

ULRR

Towards a product derivation process reference model for software product line organisations

Item Type	Thesis
Authors	O'Leary, Pádraig
Download date	2026-06-09 09:06:37
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/374



University of Limerick

OLLSCOIL LUIMNIGH

Towards a Product Derivation Process Reference Model for Software Product Line Organisations

Submitted by Pádraig O'Leary
For the award of Doctor of Philosophy

Supervised by
Dr. Ita Richardson
Prof. Steffen Thiel

Submitted to the University of Limerick
January 2010

To My Family

Pádraig O’Leary (2010)

**Towards a Product Derivation Process Reference Model for
Software Product Line Organisations**

Abstract

The derivation of individual products from a software product line is still seen as a time-consuming and expensive activity in many organisations. Despite recognition that an effective derivation process could alleviate many of the difficulties associated with product derivation, little work has been dedicated to this area. Existing approaches have very different scope and emphasise different aspects of the derivation process. Furthermore, they are frequently too specialised to a specific development technique to serve as a general solution. This leaves organisations with no centralised starting point for defining an approach to product derivation.

Accordingly there is a strong need for a structured approach to product derivation which defines activities, tasks, roles, inputs and outputs of each step in a structured and systematic way.

Through a series of research stages using sources in industry and academia, this research has 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.

Declaration

The work presented in this thesis is entirely my own work. It has not been submitted previously to this or any other institute for this or any other academic award. Where use has been made of the work of other people, it has been acknowledged and referenced.

Signed: _____

Pádraig O'Leary

Date: _____

Acknowledgements

The author wishes to acknowledge the help and support of several people and organisations, without whom, this thesis would not have been possible.

Firstly I would like to thank my two supervisors, Dr. Ita Richardson and Prof. Steffen Thiel. They filled the journey with support and encouragement. Thank you both very much!

I would like to thank my examiners, Dr. Jim Buckley and Dr. Liam O'Brien, for their time and feedback.

This research has been funded by Science Foundation Ireland through IRCSET under Grant no. RS/06/167. I would like to thank them for the opportunity. To Lero and all its members, I'm grateful for the supportive research environment over the past few years.

To my family Jerry, Margaret, Tadhg, Catherine, Helen and Kate, your support began long before I ever undertook this journey. I have been very fortunate.

*Mo mhíle buíochas do na daoine thuasluaite a
chabhraigh liom agus a thug tacaíocht dom*

Table of Contents

CHAPTER ONE: INTRODUCTION	15
1.1 INTRODUCTION	15
1.2 PROBLEM STATEMENT	16
1.3 OVERVIEW OF PROPOSED SOLUTION	16
1.4 DEVELOPMENT AND EVALUATION	17
1.5 CONTRIBUTION	18
1.6 OUTSIDE THE SCOPE OF THE WORK	19
1.7 STRUCTURE OF THESIS	19
CHAPTER TWO: LITERATURE REVIEW	21
2.1 INTRODUCTION	21
2.2 SOFTWARE PRODUCT LINES	21
2.2.1 <i>Background to Software Product Lines</i>	22
2.2.2 <i>Fundamentals and Basic Concepts</i>	22
2.3 PRODUCT DERIVATION.....	24
2.3.1 <i>Approaches</i>	25
2.3.2 <i>Conclusion</i>	38
2.4 SOFTWARE PROCESS RESEARCH.....	41
2.4.1 <i>Software Process Models</i>	42
2.4.2 <i>Process Reference Models</i>	46
2.4.3 <i>Software Product Line Assessment Methods</i>	49
2.5 CONCLUSION	50
CHAPTER THREE: RESEARCH METHODOLOGY	52
3.1 INTRODUCTION	52
3.2 RESEARCH OBJECTIVES.....	52
3.2.1 <i>Objective One – To Define a Structured Process Model for Product Derivation</i>	53
3.2.2 <i>Objective Two - To Demonstrate the Adaptability of the Process Model</i>	53
3.2.3 <i>Objective Three - To add to the Established Knowledge on Product Derivation</i>	54
3.2.4 <i>Finally</i>	54
3.3 RESEARCH PHILOSOPHY	54
3.3.1 <i>Positivism vs. Interpretivism</i>	54

3.3.2 Quantitative vs. Qualitative	55
3.3.3 Explanatory vs. Exploratory	56
3.3.4 Inductive vs. Deductive	56
3.3.5 Constructive	57
3.4 RESEARCH DESIGN	57
3.4.1 Overview of Research Design.....	58
3.4.2 Stage One – Literature Review and Expert Opinion Workshops	61
3.4.3 Stage Two – Industrial Case Study: Robert Bosch GmbH.....	66
3.4.4 Stage Three – Academic Comparative Analysis: DOPLER ^{UCON}	71
3.4.5 Stage Four – Systematic Evaluation.....	73
3.4.6 Research Design Conclusion.....	75
3.5 RESEARCH VALIDITY	76
3.6 HANDLING REFINEMENTS	79
3.7 MEETING THE RESEARCH OBJECTIVES	80
3.8 SUMMARY	81
CHAPTER FOUR: DEVELOPMENT AND EVALUATION OF PRO-PD	82
4.1 INTRODUCTION	82
4.2 STAGE ONE - LITERATURE REVIEW AND EXPERT OPINION WORKSHOPS	83
4.2.1 Steps taken in Stage One	83
4.2.2 Observations	83
4.2.3 Conclusion of Stage One	92
4.3 STAGE TWO - INDUSTRIAL CASE STUDY: ROBERT BOSCH GMBH	93
4.3.1 Observations from the Case Study.....	93
4.3.2 Impact on Pro-PD.....	97
4.3.3 Reaction of Robert Bosch GmbH Workshop Participants to Pro-PD Version One.....	102
4.3.4 Summary of Pro-PD Changes due to Stage Two of Research.....	102
4.3.5 Conclusion of Stage Two: Industrial Case Study	105
4.4 STAGE THREE - ACADEMIC COMPARATIVE ANALYSIS: DOPLER ^{UCON}	105
4.4.1 Preparing for Derivation Comparison	106
4.4.2 Product Configuration Comparison.....	109
4.4.3 Product Development and Testing Comparison.....	111
4.4.4 Conclusion of Stage Three: Academic Comparative Analysis.....	113

4.4.5 Summary of Pro-PD Changes due to Stage Three of Research	115
4.5 STAGE FOUR: EVALUATION	117
4.5.1 Evaluation Method	117
4.5.2 Inter-model Evaluation	118
4.5.3 Analysis of Existing Approaches.....	126
4.5.4 Discussion of Results	137
4.5.5 Summary of Pro-PD Changes due to Stage Four of Research	138
4.6 FORMALISING THE MODEL.....	139
4.6.1 Describing Pro-PD using EPF – Introducing Activities	140
4.7 SUMMARY	140
CHAPTER FIVE: PRO-PD DESCRIPTION	142
5.1 INTRODUCTION	142
5.2 PROCESS DESCRIPTION.....	142
5.2.1 Roles.....	143
5.2.2 Units of Work: Tasks and Activities.....	148
5.2.3 Work Products	170
5.3 WATERFALL INSTANTIATION.....	173
5.3.1 Preparing for Derivation	175
5.3.2 Product Configuration.....	175
5.3.3 Product Development and Testing.....	176
5.4 PRO-PD MATRIX	176
5.5 CONCLUSION	177
CHAPTER SIX: ADAPTING PRO-PD TO AGILE	178
6.1 INTRODUCTION	178
6.2 AGILE PRACTICES AND SPL	178
6.3 THE AGILE MANIFESTO AND PRODUCT DERIVATION	180
6.4 THE MODEL IN ACTION – AN ITERATIVE INSTANTIATION.....	184
6.4.1 Use by Specialisation - Agile Pro-PD	185
6.4.2 Model Phases.....	186
6.4.3 Phase Milestones	190
6.5 INCREASING AGILITY IN PRODUCT DERIVATION	191
6.5.1 Adoption of Early and Continuous Delivery Strategy	191

6.5.2 Automation of Product Derivation.....	192
6.5.3 Product Derivation Iterations.....	193
6.5.4 Agile Testing Techniques.....	193
6.6 CONCLUSION AND FUTURE WORK.....	194
CHAPTER SEVEN: CONCLUSION.....	195
7.1 INTRODUCTION	195
7.2 RESEARCH DESIGN	195
7.3 MEETING THE RESEARCH OBJECTIVES	197
7.3.1 Objective One – To define a structured process model for product derivation.....	197
7.3.2 Objective Two - To demonstrate the adaptability of the process model	197
7.3.3 Objective Three - To add to the established knowledge on product derivation.....	198
7.4 OVERVIEW OF SOLUTION.....	198
7.5 SUMMARY OF CONTRIBUTION	199
7.6 LIMITATIONS.....	201
7.6.1 Product Testing	201
7.6.2 Adaptation Guidelines	202
7.6.3 Case Study Bias	202
7.7 FUTURE WORK.....	202
7.7.1 Industrial Application of Pro-PD.....	202
7.7.2 Regulatory Conformance	203
7.7.3 Agile Product Derivation	203
7.8 CONCLUDING REMARKS	204
REFERENCES.....	205

List of Figures

Figure 1-1 Overview of Research Design	18
Figure 2-1 Domain and Application Engineering Phase [19]	24
Figure 2-2 FORM Process [28]	29
Figure 2-3 The Three Activities of SPL [1]	32
Figure 2-4 The ConIPF Derivation Process [3]	35
Figure 2-5 DOPLER ^{UCON} Approach[34]	36
Figure 2-6 Waterfall Model [7]	43
Figure 2-7 Spiral Model [42]	44
Figure 2-8 Use by Specialisation	49
Figure 3-1 Overview of Research Design	60
Figure 3-2 Stage One	61
Figure 3-3 Workshop Whiteboard	65
Figure 3-4 Stage Two: Industrial Case Study	68
Figure 3-5 Sample Whiteboard Drawing from the Case Study Workshop	69
Figure 3-6 Academic Comparative Analysis	71
Figure 3-7 Systematic Evaluation	75
Figure 4-1 The COVAMOF Derivation Process [22]	84
Figure 4-2 Form Application Engineering [28]	85
Figure 4-3 Overview of Pro-PD Version One	86
Figure 4-4 Framework Skeleton Discussions at Workshop	88
Figure 4-5 Impact Analysis	92
Figure 4-6 Synchronisation Caused by Platform-Product Dependency	96
Figure 4-7 Organisational Structure and Roles	99
Figure 4-8 Product Development	101
Figure 4-9 Systematic Evaluation	118
Figure 5-1 Conceptual Model of Pro-PD Structure	143
Figure 5-2 Customer Role	144
Figure 5-3 Platform Manager Role	145
Figure 5-4 Product Architect Role	146
Figure 5-5 Product Developer Role	147
Figure 5-6 Product Manager Role	148
Figure 5-7 Product Tester Role	148

<i>Figure 5-8 Overview of Pro-PD Activities</i>	150
<i>Figure 5-9 Initiate Project Activity</i>	152
<i>Figure 5-10 Identify and Refine Requirements Activity</i>	156
<i>Figure 5-11 Derive the Product Activity</i>	160
<i>Figure 5-12 Develop the Product Activity</i>	164
<i>Figure 5-13 Test the Product Activity</i>	167
<i>Figure 5-14 Project Assessment Activity</i>	169
<i>Figure 5-15 Waterfall phases and work products</i>	174
<i>Figure 5-16 Preparing for Derivation Phase</i>	175
<i>Figure 5-17 Product Configuration Phase</i>	176
<i>Figure 5-18 Product Development and Testing</i>	176
<i>Figure 6-1 Agile Pro-PD</i>	187
<i>Figure 6-2 Preparing for Derivation Phase</i>	188
<i>Figure 6-3 Product Derivation Phase</i>	189
<i>Figure 6-4 Product Development and Testing Phase</i>	190
<i>Figure 7-1 Overview of Research Design</i>	196
<i>Figure 7-2 Pro-PD as a Foundation for Automated Approaches</i>	200

List of Tables

<i>Table 2-1 The SEI Product Line Practice Areas [1]</i>	33
<i>Table 3-1 Model Notation</i>	64
<i>Table 3-2 Simplified Version of Spreadsheet Used</i>	73
<i>Table 4-1 Description of Phase Attributes</i>	88
<i>Table 4-2 Impact Analysis Phase</i>	89
<i>Table 4-3 Description of Task Attributes</i>	90
<i>Table 4-4 Task Attributes for Customer Negotiation</i>	91
<i>Table 4-5 Summary of Stage Two Changes to Pro-PD</i>	105
<i>Table 4-6 Mapping Preparing for Derivation to DOPLER^{UCon}</i>	108
<i>Table 4-7 Mapping Product Configuration to DOPLER^{UCon}</i>	110
<i>Table 4-8 Mapping Product Development and Testing to DOPLER^{UCon}</i>	113
<i>Table 4-9 Summary of Stage Three Changes to Pro-PD</i>	117
<i>Table 4-9 Relevant PLPF Practices</i>	120

<i>Table 4-10 Evaluation of Requirements Engineering Practice Area</i>	<i>122</i>
<i>Table 4-11 Practice Summary Results</i>	<i>124</i>
<i>Table 4-12 Practice Classification</i>	<i>125</i>
<i>Table 4-13 Evaluation Framework (adapted from [99]).....</i>	<i>128</i>
<i>Table 4-14 Results of Analysis Using Evaluation Framework</i>	<i>135</i>
<i>Table 4-15 Summary of Stage Four Changes to Pro-PD</i>	<i>139</i>
<i>Table 5-1 Initiate Project Activity Summary</i>	<i>151</i>
<i>Table 5-2 Identify and Refine Requirements Activity Summary</i>	<i>155</i>
<i>Table 5-3 Derive the Product Activity Summary.....</i>	<i>159</i>
<i>Table 5-4 Develop the Product Activity Summary</i>	<i>163</i>
<i>Table 5-5 Test the Product Activity Summary</i>	<i>167</i>
<i>Table 5-6 Project Assessment Activity Summary.....</i>	<i>169</i>
<i>Table 5-7 Work Product Table</i>	<i>173</i>
<i>Table 6-1 The Agile Manifesto [46]</i>	<i>181</i>

Chapter One: Introduction

1.1 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 a 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. 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 built 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.

Product derivation is the process of constructing a product from a Software Product Line's (SPL) core assets [3]. An effective product derivation process can help to ensure that the benefits delivered through using these shared artefacts across the products within a product line is greater than the effort required to develop the shared assets. In fact, 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" [4] might not hold if inefficient derivation practices diminishes the expected gains.

1.2 Problem Statement

In the context of inefficient product derivation, a number of publications speak of the difficulties associated with the process. Hotz et al. [2] describe it as “slow and error prone, even if no new development is involved”. Griss [5] 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, the derivation of individual products from shared software assets is still a time-consuming and expensive activity in many organisations [3].

Despite this, there has been little work dedicated to the overall product derivation process. Rabiser et al. [6] claim that “guidance and support are needed to increase efficiency and to deal with the complexity of product derivation”. As Deelstra et al. [3] states there “is a lack of methodological support for application engineering and, consequently, organisations fail to exploit the full benefits of software product families.” As a result current approaches fail to provide a holistic view of product derivation leaving organisations with no centralised starting point for defining an approach to product derivation. This results in ad-hoc solutions. Accordingly there is a strong need for a structured approach to product derivation which defines activities, tasks, roles, inputs and outputs of each step in a structured and systematic way. This thesis aims to fill this identified gap and the objective of the work described can be stated as:

To define a systematic process which will provide a structured approach to the derivation of products from a software product line, based on a set of tasks, roles and work products, and to demonstrate its adaptability to different process models.

1.3 Overview of Proposed Solution

In order to achieve the goal, defined above, this thesis develops the Pro-PD process for product derivation. Pro-PD is a process reference model for product derivation that is minimal, complete, and adaptable:

- Minimal – only content that is seen as essential for product derivation is included;

- Complete – it can be manifested as an entire process to build a system;
- Adaptable – it can be adapted to different process types.

Pro-PD is a minimally sufficient process reference model for product derivation. This means that only fundamental product derivation process content is included. Domain and discipline specific content is not included in Pro-PD. Pro-PD is independent of the methods and techniques used to derive a product focusing instead on the essential tasks, roles and work artefacts used to derive products from a software product line. Finally, Pro-PD is adaptable, it can be used as a foundation from which company specific product derivation process content can be developed. The process structure is based on the waterfall process model [7]; however, to demonstrate its flexibility, it is adapted to fit the characteristics of an iterative process model in Chapter Six.

1.4 Development and Evaluation

Pro-PD was constructed using development – evaluation stages (see Figure 1-1). Version one of Pro-PD was developed through sources in the literature and captured expert opinion, and evaluated during an industrial case study with Robert Bosch GmbH. Version two was developed based on a study of derivation practices at Robert Bosch GmbH and evaluated during a research collaboration with the DOPPLER laboratory. Version three was developed based on the findings of the research collaboration with DOPPLER. This version was evaluated through an inter-model evaluation and an adapted SPL evaluation framework. Finally, with the results of this evaluation integrated, version four of Pro-PD is presented.

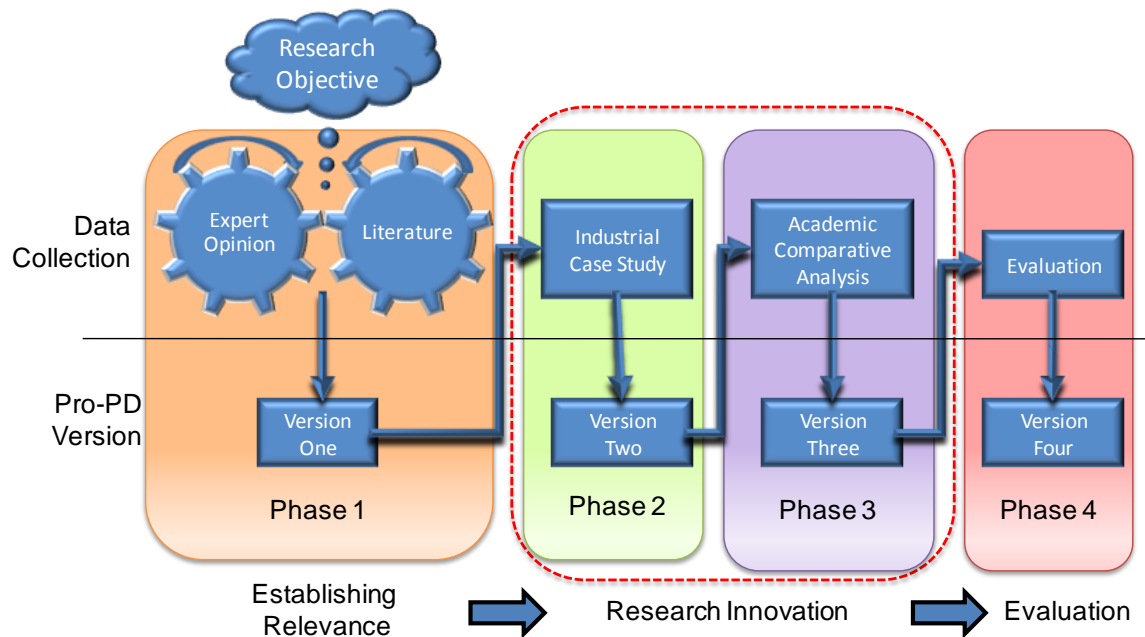


Figure 1-1 Overview of Research Design

1.5 Contribution

This research has resulted in a number of contributions. An extensive literature review is presented that highlights issues within current approaches to product derivation. Observations on product derivation practice from both academia and industry are described.

An adaptable process model for product derivation (Pro-PD) is developed and evaluated; the model describes the roles, work products and activities essential for product derivation. Pro-PD can be used as a foundation from which company specific product derivation process content can be developed therefore providing companies with a solid starting point for the definition of a product derivation approach

The research contributes to the potential integration of Agile practices in product derivation through the adaptation of Pro-PD to create A-Pro-PD. The Agile instantiation of Pro-PD provides a means of supporting the adoption of Agile practices. An integrated Agile approach could solve many of the problems associated with product derivation's complex and cumbersome nature. A combination of Agile and SPL is expected to create a leaner but more disciplined product derivation process.

