Organisational Strategies for the Derivation of Products from a Software Product Line

Pádraig O'Leary

Lero - the Irish Software Engineering Research Centre
University of Limerick, Ireland
padraig.oleary@lero.ie

Abstract. Increasingly organisations adopt software product lines to enable extensive reuse and deliver a multitude of benefits. Compared to the vast amounts of research on developing product lines and approaches to deriving products from a product line, relatively little work has been dedicated to the organisational aspects of deriving products. Much of the current research focuses on the technical aspects of product derivation. Through our collaborations with several industry partners, we have identified four strategies that are applied by organisations to derive products. The four strategies are: Tight Collaboration, Product Centric, Experience and Hierarchy. For each strategy, its characteristics and associated advantages and disadvantages are discussed. Each strategy is illustrated with an industry example.

Keywords: Software product lines, product derivation, organisation, strategies.

1 Introduction

Research in Software Product Lines (SPL) has in the past focussed more on how to scope, define, and develop product lines, rather than on how to effectively use its assets to derive products. However a recent systematic literature review [1] shows an increasing number of publications, conference tracks, and workshops focusing on product derivation. This interest has generally centred on the technical aspects of product derivation such as managing the inherent complexities associated with deriving products from a product line. Little work has been done on the organisational approaches to product derivation or the strategies organisations apply.

Through our collaboration with industrial and academic partners, we have identified and analysed four different product derivation strategies. The four strategies identified are: *Tight Collaboration*, *Product Centric*, *Experience* and *Hierarchy*. For each strategy, we provide a description, document its advantages and disadvantages, and illustrate the strategy with industry examples from our collaborations.

The identification and classification of strategies is important for a number of reasons. It allows the comparison of approaches and therefore promotes discussion on the organisational aspects of product derivation. It provides a means of describing the organisational context of a particular product derivation approach. Finally, the documentation of product derivation strategies assists in organisations choosing the

strategy that integrates best with their product line goals and is in line with their organisational structure.

The remainder of this paper is organised as follows: Section 2 discusses related work. Section 3 describes our research approach. In Section 4 based on our experiences we present the derivation strategies identified. In Section 5 we discuss the implication of these strategies for SPL organisations. We conclude the paper with a summary in Section 6 and outline further work.

2 Related Work

The SEI Product Line Practice Framework (PLPF) [2] includes 29 practice areas grouped into three categories, namely practices related to Software Engineering, Technical Management and Organisational Management. The approach used by the SEI is to identify foundational concepts underlying software product lines and activities to be considered when creating a product line. The listed practice areas comprise an extensive set of competencies and issues necessary to consider for successful adoption of product lines. There are two practice areas relevant to this work, 'Structuring the Organisation' and 'Technical Management'.

Within the Organisational Management practice area, the 'Structuring the Organisation' practice describes organisational issues for SPL. This practice describes how groups are formed within an organisation to conduct the various responsibilities.

Within the 'Technical Management' practice area, the 'Technical Planning' practice describes the planning aspects of a product line. In particular for product derivation, it refers to the use of production plans. According to the practice, production plans have to be developed to prepare the derivation process. Such plans are documents describing inputs, necessary activities, and desired outputs of product derivation. Chastek and McGregor [3] propose detailed guidelines for creating, using, and evaluating such production plans.

The practices defined by the SEI are a robust description of best practice, involving important technical and non-technical aspects of organisation and planning for product derivation. However the framework is very generic and does not define specific ways of performing the activities. There is a strong focus on planning product derivation with the ultimate goal to automate the derivation process.

Bosch [4] describes four separate organisational units after studying a number of product line corporations. These are: *Development Department*, *Business Units*, *Domain Engineering* and *Hierarchical Domain Engineering*. In the *Development Department* unit all software development is concentrated in a single unit. The unit is particularly suitable for small organisations. In the *Business Units* each unit is responsible for a specialised subsection of the product line. In the *Domain Engineering* unit a particular group is responsible for developing the platform. Product teams build the products using those core assets. In *Hierarchical Domain Engineering* teams work with specialised product lines that use a top level product line as a basis for development.

