

ULRR

Automating the product derivation process in software product line engineering

Item Type	Meetings and Proceedings
Authors	O'Leary, Pádraig;Botterweck, Goetz;Richardson, Ita
Download date	2026-05-16 23:26:00
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/3367

Automating the Product Derivation Process in Software Product Line Engineering

Padraig O’Leary, Goetz Botterweck, Ita Richardson

Lero, the Irish Software Engineering Research Centre, University of Limerick, Ireland

{padraig.oleary, goetz.botterweck, steffen.thiel, ita.richardson}@lero.ie

Abstract

A variety of automated approaches for software product line engineering in general and product derivation in particular have been proposed. Unfortunately due to a range of reasons, many development organisations fail to get the maximum benefit from these approaches. Worse, the way many organisations use these approaches actually hampers effective product derivation. As a foundation for the successful adoption of automated approaches in product derivation, a better understanding of the underlying activities in industrial product derivation practices is required.

By consolidating current knowledge from literature and industrial experience, we have developed a process framework that comprises important tasks, which product line stakeholders have to perform during product derivation. We outline how our framework can provide a link to automated approaches by providing product derivation context and facilitating tool support for the overall process.

1. Introduction

A Software Product Line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1]. The SPL approach makes a distinction between domain engineering, where a common platform for an arbitrary number of products is designed and implemented, and application engineering, where a product is derived based on the platform components [2]. The separation into domain engineering and application engineering allows the development of software artefacts which are shared among the products within that domain (see Figure 1). These shared artefacts become separate entities in their own right, subscribing to providing shared functionality across multiple products.

It is during application engineering that the individual products within a product line are

constructed. The products are constructed using a number of shared software artefacts created during domain engineering. The process of creating these individual products using the platform artefacts is known as product derivation.

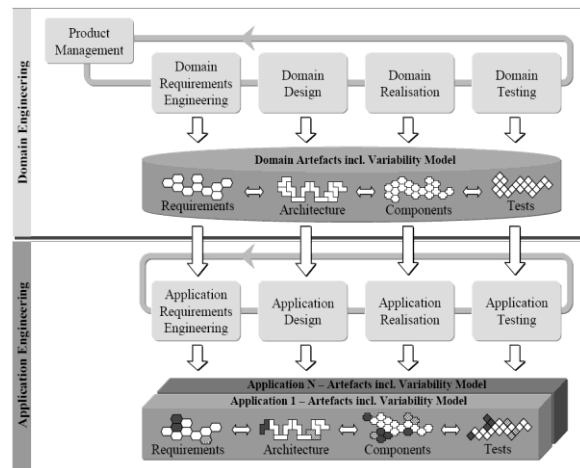


Figure 1. Domain and application engineering phases [3]

1.1 Product derivation

Product derivation is the process of constructing a product from a Software Product Lines (SPL) core assets [4]. An effective product derivation process can help to ensure that the effort required to develop the platform assets is less than the benefits delivered through using these shared artefacts across the products within a product line.

Over time a systematic and mature product derivation process can lead to automatic generation of products if the platform is suitably evolved to the needs of the product line, the organisation has institutionalised the required practice areas and the various activities of the derivation process are automated and supported by adequate tools [5].

Product derivation can be a complex and difficult activity, inefficient practices can undermine the underlying assumption in SPL that “the investments

required for building the reusable assets during domain engineering are outweighed by the benefits of rapid derivation of individual products” [6] might not hold if inefficient derivation practices diminishes the expected gains.

In the literature a number of publications speak of the difficulties associated with product derivation. Hotz et al. [7] describe the process as “slow and error prone even if no new development is involved”. Griss [8] identifies the inherent complexity and the coordination required 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”. Therefore as Deelstra [4] points out, the derivation of individual products from shared software assets is still a time-consuming and expensive activity in many organisations. Despite this, there has been little work dedicated to the overall product derivation process. Rabiser et al. [9] claim that “guidance and support are needed to increase efficiency and to deal with complexity of product derivation”. Therefore as Deelstra [4] states there “is a lack of methodological support for application engineering and, consequently, organizations fail to exploit the full benefits of software product families.”

Accordingly, there is a strong need for the identification of “best practices” in product derivation as well as on tools to support these practices.

1.2 Automating product derivation

Automatic derivation refers to the use of a set of tools to both specify a product and to transform that specification into a product using the core assets [10]. Product derivation can be a tedious and error prone activity. All tasks that can be automated may therefore significantly reduce costs and increase product derivation efficiency [4].

According to [10] the portion of the derivation process that is automated varies from one product line to another. If an entire product can be derived automatically, it is because the product line’s core assets are sufficient to produce all the specified behaviours and all the variation values can be predetermined or used as input. However, in all cases, for the portion that is automated, the product implementation is generated automatically from some form of specification.