1.6 Outside the Scope of the Work

As the proposed process model is part of a broader SPL context, a set of related aspects will not be included. The following issues are not directly addressed by this work:

- **Domain Engineering.** An important issue in the SPL process is to define the development of the artefacts for reuse. However, this aspect can be as complex as product derivation, involving the definition of activities, tasks, inputs, outputs, and roles;
- **System Delivery or System Maintenance:** Pro-PD is focused on product construction and does not consider system delivery or system maintenance;

1.7 Structure of Thesis

The remainder of this thesis is organised as follows:

- **Chapter Two: Literature Review.** The chapter commences with a discussion on the emergence of SPL and the associated concepts. The current approaches to product derivation are discussed along with outstanding issues. The concept of software process improvement is introduced with an overview of the more famous process models. Process reference models are introduced and various adaptation approaches outlined. Finally, an overview of existing SPL assessment methods is presented.
- **Chapter Three: Research Methodology.** The research objectives are presented and justification is made for the choice of research paradigm. The research design is presented including the selection of research methods and a detailed description of how these methods were executed. Potential threats to the research validity are identified and discussed. Finally, the researcher looks at how the research objectives were met through the applied research methodology.
- **Chapter Four: Development and Evaluation of Pro-PD.** The iterative development and evaluation of the model is presented. The observations made

during the various stages of development and evaluation, are described and their influence on the final model outlined.

- **Chapter Five: Pro-PD Description.** This chapter presents an overview of the proposed process reference model for product derivation (Pro-PD). The process concepts, roles, work products and tasks are described.
- **Chapter Six: Adapting Pro-PD to Agile.** In this chapter Pro-PD is adapted. The adaptation of Pro-PD to fit the characteristics of an Agile process model is demonstrated.
- **Chapter Seven: Conclusion.** A summary of the contribution of this work is presented and the directions for future work are outlined.

Chapter Two: Literature Review

2.1 Introduction

For the past decade the software product line approach has been an emerging paradigm that provides means to incorporate reuse as a part of software development [1, 8]. The basic concept of SPL is to develop a software foundation that can be used for every product line member. So rather than developing variant products independently, one reuses existing components that are designed to be reused.

This idea of software reuse has existed for some time. Software development is time-consuming and error-prone and the systems produced are complex. Reuse is seen as a strategy for solving these issues.

However, the existing reuse processes present crucial problems such as gaps in important activities like development for (Domain Engineering) and with (Application Engineering) reuse. Even today, with SPL, there is still no clear consensus about the activities to be performed (inputs, outputs, artefacts and roles) and the requirements that an effective reuse process must meet.

Bayer et al. [9] summarised this gap when they said “A flexible method that can be customised to support various enterprise situations with enough guidance and support is needed” (pp. 122). With this motivation, the remainder of this chapter presents an overview of SPL and a review of SPL approaches with a focus on product derivation. The area of Software Process Improvement is discussed including the use of process reference models. Finally, the literature review concludes with a review of SPL assessment methods.

2.2 Software Product Lines

SPL is expected to provide similar advantages as product lines in manufacturing. The main advantages anticipated are reduced time to market and lower development costs [1] once the product line for an application domain is functional. While the success of product lines in manufacturing can be seen as a source of inspiration to the software engineering

community, the original concept was first introduced by Parnas [10] when he proposed the idea of families of related software products. Some experts believe that SPL will become the dominant software development paradigm in the near future [11]. Product flexibility is currently a very important factor in the software market and, with product lines, the promise of a tailor made system built specifically for the needs of a particular customer or group of customers can be fulfilled. Especially for large companies, product lines offer an efficient way to exploit the commonalities shared by different products to achieve economies of production [12, 13].

2.2.1 Background to Software Product Lines

According to McGregor et al. [14] Whitney revolutionised the manufacturing of rifles using interchangeable parts. As Coallier et al. [15] explain this was the first widespread use of the product line approach. Whitney promised to build 10,000 muskets of different models in two years for the US government. The following citation, which is still relevant today, discusses the results of his efforts [16].

“Almost eight years was required for Whitney to fill the order, because practice still showed many gaps in his system. The number of details seemed endless. However, most of the ten thousand were turned out in the last two years. In 1811, Whitney took an order for fifteen thousand, and these were turned out within only two years.”

The example shows that while there are substantial arguments in favour of product lines, implementing a product line takes a lot of effort. Other famous product line instances are Ford’s assembly line approach and Boeing’s aircraft production line. Of course, any company who exploits commonality in their products to enable mass-production could be said to be involved in product lines.

2.2.2 Fundamentals and Basic Concepts

The fundamental idea on which software product lines are based is on the planned reuse and management of a set of artefacts from which you can systematically design, define,

build and maintain a set of related products in a given domain. This idea is captured best by Clements and Northrop [1]:

Definition 1 (Software Product Lines) a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

A software product line captures the commonalities between a set of products while providing for the differences between different instances of the product. The SPL approach is much more about strategic reuse of artefacts. As a result SPL is as much about business practices as they are about the technical practices [17]. Adopting a product line approach requires a shift in mindset for most organisations, as the focus changes from single system development to developing a product line. Alternative terminologies for SPL exist, van der Linden [18] reports that the term “product family” or “system family” is used in Europe, whereas in the United States the term “software product line” is more commonly used.

SPL organisations need to perform two intertwined activities domain engineering and application engineering. In domain engineering the commonalities and variabilities between the products within the product line are designed and realised. In application engineering these commonalities and variabilities are used to derive individual products. This separation into domain engineering and application engineering allows the development of software assets which are shared among the products within that domain (see Figure 2-1). These shared assets become separate entities in their own right, 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 and where necessary product specific development. The process of creating these individual products is known as product derivation.

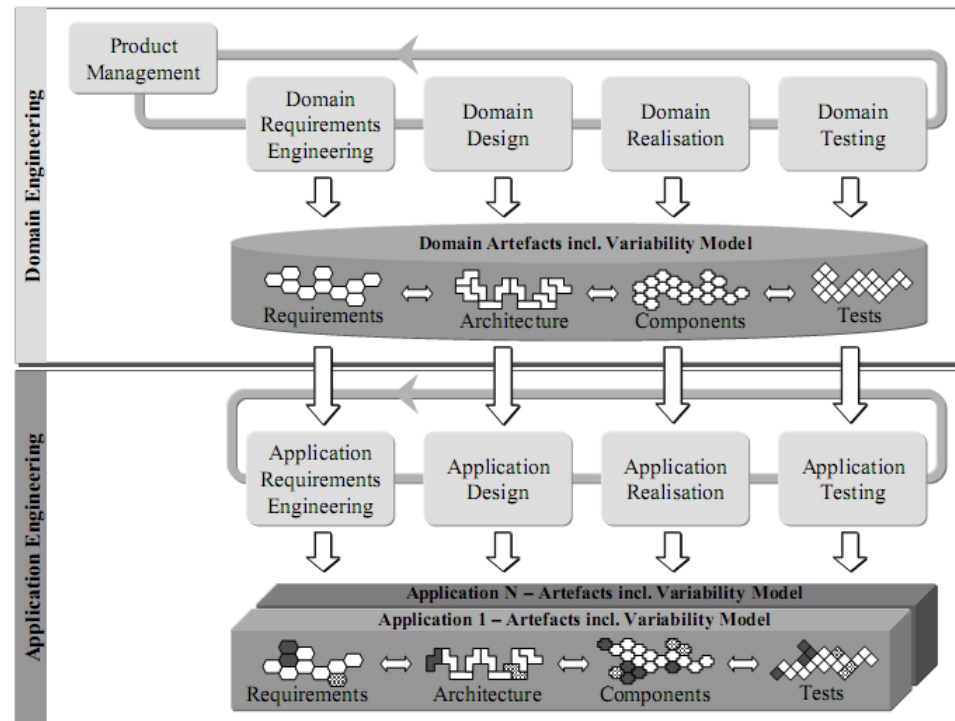


Figure 2-1 Domain and Application Engineering Phase [19]

2.3 Product Derivation

Product Derivation is the process of constructing a product from a product line of software assets [3]. An effective product derivation process within an organisation ensures 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. In fact, 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” [4] might not hold if inefficient derivation practices diminishes the expected gains. Hence, there is a strong need for the identification of a systematic structured approach to product derivation in order to achieve maximum return on investment within SPL development.

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 [20] and the various activities of the derivation process are automated and supported by adequate tools.

In an ideal SPL environment, product derivation is a configuration and integration activity rather than a development one. However in practice, many requirements are not accounted for in the shared product family artefacts and can only be accommodated by adaptation of existing components and assets or even new development [21].

Much of the SPL research to date has been focused on the design and implementation of the shared software artefacts in a way that application engineers can derive the products with greater ease. However there is little work dedicated to the overall product derivation process. As a result, the derivation of individual products from shared software assets is still a time-consuming and expensive activity in many organisations [3].

Griss [5] identifies the inherent complexity and the 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”. Other authors such as Rabiser *et al.* [6] have also re-iterated the call for further guidance and support in the product derivation process.

Many of the existing SPL approaches consider product derivation purely in terms of the configuration of platform artefacts. The SPL approaches that support or partly automate product derivation in software product line development are presented in the following section, along with the product derivation specific approaches; COVAMOF [22] and DOPLER^{UCON} [23].

2.3.1 Approaches

2.3.1.1 PuLSE/PuLSE-I

PuLSE (**P**roduct **L**ine **S**oftware **E**ngineering) [9] is a method engineering framework developed at the Fraunhofer IESE. The method consists of three major elements: Deployment Phases, Support Components and Technical Components. Deployment phases include various activities required to develop the framework. Support components supply

any additional expertise needed for process customisation and/or to solve any other organisational issues. Technical components provide the technical knowledge required to carry out different deployment activities. For this task, PuLSE has components for Customising (BC), Scoping (Eco), Modelling (CDA), Architecting (DSSA), Instantiating (I) and Evolving and Management (EM).

It is the Instantiating component that describes the “derivation part” of the PuLSE method, *PuLSE-I* (I for instantiation) [24]. PuLSE-I activities cover planning product derivation, instantiating a product architecture from the 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. The approach defines roles and tasks, however on a very high-level.

According to Atkinson et al. [25], PuLSE has been applied in a number of different contexts for different purposes. It proved to be helpful for introducing sound documentation and development techniques into existing development practices. However, where a formalised process did not exist, the introduction of PuLSE turned out to be problematic.

PuLSE-I is not a standalone derivation process. Many dependencies exist to other parts of the overall PuLSE process making PuLSE highly iterative but also highly complex. The approach is not very applicable outside the scope of a PuLSE applied product line. Overall, PuLSE-I is a complex approach with a high-level process description. The approach is not flexible and explicit tailoring possibilities should be defined to make it more usable.

2.3.1.2 KobrA

The KobrA [25] approach is centred on component-based product line development and integrates existing software engineering technologies like framework and process modelling. As described in the previous section, in situations where there were no pre-existing processes or well-defined products, the introduction of PuLSE turned out to be

problematic. With this in mind, the KobrA approach was proposed [25]. KobrA is an object-oriented customisation of the PuLSE method.

KobrA identifies commonalities and variabilities in a specific application domain and creates a reusable software infrastructure (or framework) which may be instantiated to identify the specific products. KobrA has two main activities: initially, framework engineering creates and maintains a generic framework that embodies all product variants that make up the family, including information about their common and disjoint features. Next, application engineering uses the framework built during framework engineering to construct specific applications within the domain covered by the framework. The application engineering process is explicitly defined and based on decision models to present variability to the customer and to instantiate a concrete application based on “framework models” describing the component-based product line.

While KobrA provides an alternative approach to the technical aspects of PuLSE, it does not provide further process support. Furthermore, there is no support for product specific development. FIDJI contributes to this point by defining a flexible instantiation process.

2.3.1.3 FIDJI

FIDJI is a flexible product derivation process [26], part of an overall model-driven SPL based development methodology [27]. The FIDJI product derivation process consists of writing model transformation, using a set of predefined transformation operations, which will reuse core model assets to build the product. This transformation is written by the product engineer and checked against instantiation constraints. Hence, the FIDJI product derivation process offers the flexibility required to support product specific requirements, support via transformation operations and control through instantiation constraints.

From a methodological perspective, product derivation at a given abstraction level can be seen as a reconciliation between an “ideal” model stemming from customer requirements and the existing architectural framework model. The reconciliation takes

place in writing an instantiation program that takes into account the “ideal” model and instantiation constraints in order to obtain the model of the actual SPL member.

2.3.1.4 FORM

In an approach called FORM [28], Kang et al. have extended the FODA approach [29] to consider application engineering. The idea of FORM is to analyse domain features and use the identified features to derive products (see Figure 2-2).

The FORM approach consists of the domain engineering and application engineering process. The FORM application engineering process consists of requirements analysis and feature selection, architecture model selection and application development. Product derivation begins with requirements analysis and tries to find a matching set of features. The selected features can be instantiated to a product configuration by following the dependencies and constraints documented in the feature model. The architecture model selection is facilitated through feature selection also. Application development finally consists of following the specifications and selecting pre-coded components, filling in skeletons, or instantiating parameterised templates [28].

In application engineering, the emphasis is on the analysis phase with the use of the developed features. However, few directions are defined to select the architectural model and develop the applications using the existing components.

FORM, from a software engineering perspective, is applying approved software engineering principles that support adaptability and reusability. FORM also covers domain and application engineering but does not define the complete process of software development nor does it define any tool support. To use FORM as a complete process would require a degree of customisation and integration with other methods and research for an appropriate tool chain supporting the integrated methods.

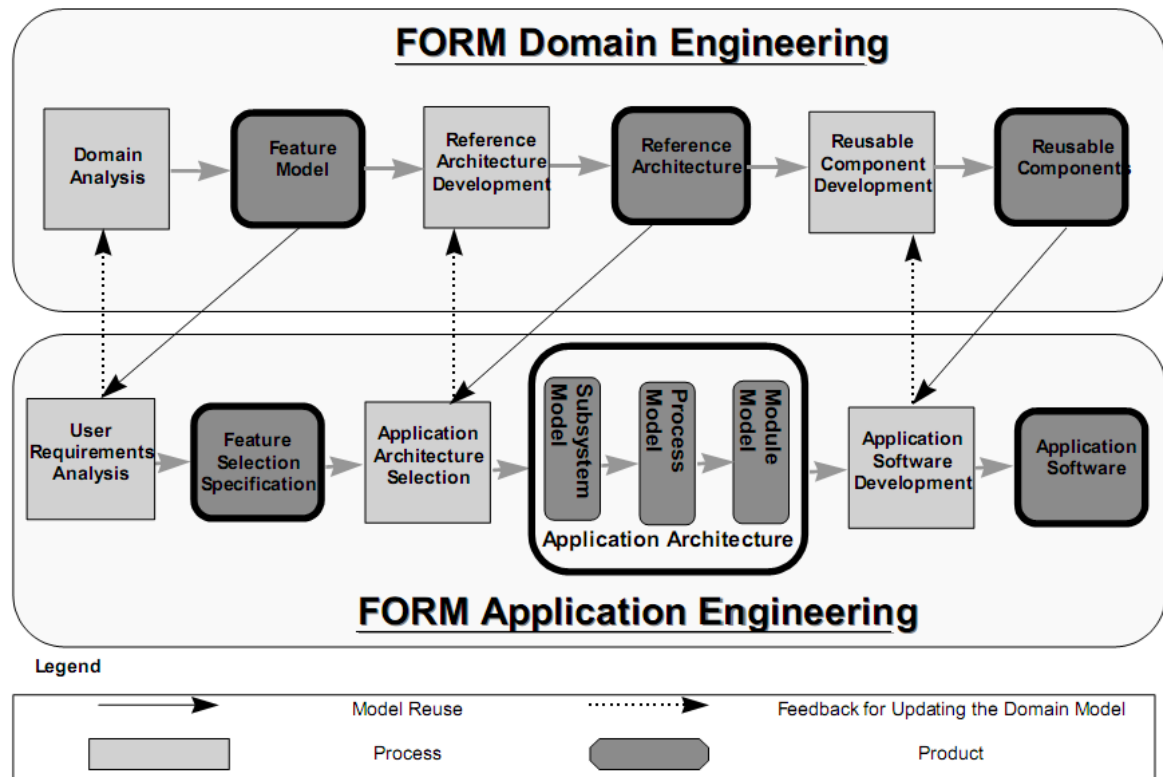


Figure 2-2 FORM Process [28]

2.3.1.5 FAST

The goal of the FAST [30] approach is to rapidly generate products from a product line using generation tools. The approach considers most of the facets of product line development.

The FAST application engineering process covers requirements elicitation, requirements analysis, product configuration and, additional development and testing. The process is described using state-based process modelling. The FAST approach is based on semi-automated product derivation where the product is developed through a combination of generation tools and manual product development. The FAST process begins by identifying variabilities and commonalities among potential product line members and then creating an application modelling language for specifying the individual products within that domain. The application modelling language is used as the basis for building a

generator. Product derivation is greatly simplified by describing the products in the application modelling language and generating the products.

If the product line requirements can be properly scoped and identified, FAST can develop individual products very quickly. However to enable automatic product derivation, system specifications must be precisely defined and specified. If there is a change or error in the requirements, specifications have to be modified and the translation needs to be altered. Additionally since requirements are specified using custom domain specification languages, other organisations may have difficulties in adopting them.

2.3.1.6 Dream

In [31] Kim et al. analyse the respective shortcomings of SPL engineering and model driven architecture (MDA). They propose an overview of a complete method integrating both approaches, which they call Dream. The Dream methodology adopts key activities of SPL and model transformation features of MDA. The Dream approach allows the semi-automatic development of products within the product line.

The Dream approach consists of two processes; framework engineering and application engineering. In framework engineering the generic architecture for the product line is designed. A decision model is developed that describes variation points, variants and mechanisms to tailor the generic architecture.

The application engineering process consists of application requirements analysis, framework instantiation, application specific construction and integration and deployment. In this phase applications that fall under the scope of the product line are derived. Derivation is performed using the decision model developed during framework engineering. A product architecture is instantiated based on the decisions in the decision model and according to the variants selected for a specific product. For requirements which were not originally envisaged by the framework engineering stage, application specific construction occurs. Application specific construction involves an integration phase where the application specific construction and the instantiated product architecture are integrated into a single product model.

The Dream approach allows the semi-automatic development of products within the product line. Kim et al. claim that it builds on earlier approaches by explicitly specifying the application construction and integration phases. However the approach does not give detailed information on the models used during the different phases of the process nor are the type of transformations required or applied discussed in detail. Additionally, there is little support for the derivation process other than a high level description of the activities required i.e. integrate the application specific construction and the instantiated product architecture. Furthermore, no guidance is provided on how the approach could be applied or tailored to an industrial setting.

2.3.1.7 SEI Product Line Practice Framework

The SEI Product Line Practice Framework (PLPF) [1] is built around what its authors term the “three essential activities” of SPL, namely Core Asset Development, Product Development, and Management. Core Asset Development is mainly concerned with defining the Product Line scope and producing the core assets from which the products will be built. Product Development consists in turning these core assets into products, and Management oversees both these activities at the project and organisational level. Figure 2-3 illustrates these three essential activities of SPL.

The PLP includes 29 practice areas grouped into three categories, namely practices related to Software Engineering, Technical Management and Organisational Management. Table 2-1 summarises the Practice Areas.

Clements and Northrop [1] define the term “practice area” as a body of work or a collection of activities that an organisation must master to successfully carry out the essential work related to an SPL. Northrop [17] further elaborates the effectiveness and importance of practice areas because they help in making the essential activities more achievable by dividing activities into smaller parts that can be easily traced.



Figure 2-3 The Three Activities of SPL [1]

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. The PLPF supports product line planning and management, rather than giving concrete instructions on implementing specific engineering tasks.

In addition to the PLPF, the SEI also supports the automation of product derivation. McGregor [32] describes a high-level framework of practices for deciding when to automate product derivation, how to choose the right technology, and how to plan and carry out the derivation process. According to the framework, *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 [20] propose detailed guidelines for creating, using, and evaluating such production plans.

Software Engineering Practice Areas	Technical Management Practice Areas	Organisational Management Practice Areas
Architecture Definition	Configuration Management	Building a Business Case
Architecture Evaluation	Make/Buy/Mine/Commission Analysis	Customer Interface Management
Component Development	Measurement and Tracking	Developing an Acquisition Strategy
Mining Existing Assets	Process Discipline	Funding
Requirements Engineering	Scoping	Launching and Institutionalizing
Software System Integration	Technical Planning	Market Analysis
Testing	Technical Risk Management	Operations
Understanding Relevant Domains	Tool Support	Organisational Planning
Using Externally Available Software		Organisational Risk Management
		Structuring the Organisation
		Technology Forecasting
		Training

Table 2-1 The SEI Product Line Practice Areas [1]

The framework defined by the SEI is a robust description of best practice involving important technical and non-technical aspects grouped in software product line practical areas. 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.