The work builds considerable on previous work which considered only domain and application engineering units. However Bosch discriminates between the units based

only on organisational size; the work does not take into account other factors such as the impact on product derivation approaches of a particular organisational unit.

3 Research Method

We investigated product derivation strategies applied within industrial software product line organisations. The research was conducted over a series of research phases involving both academic and industrial partners. Through the phased research approach applied, we were able to identify and, compare and contrast product derivation strategies. The research followed a complementary approach where we sought confirmatory finding on strategies between the research partners.

The primary research method applied was case study research. Case study is appropriate in situations where researchers are seeking to develop understandings of the dynamics of a phenomenon in its natural context [5] and where the aim is to represent the case authentically "in its own terms" [6]. Therefore, in terms of this research it was deemed appropriate.

We conducted a case study with the Corporate Research division of Robert Bosch GmbH[1]. Robert Bosch GmbH was interested in identifying and analysing product derivation strategies applied within the business units of the software-intensive automotive product lines division.

The research was further developed through a six month collaboration with LASSY[2]; the six month collaboration involved a workshop on derivation strategies with IEE[3]. IEE are manufacturers of driver restraint systems for the automotive market. Their customers include many of the major car manufacturers worldwide. Through our involvement with IEE, we were able to contrast their product derivation strategies with the findings from the Robert Bosch GmbH case study.

We performed a third research collaboration involving JKU[4]. JKU developed the DOPLER^{UCon} (Decision-Oriented Product Line Engineering for effective Reuse: User-centered Configuration) approach which was driven by industry needs with the goal to define a user-centred, tool-supported product derivation approach [7]. The approach was mainly influenced by a research-industry collaboration with Siemens VAI Metals Technologies, the world leader in engineering and building steel plants. The goal of the collaboration was to support modelling and utilising the variability of Siemens VAI's software system for the automation of continuous casting in steel plants. The strong industry focus of the DOPLER approach allowed us to perform a type of indirect industrial study, using the DOPLER findings to supplement our research.

Based on our experiences [10 - 18], we gathered insights on how companies utilize product lines. This paper presents the generalized results of our observation.

---

[1] http://www.bosch.com
[2] Laboratory of Advanced Software Systems (LASSY), University of Luxembourg
[3] http://www.iee.lu
[4] Johannes Kepler University Linz, Austria

4 Product Derivation Strategies

In this section we discuss the four product derivation strategies identified. For each strategy, we provide a description, outline the advantages and disadvantages and describe the strategy in an industrial context. The four product derivation strategies identified and discussed are:

- *Tight Collaboration*
- *Product Centric*
- *Experience*
- *Hierarchy*

4.1 Tight Collaboration

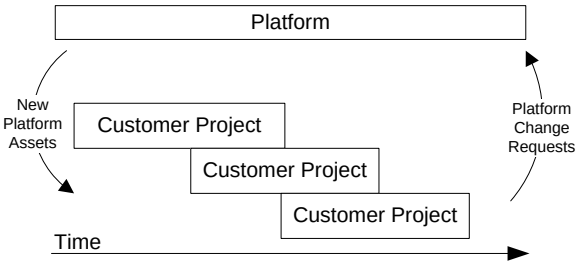

Fig. 1. Tight Collaboration Strategy

In the *Tight Collaboration* strategy (see Fig. 1) the product line is concentrated on the maintenance and evolution of the platform. Typically, the product line has one or several customer projects running simultaneously. All features requested by a customer are considered by a change control board (CCB). The CCB is comprised of key members of both the Product Team and the Platform Team. The CCB decides whether the requested feature will result in product-specific code or in adaptation of the entire product line (platform). The CCB must ensure that practical arguments such as time to market and short term cost do not result in scoping solutions that are neither optimal for the product itself nor for the product line as a whole [8]. Market forecasts, resources, implementation costs and intellectual property issues all influence the CCB's scoping decision. Intellectual property may become an issue when a customer wants a customer-only-solution and does not want their features to become part of the general platform features.