Automating the steps of the derivation process requires significant upfront investment, including both the core assets that will be assembled as products and the core assets that will perform the assembly, as well

as the tool environment setup and training of personnel [4, 10]. A SPL organisation has the ability to amortise the cost of this upfront investment over a set of products.

However tools alone are not the silver bullet for product derivation, as Humprey [11] explains, “Tools can be enormously helpful, but they alone will not fully solve software engineering’s problems. Neither will process alone. Both are needed to obtain a balanced result.”

Current product derivation process automation efforts fail to appreciate the sometimes chaotic aspects of product derivation. Until these aspects of product derivation are handled, approaches will be limited to partial solutions. Automated derivation approaches need to re-identify themselves as collaborative technology, where there is clear evidence of alignment between process and tool. A first step towards the development of these collaborative technologies is the development of a product derivation process.

1.3 Contribution

This paper presents a Product Derivation Process Framework (PDPF) that can assist the development and use of automated product derivation approaches. The framework builds on work done by Deelstra et al. [4]. It identifies tasks, roles, artefacts and process flows within product derivation. A framework, as proposed here, provides benefits to both academia and industry.

For academia, the framework provides structure to the area under concern. Hence, it becomes easier to place a particular research topic in context. There is historical evidence of certain fields achieving progress at the expense of others, through the establishment of a core, theoretical structure [12]. As a roadmap, the framework points to areas of uncertainty and helps identify remaining challenges. Such a roadmap encourages the insertion of those pieces that may be missing, or the extra detail that may be needed for a particular purpose or group.

For industry, it is envisaged that the development of the framework will help the advancement of product derivation practices. It will assist organisations by providing a structured approach to product derivation, making the process more predictable and manageable. Moreover, this enables the integration of non-standard techniques such as agile practices, at appropriate times of the development process [13, 14].

It can be argued that without some established process framework for product derivation, the applied practices will be motivated by technological solutions (e.g., available tools), rather than good practice.

Standardisation of the development process around a methodology rather than a tool facilitates technology change. Tools can be interchanged more easily once seen to work within the bounds of the methodology. The framework can also support the development of these tool environments to make the derivation of products more efficient and effective.

The framework assists organisations in using a structured approach for product derivation activities and thus for achieving maximum return on investment using an SPL development approach. The identification of tool and automation needs can help establish an integrated tool environment to support the product derivation process.

The paper is organised as follows: In section 2 we present our Product Derivation Process Framework (PDPF). In section 3 we describe our work formalising the PDPF using the Eclipse Process Framework. In section 4, we present how the framework can be used as a foundation for automated approaches. In section 5, we discuss related work. In section 6, we present our conclusion and future work.

2. A Product Derivation Process Framework

The preparatory stage of this research was conducted as a review of existing SPL whitepapers, product derivation papers and software process improvement (SPI) practices ([4], [6], [7], [9], [15], [5]). The research aimed to identify the fundamental practices of product derivation, to study the existing product derivation practices as well as to chart the available empirical evidence on the topic – scientific as well as anecdotal. The initial results were further developed and assessed through a series of iterative workshops over a four month period. Evidence and feedback from SPL practitioners and researchers was collected from these organised workshops. The output of this process was the PDPF, as presented in the following section.

The PDPF proposed is independent of tools or configuration models. In many cases these tools and models can assist the derivation process and we will explore their potential implementation within the framework. The framework itself could be executed in a manual, partially automated or fully automated manner depending on the organisation and the availability and integrability of adequate tools.

2.1 Overview of the PDPF

The PDPF is structured into four main steps. Figure 2 provides an overview of the PDPF, showing the interactions between the main steps:

1. **Impact Analysis** is aimed at gathering product-specific requirements based on customer requirements and negotiation with the platform team.
2. **Reusability Analysis** purports to create a partial product configuration based on the product specific requirements and by using the available core assets.
3. During **Component Development and Adaptation**, new components are developed (if required) and existing components are adapted to satisfy requirements which could not be satisfied by configuring existing core assets.
4. Finally, **Product Integration and Validation** aims to integrate the core asset configuration and newly developed components. The integrated product is then validated by performing appropriate testing procedures.

We will now discuss these product derivation steps in more detail.

2.2 Impact Analysis

In this step the customer requirements are mapped to platform features. The product team determines the list of the customer requirements that can be satisfied through a configuration of the platform assets. Ideally the complete set of customer requirements are met by the platform however in reality some customer requirements will fall outside the scope of the platform. Customer requirements which cannot be satisfied by existing assets must be negotiated with the customer.

Customer negotiation is a critical aspect of product derivation. Time to market requirements for a particular customer project can cause the product team to make their own product-specific modifications to core assets – modifications that might lead to uncontrolled variability in the product line [16]. There is a trade-off for the product team between meeting all of the customer's needs while retaining the profitability of the platform assets for the whole product line. The product team can discourage customer requirements that fall outside the platform scope through a fixed pricing structure based on the amount of customer specific development required. Involving the platform architects in customer negotiation can solve many of the problems that arise from these conflicting requirements [17]. The satisfied customer requirements

and the negotiated requirements are merged to form the product specific requirements. These product specific requirements are used to create the product specific test cases. The product team uses the platform test cases artefacts as a basis for the creation of the product specific test cases.

