.

Dhungana, D., Grünbacher, P., Rabiser, R., Neumayer, T., 2010. Structuring the modeling space and supporting evolution in software product line engineering. J. Syst. Software 83 (7), 1108–1122.

Gomaa, H., 2004. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison Wesley Longman Publishing Co., Inc.

Griss, M.L., 2000. Implementing Product-Line Features with Component Reuse. Springer-Verlag, London, UK.
Grünbacher, P., Rabiser, R., Dhungana, D., Lehofer, M., 2009. Model-based customization and deployment of eclipse-based tools: industrial experiences. In: 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009), IEEE/ACM, Auckland, New Zealand, pp. 247–256.
Halmans, G., Pohl, K., 2003. Communicating the variability of a software-product family to customers. Informatik - Forschung und Entwicklung 18 (3–4), 113–131.
Hammersley, M., Gomm, R., Foster, P., 2000. Case Study Method: Key Issues. Key Texts. Sage Publications, London.
Heider, W., Rabiser, R., 2010. Supporting evolution of product lines through rapid feedback from application engineering. In: Proc. 4th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2010). ICB-Research Report No. 37, University of Duisburg Essen, Linz, Austria, pp. 167–170.
Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-Oriented Domain Analysis (FODA) feasibility study. CMU/SEI-90TR-21. USA Carnegie Mellon Software Engineering Institute, Pittsburgh, PA.
Kauppinen, R., 2003. Testing Framework-Based Software Product Lines. Technical Report C-2003-20. University of Helsinki, Department of Computer Science.
Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., Rosenberg, J., 2002. Preliminary guidelines for empirical research in software engineering. IEEE Transactions on Software Engineering 28 (8), 721–734.
Matinlassi, M., 2004. Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA, software engineering, ICSE 2004. In: Proceedings. 26th International Conference on, EICC, Scotland, UK, pp. 127–136.
McGregor, J.D., 2005. *Preparing for Automated Derivation of Products in a Software Product Line*, CMU/SEI-2005-TR-017. Carnegie Mellon Software Engineering Institute.
O'Leary, P., 2010. Towards a Product Derivation Process Reference Model for Software Product Line Organisations. Department of Computer Science and Information Systems. University of Limerick, Limerick, p. 277.
O'Leary, P., Ali Babar, M., Thiel, S., Richardson, I., 2007. Product derivation process and agile approaches: exploring the integration potential. In: Proceedings of 2nd IFIP Central and East European Conference on Software Engineering Techniques, Wydawnictwo NAKOM, Poznań, Poland, pp. 166–171.
O'Leary, P., McCaffery, F., Thiel, S., Richardson, I., in press. An agile process model for product derivation in software product line engineering. J. Software Maint. Evol.: Research and Practice, doi:10.1002/smr.498.
O'Leary, P., Rabiser, R., Richardson, I., Thiel, S., 2009. Important issues and key activities in product derivation: experiences from two independent research projects. In: Software Engineering Institute, C. (Ed.), Software Product Line Conference. Proc. of the 13th International Software Product Line Conference (SPLC 2009). San Francisco, CA, USA, pp. 121–130.
O'Leary, P., Richardson, I., Thiel, S., 2008a. Developing a Product Derivation Process Framework for Software Product Line Organisations, EuroSPI 2008 Doctoral Symposium. Dublin, Ireland.
O'Leary, P., Thiel, S., Botterweck, G., Richardson, I., 2008b. Towards a product derivation process framework. In: 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques CEE-SET 2008, Brno (Czech Republic), pp. 189–202.
Perrouin, G., Klein, J., Guelfi, N., Jezequel, J.M., 2008. Reconciling automation and flexibility in product derivation. In: 12th International Software Product Line Conference (SPLC), pp. 339–348.
Pohl, K., Böckle, G.v.d., Linden, F., 2005. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Heidelberg.
Rabiser, R., 2009. A User-Centered Approach to Product Configuration in Software Product Line Engineering, Christian Doppler Laboratory for Automated Software Engineering. Institute for Systems Engineering and Automation, Johannes Kepler University, Linz.