If the CCB decides that the requested feature will result in platform development, the feature will be passed to the customer project where it is integrated into the final customer product.

In the *Tight Collaboration* strategy staff should be moved from the customer project to the platform team and vice versa. This gives SPL members a good overview of the working of the product line and makes the various product teams more understanding of the platform demands. This is particularly important giving the collaborative nature of the strategy.

The advantages of the *Tight Collaboration* derivation strategy are:

- The high reuse potential of artefacts particularly if products have a high degree of commonality;
- The evolution of the product line can be planned;
- If a feature is implemented in the platform other customer projects will not have to perform feature mining.

The disadvantages of the *Tight Collaboration* derivation strategy:

- The size of the platform team may fluctuate strongly due to tight time constraints on new customer specific features. These customer specific features may need to be implemented on a short time scale therefore platform developers are moved back to product teams;
- Customer specific features which are implemented at the platform level force other customer projects, which do not use this feature, to develop a handling mechanism i.e. can they remove or disable the feature from their customer project;
- There is a high risk of feature creep or uncontrolled growth of the platform complexity due to various customer feature requests;
- Customer specific platform requests can also make the development of an 'optimal' platform more difficult;
- The co-ordination demands of this strategy put a heavy burden on the platform team to deliver features within specific customer deadlines particularly when there are multiple customer projects working in parallel.

Industry Example. Within Robert Bosch GmbH, we observed the *Tight Collaboration* strategy in operation within the software intensive automotive division. The *Tight Collaboration* strategy requires a high degree of coordination and communication, as the heavy dependencies across the platform product divide is managed. In Fig. 2 we have illustrated these platform product dependencies.

The product team designed and implemented customer specific components based on the customer requirements. The platform team received the platform software requirements containing the required extensions to the existing platform in order to facilitate the new customer requirements. Both the customer-specific development and platform development occurs in parallel. The product team need to interface correctly with the new platform release in order to leverage these extensions.

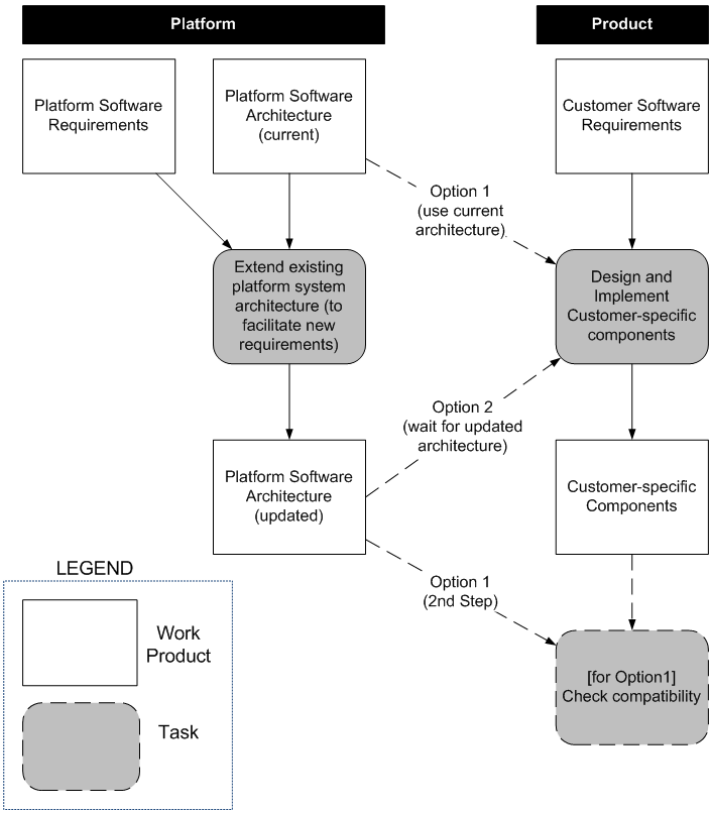

Fig. 2. Platform Product Divide in Tight Collaboration [9]