Although full automation is hard to achieve, Impact Analysis can be supported by tools which take care of routine tasks. Some of these include tools to manage requirements and map between requirements, features and components including visualisation of the various artefacts and the dependencies among them [18]. Also tools that can automatically generate and manage test cases can support this step.

2.3 Reusability Analysis

The main goal of Reusability Analysis is to create a partial product configuration that makes maximum use of the platform artefacts and minimises the amount of product specific development required.

The product team use one of two approaches to create a base product configuration. In the first approach, the product team selects a base configuration from the set of existing platform configurations. According to Deelstra et al. [4], in contrast to a reference and old configuration, where the focus is on reselecting components, the focus of product derivation with a base configuration is on adding components to the set of components in the base configuration. Selection of an existing configuration from the platform is especially viable in cases where a large system is developed for repeat customers, i.e. customers who have purchased similar types of systems before. Typically, repeat customers desire new functionality on top of the functionality they ordered for a previous product. In this respect, configuration selection is basically reuse of choices. Configuration selection can also help to speed up the development process by choosing a previously tested solution especially in cases when two or more configurations can be used. In the second approach, where no appropriate existing configuration can be selected, the product team must derive a new base configuration from a subset of the overall Platform Architecture.

Once the base configuration has been created, components are selected from the collection of platform components for, addition to or replacement of, existing components. The initial product configuration can be either a selection of existing configurations or a newly derived configuration. The selection of components allows the product team to fulfil requirements which could not be met through

configuration selection. Whether a component fits in the configuration depends on whether the component correctly interacts with the other components in the configuration and no dependencies or constraints are violated. A partial configuration is now created. A partial configuration partially implements a software product in the sense that not all variants are yet selected.

At this stage, the product team can identify which product specific requirements are satisfied by this

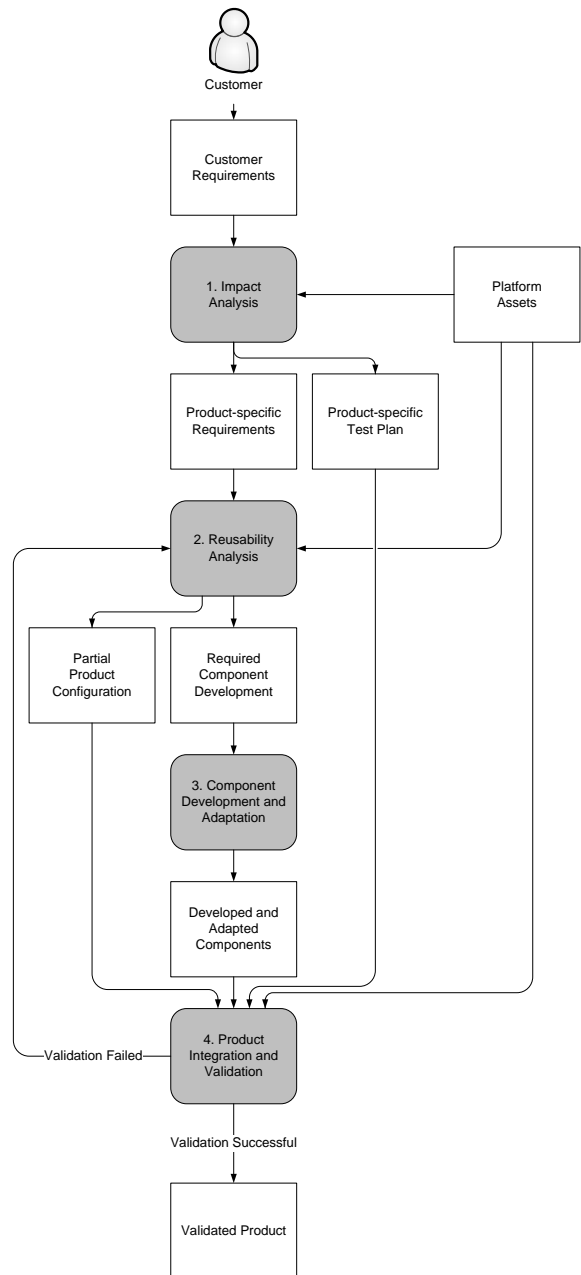


Figure 2. Overview of the PDPF

partial configuration. The requirements which could not be accommodated by the partial product configuration are handled by the product team. The product team is responsible for the development of new components and adaptation of existing components.

Reusability Analysis can be supported by some of the tools mentioned in the previous section (requirements management and mapping between requirements, features and components). Tools that can automatically generate and manage configurations (which may include an inference engine) can also be used. Tools to identify where changes are required within software components and the impacts of those changes are also useful.