2.3.1.8 COVAMOF

Deelstra *et al.* [3] present a product derivation framework developed based on two industrial case studies. The framework consists of two phases: an initial and an iteration phase (see Figure 2-4). 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 authors derivation framework have been implemented in a research tool called COVAMOF-VS [33], a variability modelling framework which purports to solve the product derivation problems associated with dependencies.

The initial work by Deelstra *et al.* provides a framework of terminology and concepts for product derivation. The approach is very tool-focused and centred on the configuration of products. There is no support for the early phases of product derivation or product specific development and testing. The framework assumes ‘engineers’ perform all the work and does not specify responsibilities, or define role and task structures. The framework focuses on product configuration and is a high-level attempt at providing the methodological support that Deelstra *et al.* [3] and others [24, 32, 33] agree is required for product derivation.

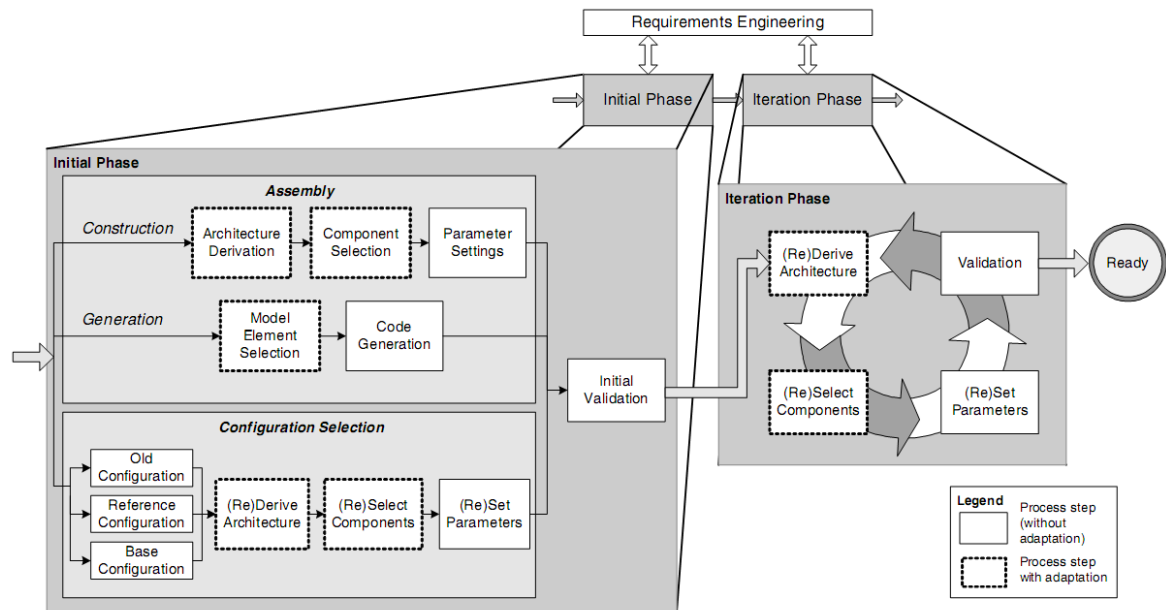


Figure 2-4 The ConIPF Derivation Process [3]

2.3.1.9 DOPLER^{UCON}

DOPLER^{UCON} [34] is a tool-supported approach for product configuration with capabilities for adapting and augmenting variability models to guide sales people and application engineers through product derivation (see Figure 2-5). DOPLER^{UCON} builds on existing decision-oriented approaches and aims to support domain experts as well as engineers through the provision of user centred tool support.

The approach assumes the creation of domain engineering models which describe the commonalities and the variability of relevant core assets such as requirements, features or architectural elements.

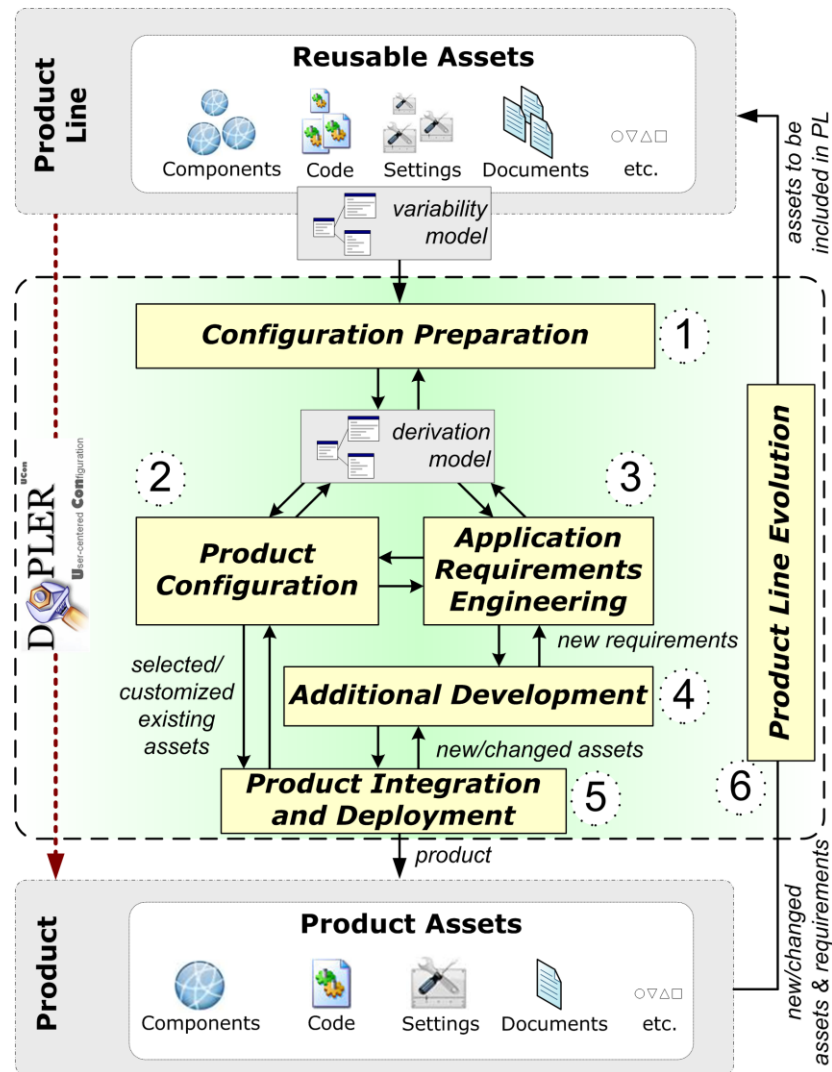


Figure 2-5 DOPLER^{UCON} Approach[34]

From a high-level point of view DOPLER^{UCON} comprises the following activities which need to be conducted in an iterative manner (see [34] for details):

In *Configuration Preparation* (see 1 in Figure 2-5) project managers prepare variability models for a concrete project/customer. They capture customer information and resolve variability based on high-level requirements known early on. They further define the roles and tasks of the people involved in product derivation. Additionally, domain experts model guidance on decisions 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 [6]. The output is a project-specific version of the original variability model called the derivation model.

Product Configuration (see 2 in Figure 2-5) starts with presenting decisions to sales people and engineers according to their roles and tasks defined in the derivation model. Sales people communicate with customers to elicit their detailed requirements and take decisions accordingly. Engineers perform more technical configuration based on sales peoples' decisions. The sub-activities of product configuration are: review available variability, communicate variability, take decisions and customise assets, and generate configurations. Product configuration is supported by the tool ConfigurationWizard [35]. The outputs are selected and customised assets.

Application Requirements Engineering (see 3 in Figure 2-5) aims at capturing, negotiating, and managing requirements that cannot be fulfilled by the product line. These will likely arise during product configuration. The sub-activities of application requirements engineering 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 [6].

During *Additional Development* (see 4 in Figure 2-5) product-specific requirements are addressed. Developers have to take into account the already existing assets and their relationships. New developments need to be tested. Activities like prototyping and unit testing are therefore typically involved. DOPLER^{UCon} does not define concrete sub-activities for additional development because it assumes this activity to be too domain-specific.

Product Integration and Deployment (see 5 in Figure 2-5) means integrating derived assets with new developments and preparing them for deployment. Again, the concrete steps involved differ from company to company. ConfigurationWizard can be extended with domain-specific tools, e.g., to enable generating build files or settings files.

In *Product Line Evolution* (see 6 in Figure 2-5) 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: analyse newly developed assets and analyse new requirements.

DOPLERPUCon is focused on providing user-centred tool support for product derivation. There is little support for the product derivation process within the approach. The defined process is too abstract to be of use. For instance, DOPLER^{UCON} does not define how to handle or negotiate unsatisfied customer requirements.

In Chapter Four, the DOPLER^{UCON} approach is discussed in more detail (See Section 4.4).

2.3.2 Conclusion

The methods evaluated have very different scope and emphasise different aspects of the derivation process. Some of them, like FIDJI (see Section 2.3.1.3), capture only a small part of the process while others, like PuLSE-I (see Section 2.3.1.1) are much broader. All of them (with the exception of the SEI PLPF) come with different amounts of prescription and tool support. Some describe a generic process rather vaguely and others are very close to practise and prescriptive in the definition of their process steps and documentation.

FORM was directly derived and developed from FODA by the original author and extends it to the software design phases. It applies software engineering principles but gives only a few suggestions regarding the implementation. KobrA (see Section 2.3.1.2) has been derived as an instantiation of PuLSE with the intention of rapid use. It makes use of UML diagrams and templates for the defined development items and is component-oriented. However it does not provide the required process detail. Other approaches like QADA [36] (not discussed) introduce quality assurance, and so each of them has its own strengths. What most of them have in common is that they are quite abstract, and lack process specifics.

From the study of these approaches, the key areas for concern are:

- Defining a flow of artefacts

- Defining roles and responsibilities
- Providing process support
- Ensuring adaptability

2.3.2.1 Defining a Flow of Artefacts

Product development within a SPL requires a high degree of coordination and communication. The product team design and implement customer-specific assets based on customer requirements. The platform team receives the required extensions to the existing platform to facilitate the new customer requirements. Frequently both customer-specific and platform development occur in parallel. There is a need for awareness of the artefacts and stakeholders involved in product derivation.

Many of the existing approaches [22] do not explicitly define the flow of artefacts within product derivation. In Dream [31] little information is provided on the usage of model artefacts during the different phases of the approach. DOPLER^{UCon} does provide some description of artefact flow however they are specified level of granularity is too abstract to give useful guidance for practitioners. A good starting point could be PuLSE-I [24], as it names the development items in a descriptive manner. However, it does not provide detailed description of artefacts usage within the process,

2.3.2.2 Defining Roles and Responsibilities

Diverse people with diverse tasks, roles, and responsibilities are involved in product derivation. Current approaches do not provide sufficient support for the managing of roles and assignment of roles to tasks and artefacts within the product derivation process. DOPLER^{UCON} acknowledges the need for defining roles and responsibilities during a derivation project but no guidance is provided on how this should occur. FAST [30] assigns activities to one of the three defined derivation roles but this is done at a very high level and is unusable in any practical setting.

2.3.2.3 Providing Process Support

The arguments for defining process should be a familiar one. A well-defined process can be managed, measured and observed, and therefore improved. An emphasis on processes helps software development to become more like engineering, with predictable time and effort constraints, and less like art [37]. Boeckle [38] finds that transforming an organisation to create products as members of a product line required installing corresponding processes, and organisational structure and methods.

Clements and Northrop explain the fundamental need for documented processes within SPL [1] as follows:

An essential aspect of software engineering is the discipline it requires for a group of people to work together cooperatively to solve a common problem. Defined processes set the bounds for each person's roles and responsibilities so that the collaboration is a successful and efficient one. ... If software engineering is about a group of people working together to solve a problem cooperatively, then product line software engineering requires cooperation in spades. ... In fact, organisations that do not have a strong process culture will find deploying a successful product line a perilous proposition.

Furthermore, key authors in the area have called for process support within product derivation [3, 6, 22, 32].

Despite this there is relatively modest support for the derivation process in existing approaches. In Dream little information is provided on the models employed in the different phases of the Dream process. While Kobra provides a detailed analysis model and some methodological guidelines it does not provide any detailed description. Deelstra et al. [3] provide a framework of terminology and concepts, however it is presented at a high level.

A good starting point could be PuLSE-I [24], as it names and briefly describes the development items, which roles are responsible for which tasks and which artefacts are consumed or produced by a certain activity. Furthermore, many dependencies exist to other

parts of the PuLSE process which means that the approach is not very applicable outside the scope of a PuLSE applied product line.

All in all, there are a number of defined processes but those described are too abstract or not explicit in their description of the process. Furthermore, they are frequently too specialised to a specific development technique to serve as a general solution.

2.3.2.4 Ensuring Adaptability

All of the various approaches presented have been developed with different goals, for different purposes, and in different domains. Some are rather focused on the early phases of derivation [39, 40], some are intended to support specific aspects, such as variability within product derivation [33], and others focus on tool support [6].

All the different people involved in product derivation are supported in their tasks by these different approaches and tools. Because of the difficulty of integrating these different approaches and tools, product derivation can become an error-prone and tedious task. Furthermore every domain and organisation has its specific environmental characteristics. It is challenging to integrate an approach with an existing environment and process. The difficulty for standardization is that every process still requires adaptation and customisation for each company. Otherwise it will not be possible to apply them in practice.

Organisations need a product derivation approach that can be easily integrated with different existing approaches and environments. To support this, research into new approaches must ensure that they are adaptable to allow integration with existing processes and environments.

2.4 Software Process Research

A software process is the collection of related tasks leading to a product. A process is important to define how an organisation should perform its activities, and how people work and interact, in order to ensure efficiency, reproducibility, and homogeneity [41].

In the previous section, the lack of process support for product derivation was identified. In this section, the subject of Software Process Improvement is discussed. The evolution of software process models over time is shown, beginning with a discussion on the waterfall model [7], then the spiral model [42] and, finally Agile methods [43, 44]. How the construction and use of process reference model can streamline the design of (particular) situation specific models is discussed in Section 2.4.2. This provides additional rationale for the establishment of Pro-PD. Later in the thesis the original Pro-PD, developed in the context of a waterfall process, is adapted to be employed in an agile context. This adaptation is achieved by a process called Use by Specialisation and this is defined in Section 2.4.2.1.1. Finally, the state of the art in SPL process assessment is reviewed in Section 2.4.3.

2.4.1 Software Process Models

The software process is the technical and management framework established for applying tools, methods, and people to the software task. In short, a defined process provides the software professionals with the framework they need to do a consistently professional job [45].

The classic software life cycle is often represented as a simple prescriptive waterfall model (see Section 2.4.1.1). In the waterfall model, software development proceeds through an orderly sequence of transitions from one phase to the next in order. The spiral model (see Section 2.4.1.2) represents a risk driven approach to software development. Agile methods (see Section 2.4.1.3) refers to a group of development methodologies based on iterative development, where requirements and solutions evolve over time.

These process models have developed over the decades and demonstrate the evolution of process models from traditional (waterfall model) to contemporary (Agile methods). The described process models also vary in terms of their ability to support evolving requirements which are identified late in product development. In later chapters, the waterfall model and Agile methods will be used as the basis for Pro-PD instantiations.

The waterfall model and Agile methods are contrasting approaches in terms of both their maturity and approach to software development,

2.4.1.1 The Waterfall Model

The waterfall model was first proposed by Royce [7] in 1970. The model contains distinct development stages that are in strict sequential order, which means an activity begins only after the preceding activity is finished. Additionally, each phase represents a milestone to indicate the end of a particular phase. The major stages of the model are shown in the following figure.

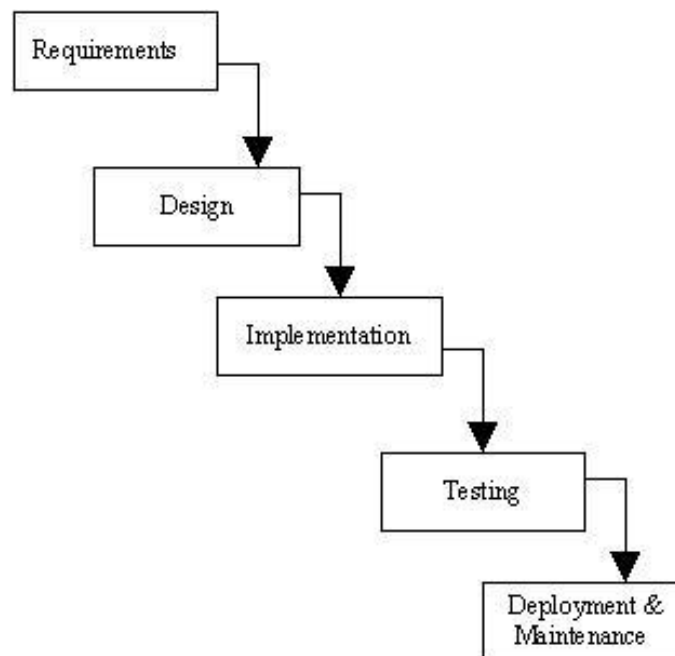


Figure 2-6 Waterfall Model [7]

In an ideal world, changing back to a prerequisite activity is not encouraged in the Waterfall Model. However, in reality, projects rarely follow the sequential flow: requirement changes can happen. Another disadvantage of the Waterfall Model is that it only describes high level activities occurring during the software life cycle.

2.4.1.2 The Spiral Model

The Spiral model, originally developed by Boehm [42], is an evolutionary development process model (see Figure 2-7) which explicitly takes risk into consideration when performing software development activities. By iterating through the software development activities, increasingly versions of software are released. Boehm describes the model “a risk-driven process model” and “is used to guide multi-stakeholder concurrent engineering of software intensive systems”. It emphasises two distinguishing features. The model takes the cyclic approach, and each cycle can be connected to a phase within the Waterfall model. The initial round is to analyse the system feasibility; subsequent passes round the spiral are to develop the requirement specification and then progressively more sophisticated versions of the software.

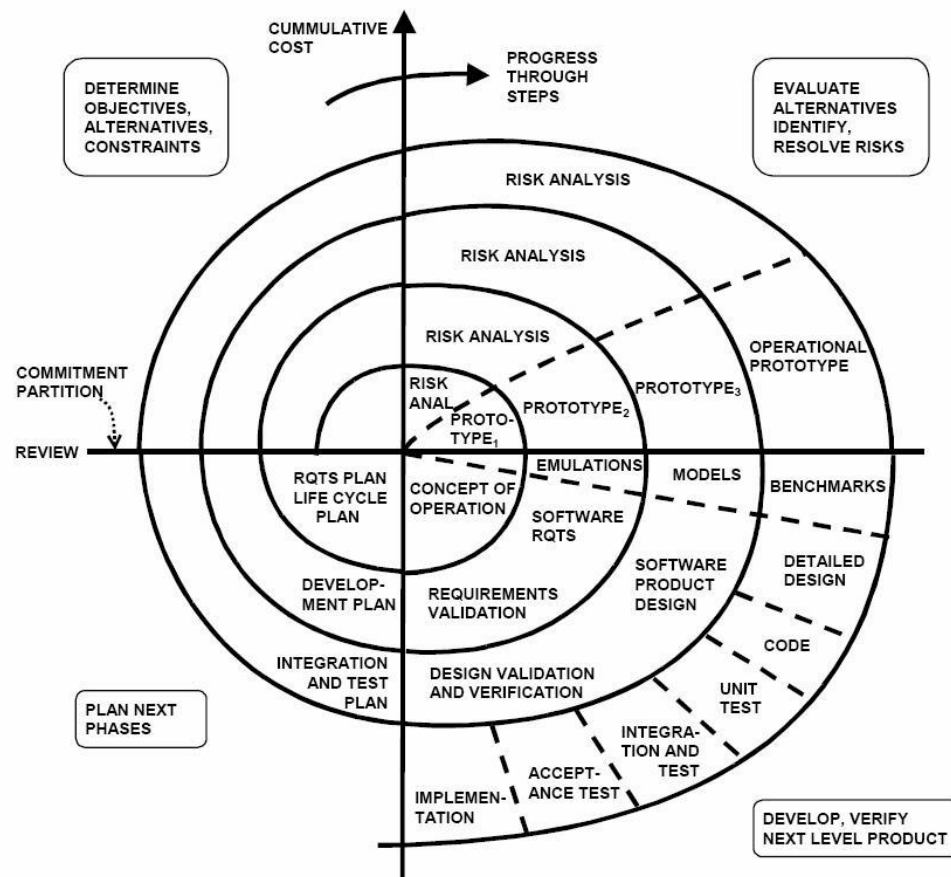


Figure 2-7 Spiral Model [42]

2.4.1.3 Agile Methods

Agile Methods have recently gained popularity among large numbers of companies as a mechanism for reducing costs and increasing ability to handle change in dynamic market conditions. Researchers and practitioners have proposed several software development approaches based on the principles of the Agile manifesto [46]. The Agile manifesto defines four basic core values:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

In addition to these four values, the Agile manifesto defines a set of principles that form guidelines for development. The Agile manifesto provides a foundation for all of the Agile methods including: eXtreme Programming (XP) [47] and Scrum [44].