The product team typically chooses between three alternative development strategies. Option 1 is to design and implement customer specific components using the current platform release, which has not yet been updated, as a basis for development. Consequently, when the new platform architecture is released, the product team has to check the compatibility of the developed components with the new architecture. Option 2 for the product team is to wait for the updated platform release. This is suggested when potentially large compatibility issues are expected with the risk of wasted development effort. A third hybrid option is for the product team to first negotiate a platform interface with the platform team before proceeding to develop in parallel against the platform team. Alternatively, the product team can make assumptions on expected interface changes, and work from these expectations. After the updated platform release the product team check the compatibility of the developed components with the new architecture.

4.2 Product Centric

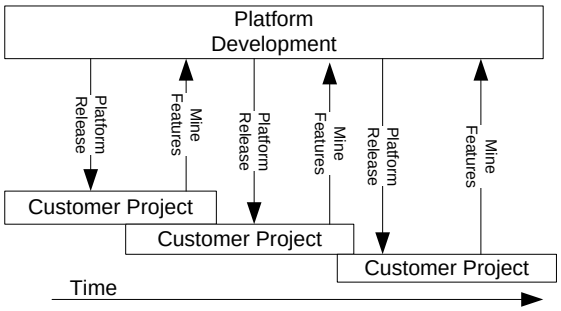

Fig. 3. Product Centric Strategy

In the *Product Centric* derivation strategy (see Fig. 3), the product and the platform development occur in two different and distinct organisational teams. Customer features are directly implemented in the customer project. The platform team monitor customer projects for new features which have a high reuse potential across the product line. Customer features which are deemed to have a sufficiently high reuse potential are mined from the relevant customer product by the platform team.

There are two considerations when considering the *Product Centric* derivation strategy. Firstly, the expectation is that the new customer projects will be based on the platform. There is no provision for new platform releases to provide required customer features, these required customer features must be implemented at product level. Secondly, the product architecture for customer projects should deviate as little as possible from the platform architecture in order to reduce the effort required for new platform releases.

The advantages of the *Product Centric* derivation strategy are:

- The strategy is ideal when collaboration and co-ordination between the product and platform teams is weak;
- On the organisational side, the strategy reduces platform workforce overhead as the platform team does not have to spend time implementing customer features;
- Project planning aspects such as staff allocation within the product line is simplified;
- The effort required to design an optimum platform solution is eased as the platform team are not performing reactive development to specific customer features (as in the *Tight Collaboration* strategy).

The disadvantages of the *Product Centric* derivation strategy are:

- The cost per each product derived is higher than other derivation strategies.
- The effort and cost required to mine features for the platform is costly.
- Project budgeting is more difficult due to uncertainties regarding effort required to reuse features and what number of features can be mined from existing projects.
- A new customer product for an existing customer may prefer an alternative project strategy such as the *Experience* strategy. Such a strategy has an increased ability to provide better savings for the customer.

Industry Example. Within IEE we observed the *Product Centric* strategy. The platform was referred to as the 'innovate product' within the organisation. The platform is the baseline from which each product is constructed. Subsequent reuse of platform artefacts is mainly opportunistic. It is the responsibility of the platform manager to identify artefacts for reuse in the platform or products within the product line. The customer requirements are managed by mapping them to a requirement model of the platform and dealing with the gaps as 'change'. This 'change' is product specific development. This strategy for product derivation allows the company to develop an optimum platform for a particular domain. Subsequent customer products use this optimum platform as a base. For the company, this strategy is a good light-weight approach to SPL development due to the minimising of costly upfront platform development.