2.4 Component Development and Adaptation

In Component Development and Adaptation the product team facilitates requirements which could not be satisfied by the partial product configuration by adapting existing and developing new components. The decision of whether the required component development or adaptation will result in product-specific code or adaptation of the product line (platform) is determined through a Change Control Board (CCB). The CCB is usually comprised of members of the product team and the platform team. Scoping new development is a difficult task. Practical arguments such as time to market and short term costs frequently cause scoping solutions to be selected that are neither optimal for the product itself nor for the product line as a whole [4]. While platform development must provide a consistently high-quality platform, product development must meet delivery dates and customer requirements. So, with any required development the CCB must decide whether to integrate a given requirement into the platform or into an individual product only [17].

If the CCB decides that the component development should occur at the platform level then the platform team has to adapt or develop new shared artefacts and release a new version of the platform. Based on the new platform, the product team must repeat Reusability Analysis for the products under consideration. If the development or adaptation is designated to be product-specific then it is the responsibility of the product development team to implement the required component changes at the product level.

When a component is built or adapted, initial or tailored versions of a component will need to be tested rigorously through unit testing. According to Kauppinen [19], conventional unit test methods must

be utilized as no product line specific methods have been developed so far.

Component Development and Adaptation can be supported by tools that identify where changes are required within software components and the impacts of those changes. Tools to support new development and adaptation of existing components and support determination of effort and cost for such tasks can also support this step. Tools to generate unit test cases and automate the testing process are also useful.

2.5 Product Integration and Validation

In Product Integration and Validation, an integrated product is created from the partial product configuration and the developed components. The product is validated by integration and system testing.

During product integration, the newly developed or adapted components are integrated with the partial product configuration. The product team integrates the developed or adapted components and the partial product configuration by writing sufficient “glue” code to interface with the components [16]. This includes implementing any required architectural changes to facilitate the developed or adapted components.

The integrated product configuration must undergo product validation. Before product validation can begin the product team must confirm that no changes in the customer requirements have occurred. If the customer requirements have changed, the product team must return to a previous step. Major customer requirement changes can be handled by returning to Step 1 and performing Impact Analysis again. Minor changes or changes required under tight time constraints are handled through returning to Step 3 for development at the product level.

If the customer requirements are consistent then the product team begins product validation. During product validation the product is checked for the consistency and correctness of the component configuration in Integration Testing and for compliance with the product specific requirements [4] in System Testing. The product specific test cases are used to assist the product team in the verification of requirements [20].

Due to the variability defined in the platform assets, completely testing the platform assets for all possible configurations is impossible except for trivial cases. Hence, Integration Testing only validates the Platform assets for this particular configuration. The integration tests should reuse Platform Test Artefacts. This also ensures that no new errors appear due to the integration of core assets with product specific assets [21].

After Integration Testing, System Testing is performed. System Testing verifies if the product as a whole conforms to the product specific requirements. System test artefacts such as the product specific test cases are already derived from the product specific requirements [21] in Step 1, Impact Analysis.

If the product fails Integration Testing or System Testing then the current configuration may not provide the required functionality, or some of the selected components simply do not work together as expected. In this case, the product team should repeat, starting either from Reusability Analysis or Components Development and Adaptation depending on the scope of the required changes. There are two main reasons why a product may fail Integration or System Testing. Firstly the requirements set may change or expand during product derivation, for example, if the organization uses a subset of the customer requirements to derive the initial configuration, or if the customer has new wishes for the product. Secondly, if the configuration may not completely provide the required functionality, or some of the selected components simply do not work together at all [4].

If the product is validated through system and integration testing the process is complete and the customer product has been derived.

Product Integration and Validation can be supported by tools to build products from a configuration and carry out both integration and system testing of the product. Tools for architecture conformance, verification and validation when combining new or adapted components into a configuration can also be used. Tools to determine the causes of test failure would also support this step.

3. Formalising the Framework

We are using the Eclipse Process Framework (EPF) to model the product derivation process and create a formalised version of our PDPF. Using as input information collected on product derivation practices from an extensive literature review, discussions with SPL practioners and researchers, and a study of product derivation practices in a large automotive supplier, we have built up an EPF version of our PDPF.

By enabling inbuilt process variability within EPF we can select, tailor and or remove content from a process in order to strike the right balance for a particular situation. Therefore process models can be adapted and customized for a particular industry and organization, allowing for situational method content.



Figure 3. Impact Analysis modelled using EPF

This has the potential for making the PDPF as applicable to a small software development team working on a mobile application as it is for large aerospace and defence contractor building a system of systems. For instance, in the case study company we observed that embedded software development is a cross discipline activity. In this context, “discipline mapping” where requirements are allocated to software, hardware or mechanical disciplines, would be a relevant task. In Figure 3, you can observe the discipline mapping task within the Impact Analysis activity modelled using EPF. However this particular activity may not be necessary when developing a product line in another domain, therefore is domain specific. This process flexibility allows us to model generic product derivation practices and domain specific practices in one uniform process framework. The ability to perform such adaptations is tool-supported by EPF Composer [22].