XP evolved from the problems caused by the long development cycles of traditional development models [43]. The individual practices of XP are not new however those practices have been collected and lined up to function with each other in a novel way. The term ‘extreme’ comes from taking these commonsense principles and practices to extreme levels [48].

Scrum provides a project management framework that focuses development into 30-day Sprint cycles in which a specified set of Backlog features are delivered [44]. The core practice in Scrum is the use of daily 15-minute team meetings for coordination and integration. Scrum does not define any specific software development techniques. Scrum concentrates on how team members should function in order to produce good quality code and maintain flexibility in a changing environment.

Although XP and Scrum are based on a common guideline defined by the Agile manifesto, they vary in focus and presentation. XP emphasises technical elements of the development lifecycle, while Scrum concentrates on the project management. One of the central principles of the Agile approach is the focus on people. People, coupled with

effectiveness and manoeuvrability, are considered the primary drivers of a project's success [49].

Agile approaches have their sceptics, authors such as Rakitin view Agile as a step backwards from traditional engineering practices, a disorderly “attempt to legitimize the hacker process” [50]. Where processes such as Waterfall and Spiral stress lengthy upfront planning phases and extensive documentation, Agile approaches tend to shift these priorities elsewhere. XP, for example, holds brief iteration planning meetings in the Planning Game to prioritize and select requirements, but generally leaves the system design to evolve over iterations through refactoring, resulting in hacking [50].

This accusation of Agile approaches being no more than hacking is refuted by XP creator, Kent Beck. Beck states: “Refactoring, design patterns, comprehensive unit testing, pair programming—these are not the tools of hackers. These are the tools of developers who are exploring new ways to meet the difficult goals of rapid product delivery, low defect levels, and flexibility” [51].

2.4.2 Process Reference Models

The main objective of a reference model is to streamline the design of (particular) models by providing a generic solution that can serve as a template for defining a model for a particular enterprise [52]. It can thus serve as a tool for simplifying process problem solving, and enables users to have a degree of confidence that the process begins on a solid foundation.

A reference model is (usually) created to represent already existing processes, e.g. by observing current practice in industry and academia, and thus serves as a blueprint for others. Reference models have to be universally applicable for a certain situation, and serve as a recommendation on how to solve or organise that situation.

The possible benefits of reference models are to raise the quality of the models produced, a cost, time and risk reduction, the reuse of knowledge, and access to industry best practices [53, 54]. Reference models accelerate the modelling and configuration process by providing a repository of potentially relevant models [55].

Although there is a lack of empirical evidence to support these claims, it is a software engineering assumption, that when you reuse, instead of developing from scratch, a positive influence can be observed on the time-to market, product quality and development cost of the intended product. It is based on this assumption that reference models are believed to bring benefit.

2.4.2.1 Reference Model Use

Once a reference model has been constructed and is available for use, it needs to be instantiated for the specific situation at hand. Different techniques are used to instantiate reference models. Braun [56] suggests these instantiation techniques are either:

1. Rule-based: This set of techniques is based on specific constructs in the reference model's grammar. These constructs provide formal means for adapting the reference model.
2. Formative: Formative techniques are based on creative activities of the person who adapts the reference model. In contrast to rule-based techniques, formative techniques do not only remove content from the reference model but also add additional knowledge.

Both of these techniques can be distinguished by their degree of formality. While in a strict instantiation approach the reference model is derived using formal notions within the reference model, in a loose instantiation approach the process model is derived by using creative techniques. In the first case the relation between both models is formally defined while in the second case the relation exists only informally.

Based on the work of [52] some categories of formative instantiation for a process reference model to a company-specific model are described.

In *Use by Adoption* the model is intended to be reused as is, without modification. The user of the reference model does not change the model to fit its own organisation, but adapts the organisation to fit the software in which these reference models are implemented. However, this type of process reuse is unlikely as it is expected that

organisations prefer to adapt (parts of) the model to their own situation instead of adapting the organisation to comply with the model

In *Use by Assembly*, the reference model is not to be taken as a whole, but is composed of “building blocks” which can be assembled to provide the best solution. These models provide flexibility by the ability to select from a variety of building blocks, but there are no mechanisms to make changes to these blocks.

Reuse by Configuration is targeted at reference models attached to enterprise systems, and based on the reuse by adoption approach. The difference with the reuse by adoption approach is that the models contain explicit variations; from the models it is clear where different options are available, and dependencies are explicitly defined.

In *Use by Specialisation* the defined process needs to be specialised and a lower level frequently needs to be constructed in order to create a working company specific model. So, in contrast with the previous approaches, the model is not complete yet. The advantage is that it provides a high level of abstraction, and it serves as a basis without providing a solution.

In Chapter Six Pro-PD will be instantiated using the formative *Use by Specialisation* technique.

2.4.2.1.1 Use by Specialisation

In *Use by Specialisation* [56], general aspects of a domain that are captured within a reference model allow the plugging in of process aspects considered special requirements for a domain. The goal of specialising a reference model is to add additional process know-how to fit with the specific process needs, in areas where the coverage of the reference model is deemed insufficient for the project.

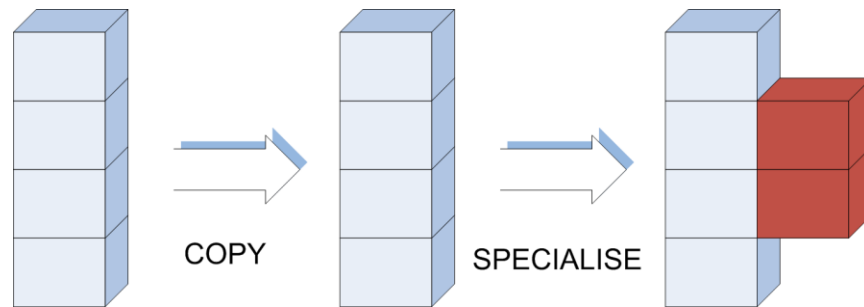


Figure 2-8 Use by Specialisation

This process instance is fine-tuned to fit the needs of the project (see Figure 2-8). If the reference model is an accurate representation of its target domain then the task of instantiating the model will be fairly light. The work of instantiating the process through specialisation includes:

- Making a copy of the original reference model
- Defining the process instance specific content required to fit the characteristics of the process type, organisation or domain for which the process is applied
- Customise the lifecycle model to fit the characteristics of the project. This involves amending the process content including activities, phases, roles or tasks as is required and defining the main goals of the process phases under this particular process instance

2.4.3 Software Product Line Assessment Methods

To support software process development and improvement, a variety of process assessment methods have been proposed. However the majority of existing assessment methods such as the Capability Maturity Model Integration (CMMI) [57] mostly deal with project-type processes, although they do include operational processes such as Process Management. Since SPLs can have quite a long life-cycle it is normal that the processes and the organisational structures associated with them are more permanent in nature, leading to a fundamental paradigm shift compared to a project-based approach.

Furthermore, current software process assessment approaches such as the CMMI do not address product line engineering and are more concerned with the issues

surrounding project based single system development. As a result, they are not capable of assessing the product line maturity of an organisation or indeed its ability to successfully derive products from the product line. SPL extensions to the existing assessment approaches have been developed [58], especially for the CMMI, but these are not yet generally known and accepted.

Van der Linden proposes the Families Evaluation Framework [59]. The framework creates the foundation of a systematic and comprehensive strategy for the software product line process evaluation. The framework highlights four potential dimensions: business, architecture, process, and organisation as a set of four variables to describe the maturity of the software product line process. This initial framework further defines various maturity scales for individual dimensions of business, architecture, process, and organisation. The framework proposes a conceptual layout for software product line process assessment. This framework can be regarded as an initial attempt to develop a comprehensive strategy for software product line process maturity assessment.

An assessment method for product line capability exists called the Product Line Technical Probe (PLTP) [60]. The PLTP is a method for examining an organisations readiness or ability to succeed with a software product line approach. The PLTP is designed to be used within an organisation and is highly interactive. Furthermore, the PLTP is not available in the public domain and anyone interested in having their organisation assessed must request a technical probe from the SEI and pay for their consulting services.

2.5 Conclusion

Product derivation is a critical part of SPL and central to ensuring return on investment for the product line effort. Existing approaches have very different scope and emphasise different aspects of the derivation process. Furthermore, they are frequently too specialised to a specific development technique to serve as a general solution. Accordingly there is a strong need for a structured approach to product derivation which defines activities, tasks, roles, inputs and outputs of each step in a structured and systematic way.

The literature review in this chapter has led to a number of research objectives and these will be discussed in the following chapter, along with the research methodology adopted to meet these objectives.

Chapter Three: Research Methodology

3.1 Introduction

In this chapter the research approach adopted in this study is discussed. Section 3.2 introduces the research objectives. Philosophical and epistemological assumptions are discussed in Section 3.3. The research design is presented including selection of research methods and detailed description of how these methods were executed in Section 3.4. In Section 3.5 the research validity is discussed. The researcher looks at how the research objectives were met in Section 3.6. Finally, the limitations of the research are discussed in Section 3.7.

3.2 Research Objectives

The definition of a suitable objective is the most critical stage of any research project. Jenkins [61] stated that it should be clear, unambiguous and enable the researcher to select an appropriate strategy.

In Chapter Two gaps within current product derivation approaches are identified. Existing approaches do not define activities, tasks, roles, inputs and outputs of each step in a structured way (see Section 2.3). Thus the primary objective of the work described in this thesis can be stated as:

To define a systematic process which will provide a structured approach to the derivation of products from a software product line, based on a set of tasks, roles and work products, and to demonstrate its adaptability to different process models.

From the primary objective three sub-objectives can be defined.

1. To define a structured process model for product derivation
2. To demonstrate the adaptability of the process model
3. To add to the established knowledge on product derivation

3.2.1 Objective One – To Define a Structured Process Model for Product Derivation

The first objective of this research is to develop a process model that provides a structured approach to product derivation. The review of product derivation literature revealed a need for process guidance and support among key authors in the area [3, 6, 24, 32].

Despite this, there has been little work dedicated to the overall product derivation process. Rabiser et al. [6] claim that “guidance and support are needed to increase efficiency and to deal with complexity of product derivation”. As Deelstra et al. [3] states there “is a lack of methodological support for application engineering and, consequently, organisations fail to exploit the full benefits of software product families.” As a result current approaches fail to provide a holistic view of product derivation leaving organisations with no centralised starting point for defining an approach to product derivation. This results in ad-hoc solutions. Meeting this objective will be a step towards providing this process support that is required within product derivation (see 2.3.2.3).

Furthermore, through the definition of a structured process for product derivation this research focuses on the essential tasks, roles and work artefacts used in the derivation process, satisfying the need for a defined flow of artefacts (see 2.3.2.1) and, defined roles and responsibilities (see 2.3.2.2) within product derivation.

To meet this objective, the research seeks to collate the disparate information gathered from literature, industry practice, research projects and documented best practice to create a process reference model for product derivation (Pro-PD). Pro-PD aims to provide guidance to SPL organisations on what product derivation activities, tasks, work products and roles are required during a product derivation project.

3.2.2 Objective Two - To Demonstrate the Adaptability of the Process Model

Pro-PD should be adaptable; allowing organisations to manipulate the approach to suit their specific circumstances. In Chapter Six Pro-PD will be adapted to fit Agile characteristics in order to demonstrate the adaptability of the approach. In meeting this objective, the research will be meeting the need for adaptable approaches (see Section 2.3.2.4) to product derivation.

3.2.3 Objective Three - To add to the Established Knowledge on Product Derivation

The final objective is to add to the established knowledge in product derivation. The research proposes an Agile approach to product derivation (A-Pro-PD) and identifies Agile practices that could be adopted within an Agile product derivation project (see Chapter Six). The research has added to established knowledge through publication of the research (see Appendix H).

3.2.4 Finally

It is important to note that while the above sub-objectives are defined linearly, they do not carry equal significance. Achieving objective one, through the development of Pro-PD, is the main focus of this research. Objectives two and three are supporting sub-objectives.

3.3 Research Philosophy

Klein and Myers [62] recommend that “the intellectual basis for the research should be as transparent as possible to the reader”. Therefore, in this section, the philosophical and epistemological debates underlying this research are discussed. The main debates considered are positivism versus interpretivism, quantitative versus qualitative, inductive versus deductive, and constructive research.

3.3.1 Positivism vs. Interpretivism

Positivist research is seen as a progression and adoption of the natural sciences. Positivist research is typically associated with the testing of a predefined theory and therefore concerns itself with control, reduction and explanation. To achieve control the research is performed in a laboratory type setting where the researcher is primarily an observer of events [63].

The research in this study is not suited to the positivist paradigm because it is not feasible to operate the research in the controlled and managed laboratory type environment

required by the positivist ideal. The processes of industrial product derivation organisations are fluid due to the nature of their activities which are an integral part of daily business. Also prior to the research there were no suitable and quantifiable measure of variables and the research is not concerned with testing a hypothesis.

The main goal of interpretive research is “to describe, interpret, analyse, and understand the social world from the participants’ perspective” [64]. An important aspect of interpretive research is the subject matter must be “set in its social and historical context so the reader can see how the current situation emerged” [62]. The research in this thesis utilises an industrial context during the case study research. The results therefore must be observed and interpreted in this context. Findings of the research are applicable to others who also operate in this environment. Given that the research is set in a defined context with which the results may be observed and interpreted, the interpretive research paradigm is appropriate.

3.3.2 Quantitative vs. Qualitative

The contentious issues surrounding the debate between the positivist and interpretivism dichotomies are also prevalent in the debate between quantitative and qualitative research. Quantitative techniques are based around formality, mathematics and statistics. They have their roots firmly in the positivist tradition. Qualitative research tends to be associated with the interpretivist mindset.

A qualitative study is an appropriate way to research an area where few previous studies have been carried out [65, 66] and when it is the process that is being studied rather than the product [67]. Product derivation is an example of such an emergent topic of research. Furthermore, the researcher is also looking at the practices and processes involved in the derivation of products from a SPL rather than any end product of the derivation process. A qualitative approach is therefore considered to be most appropriate for this study.

3.3.3 Explanatory vs. Exploratory

The goal of explanatory research is to develop “statements which make something intelligible about why things are the way they are” [68]. A focus is usually placed on testing hypotheses and a scientific, positivist stance is typically adopted [66, 69].

Exploratory research promotes understanding and is suitable for “new fields of study where little work has been done, few definitive hypotheses exist, and little is known about the nature of the phenomenon” [66, 70]. This study is exploratory due to (1) the lack of theoretically grounded literature on product derivation and (2) the research is laying the basic foundation for a defined process approach to product derivation. Yin [66] dispels the notion that exploratory studies “fumble in the dark” stating that “rationale and direction should underlie” such studies.

3.3.4 Inductive vs. Deductive

Deductive reasoning is the belief structure behind much of mathematics. Deductive researchers believe their “justifications to be conclusive, in the special sense that it is impossible, on pain of self-contradiction, for the beliefs which are our reasons to be true and the conclusions that we draw from them to be false” [71].

Inductive reasoning occurs “when we take our reasons to be sufficient to justify our conclusion without conclusiveness”, or “we have some but not yet sufficient reason for the conclusion, hoping that further reasons may yet be found” [71]. The key distinction between induction and deduction is that inductive reasoning does not involve such strong relationships between reasons and conclusions, as there “is an inferential jump beyond the evidence presented” [72].

Inductive methods such as grounded theory [73] or case study research [66, 74] seek to gather information and build theories from these methods. That is they argue from the particular to the general. This research is investigating industry and academic approaches to product derivation and is building a general theory. Therefore this research is inductive research.

3.3.5 Constructive

The study itself can be classified as applied research and more specifically, as constructive research. Järvinen [75] defines constructive research as typically involving the building of a new innovation based on existing (research) knowledge and new technical or organisational advancements. Furthermore, Järvinen suggests that constructive research also involves an evaluation of the innovation. According to Järvinen it is possible to accept a prototype or even a plan as a research outcome as an alternative for the final product in constructive research.

3.4 Research Design

The literature review in Chapter Two highlighted among other issues the lack of any defined methodological process for product derivation. It was observed that product derivation approaches to date were centred on technological solutions that supported or partly automated the product derivation process. Evidence of product derivation approaches to date is based largely on anecdotal rather than scientific evidence.

While such a large gap between research and practice presents many opportunities, it also brings many potential pitfalls. Without a mature body of theoretically founded knowledge to use as a guiding light, systematic research becomes somewhat more difficult. Cooper and Schindler [72] recommend that in instances where the study is broad and exploratory and where limited research currently exists, it is vital that the researcher parses the research project into a set of clearly defined steps.

To define the research steps, the research methods must be selected. No single research method however is universally applicable. According to Gill & Johnson [76] “all research approaches may have something to offer and there is no independent form of evaluating different research strategies in absolute terms”. There is a considerable range of research methods available [77], all of which have distinct strengths and weaknesses.

To compensate for these weaknesses, Franz et al. [78] “recommend an ample dose of complementary qualitative research” or multi-method research design. According to Morse [79] multi-method design is “the conduct of two or more research methods, each

conducted rigorously and complete in itself, in one project.” A multi-method approach can allow for the counter-balancing of the strengths and weaknesses. By triangulating between methods and data, more plausible interpretations can emerge.

According to Woods et al. [80] multi-method research may be conducted from a complementary or evolutionary perspective. In this research, an evolutionary approach was followed. An evolutionary approach is used when there is little research conducted on a particular phenomenon. Rather than investigating an effect through two or more different empirical methods, seeking confirmatory power between them, an initial exploratory study gathering qualitative data is undertaken. At this early stage, the initial study is designed to explore a wide range of topics in the area under investigation. The collected data is then analysed, and the important findings from the initial study are refined and used in the following study. This process is then repeated, usually using a different research method.

3.4.1 Overview of Research Design

In establishing the research design, it has to be considered that the option of spending extended periods in the field was not available. Ethnography and action research were both automatically ruled out as potential research methods because of time, resource, and access constraints. Field and laboratory experiments were also considered inappropriate, as their implicit assumptions are incongruent with the epistemological beliefs underpinning this research.

The research design adopted in this study was influenced by Ahlemann et al. [81] which focused on empirically grounded and valid process model construction.

The first stage of the research design is the core construction stage. This entailed a literature review from which a preliminary version of the model was constructed, followed by a series of expert opinion workshops. According to Ahlemann et al. [81] if sufficient existing research results are available, a literature-based verification should be undertaken first. Preparation for this initial empirical inquiry can commence as soon as the first version of the frame of reference is available [81]. The initial inquiry is carried out to capture the experts' domain knowledge as a basis for the first cycle of the reference model

construction. Both Rosemann and Schütte [82] and Schlagheck [83] emphasise the participation of users in the core construction stage, as the users are the subject-matter experts of the problem domain.

The core construction stage needs to be followed by an inductive, empirical validation [81]. Furthermore, as previously mentioned, the researcher identified the research as an interpretive study (see Section 3.3.1). Walsham [84] stated that “case studies provide the main vehicle for research in the interpretive tradition”. The case study approach has always been one of the most popular research strategies [64]. It is a powerful and flexible technique, considered suitable for exploratory research both prospectively and retrospectively [85]. 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 [66]. It is often considered to be the optimal approach for researching practice based problems, where the aim is to represent the case authentically “in its own terms” [86].

Therefore, the second and third stage of the research design is an industrial case study and an academic comparative analysis. Here, Pro-PD is mapped and compared to real world structures from both industry and academic approaches to product derivation. The industrial case study (second stage of research) is with Robert Bosch GmbH. The academic comparative analysis (third stage of research) is a research collaboration with JKU (Johannes Kepler University Linz, Austria). JKU developed the DOPLER^{UCon} (**D**ecision-**O**riented **P**roduct **L**ine **E**ngineering for effective **R**euse: **U**ser-centered **C**onfiguration) approach which was driven by industry needs with the goal to define a user-centred, tool-supported product derivation approach [6]. The approach was mainly influenced by a research-industry collaboration with Siemens VAI Metals Technologies

The fourth and final stage of the research was to evaluate the solution. Evaluation was done in two steps:

- (1) Systematically analysed the support for Pro-PD facets in prominent existing approaches and
- (2) An inter-model relationship evaluation with the SEI Software Product Line Practice Framework.

According to Ahlemann et al. [81] such standards should be considered during the construction process, and process models that are compatible with such standards and norms are regarded as high quality.