4.3 Experience

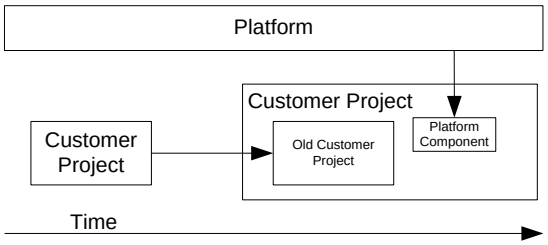

Fig. 4. Experience Strategy

In the *Experience* derivation strategy (see Fig. 4), product derivation is based on the use of an earlier release of a product as a foundation for the new product. Typically the release that is used as a starting point is an earlier product developed by the same team for the same customer. The product team then modify the old release to match the new customer requirements.

The product team can reuse parts of the platform to provide the necessary new functionality if they deem it is more efficient then product specific development. These platform components are then copied and integrated into the new product with any required modifications.

Typically, in an organisation that adopts the *Experience* strategy there are several co-existing *Experience* strategies in operation. The platform is used as the original product baseline and further releases build on the previous product releases for that customer. The strategy works well when a product is being co-developed together with the customer. The goal of the *Experience* strategy from the SPL point of view is to reduce as much as possible deviation from the platform. This reduces the cost of individual product development.

The advantages of the *Experience* derivation strategy are:

- Allows for product derivation to be based on the use of a consistent platform configuration for a particular customer.

- Allows a build of tacit knowledge within the product team for a particular customer, as team members become accustomed to the requirements for that customer.
- Undefined features can be carried over from one project to the next for the same customer, this provides continuity and reduces the need for a detailed requirements specification.
- The customer's IP (Intellectual Property) is protected.
- The development of a cost model for a particular customer project is easier and budgeting is more predictable.
- The emphasis and reliance on the platform is greatly reduced, this reduces both the associated overhead, size of the platform team and the need for platform maintenance.
- The *Experience* derivation strategy works well if requirements for different customer are very diverse or if customer products are evolving in different directions.

The disadvantages of the *Experience* derivation strategy are:

- It does not support consistent development of reusable assets.
- There is a lack of co-ordination between the product and platform team on future development for the platform.
- The opportunity for reuse between product teams is weak.
- There is a high risk of feature creep between a sequence of customer products.
- There is no centralised control over features contained in the customer specific platform.
- Product teams are reacting to customer feature requests individually.
- Product bug fixing is isolated and improvements at product level are not reflected back to the platform.
- In time the platform for different customers will diverge and can become increasingly isolated. This makes the argument for an investment or restructuring of the product line more difficult as it will typically only serve one customer.
- It becomes increasingly difficult to centralise knowledge within the platform.
- It encourages team members to become customer specific specialists. These specialists become more difficult to move to other customer projects.
- Finally, there are multiple development efforts for different customers.

Industry Example. In Siemens VAI, as reported in [10], customers often wish to upgrade existing steel plants in order to improve steel quality by deploying Siemens VAI's most recent casting technologies. It was often possible to reuse configurations from past projects as a starting point. As previous projects to the customer had handled customer requirements needs such as to interoperate with diverse legacy software and existing 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, before the Siemens team modify the old release to match the new customer requirements.

Typically the release that is used as a starting point is an earlier product developed by the same team for the same customer. The product team then modifies the old release to match the new customer requirements.

This approach allowed undefined features to be reused by the customer and helped Siemens VAI build up knowledge on the needs of particular customers.

Hierarchy

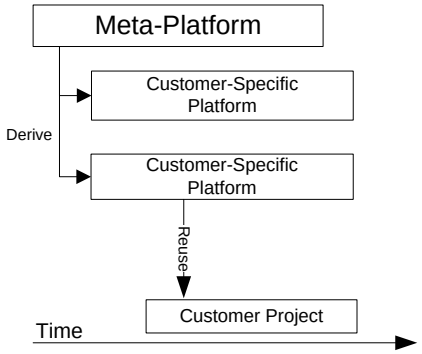

Fig. 5. Hierarchy Strategy

Typically two products for one customer are mutually much more similar than two products for two different customers. Incorporating features into the platform that are similar for the same customer but very different from other customers features, leads to a large amount of variation points within the platform. Consequently, this leads to a high degree of platform complexity.