Figure 4 shows the role the PDPF can play in bridging the gap between current industrial practices and the adoption of automated approaches. The PDPF can act as a type of roadmap towards automation. As we move from left to right, the need for further formalisation and abstraction of product derivation activities is required; the use of EPF is a means of achieving this required formalisation and abstraction.

We have illustrated how industrial product derivation practices have helped augment and refine the PDPF. The main activities of product derivation, as described in the PDPF were observed in the industrial case study however the everyday realities of industrial product derivation meant the activity boundaries are less well defined and contain company specific practices and tasks.

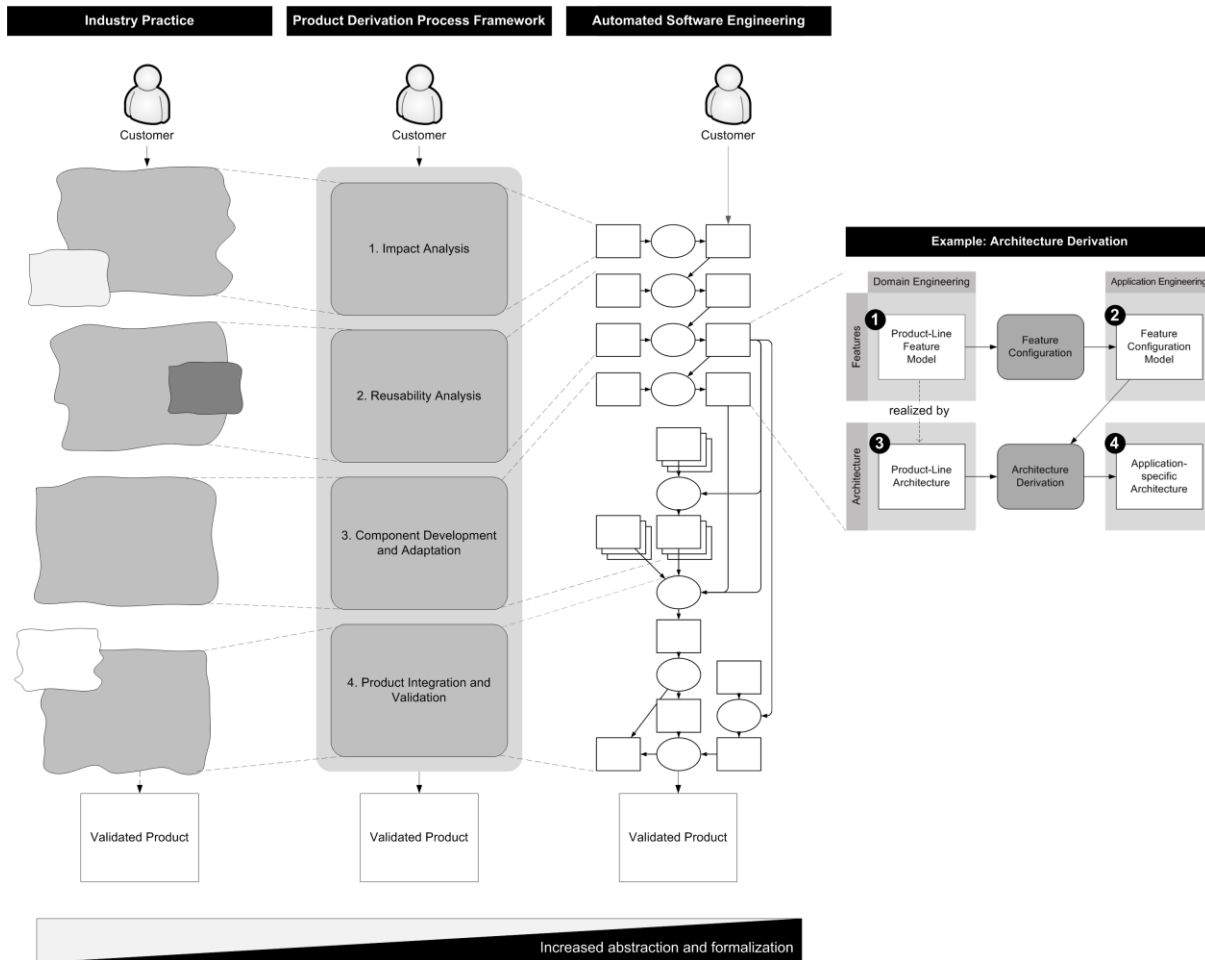


Figure 4. Industry practice, PDPF, and ASE

4. The Framework as a Foundation for Automated Approaches

In this section we discuss the contributions the PDPF can provide in the context of automated software engineering.

We have derived our PDPF from literature and industry practice (Figure 4 left). We abstracted and modelled the discovered process structures into our process framework. With this as a foundation, we see the links between our framework and automated product derivation approaches (Figure 4) in two aspects: The framework provides context for automated approaches and it facilitates tool support for the overall process.