Figure 3-1 presents a summary of the research design. In an analogy with systems engineering, the overall construction process is based on a cyclic structure to allow for model corrections on preceding construction stages via feedback-loops. Although the stages are dealt with sequentially, they contain cyclic sub-processes.

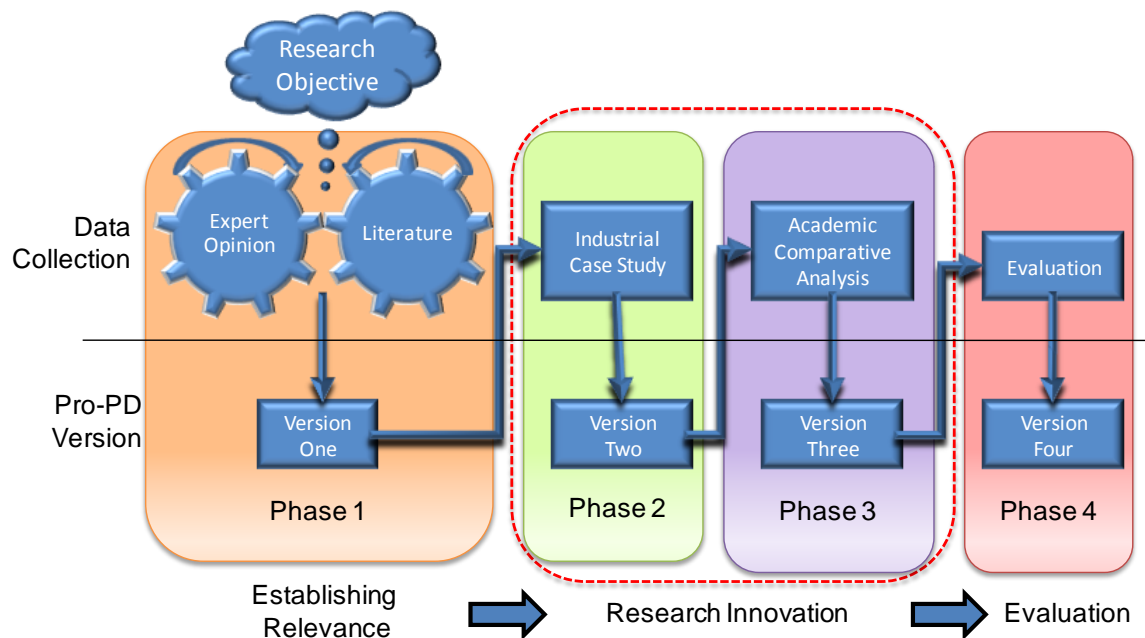


Figure 3-1 Overview of Research Design

While the first stage was designed to set up and construct Pro-PD, it also established its relevancy. Stages two and three stabilised and refined Pro-PD, this is the research innovation. The final stage evaluated the model.

The research design is compatible with common suggestions for qualitative research designs in process models [87]. Other guidelines on design in systems analysis and social sciences are described by Krallmann et al. [88] and Patton [89].

In the following sub-sections, the different stages of the research design are discussed in more detail. Firstly, the literature review is addressed. Secondly the iterative

expert opinion workshops are described. Thirdly the industrial case study is presented followed by the academic comparative analysis. Finally the systematic evaluation of Pro-PD is described.

3.4.2 Stage One – Literature Review and Expert Opinion Workshops

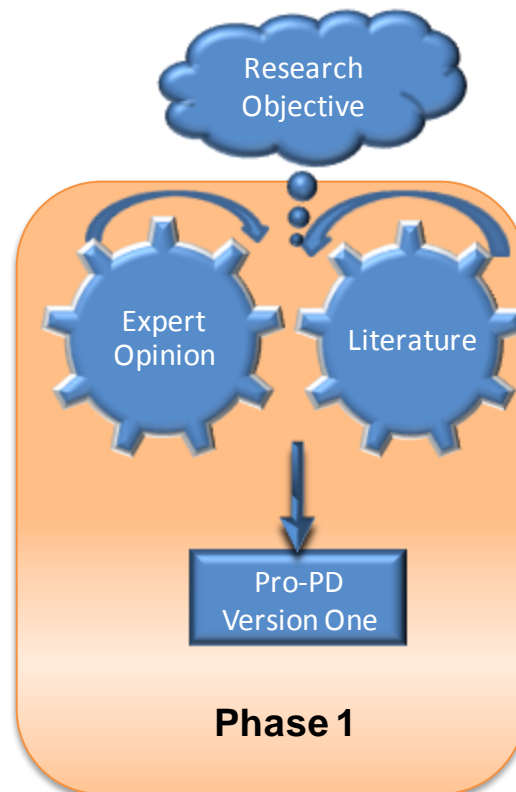


Figure 3-2 Stage One

3.4.2.1 Literature Review

The preparatory stage of this research was conducted as a review of SPL literature focusing in particular on product derivation. As suggested by Cooper [90], the literature review is aimed at defining the research topic's current state of knowledge. More specifically the research aimed to identify the fundamental practices of product derivation, to study the existing product derivation approaches as well as to chart the available empirical evidence on the topic – scientific as well as anecdotal.

According to Cooper [90], the cumulative nature of science requires trustworthy accounts of past research to form a necessary condition for orderly knowledge building. In this research, the literature review contributed to determining the focus of the incipient research in product derivation, and developing insightful research questions on the topic, as suggested by Yin [66].

Process model construction should always begin with an analysis of current domain knowledge. A first analysis of the problem domain is achieved by constructing a frame of reference [81]. This frame of reference represents a high-level perspective on the problem domain, and can be used for structuring during expert analysis. In Section 4.2.2.1, the construction process for a high level framework skeleton for Pro-PD from the product derivation literature is detailed.

While the research did not conduct a full Systematic Literature Review (SLR) as proposed by Kitchenham [91], an SLR protocol was developed and used during the latter stages of Stage One of the research. The SLR protocol ensured that the research had considering all key literature in the area. The developed SLR protocol can be seen in Appendix F.

The output of the literature review was Pro-PD Version One. In section 4.1, how the initial version of Pro-PD was constructed using sources from the literature is described. This version is validated theoretically through the use of existing literature from the SPL field and the feedback of experts. This is described in further detail in Chapter Four.

3.4.2.2 Expert Opinion Workshop Series

Forming a common perception of reality among observers of a system requires them to exchange and discuss the results and through constructive criticism. Then through purpose-means discussions they are able to develop complex suggestions for human action [81]. The research approach in this study uses expert opinion workshops to form this type of consensus. In expert opinion research, a declaration is considered to be truthful if each member of the group of experts grants their acceptance [92]. Applying this concept to the research design of this study means that all propositions contained in the model are to be

scrutinized with regard to their acceptance by a group of subject-matter experts. With this in mind it must be clarified which persons can be considered subject-matter experts.

Rosemann and Schütte [82] detail recommendations on the people that should be involved as experts. According to this approach, people should be selected based on their ability to adopt a specific perspective that corresponds to one of the intended research uses. In effect all perspectives should be sufficiently represented by the group of users involved [82, 93].

The research is designed for use by both industry and academia. It is investigating process support for the derivation of products from a SPL. Thus the identified expert participants were:

- Two academic SPL experts
- An industrial SPL expert
- An SPI expert
- The researcher of this study

According to Ahlemann et al. [81] in order for experts to be able to discourse on a model the inter-subjective comprehensibility of these propositions first needs to be guaranteed. This is achieved in three ways: First, the reference model documentation must be based on a common linguistic system, as the exchange of linguistic artefacts ultimately presents the foundation for every validation effort [92]. In the research this was achieved by the adoption of a simple model notation (see Table 3-1) during the early stages of development which accompanied a description of the latest version of the framework. The documented framework was sent to the participants in advance.





Notation	Description
	Artefact
	Process or activity
	Decision
	Stakeholder

Table 3-1 Model Notation

Secondly, the explication of the construction process on the grounds of which the reference framework was obtained contributes to inter-subjective comprehension. The researcher organised a series of iterative workshops over a four months period with participants meeting twice a month. A minimum of three of the five participants were present at each workshop. Prior to each workshop the framework was sent to each participant allowing them to familiarise themselves with the version being analysed. During each workshop an interactive presentation on the framework was given by the researcher. Issues that were raised or requiring clarification were discussed. The discussion was facilitated through use of a whiteboard (see Figure 3-3) where clarification and agreement was sought on varying issues.

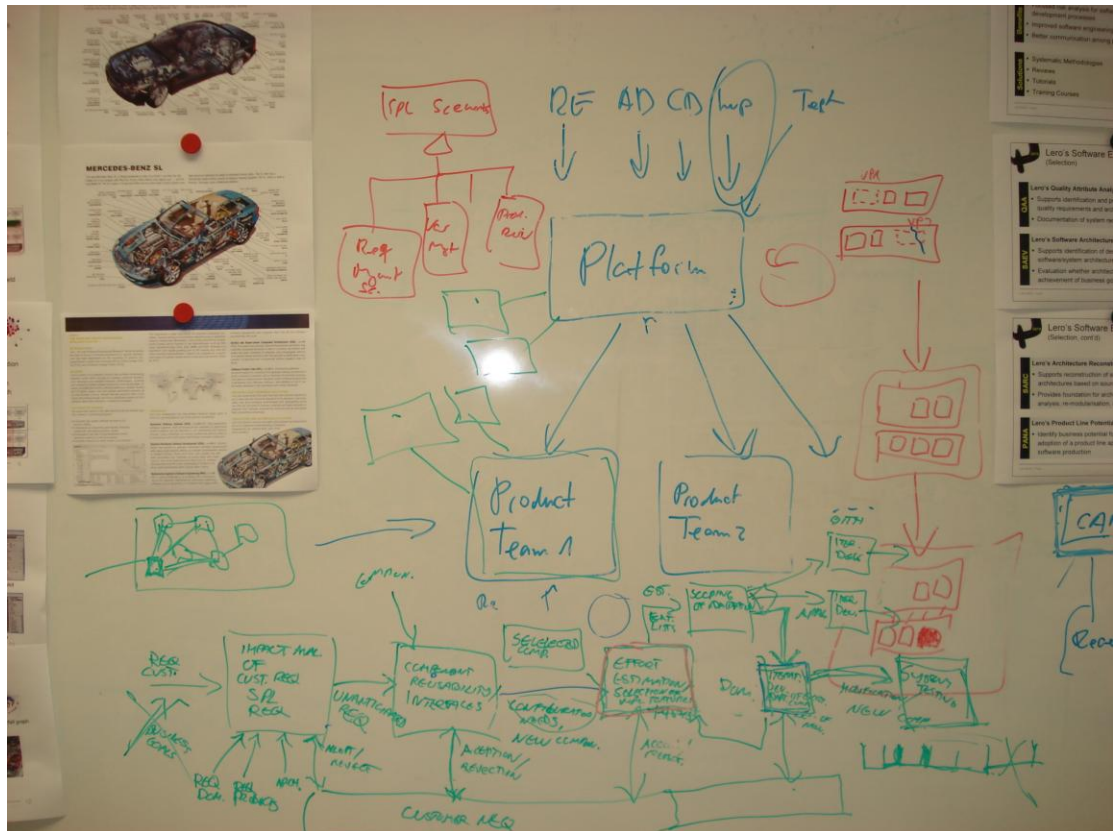


Figure 3-3 Workshop Whiteboard

Thirdly and finally, single reference framework elements should be linked with corresponding theoretical and empirical references. The validation process itself, namely the discourse and its implications for the modelling process, needs to be documented appropriately. This version of Pro-PD is validated through the literature and expert agreement. The expert opinion workshop series comprised as many iterations as necessary until the experts were largely in agreement on the framework structure and insights gained from preceding workshops was diminishing. In the case of this research, ten workshops were held. The development and evaluation undertaken during this stage of the research is discussed in Chapter Four.

3.4.3 Stage Two – Industrial Case Study: Robert Bosch GmbH

For the case study, the researcher collected data on the product derivation process of software-intensive automotive product lines in different business units within Robert Bosch GmbH. The company was chosen for the case study because previous SPL efforts within the company have been judged a success by their peers [94]. The case study was carried out in conjunction with the corporate research division.

3.4.3.1 Data Source

The researcher met with the case study company where he gave a presentation on the research project and the nature of the commitment required for any participant company. In return for company participation, on completion of the case study the researcher would organise a workshop where he would present a set of recommendations for improvements within the company's product derivation process. The researcher would introduce some alternative derivation practices as identified in the literature. On completion of the research project, the researcher would make any findings available to the participating company.

The company agreed to participate and organised for the researcher to work with Robert Bosch Corporate Research. Robert Bosch Corporate Research was interested in analysing product line product derivation methods applied in different Robert Bosch GmbH Business Units within the organisation. The goal of the work was to strengthen the ability of Robert Bosch Corporate Research to support and advise different Business Units in product derivation. The Business Units seeking support and advice were within the software-intensive automotive product lines division. Each Business Unit platform contained software and algorithm components, hardware modules and housing concepts. Members of two Business Units were made available to Robert Bosch Corporate Research for the duration of the project.

Robert Bosch Corporate Research nominated a person to liaise with the project for planning and execution of the research and four staff member from various business units were made available for the onsite visit. After discussion three primary work tasks were identified for the case study. These tasks were:

1. Identification of product derivation practices within the company
2. Evaluation of product derivation practices within the company
3. Recommendations to improve product derivation practices

A timeline was established with expected delivery dates on interim results and final delivery of recommendations agreed upon.

3.4.3.2 Data analysis procedure and methods

According to Yin [66], the use of multiple sources of evidence in a case study allows an investigator to address a broader range of historical, attitudinal, and behavioural issues. The use of multiple sources of empirical evidence provide an opportunity for triangulation in order to make any finding or conclusion of the study more convincing and accurate [66]. Yin identifies six sources of evidence: documentation, archival records, interviews, direct-observation, participant-observation, and physical artefacts [66].

While conducting the case study the researcher had access to the following sources of evidence:

- Company documentation
- Collective group notes from workshop
- Whiteboard drawings of company practices collected from workshop
- Researcher notes of workshop discussion
- Anecdotal evidence from company employees

Prior to an onsite visit to the case study company, the researcher had access to internal company documentation. These documents included information on product derivation practices within a particular business unit, organisational structure of the company's SPL teams and information on various derivation techniques applied within the various business units. The researcher performed an initial review of company practices based on this documentation. This initial review of company practices would be used to facilitate discussion during the onsite visit. This is illustrated as Bosch Model Version One in Figure 3-4.

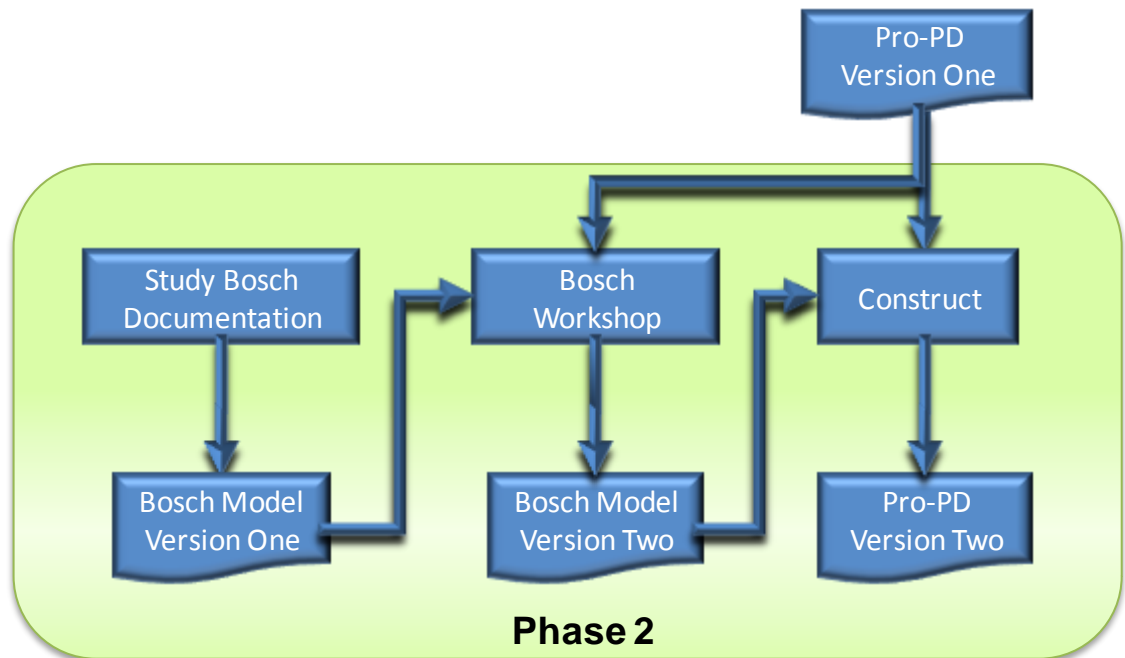


Figure 3-4 Stage Two: Industrial Case Study

For the onsite visit to the company, the researcher organised a two day workshop. The main theme of the workshop was product derivation practices within the case study company. The researcher examined the company product derivation practices under a number of headings:

- Activities
- Artefacts
- Stakeholders
- Tools
- Techniques
- Constraints
- Open issues

The researcher was careful not to present the Pro-PD Version One until the end of the workshop. This prevented any bias being introduced to the group discussions.

The primary researcher was accompanied by two other researchers, one of whom had extensive experience in conducting case study research. Each researcher had a specific

role for the workshop; One recorded the workshop discussion through note taking and a second recorded the main points on a projected PowerPoint presentation. Note taking in this manner helped the workshop discussion in two ways. Firstly, it kept everybody involved in the discussion and kept the discussion on track. Secondly, projecting results encouraged the workshop group members to form a consensus on topics. The third researcher was responsible for the whiteboard. The whiteboard was used to illustrate company practices, and all company employees were encouraged to actively engage with the whiteboard. Each whiteboard drawing was printed before the board was wiped clean to serve as a record of the discussion. In Figure 3-5, a sample whiteboard drawing shows one of the steps in the company's product derivation process. This figure describes how the step 'Resolve Variation Points According to Customer Requirements' took various inputs such as the Product Software Requirements Specification (P-SRS) and produced outputs such as the Product Software Architecture.

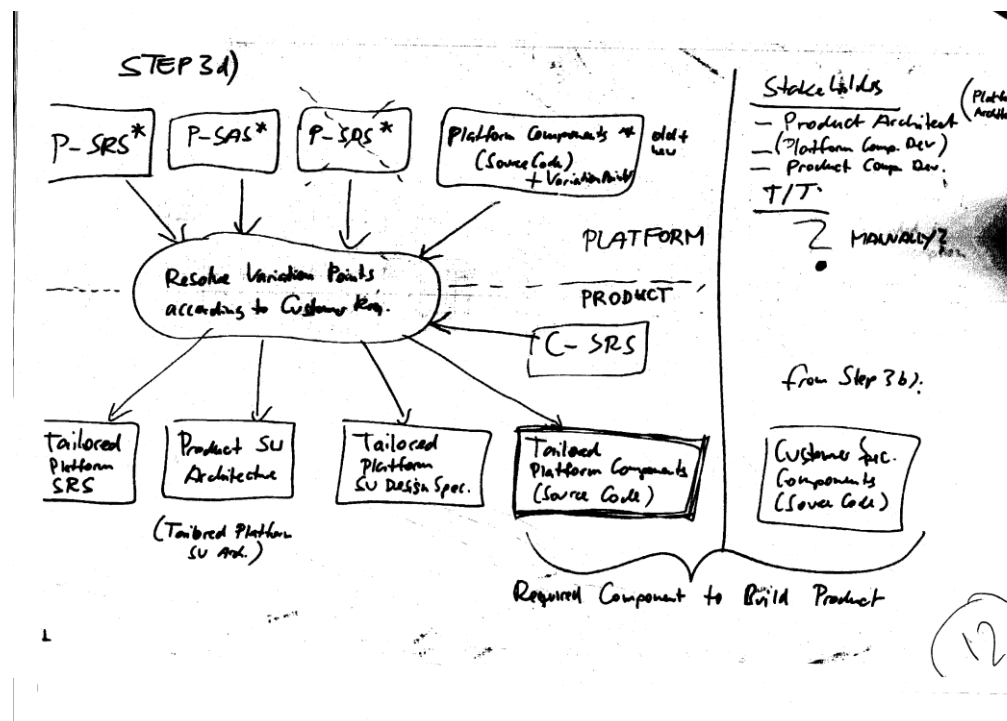


Figure 3-5 Sample Whiteboard Drawing from the Case Study Workshop

After the workshop the collected data was used to create a technical report on the company derivation practices (See Appendix G). The technical report described the Robert Bosch GmbH process of deriving an individual customer product from the product line platform. The report contained information on tools used to assist in derivation projects and organisational structure and roles within those derivation projects. The researcher made recommendations for the introduction of Agile elements in the Robert Bosch GmbH derivation process. The recommendations were based on a review of the literature (see Section 2.4.1.3) and current research into the adoption of Agile elements in SPL (see Section 6.2). These Agile recommendations can be seen in Chapter Six (see Section 6.5).