Rabiser, R., Dhungana, D., 2007. Integrated support for product configuration and requirements engineering in product derivation. In: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 219–228.
Rabiser, R., Grünbacher, P., Dhungana, D., 2007. Supporting product derivation by adapting and augmenting variability models. In: 11th International Software Product Line Conference, Kyoto, Japan.
Rabiser, R., Grünbacher, P., Dhungana, D., 2010. Requirements for product derivation support: results from a systematic literature review and an expert survey. Information and Software Technology 52 (3), 324–346.
Rabiser, R., Wolfinger, R., Grünbacher, P., 2009. Three-level customization of software products using a product line approach. In: Proc. of the 42nd Hawai'i International Conference on System Sciences. IEEE CS, Waikoloa, Big Island, HI, USA.
Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Eng. 14 (2), 131–161.
Schmid, K., John, I., 2004. A customizable approach to full lifecycle variability management. Sci. Comput. Program. 53 (3), 259–284.
Schmid, K., John, I., Kolb, R., Meier, G., 2005. Introducing the PuLSE approach to an embedded system population at Testo AG. In: Proceedings of the 27th International Conference on Software Engineering, ACM, St. Louis, MO, USA, pp. 544–552.
Sinnema, M., Deelstra, S., 2008. Industrial validation of COVAMOF. J. Syst. Software 81 (4), 584–600.
Sinnema, M., Deelstra, S., Hoekstra, P., 2006a. The COVAMOF derivation process. In: Proceedings of the 9th International Conference on Software Reuse (ICSR 2006), Springer Berlin Heidelberg, Turin, Italy, pp. 101–114.
Sinnema, M., Deelstra, S., Nijhuis, J., Bosch, J., 2004. COVAMOF: a framework for modeling variability in software product families. In: Proc. 3rd Int'l Conf. Software Product Lines (SPLC 04), San Diego, CA, pp. 197–213.
Sinnema, M., Deelstra, S., Nijhuis, J., Bosch, J., 2006b. Modeling dependencies in product families with COVAMOF. In: 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006), Potsdam, Germany.
van der Linden, F., Schmid, K., Rommes, E., 2007a. The Family Evaluation Framework, Software Product Lines in Action. Springer-Verlag, New York, USA, pp. 79–108.
van der Linden, F., Schmid, K., Rommes, E., 2007b. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer-Verlag, New York, Inc.
Vierhauser, M., Dhungana, D., Heider, W., Rabiser, R., Egyed, A., 2010. Tool support for incremental consistency checking on variability models. In: Proc. 4th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2010). ICB-Research Report No. 37, Linz, Austria, pp. 171–174.
Weiss, D.M., Lai, C.T.R., 1999. Software Product Line Engineering: A Family-based Software Development Process, 1st ed. Addison-Wesley Professional.
Yin, R., 2003. Case Study Research: Design and Methods. SAGE Publications, Beverly Hills, CA.

Rick Rabiser is a postdoctoral researcher at the Christian Doppler Laboratory for Automated Software Engineering at Johannes Kepler University, Linz, Austria. He received his PhD in Business Informatics in 2009 from Johannes Kepler University. Contact him at rabiser@ase.jku.at.

Pádraig O'Leary is a Post-doctoral researcher with the Reuse in Software Engineering (RiSE) group based in the Federal University of Bahia (UFBA). He received his PhD in 2010 from the University of Limerick. He was a member of Lero - the Irish Software Engineering Research Centre from 2006 until 2010. Previously, he also worked as a software developer in the insurance and financial services industry. Contact him at padraig.oleary@rise.com.br.

Dr Ita Richardson is a Senior Lecturer in the Department of Computer Science and Information Systems at the University of Limerick, Ireland, and a Research Area Leader for Process, Practice and Methods within Lero–the Irish Software Engineering Research Centre. Dr Richardson's main research interests are in software quality and software process improvement, focusing particularly on Global Software Development and regulated industry such as Medical and Health Systems. She publishes internationally and supervises full-time and part-time PhD students. She has received funding from sources such as Science Foundation Ireland, Enterprise Ireland and the European Community.