4.1 Context for automation

When focussing on a particular task within product derivation, it is possible to use automated approaches. The PDPF can serve as a foundation for these approaches, by providing the bigger context and describing overall process structures for product derivation. Within this context, approaches that focus on the automation of particular derivation steps can be set against the bigger product derivation picture.

One example for such approaches (Figure 4 right) is the derivation of application-specific architectures **4** from the product-line architecture, PLA **3**. This derivation is based on the domain feature model **1** and the application-specific feature configuration **2**.

The PLA contains variability to cover the full range of products which can be created from the product line. In the Application Architecture, after the derivation process, this variability is gone since the feature configuration is fixed and all decisions whether components should be included or not, have been implemented.

In earlier work in this topic [23] we described how this process can be fully automated, when we assume that the PLA contains enough elements to cover all possible Application Architectures and the process of derivation is thereby simplified to the task of filtering the right elements from the PLA. The filtering is directly based on the feature configuration ② and the links between domain-feature model ① and PLA ④ which describes “realized-by” relationships between features and architectural components. In our approach the process of architecture derivation is implemented as an model transformation in the Atlas Transformation Language (ATL) [24].

4.2 Tool-support for the overall process

When we extend the scope of our PDPF to the overall process of product derivation we have to abandon the goal of full automation, but can strive to use our EPF-based framework to provide as much tool support as possible, which will free the application engineer from tedious routine work.

Such tools could increase the visibility of the status of tasks, milestones and key deliverables. For instance, a tool could detect the current tasks and artefacts the developer is working on, by interaction (hotkeys) or by analysing access operations to a version control server. Integrated with the status overview (of tasks and artefacts) tools could provide functionality to start task specific activities or jump directly to an artefact. An instance of this would be where a user selects the activity Impact Analysis and is presented with an editor containing the high level design form template for the product specific requirements, a hypertext on-line browsing tool containing design documentation and a design tool to assist with development of the data flow diagrams, structure charts, etc. When the task is completed, the process enactment tool then sends the completed artefact to the next phase, Reusability Analysis.

EPF can allocate dependencies between tasks so that the order in which the tasks are executed can be controlled. Dependencies can be used so that, tasks or task specific tools requiring inputs from a previous step will occur sequentially, for instance, the task ‘create product specific components’ occurs after the new platform development is complete. The creation of task dependencies gives tool developers a formal blueprint of the requirements for any product derivation tool.

The framework can also facilitate current automated tool approaches by facilitating tool replacements. Standardisation of the development process around a

methodology rather than a tool facilitates technology change. New tools can be interchanged in and out once seen to work within bounds of the framework.

Once the PDPF has been defined and automating tools have been adopted, efforts aimed at improving the quality and the productivity of these tasks could begin. Process tracking tools can record the time spent on particular product derivation tasks and reports can record where the process was followed. These and other metrics can allow the organisation to determine where bottlenecks occur within product derivation and encourage a culture of continuous process improvement through the use of collected metrics. This can help identify areas where further staff training is required or where specific tools are not hampering rather than facilitating product derivation activities.

5. Related Work

To date, several approaches and tools that support or partly automate product derivation activities in SPL have been proposed. Krebs *et al.* [25] outline a derivation methodology that uses a configuration model to represent functionality and variability in a product line. The configuration model supports two types of artefacts: components and features. The components are derived from the physical architecture of the product line. Features represent a customer’s view of the functionality in the components. A mapping between features and components allows for automated inferring of the components required for a given selection of features.

Asikainen *et al.* [15] provide a product configuration modelling language (PCML) and configuration tool (WeCoTin). PCML supports the creation of feature models for a software product line. WeCoTin is used to derive valid feature models for particular products of the product line.

The ConIPF Methodology [7] proposed by Hotz *et al.* tackles the challenges of product derivation by combining concepts from product line engineering and knowledge-based configuration.

Rabiser *et al.* [9] present an approach for supporting product derivation using feature specifications. The approach introduces business decision-making into product derivation through a combination of modelling stakeholder needs, product features, architectural elements, and variability. The approach emphasises supporting the requirements acquisition and management mechanism through the use of variability models.

McGregor [5] introduces the *production plan*, which prescribes how products are produced from

platform assets. It contains the attached processes of the platform assets as well as an overall scheme of how the processes are combined to build products. The product plan facilitates the passing of knowledge between the platform developers and the product developers. An example of the production plan in use is given in [16]. McGregor [10] also provides an overview of technologies and approaches to automate product derivation.

Deelstra *et al.* [4] present a product derivation approach developed based on two industrial case studies. The framework consists of two phases: an initial and an iteration phase. During the initial phase, a first product configuration is derived from the product line artefacts. The initial configuration is modified in a number of subsequent iterations during the iteration phase until the product sufficiently implements the imposed requirements. Requirements that cannot be accommodated by existing assets are handled by product-specific adaptation or reactive evolution. Parts of the derivation framework have been implemented in a research tool called COVAMOF [26], a variability modelling framework which purports to solve the product derivation problems associated with dependencies.