The Robert Bosch GmbH process description is illustrated as Bosch Model Version Two in Figure 3-4. The results and the construction of the Bosch Model Version Two led to changes in Pro-PD. These changes are described in Section 4.3.1. The output of this stage of the research is Pro-PD Version Two.

3.4.4 Stage Three – Academic Comparative Analysis: *DOPLER*^{UCon}

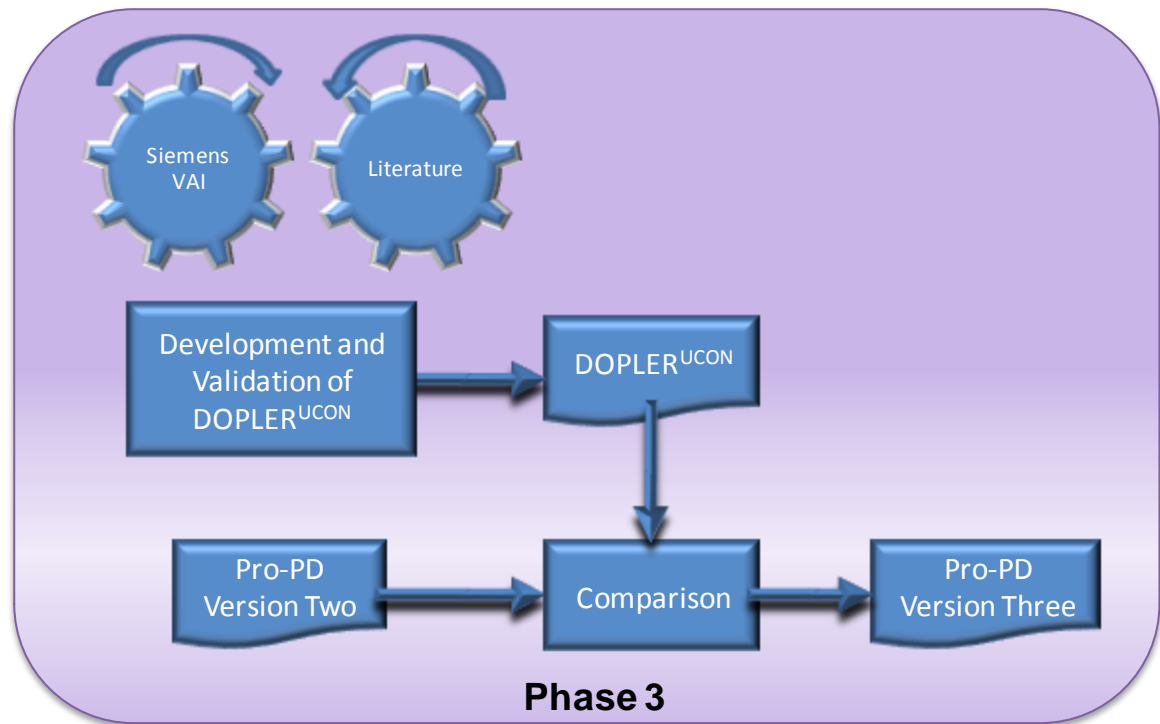


Figure 3-6 Academic Comparative Analysis

For the academic comparative analysis the researcher collected data on the product derivation process applied by *DOPLER*^{UCon} (**D**ecision-**O**riented **P**roduct **L**ine **E**ngineering for effective **R**euse: **U**ser-centered **C**onfiguration). Figure 3-6 gives an overview of this stage of the research.

DOPLER^{UCon} was chosen as a case study for a number of reasons. Firstly, the approach was driven by industry needs with the goal to define a user-centred, tool-supported product derivation approach [6]. The approach was mainly influenced by a research-industry collaboration with Siemens VAI, therefore, there was a strong practical industry focus in the approach. Secondly, the approach was focused on adaptable tool support usable in practical settings. The approach therefore was suitable in terms of the second objective of this research, to develop an adaptable approach. Thirdly, the approach had academic credibility, as it has been widely publicised and judged an appropriate tool approach to product derivation by peers through publications in leading journals and SPL

conferences [6, 23, 34, 35, 95-97]. Finally, DOPLER^{UCon} was designed to be generic, without focusing on a particular organisation or domain. The approach had a strong industry focus and through choosing this case for the comparative analysis, the researcher was performing a type of indirect industrial study.

While Pro-PD was influenced by Deelstra *et al.* [3] and a case study with Robert Bosch GmbH, DOPLER^{UCon} was mainly influenced by the research-industry collaboration with Siemens VAI. While the first approach was developed as a generic methodology, the latter was developed with a focus on adaptable tool support usable in practical settings.

3.4.4.1 Data Source

Pro-PD was presented to the DOPLER group during a visit to Johannes Kepler University (JKU) where the details of the research collaboration and commitment required were discussed. The group agreed to participate in a collaborative research project and nominated a person to liaise with the project for planning and execution of the research.

After discussion with the DOPLER group three primary work tasks were identified. These tasks were:

1. Perform a comparison of approaches
2. Discussion on industry experiences with Robert Bosch GmbH and Siemens VAI
3. Report and document mutual industry experiences

A timeline was created and expected delivery dates on interim results were agreed.

3.4.4.1.1 Data Analysis Procedure

Following the preliminary meeting with DOPLER, a second onsite visit to Lero was organised with the research group. There was an initial discussion on both approaches. Following this there was extensive studying of existing documentation on both approaches.

Pro-PD Task	Purpose	Supported by DOPLER^{UCon}? (none/partial/full)	DOPLER^{UCon} Activities involved
Allocate Requirements	Allocate requirements to relevant disciplines, e.g., hardware discipline, algorithms, Prioritise implementation iteration of particular product requirements.	Partly supported (Tasks in derivation models group related decisions. Requirements related with decisions in a task also allocated to a discipline. Prioritisation of implementation iterations not supported in DOPLER ^{UCon}).	Define roles and tasks
....

Table 3-2 Simplified Version of Spreadsheet Used

Based on initial discussions and existing documentation of both approaches, a first high-level mapping was created in a distributed manner using spreadsheets to visualize commonalities and differences between the two approaches (See Table 3-2). Using such a high-level mapping, the researchers of both approaches paper met in person to analyse the first results, discuss open issues, and detail the comparison. During the six month period that followed there were regular telephone conferences to work on the details of the comparison. The results of this stage of the research was published at SPLC 2009 [98].

In Chapter Four (see Section 4.3) the findings and results from the comparative analysis are presented. The case study impact on the research is discussed.

3.4.5 Stage Four – Systematic Evaluation

To evaluate Pro-PD, the researcher analysed prominent existing approaches for their support for Pro-PD activities. SEI's Product Line Practice Framework (PLPF) [1], COVAMOF [22] and PuLSE-I [24] were chosen because of they are well-known in the

community, mature enough for such an analysis and have been validated in different domains.

When performing a compatibility evaluation such as this it is important to evaluate the process model and not an instance of the model [81]. Instances of a process depend on particular contexts. The knowledge that Pro-PD was successfully applied in a project or SPL organisation is a specific experience. It can be a useful piece of information but is not a fair evaluation of the process. To have a fair evaluation based on empirical experiences a large number of case studies are needed. Therefore the work in this stage is described as an evaluation of Pro-PD rather than an assessment.

The systematic evaluation was conducted in two parts (see Figure 3-7). The first part of the evaluation was an inter-model evaluation with the PLPF. According to Ahlemann et al. [81] process models that are compatible with standards and norms are regarded as high quality. Compliance with standards and norms however can only be proven by a direct mapping of reference information model elements to standard or norm elements. In Section 4.5.2, the research describes a mapping between Pro-PD and the PLPF.

The second part of the evaluation is to systematically analyse two prominent product derivation approaches for their support for the activities of Pro-PD. COVAMOF [22] and PuLSE-I [24]. To enable systematic analysis, a suitable evaluation framework was required. While a framework for evaluating product derivation approaches does not exist, a framework developed for the purpose of evaluating software product line architecture design methods [99] was identified¹ which could be adapted for the purpose of this research. The goal was not to provide a detailed survey on existing product derivation approaches but to validate key activities in Pro-PD by studying how they are supported or prescribed by prominent existing approaches. These approaches have been developed independently, for different purposes and in different contexts. The aim was not to judge

¹ Dr. Rabiser and the researcher collaborated to develop the first version of this evaluation framework.

their usefulness or compare them in detail but to find out if they support or prescribe the Pro-PD components. In Section 4.5.3 this part of the research is described.

The evaluation was broken into two parts as the description of the PLPF is far more abstract than COVAMOF or PuLSE-I. Furthermore, both COVAMOF and PuLSE-I are technical approaches to product derivation while the PLPF is a framework of key practices for SPL. The process and results of this evaluation are described in Chapter Five.

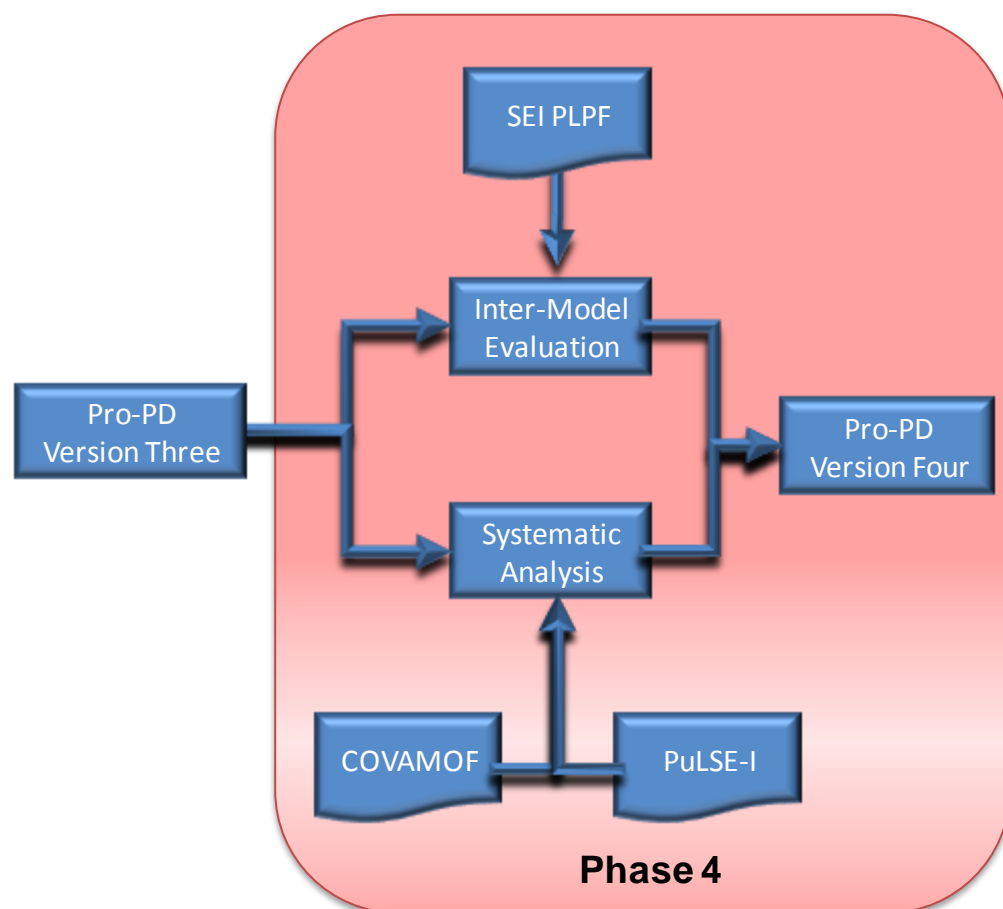


Figure 3-7 Systematic Evaluation

3.4.6 Research Design Conclusion

In this research, an evolutionary multi-method approach was followed. The research design adopted in this study is influenced by an approach by Ahlemann et al. [81] and is focused on empirically grounded and validatable process model construction.

The first stage of the research design is the core construction stages. This involved a literature review from which a preliminary version of the model was constructed followed by a series of expert opinion workshops. The second stage is an industrial case study. The third stage is an academic comparative analysis. The final stage of the research was to evaluate Pro-PD by systematically analyzing the support for its activities in prominent existing approaches.

3.5 Research Validity

All research only becomes valuable when it is first deemed credible. Many authors such as Creswell [100] however point out that there is no consensus or right way of verifying the credibility of qualitative research. In this context, Marshall [101] argues that the quality of research is dependent on honest and forthright investigations. This dependency means it is difficult to verify the quality of qualitative research. Another difficulty in qualitative research is the introduction of bias and the danger of multiple interpretations of data. Therefore, a self critical attitude is essential in qualitative research. Demonstrating how you know is as important as demonstrating what you know.

There have been a number of suggestions in the literature about how best to evaluate the quality of qualitative research. Common suggestions are to evaluate the reliability and validity of the research results. Patton [70] suggests that a credible qualitative study needs to address the following three questions:

1. What techniques and methods are used to ensure the integrity, validity and accuracy of the results?
2. What does the researcher bring to the study in terms of qualifications, experience and perspective?
3. What paradigm orientation and assumptions underpin the study?

What techniques and methods are used to ensure the integrity, validity and accuracy of the results?

Internal validity and credibility is achieved through prolonged engagement in the field, persistent observation and triangulation exercises [102, 103]. From their discussions on triangulation, Liamputtong and Ezzy [104] identify four types of triangulation, which may be used to strengthen a potentially weak case:

- Data Source Triangulation: The use of multiple information sources.
- Method Triangulation: The application of findings generated by different data collection methods.
- Researcher Triangulation: The inclusion of a variety of researchers in the research process.
- Theory Triangulation: the researcher draws on multiple theoretical perspectives to provide new insights.

This research uses data source triangulation. In the initial framework development, multiple sources of literature were used as well as anecdotal evidence from SPL experts. Industrial practice was integrated directly through case study research and indirectly through the experiences of DOPLER^{UCon}. In the Robert Bosch GmbH case study multiple data sources were used (See Section 3.4.3.1). In the DOPLER^{UCon} academic comparative analysis, the researcher had access to documentation and to the developers of the approach (See Section 3.4.4.1).

The research uses method triangulation. The research design includes case study research, expert opinion, facilitated workshops and SPL literature.

The research uses researcher triangulation. When it was possible the services of other researchers to facilitate the main researcher were engaged. In the facilitated industrial case study workshop session (Section 3.4.3.1) two other researchers assisted in the organisation and recording of results in the workshop sessions. During the collaboration with DOPLER, the research involved two members of the DOPLER team.

The research uses theory triangulation. In the development of the initial version of Pro-PD, the researcher used a multitude of evidence from literature to theoretically validate aspects of the model. The researcher synthesised this existing theory. This is a form of theory triangulation.

Another method of strengthening credibility is through respondent validation [105]. Lincoln and Guba state that respondent validation is “the most critical technique for establishing credibility” [103]. In respondent validation, the researcher goes back to subjects with tentative results and refines them in light of their reactions. Section 3.4.3 describes how respondent validation was used during the iterative development of the framework in the expert opinion workshop series. In the academic comparative analysis, the results were validated by the DOPLER team, through workshops, conference calls and documentation.

What does the researcher bring to the study in terms of qualifications, experience and perspective?

Patton [70] recommends verifying the credibility of the researcher. Patton [70] suggests that a qualitative report must include information about the researcher because the researcher is a central aspect to the study.

The researcher of this study has two years industrial software engineering experience, part of which was spent working on a large scale software project for the insurance industry. While the project was not part of a product line, the development environment involved using many key SPL facets such as centralised architecture for multiple products and implementing product variability. The researcher also has industry experience of the adoption of specific Agile practices such as automated testing approaches.

The researcher has had exposure to state-of-the-art literature within the SPL area and through the case study research has seen industrial SPL projects. He has been exposed to anecdotal evidence from SPL projects running in parallel to this research. The researcher undertook several generic research skills training session. He attended tutorials and workshops on SPL related topics such as product line testing, product production, goal driven product derivation. Finally, he does not have any personal connection to the participants of the study.

What paradigm orientation and assumptions underpin the study?

Generalisation

Patton [70] points out that the small size involved in qualitative methods make it impossible to generalise the results. This is especially true in case study research where the focus on a particular case makes it unable to produce a general conclusion. In an effort to counteract any weakness in the research in terms of generalisability, a multi-method research design was adopted. By considering expert opinion, literature, case studies and documented best practice, the generalisability of Pro-PD can improve and more plausible interpretations of the data can emerge.

Objectivity

The most frequent charge heard on qualitative research is its inevitably 'subjective' nature. Patton [70] considers subjective to be biased, unreliable and irrational. When examining any research data, a researcher wishes to have an objective view in order to ensure that any conclusions are unbiased. Despite this, any conclusions taken from the data will be based on personal interpretation and reflect to some extent the researchers view of the world. Therefore, the study aims to be as objective as possible, while acknowledging that any conclusions from data will contain researcher's personal insights and interpretations.

3.6 Handling Refinements

Each model correction proposal derived from the research provided the basis for the revision or refinement of Pro-PD. A major challenge during the research was the evaluation of different suggestions with respect to each other. For example, before a correction is integrated it has to be determined whether the proposal can be characterized as being universally valid or whether it is tied to a specific context and therefore not suitable for model refinement. Furthermore, it is possible that improvement suggestions made by different persons are quite contradictory.

There were two main options to resolve these situations: First, one proposal was chosen over another if the source was deemed to be of a better quality, either through its experience or the location of the source. This evaluation was conducted by the researcher, and involved a degree of researcher interpretation as to the quality of the various sources. The alternative approach was to consider both suggestions and integrate them both into the model.

3.7 Meeting the Research Objectives

Objective One – To define a structured process model for product derivation

Pro-PD provides the process support that is required within product derivation (see Section 2.3.2.3). Pro-PD focuses on the essential tasks, roles and work artefacts used in the derivation process. It satisfies the need for a defined flow of artefacts (see Section 2.3.2.1) and defined roles and responsibilities (see Section 2.3.2.2) within product derivation.

Pro-PD is an output of the applied research design described in Section 3.4. The research meets the first objective through the development of Pro-PD. Pro-PD is described in Chapter Five of this thesis.

Objective Two - To demonstrate the adaptability of the process model

The process structure of Pro-PD is based on the waterfall process model. In Chapter Six it is adapted to fit the characteristics of an iterative process model and in doing so the research will be meeting needs for more adaptability (see Section 2.3.2.4) in product derivation approaches.

In Chapter Six, the research looks at the literature on Agile practices within SPL. Agile approaches are identified as a potential solution to issues in product derivation including the difficulty in handling product specific development, how product derivation is still a time-consuming and expensive activity in many organisations [3]. In Chapter Six, the research demonstrates the adaptability of Pro-PD by proposing an Agile Pro-PD (A-Pro-PD), the results of which, have been published [106].

Furthermore, the researcher of this study believes that Agile integration could strengthen Pro-PD's ability to support product derivation, particularly when dealing with small organisations. An Agile Pro-PD would be meeting calls from industry for research into this area [107]. A combination of Agile and SPL is expected to create a leaner but more disciplined product derivation process [108].

Objective Three - To add to the established knowledge on product derivation

The research proposes an Agile approach to product derivation (A-Pro-PD) and identifies Agile practices that could be adopted within an Agile product derivation project (see Chapter Six).

3.8 Summary

In this chapter the research approach undertaken in this project was described. The research uses empirical evidence in the development of the process reference model for product derivation while an evolutionary multi-method approach was followed. The different stages form a continuum in which the focus of the research, the model, is continually adjusted based on the results of the previous stage. In the next chapter, observations made during these research stages, and how these impacted on the development of Pro-PD are reported.

Chapter Four: Development and Evaluation of Pro-PD

4.1 Introduction

The goal of this chapter is to describe the development and evaluation cycles during which Pro-PD was iteratively constructed. For each version of Pro-PD, the development and evaluation is described and documented.