To alleviate the problems associated with high complexity, customer-specific platforms can be derived from the original platform (or meta-platform in this context). The relation between the platforms and the customer projects can be presented in Fig. 5. The *Hierarchy* strategy is most suitable in mature SPL organisations with a large number of staff.

The meta-platform is not a full product. It contains different layers, where you have different types of variables. The customer-specific platform can be maintained in a customer project. However, division of a meta-platform into customer-specific platforms may not always be beneficial and other derivation strategies should be first considered.

The advantages of the *Hierarchy* derivation strategy are:

- The degree of reuse for building products for a certain customer out of the customer-specific platform can be extensive.
- It provides a means of organising the product derivation effort if a large number of staff are involved
- It can simplify the complexity associated with large amounts of variability in the platform

The disadvantages of the *Hierarchy* derivation strategy are:

- There is a large co-ordination effort to keep the various platforms in sync.
- Suitable for large organisations only
- Not a 'light-weight' approach to product derivation

Industry Example. Within Robert Bosch GmbH, we observed the *Hierarchy* strategy. The organisation had an airbag control system product line. This was the meta-platform from which customer-specific platforms were derived for customer such as BMW. Each customer-specific platform was broken into three broad disciplines, software, hardware and mechanics. Within each of these disciplines there are further sub-disciplines. For instance, the hardware discipline has a microcontroller team and an ECU (Electronic Control Unit) team. The mechanics discipline has housing, mechanical quality, interfaces and plugs teams. The software discipline had basic software and algorithms teams.

The *Hierarchy* strategy created the need for intricate role structures within the product line that were broken down according to the hierarchical structure of the product line. These intricate role structures are reflected by appropriate communication and task structures. For instance, the allocation of requirements to responsible teams requires the various disciplines and sub-disciplines to have a finer degree of granularity than in other strategies.

Another consequence of the *Hierarchy* strategy, within Robert Bosch GmbH, was raised importance of modularisation as a result of the distributed development across both disciplines and platform and product teams. Consequently, interface management is performed as an explicit task and encapsulation is a key design property for development; a software component should ideally be independent of how a sensor, actuator or microcontroller works internally.

The use of the *Hierarchy* strategy relied heavily on documentation to drive the product derivation process. Documentation was used to facilitate communication and synchronise development between the product and platform teams, in the various platforms, between the different hardware, software and mechanical disciplines and also between the sub-disciplines.

5. Discussion

This research identifies and classifies four organisational strategies for product derivation. Up to now, no classification of these strategies is documented. The classification of strategies adopted by an organisation plays two roles. Firstly, classification allows the comparison of strategies and the documentation of the organisational environment to which a strategy is best suited. Secondly, classification provides a means of describing the strategic context for a particular product derivation approach. The strategic context impacts on the approach, for instance whether a customer requirement is automatically handled at product or platform level.

For organisations the strategy and role structure within the organisation are inextricably linked. Ideally the product derivation strategy should be motivated by the business goals for the SPL; however situations arise where the role structure is fixed and the strategy applied for product derivation should reflect this role structure. For instance, a SPL organisation having a structure centred on Development Departments [4] when the strategy being applied is *Experience* could result in the organisation failing to develop and retain the tacit knowledge that would be expected from that

strategy. If the structure and strategy are not considered in tandem then the result could be a mismatch within the SPL organisation. This mismatch between structure and strategy could be a contributing factor to a poor return on the SPL effort.

An organisation should carefully consider before deciding on their product derivation strategy. The strategy should be in line with their SPL goals and the environment in which they operate. For instance, an organisational with a well institutionalised SPL approach might find that a *Tight Collaboration* strategy is best suited. However, for a relatively new SPL effort, where the organisation is more accustomed to traditional software development, then *Product Centric* is likely to be more intuitive.