The work by Deelstra *et al.* presents 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.* [6] agree is required for product derivation.

Through a series of research stages using sources in industry and academia, this research [27-36] developed a process reference model for product derivation (Pro-PD). Pro-PD focuses on the essential activities, tasks, roles and work products used to derive products from a software product line. Pro-PD is an adaptable approach and can be tailored to suit different process environments. It describes essential activities and tasks that should form part of any derivation process.

6. Conclusion and Future Work

This research is motivated by the assumption that despite the adoption of SPL within industry, product derivation remains an expensive and error-prone activity – which is hard to support by tools, let alone automate. This stimulates our research which aims at supporting the adoption of automated product derivation approaches, through the structuring of the involved activities into a larger process model.

Hence, we have presented a product derivation process framework which is based on an extensive

literature review and workshops with SPL practitioners and researchers. We have studied the product derivation practices of a large automotive supplier, the observations from which we have used to augment our framework, in particular focusing on software-intensive systems (which include other aspects besides software).

We see the contribution of this framework to the automation of product derivation as twofold: First, it allows us to put automated approaches, which tackle one particular task, into a bigger context and, second, it lays the foundation for tools which support the overall process.

Our goal in the near future is to provide a version of the PDPF which is completely described electronically, i.e. in models used by the Eclipse Process Framework (EPF). First, this would allow easy access to documentation and guidelines, since these models can be published in electronic form. Second, it allows us to employ variability in the process framework. Consequently, the models can be adapted and customized for a particular industry and organization.

In the long run, we hope this framework will assist the development and adoption of automated product derivation activities in industry. Automated approaches can be a means to ease the complexity associated with the product derivation process.

7. Acknowledgements

This work is partially supported by Science Foundation Ireland under grant number 03/CE2/I303_1 and IRCSET under grant number RS/06/167.

8. References

- [1] R. Kurmann, "Agile SPL-SCM Agile Software Product Line Configuration and Release Management," in 1st International Workshop on Agile Product Line Engineering (APLE'06). Maryland, USA, 2006.
- [2] P. Trinidad, D. Benavides, A. Ruiz-Cortes, and S. Segura, "Explanations for Agile Feature Models," in 1st International Workshop on Agile Product Line Engineering (APLE'06), Maryland, USA, 2006.
- [3] K. Pohl, G. Böckle, and F. v. d. Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Heidelberg: Springer, 2005.
- [4] S. Deelstra, M. Sinnema, and J. Bosch, "Product Derivation in Software Product Families: A Case Study," in *J. Syst. Softw.* vol. 74 New York, NY, USA: Elsevier Science Inc., 2005, pp. 173-194.
- [5] G. Chastek and J. D. McGregor, "Guidelines for Developing a Product Line Production Plan," *Software Engineering Institute, Pittsburgh, PA, Technical Report CMU/SEI-2002-TR-006*, June 2002 2002.