The chapter begins by discussing the development and evaluation of Pro-PD during stage one of the research (see Section 4.2) where sources in the literature and captured expert opinion are viewed in light of their impact on version one of the model. Version two of the model was influenced by the case study research conducted at Robert Bosch GmbH. The researcher describes how practices identified in the company influenced the model (see Section 4.3). Version three of the model drew on a research collaboration with DOPLER laboratory from the University of Linz. The researcher discusses how the collaborative work undertaken has further influenced the model (see Section 4.4). The final version of the model is the result of an inter-model comparison and an evaluation model developed through existing approaches (See Section 4.5). This version is the final result of the research design and is presented in Chapter Five.

It can be concluded that each version of the model has had flaws in different aspects, and this is natural. The purpose of the different research iterations has been to sort out the weak as well as the strong parts of the model. The interesting questions are how, why and what impact these research stages have had on the model design. Therefore the objective of this chapter is to present the arguments for the design decisions taken throughout the research and describe the evaluation during each stage.

4.2 Stage One - Literature Review and Expert Opinion Workshops

This section reports on how the initial version of Pro-PD was developed using sources from the literature (see 3.4.2.1) and through feedback received during the iterative workshop series (see Section 3.4.2.2).

4.2.1 Steps taken in Stage One

The literature review 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. Concurrently to this, a series of iterative workshops was organised over a five month period, with participants meeting twice a month. At each workshop the model was presented to the experts. The model was evaluated and discussed amongst the group. After each workshop the researcher returned to the literature and based on feedback iteratively developed the model.

4.2.2 Observations

During the initial literature analysis two levels of model granularity were defined. First an appropriate model skeleton was created to organise the overall structure of the model. The model skeleton would be presented in the form of phases of development. Then the researcher determines how to fill the skeleton phases with practices which fit better with the phase context.

4.2.2.1 Step One: Model Skeleton Construction

SPL remains a relatively young discipline within software engineering indeed, this characteristic brings a lack of formal scoping to the domain and thus difficulty in constructing a skeleton for the model. For the development of the model skeleton the researcher focused on prominent existing approaches within the literature [19, 20, 22, 24, 28]. These approaches were selected as they proposed a process overview for product derivation that considers all or almost all of the derivation process. Furthermore the considered approaches all had a derivation skeleton themselves i.e. a high level

organisation of derivation tasks. Therefore these approaches were the most useful for the task of creating an initial model skeleton.

The work by Sinnema et al. [22] presents a product derivation approach developed primarily on two industrial case studies conducted by Deelstra *et al.* [3]. The *COVAMOF* (**C**Onfiguration in Industrial Product Families **V**ariability **M**odeling **F**ramework) derivation process is divided into four steps, i.e. Product Definition, Product Configuration, Product Realization, and Product Testing (see also 2.3.1.8).

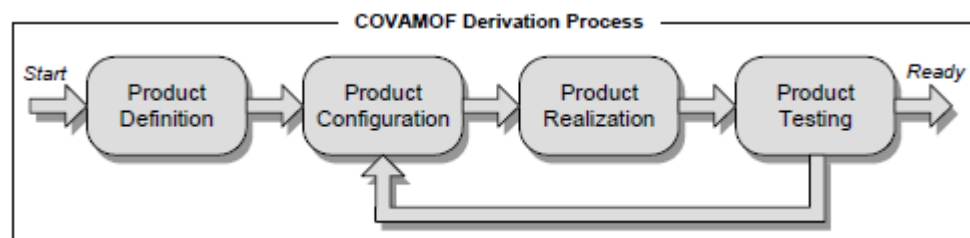


Figure 4-1 The COVAMOF Derivation Process [22]

PuLSE-I [24] defines several process steps based on other PuLSE artefacts, e.g., reference architecture, domain decision model, or scope definition (see also 2.3.1.1). PuLSE-I activities cover planning product derivation, instantiating a product architecture from the reference architecture using decision models, and additional designing, implementation, and testing activities. Delivery and maintenance processes are also addressed.

In FORM [28], product derivation begins with requirements analysis and tries to find a matching set of features (see also 2.3.1.4). The selected features can be instantiated to a product configuration by following the dependencies and constraints documented in the feature model. The architecture model selection is facilitated through feature selection also. Application development finally consists of following the specifications and selecting pre-coded components, filling in skeletons, or instantiating parameterised templates.

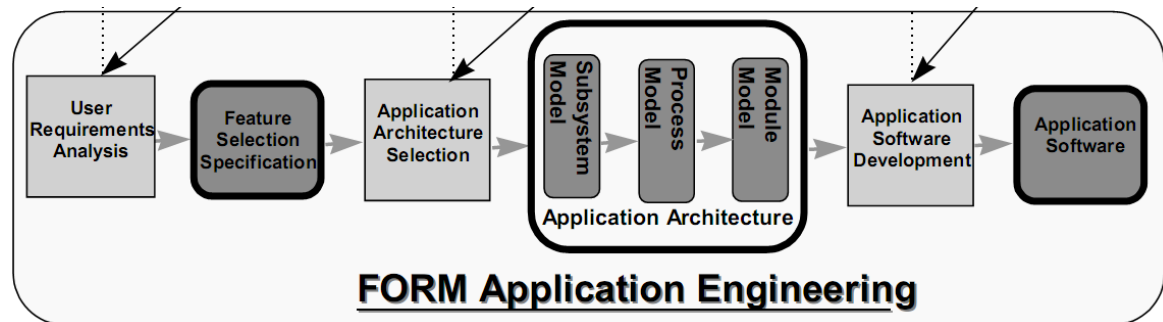


Figure 4-2 Form Application Engineering [28]

Based on these three approaches, a framework skeleton consisting of four main phases was constructed:

1. Impact Analysis;
2. Reusability Analysis;
3. Component Development and Adaptation
4. Product Integration and Validation.

Below Figure 4-3 provides a diagrammatic representation of these phases. Impact Analysis (phase 1.0) is aimed at gathering product specific requirements based on customer requirements and negotiation with the Platform Team. Reusability Analysis (phase 2.0) purports to create a partial product configuration based on the product specific requirements and by using the available core assets. During Component Development and Adaptation (phase 3.0), new components are developed (if required) and existing components are adapted to satisfy requirements which could not be satisfied by configuring existing core assets. Finally, Product Integration and Validation (phase 4.0) aims to integrate the core asset configuration and newly developed components and to validate the integration by performing appropriate testing procedures.

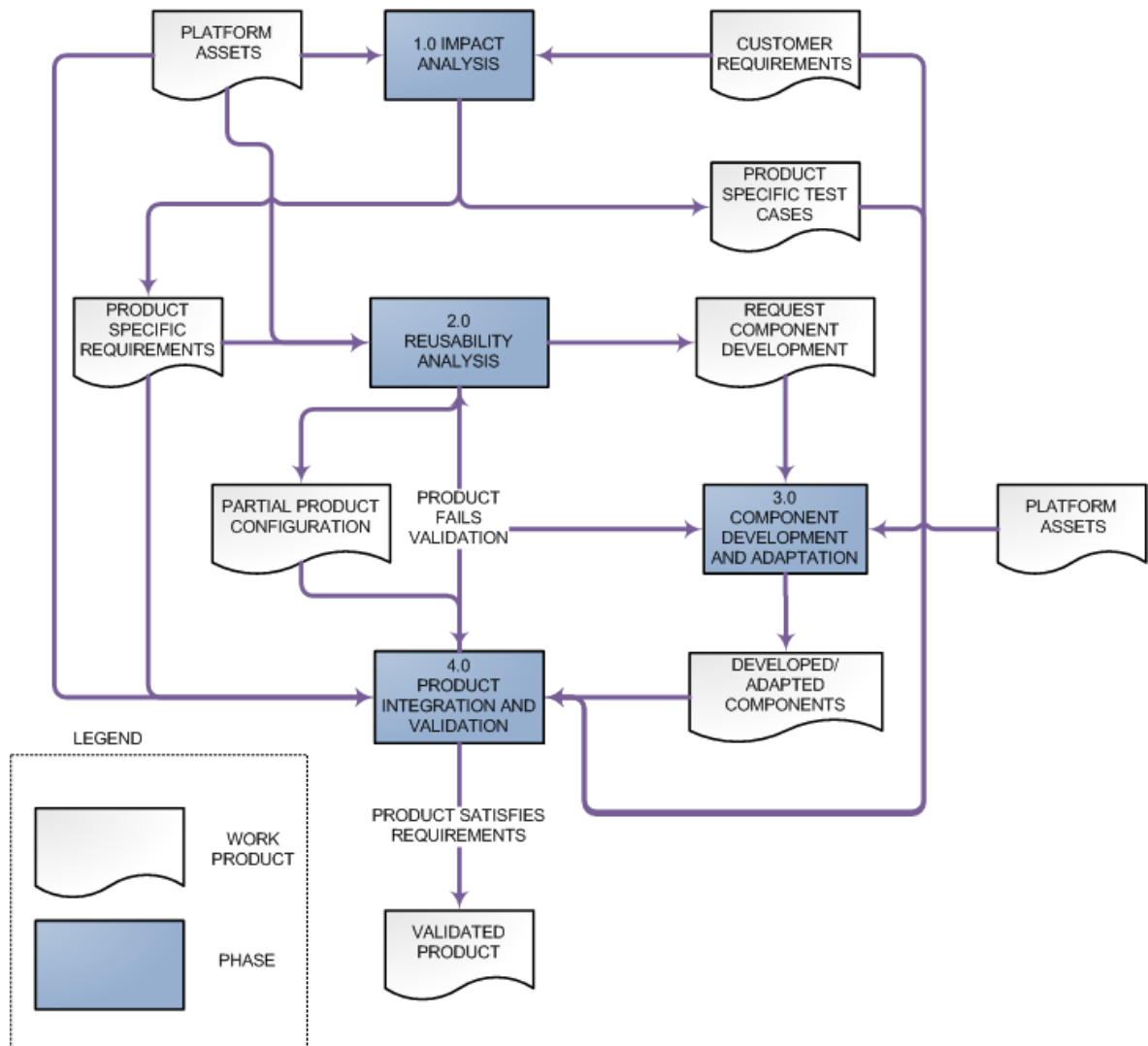


Figure 4-3 Overview of Pro-PD Version One

This initial configuration of Pro-PD was structured in the form of a “stairs” model, a type of “waterfall” [7] development model. Through references to Sommerville [109] the researcher were aware of the two main disadvantages of this type of process model. Firstly, the premature freezing of some parts of the product design which occurs after a small number of iterations. This may result in a product that does not correctly reflect the customer expectations since his voice is discarded early in the process. Secondly, this type of process should only be used when requirements of the software to be produced are well-

understood as well as when the risks incurred as a result of designing and implementing such a system are well-known.

However, despite this insight, the researcher argues in favour of this linear staged representation of the model. In a SPL approach, most product requirements are well-understood since they have been carefully defined and analysed according to a particular market and to be valuable enough to develop reusable assets supporting them. Moreover, design and implementation risks are largely reduced by the (re)use of an architectural framework since it provides a core architecture and its implementation (in the form of an object-oriented framework) on which all products are based.

The model was then discussed over a number of workshop meetings. Prior to each workshop, the model was sent to each expert, allowing them to familiarise themselves with the version being analysed. At each workshop an interactive presentation on the current model was given where justification for the model design was given. Issues requiring clarification or which were disputed among participants were discussed.

It was noted that in time it would be interesting to consider a more incremental development process (such as Boehm's spiral model [42]). In particular, this would be worthwhile for the customer-specific part of the application; an incremental process would allow us to validate several pieces of the prototypes with the customer before the product is finished. Figure 4-4 shows one such whiteboard discussion where experts felt an iterative design was more reflective of product derivation practice.

While the discussion on an incremental development process was noted, the workshop participants felt that, until further evidence suggested otherwise, a product derivation process model should be represented as linear and not iterative.

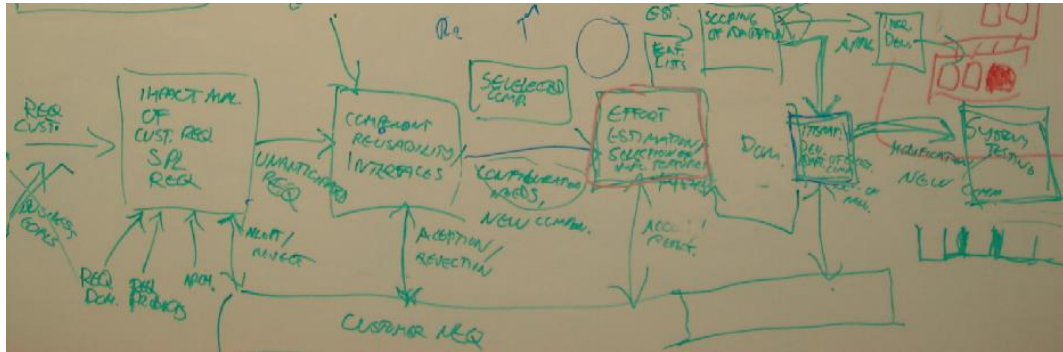


Figure 4-4 Framework Skeleton Discussions at Workshop

For each phase, based on SPL literature, attributes were used to characterise that phase. The attributes used for the characterisation were based on their ability to help define a systematic process with task, work products and roles. The attributes used are listed in Table 4-1.

Attribute	Description
Phase Name	The name of the phase
Goal	The overall aim of the phase
Input	The artefacts which are input to the phase
Entry Criteria	The conditions which have to be met before the phase can commence
Tasks	Work which must be completed during the phase
Output	Work products which are produced or changed by the phase
Stakeholders	People who affect or are affected by the step
Tools	Objects which assist the performance of this phase
Exit Criteria	The conditions which have to be met before the step can be completed

Table 4-1 Description of Phase Attributes

The phase attribute tables were the foundation upon which skeleton filling could occur. These tables were used as a basis for identifying which tasks should fall under

which phases. For example in Table 4-2 the ‘Impact Analysis’ phase is defined. The other tables are included in Appendix B.

Attribute	Content
Phase Name	Impact Analysis
Goal	To form a set of product specific requirements which satisfy the customers’ needs and ensures long term viability of the product line
Input	Platform Assets Customer Requirements Platform Test Case Artefacts
Entry Criteria	The product is requested by the customer and falls under the scope of the Product Line
Tasks	Customer requirements are mapped to platform features Unmapped requirements are negotiated with the customer The Product Specific Requirements are formed The Product Specific Test Cases are created
Output	Set of Product Specific Requirements Set of Product Specific Test Cases
Stakeholders	Customer Product Team Platform Architect
Tools	Requirement Management Tools Support for mapping between requirements, features and components including visualisation of the various artefacts and the dependencies among them [110] Automatic generation and management of test cases
Exit Criteria	The requirements set is sufficiently complete to begin next stage The scoping of the product may need to be re-examined if a sufficient amount of requirements need to be re-negotiated

Table 4-2 Impact Analysis Phase

4.2.2.2 Step Two: Skeleton Filling

After identifying the main model phases during skeleton construction, the specific tasks to fill these phases were defined. For example, looking at the skeleton filling for the ‘Impact Analysis’ phase, the researcher focused the literature review on tasks that contributed to the primary goal of this phase, “to form a set of product specific requirements which satisfy the customer’s needs and ensures long term viability of the product line.” When two approaches differed in what accounted for an Impact Analysis task, the variation was documented. The end result was a documented overview of the variation and commonalities for that particular phase which could be presented to experts.

Specific task attributes were elicited from the literature. They are listed in the table below along with a short description. The attributes chosen for the characterisation helped define a systematic process with task, work products and roles. The task tables are included in Appendix J.

Attribute	Attribute Description
Name	Name of the task
Purpose	The primary function of the task
Inputs	Defined work products which are consumed within the task
Outputs	Defined work products which are produced within the task
Main Description	Detailed overview of task
Roles	The primary performer for this task. Optionally one or more roles as additional performers

Table 4-3 Description of Task Attributes

The identification of task attributes was of particular benefit when eliciting expert opinion during the workshops. The researcher formulated these task attributes both as short statements and as questions, where the latter could be answered with one of the attributes values. For example, the attribute ‘Purpose’ was reformulated into a question ‘What is the

purpose of the task?’ In the case of the ‘Customer Negotiation’ task (see Table 4-4), the discussion would run until a consensus among workshop experts as to the purpose of ‘Customer Negotiation’ was formed. The use of questions in a workshop context made it easier to find and form consensus among the experts.

Attribute	Attribute Description for this Task
Name	1.2 - Customer Negotiation
Purpose	To meet (ideally) all of customer’s needs while retaining the profitability of the platform assets for the whole product line
Inputs	Customer Specific Product Requirements
Outputs	Negotiated Product Requirements
Main Description	Customer requirements which cannot be satisfied by existing assets are negotiated. Customer negotiation is an important and critical aspect of product derivation. The trade-off here is to meet (ideally) all of the customer’s needs while retaining the profitability of the platform assets for the whole product line.
Roles	Product Manager Customer

Table 4-4 Task Attributes for Customer Negotiation

The identification of the task attributes was a pre-cursor to deciding the final artefact flow for a particular phase. The task attributes helped structure the process flow for a phase. In Figure 4-5 a diagrammatic representation of the ‘Impact Analysis’ phase can be found. It illustrates the tentative configuration of the tasks and work products of the initial version of the model when viewed in terms of the final version description.

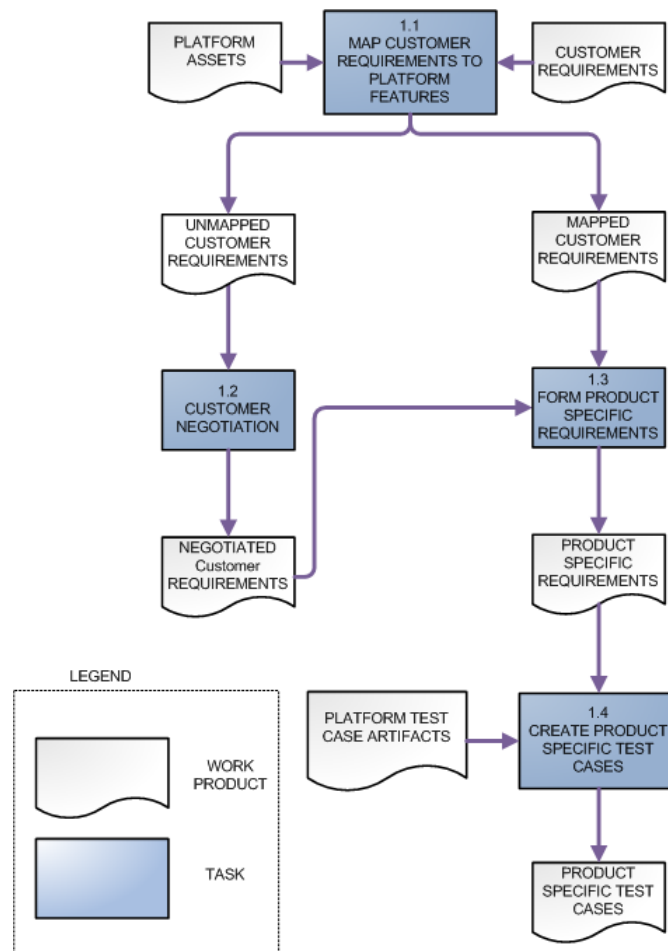


Figure 4-5 Impact Analysis

4.2.3 Conclusion of Stage One

The output of the literature review and expert workshop series was a first version of the product derivation process model. The existing literature in product derivation was organised and synthesised to form the first version of Pro-PD. This version of the model [111, 112] was validated theoretically through the use of existing literature from the SPL field and through agreement with the members of the expert panel. Furthermore, it was presented to Robert Bosch GmbH during a workshop on product derivation practices within the company. Feedback and areas for improvement were identified. Version one of Pro-PD is included in Appendix C and is an extract from [112].

4.3 Stage Two - Industrial Case Study: Robert Bosch GmbH

In the previous section the researcher described how sources in the literature and garnered expert opinion influenced the development of the initial model. In these section observations on the product derivation process obtained during case study research at Robert Bosch GmbH (See Section 3.4.3) is reported and how these observations influenced the development of Version Two of the model is discussed.

4.3.1 Observations from the Case Study

The main observations obtained from the case study on industrial product derivation practices were:

- Additional Development Disciplines
- Additional Roles and Tasks
- Platform-Product Synchronisation
- Use of Documentation

4.3.1.1 Additional Development Disciplines

The organisational structure for a particular business unit engaged in product derivation is broken into three broad disciplines in Robert Bosch GmbH; 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 and interfaces and plugs teams. The software discipline had basic software and algorithms teams.

4.3.1.2 Additional Roles and Tasks

During the case study a number of novel product derivation roles were observed. For example, the software product team consists of architects, developers, integrators, testers, and customer specific component developers. These roles are replicated across the

independent product sub-discipline teams and platform teams. Moreover, similar roles exist for hardware and mechanics.