Limitations and Future Work

In this paper we report on four product derivation strategies which we identified over the course of our research into product derivation. The results presented in this paper are not the definitive guide to organisational product derivation strategies. Nonetheless, it is presented as a step towards a consensus on organisational strategies for product derivation and forms the anchor on which further improvements can be built.

It is almost certain that other organisational strategies for product derivation exist and an empirical inquiry in relation to this should be conducted.

In our future work, we aim to conduct systematic review of the literature on organisational strategies. Through the literature we hope to perform a literature based verification of our finding while also identifying other strategies. A survey on product derivation strategies within industry could also validate identified strategies and ensure identification of remaining strategies.

Ideally, research into the relationship between the organisational structure and the product derivation strategy would be conducted. Of particular interest would be investigating the relationship between the organisational units defined by Bosch [4] and the strategies documented here.

6. Conclusion and Future Work

Through our research with several software product line organisations we have identified four strategies for deriving products from their product line. The four strategies we identified are *Tight Collaboration, Product Centric, Experience* and *Hierarchy*. For each strategy, its characteristics, advantages and disadvantages are discussed. We illustrate these strategies with examples from industry collaborations.

Tight Collaboration: In this strategy the product line is concentrated on the maintenance and evolution of the platform. Customer feature requests are scoped by a CCB for product or platform implementation. The strategy is particularly suitable when products have a high degree of commonality. However, the strategy can lead to platform feature creep or a growth in complexity within the platform due to various customer requests.

Product Centric: In this strategy product and platform development occur in two distinct organisational teams. Customer features are directly implemented in the customer project. Customer features which are deemed to have a sufficiently high reuse potential are then mined by the platform team. The strategy is ideal when collaboration and co-ordination between the product and platform teams is weak. However, the strategy can lead to higher costs for deriving products and due to the effort required to mining features for the platform.

Experience: In this strategy product derivation is based on the use of an earlier release of a product as a foundation for the new product. The product team then modify the old release to match the new customer requirements. The product team can reuse parts of the platform to provide the necessary new functionality if they deem it is more efficient than product specific development. The strategy is most suitable when organisations want to reduce the associated overhead and costs associated with platform maintenance or if particular customer products are evolving in different directions.

Hierarchy: In this strategy when products for one customer are mutually much more similar than products from other customers, organisations derive customer-specific platforms. The meta-platform is not a full product. It contains different layers, in which there are different types of variables. The advantages of this strategy is that there can be a high degree of reuse for building products for a certain customer out of the customer-specific platform. However, division of a meta-platform into customer-specific platforms may not always be beneficial and other derivation strategies should first be considered.

7. Acknowledgements

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie) and by IRCSET under grant no. RS/06/167. We also want to thank Klaas-Jan Stol for valuable feedback on the paper.

References

1. Rabiser, R., Grünbacher, P., and Dhungana, D., Requirements for Product Derivation Support: Results from a Systematic Literature Review and an Expert Survey (in press). *Information and Software Technology*, 2009(Elsevier).

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

3. Chastek, G. and McGregor, J.D. (2002). *Guidelines for Developing a Product Line Production Plan*, CMU/SEI-2002-TR-006. Pittsburgh, PA: Carnegie Mellon Software Engineering Institute.

4. Bosch, J., *Software Product Lines: Organizational Alternatives*, in *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*. 2001.

5. Yin, R., *Case Study Research : Design and Methods*. 2003: SAGE Publications.

6. Hammersley, M., Gomm, R., and Foster, P., *Case Study Method: Key Issues, Key Texts*. 2000, London: Sage Publications.
7. Rabiser, R., Grünbacher, P., and Dhungana, D., *Supporting Product Derivation by Adapting and Augmenting Variability Models*, in *11th International Software Product Line Conference*. 2007: Kyoto, Japan.
8. Deelstra, S., Sinnema, M., and Bosch, J., Product Derivation in Software Product Families: A Case Study. *Journal of Systems and Software*, 2005. **74**(2): p. 173-194.
9. 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).
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.