<http://www.sei.cmu.edu/publications/documents/02.reports/02tr006.html>

- [6] S. Deelstra, M. Sinnema, and J. Bosch, "Experiences in Software Product Families: Problems and Issues During Product Derivation," in *Software Product Lines*, Third International Conference Boston, MA, USA: Springer, 2004.
- [7] L. Hotz, A. Gunter, and T. Krebs, "A Knowledge-based Product Derivation Process and some Ideas how to Integrate Product Development," in *Proc. of Software Variability Management Workshop Groningen*, The Netherlands, 2003.
- [8] M. L. Griss, *Implementing Product-Line Features with Component Reuse*. London, UK: Springer-Verlag, 2000.
- [9] R. Rabiser, P. Grunbacher, and D. Dhungana, "Supporting Product Derivation by Adapting and Augmenting Variability Models," in *Software Product Line Conference, 2007. SPLC 2007. 11th International Kyoto, Japan, 2007*.
- [10] J. D. McGregor, "Preparing for Automated Derivation of Products in a Software Product Line," *Software Engineering Institute., Technical Report CMU/SEI-2005-TR-017*, September 2005 2005.
- [11] W. S. Humphrey, *A Discipline for Software Engineering*. Boston, MA, USA: Addison-Wesley, 1995.
- [12] B. Latour, *The Pasteurization of France*. Cambridge, MA: Harvard University Press, 1988.
- [13] P. O'Leary, M. Ali Babar, S. Thiel, and I. Richardson, "Product Derivation Process and Agile Approaches: Exploring the Integration Potential," in *Proceedings of 2nd IFIP Central and East European Conference on Software Engineering Techniques*, Poznań, Poland, 2007, pp. P. 166-171.
- [14] P. O'Leary, M. Ali Babar, S. Thiel, and I. Richardson, "Towards Agile Product Derivation in Software Product Line Engineering," in *RISE 2007, 4th International Workshop on Rapid Integration of Software Engineering techniques*, Luxembourg, LUXEMBOURG, 2007.
- [15] T. Asikainen, T. Männistö, and T. Soininen, "Using a Configurator for Modelling and Configuring Software Product Lines Based on Feature Models," in *3rd Software Product Line Conference (SPLC 2004)* Boston, MA., 2004.
- [16] G. Chastek, P. Donohoe, and J. D. McGregor, "Product Line Production Planning for the Home Integration System Example," *Software Engineering Institute, Pittsburgh, Technical Note CMU/SEI-2002-TN-029*, September, 2002 2002.
- [17] A. Birk, G. Heller, I. John, T. v. d. Maßen, K. Müller, and K. Schmid, "Product Line Engineering: The State of the Practice," in *IEEE Computer Society*. vol. 20, 2003, pp. 52-60.
<http://doi.ieeecomputersociety.org/10.1109/MS.2003.1241367>
- [18] D. Nestor, L. O'Malley, A. Quigley, E. Sikora, and S. Thiel, "Visualisation of Variability in Software Product Lines," in *Proceedings of the 1st International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS-2007)* Limerick, Ireland, 2007.
- [19] R. Kauppinen, "Testing Framework-Based Software Product Lines," in *Technical Report C-2003-20*,: University of Helsinki, Department of Computer Science, 2003.
- [20] C. Kuloor and A. Eberlein, "Requirements Engineering for Software Product Lines," in *Proceedings of the 15th International Conference on Software & Systems Engineering and their Applications (ICSSEA'02)* Paris, France, 2002.
- [21] K. Pohl and A. Metzger, "Software Product Line Testing," in *Commun. ACM*. vol. 49: Communications of the ACM, 2006, pp. 78-81.
- [22] "Eclipse Process Framework Composer," 2007. http://www.eclipse.org/epf/downloads/tool/tool_downloads.php
- [23] G. Botterweck, L. O'Brien, and S. Thiel, "Model-Driven Derivation of Product Architectures," in *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, Atlanta, GA, USA, 2007. <http://doi.acm.org/10.1145/1321631.1321711>
- [24] F. Eclipse, "ATL (ATLAS Transformation Language)." <http://www.eclipse.org/m2m/at/>
- [25] T. Krebs, K. Wolter, and L. Hotz, "Model-based Configuration Support for Product Derivation in Software Product Families," Koblenz, Germany, 2005.
- [26] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "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, 2006.
- [27] O'Leary, P., Thiel, S., Botterweck, G., and Richardson, I. Towards a Product Derivation Process Framework. in *3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques CEE-SET 2008*. 2008. Brno (Czech Republic).
- [28] R. Rabiser, P. O'Leary, and I. Richardson, "Key activities for product derivation in software product lines," *Journal of Systems and Software*, vol. 84, no. 2, pp. 285-300, Feb. 2011.
- [29] P. O'Leary, M. Ali Babar, S. Thiel, and I. Richardson, "Product Derivation Process and Agile Approaches: Exploring the Integration Potential," in *Proceedings of 2nd IFIP Central and East European Conference on Software Engineering Techniques*, 2007, p. P. 166-171.
- [30] P. O'Leary, S. Thiel, G. Botterweck, and I. Richardson, "Towards a Product Derivation Process Framework," in *3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques CEE-SET 2008*, 2008, pp. 189-202.
- [30] F. McCaffery, S. Thiel, I. Richardson, P. O'Leary, F. McCaffery, S. Thiel, I. Richardson, P. O'Leary, F. McCaffery, S. Thiel, and I. Richardson, "An Agile process model for product derivation in software product line engineering," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 24, no. 5, pp. 561-571, Aug. 2010.
- [32] P. O'Leary, I. Richardson, and S. Thiel, "Towards a Product Derivation Process Reference Model for Software Product Line Organisations," University of Limerick, Limerick, 2010.
- [33] P. O'Leary, F. M. Caffery, I. Richardson, and S. Thiel, "Towards Agile Product Derivation in Software Product Line Engineering," *16th European Conference on Software Process Improvement (EuroSPI 2009)*, pp. 8.1 - 8.6, 2009.

- [34] P. O’Leary, E. S. de Almeida, and I. Richardson, “The Pro-PD Process Model for Product Derivation within software product lines,” *Information and Software Technology*, vol. 54, no. 9, pp. 1014–1028, Sep. 2012.
- [35] P. O’Leary and I. Richardson, “Process Support for Product Line Application Engineering,” in *Systems, Software and Service Process Improvement*, 18th European Conference, EuroSPI 2011, 2011, vol. 172, pp. 191–202.
- [36] P. O’Leary and I. Richardson, “Process reference model construction: implementing an evolutionary multi-method research approach,” *IET Software*, vol. 6, no. 5, p. 423, 2012.