These intricate role structures are reflected by appropriate communication and task structures. For instance, the allocation of requirements to responsible teams has to consider the various disciplines and sub-disciplines. This requires a higher degree of granularity for requirements management tasks than originally envisaged. This can be observed when the case study company starts a product-specific project. During the early phases, the customer requirements are translated into a set of internal company documents. These documents are processed and augmented during various tasks where requirements are analysed for reuse potential and then assigned to responsible disciplines and sub disciplines.

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

4.3.1.3 Platform-Product Synchronisation

Within the case study company, product development requires a high degree of coordination and communication as the heavy dependencies across disciplines and the platform product divide is managed. In Figure 4-6 the observed platform product dependencies are illustrated.

The product team designs and implements customer specific components based on the customer requirements. The platform team receives the platform software requirements containing the required extensions to the existing platform in order to facilitate the new customer requirements. Both the customer-specific and platform development is occurring in parallel. The product team needs to interface correctly with the new platform release. Here, the product team can choose between two 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, not illustrated in Figure 4-6, 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.

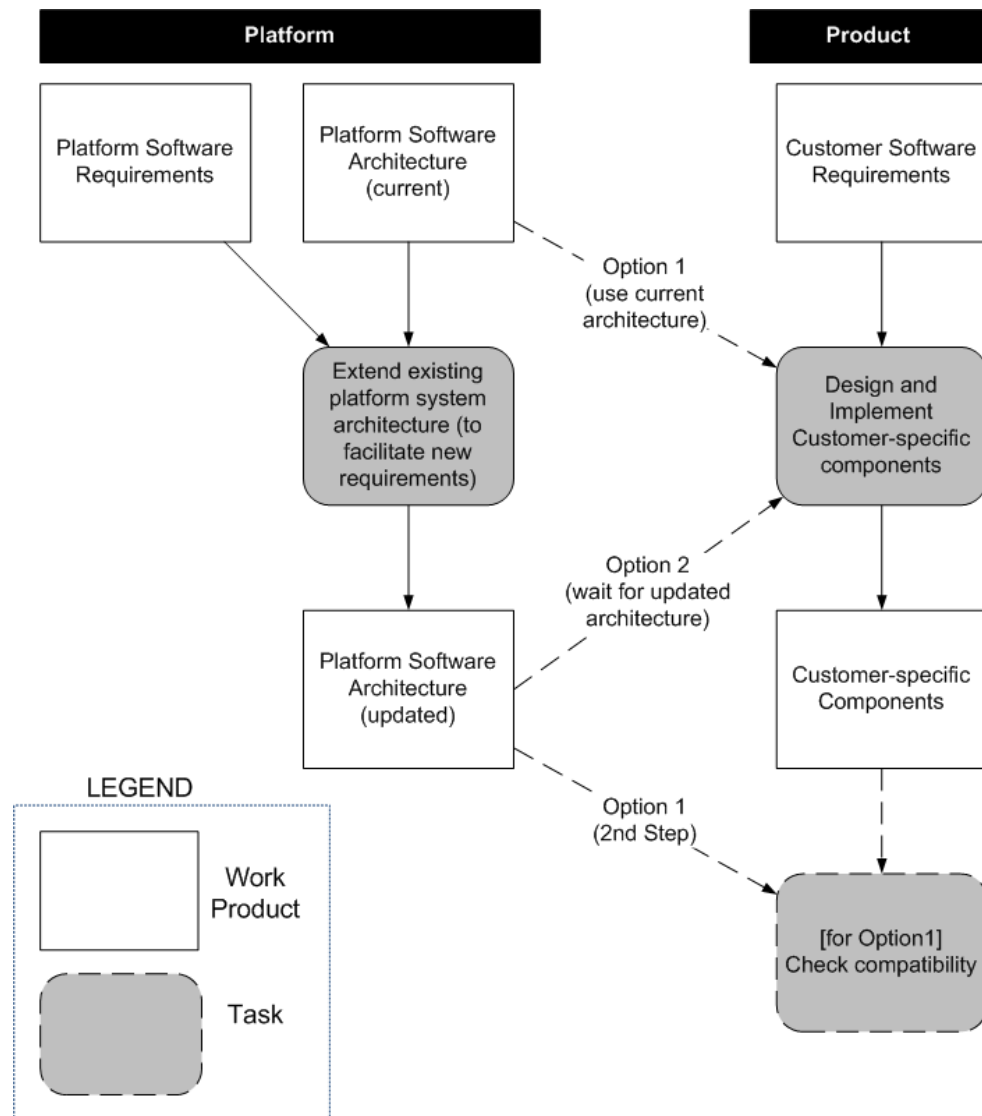


Figure 4-6 Synchronisation Caused by Platform-Product Dependency

These outside dependencies that the product team must handle are a reoccurring process pattern within product derivation. Similar dependencies can be seen during software integration with the hardware modules. The software product team can choose one of the development strategies similar to those described, for handling the hardware interfaces during parallel software and hardware development.

4.3.1.4 Use of Documentation

The case study company relies heavily on documentation to drive the product derivation process. Documentation is used to facilitate communication and synchronise development between the product and platform teams, between the different hardware, software and mechanical disciplines and also between the sub-disciplines. It is also used as a milestone to plot project progress and as a driver to trigger certain tasks within the project. Additionally, in certain domains evidence of due process is required by law, some documents satisfy this condition.

4.3.2 Impact on Pro-PD

In the preceding section, observations made during the case study research were discussed. These observations have resulted in the augmentation and refinement of version two of Pro-PD:

- Inclusion of Additional Roles
- Refined Impact Analysis
- Synchronisation between Teams

4.3.2.1 Refined Impact Analysis

In section 4.3.1.2, the case study company was observed to have a mature requirements management process. The customer requirements are used to create a system requirements specification which is in turn broken into individual discipline requirements, allocated to sub-discipline teams and categorised as platform or product requirements.

This caused the researcher to recognise the need for a more sophisticated requirements management process.

The task *Translating the Customer Requirements* was included which translates the customer requirements from a customer specific document into an internal organisational document. The product team use the ‘Glossary’ document which contains customer terms and their Product Line equivalent as a guide during this task.

The *Find and Outline Requirements* task allocates requirements to the relevant disciplines; the allocated requirements are held in separate requirements documents, such as the platform software requirements specification and the customer hardware requirements specification.

4.3.2.2 Inclusion of Additional Roles

Earlier the researcher reported on the unexpected number of product derivation roles within the case study company (see Section 4.3.1.1 and 4.3.1.2). This was generalised within Pro-PD.

First, even if the researcher generalises from the special case reported earlier, he wants to keep the notion of software-intensive systems which incorporate *non*-software subsystems. This is reflected by the concept of disciplines which specialise on overall aspects, such as Software, Electronics, or Mechanics (see second level of the hierarchy shown in Figure 4-7).

Within each discipline the model allows for teams which, depending on the particular industry, focus on specialized aspects within that industry. For instance, in the case of automotive systems, there is a special focus on algorithms which detect certain physical conditions such as an imminent collision. Hence, there is a separate team within the software discipline assigned to this aspect. The model allows for the adaptation and parameterisation of disciplines and teams depending on the concrete organisational structure.

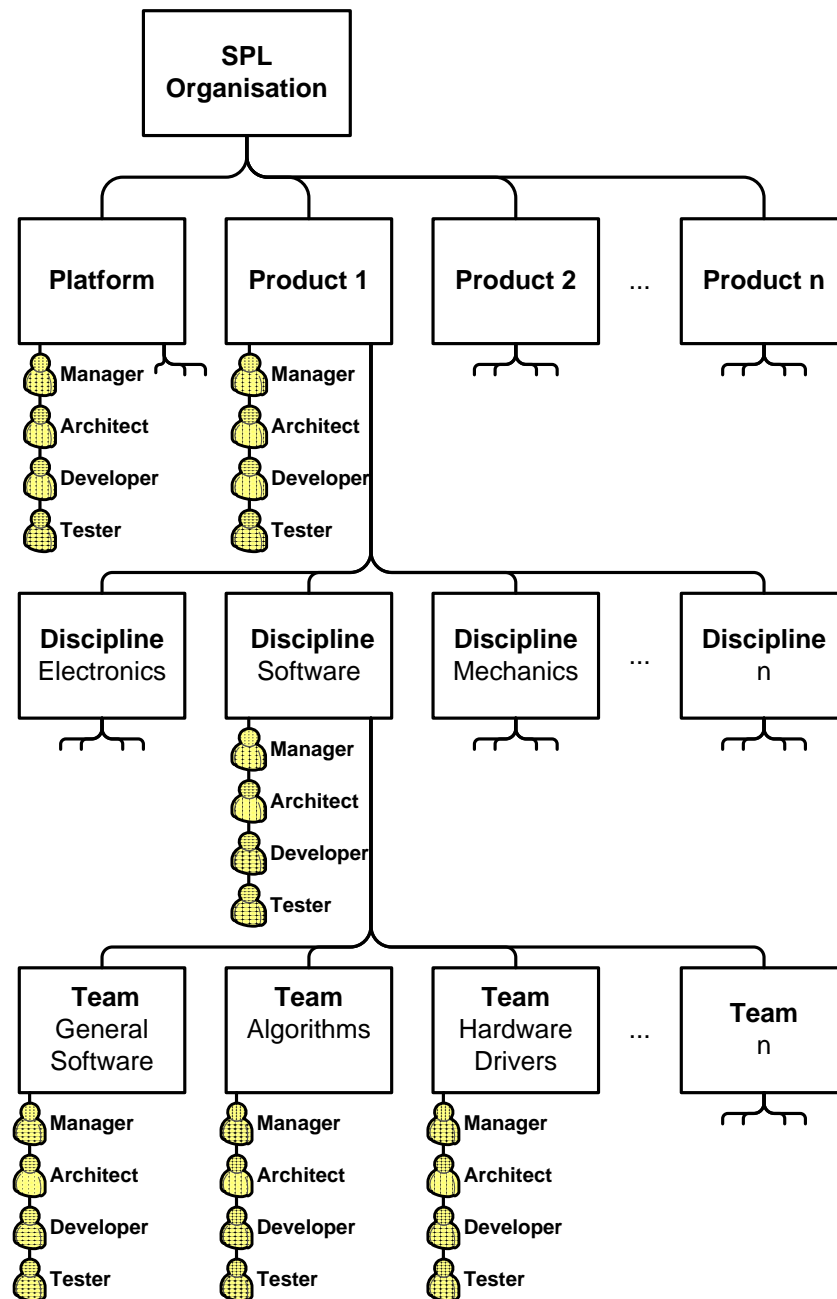


Figure 4-7 Organisational Structure and Roles

Second, general roles such as Manager, Architect, Developer, and Tester, are defined which can be instantiated in a certain discipline or team context (see the role symbols in Figure 4-7). It should be noted, that it is not mandatory to use each role for each

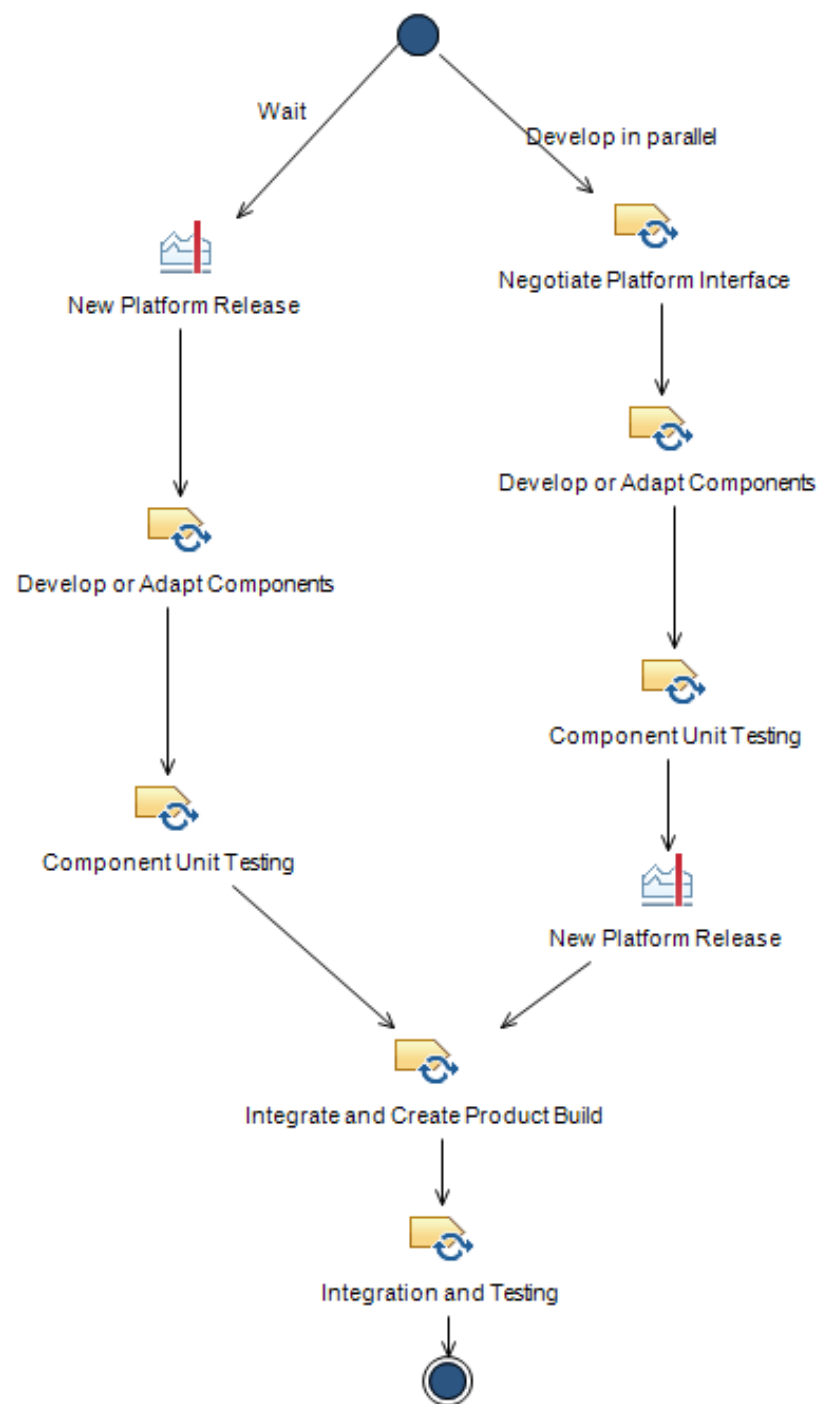
particular discipline or team. In some cases, team-specific roles might be collapsed with similar roles on a higher level. For instance, it might not be necessary to have a separate manager for Hardware Drivers; instead these responsibilities can fall to the overall Software Discipline's manager. However in other cases, it might be advisable to have a full set of roles including a separate manager and architect even on team level, for instance for the business critical Algorithms team.

4.3.2.3 Synchronization between Teams

In section 4.3.1.3, the possible synchronisation process patterns used by the product team to handle development dependencies are described. Pro-PD is extended to handle these development dependencies within the *Component Development* task. The product team can decide on an implementation strategy based on their development needs. Figure 4-8 shows the two development strategies of Pro-PD modelled in EPF.

Option one. The product team waits for the new platform release and then proceeds to design, implement and test customer specific components.

Option two. The product team bases new component development on a model of how the platform will change; this model being an interface or an internal assumptions model. The product team first negotiates the model changes with the platform team before proceeding to develop in parallel. Alternatively, the product team will make assumptions on interface changes, working off expected changes to the interface. If conflicts are detected when the new platform architecture is released, then the product team makes the alterations.

**Figure 4-8 Product Development**

4.3.3 Reaction of Robert Bosch GmbH Workshop Participants to Pro-PD Version One

At the end of the workshop, the researcher presented Pro-PD version one to the workshop participants. The participants were satisfied that version one was a reasonably accurate representation of the product derivation process however there was room for improvement.

The workshop participants recommended that the phases of Pro-PD should be re-organised from four phases into three as the case study participants felt that the phase names did not always reflect the tasks which were contained within that phase. The participants found some phase and task naming caused confusion, for instance when describing the *Product Requirements* as *Product Specific Requirements* the participants found the use of the term ‘specific’ caused confusion. The participants also found that the analysis of product requirements within Pro-PD version one lacked detail and missed important tasks required for preparing for product derivation.

In Table 4-5 a summary of the changes to Pro-PD version one as a result of the feedback and observations from the Robert Bosch GmbH workshop is detailed.

4.3.4 Summary of Pro-PD Changes due to Stage Two of Research

The result of stage two of the research was Pro-PD version two. The changes made from version one to version two are the results of observations from the case study and feedback from the case study participants. In Table 4-5 a summary of these changes are presented along with the primary motivation for these changes.

Change	Motivation
Change phase name from <i>Impact Analysis</i> to <i>Preparing for Derivation</i>	The phase name did not reflect all of the tasks which were contained within that phase. For instance Allocate Requirements had no element of analysis.
Added new task <i>Translating the Customer Requirements</i> to <i>Preparing for Derivation</i> Phase	Translating the customer requirements into internal platform language is an important Robert Bosch GmbH task when preparing for derivation. This prevented terminology confusion and customer

	specific description of assets.
New artefact <i>Glossary</i> input to <i>Translating the Customer Requirements</i> task	Robert Bosch GmbH used a customer terminology glossary to assist in translating specific customer terms to platform specific terminology within requirements specifications
Added artefact <i>Translated Customer Requirements</i> as output of <i>Translating Customer Requirements</i> task	As observed in Robert Bosch GmbH, this artefact is an output of the <i>Translating Customer Requirements</i> task
Modified <i>Map Customer Requirements to Platform Features</i> task name to <i>Coverage Analysis</i>	Robert Bosch GmbH described the task of mapping customer requirements to platform features as <i>Coverage Analysis</i>
Replaced artefact <i>Platform Artefacts</i> with <i>Platform Requirements</i> as input into <i>Coverage Analysis</i> task	In Robert Bosch GmbH coverage analysis was a comparison between the Customer System Requirements Specification and the platform system requirements specification
Renamed <i>Unmapped Customer Requirements</i> to <i>Customer Specific Product Requirements</i>	Customer requirements which did not fall under the scope of the platform were labelled Customer Specific Product Requirements within Robert Bosch GmbH.
Renamed <i>Mapped Customer Requirements</i> to <i>Translated Customer Requirements</i>	Customer requirements which did fell under the scope of the platform were labelled Translated Customer Requirements within Robert Bosch GmbH.
Renamed <i>Form the Product Specific Requirements</i> to <i>Create the Product Requirements</i>	The task of creating the product requirements using the negotiated product requirements, the translated product requirements and the platform requirements used as a baseline was called Create the Product Requirements within Robert Bosch GmbH.
Renamed artefact <i>Product</i>	The use of term ‘specific’ caused confusion for the

<i>Specific Requirements to Product Requirements</i>	case study participants. For instance, was there a difference between ‘specific product requirements’ and ‘product requirements’.
Renamed task <i>Create the Product Specific Test Cases to Create the Product Test Cases</i>	Removed the term ‘specific’ from Create the Product Specific Test Cases. In line with removal of ‘specific’ from Product Requirements artefact
Renamed artefact <i>Product Specific Test Cases to Product Test Cases</i>	Removed the term ‘specific’ from Product Specific Test Cases. In line with removal of ‘specific’ from the task Create the Product Specific Test Cases
Added task <i>Find and Outline Requirements to Preparing for Derivation</i> phase	This task was observed within Robert Bosch GmbH. It involves analysis of the product requirements. The functional and non functional requirements of the system are specified. See Section 4.3.2.1
Added task <i>Allocate Requirements to Preparing for Derivation</i> phase	This task was observed within Robert Bosch GmbH to involve the allocating of requirements from the product system requirements specification to different organisational disciplines within the product team, such as hardware, algo (algorithms) or software disciplines. See Section 4.3.2.1
Renamed <i>Reusability Analysis</i> phase to <i>Product Configuration</i> phase	The original name was deemed confusing to Robert Bosch GmbH workshop participants. There is no analysis aspect to this phase. The primary purpose of this phase is to configure platform components hence the renaming to Product Configuration phase
Renamed and modified <i>Product Integration and Validation</i> phase to <i>Product Development and Testing</i> phase.	In Robert Bosch GmbH the goal of this phase was to perform product specific development and testing. Any specific development or adaptation at product level should occur working closely with the product test teams.

Removed phase <i>Component Development and Adaptation</i>	The tasks of this phase are now contained within the other phases. In the Find and Outline Requirements task in the Preparing for Derivation phase the development scoping decisions are made. The tasks relating to product specific development are moved to the Product Development and Delivery phase.
Added artefact <i>New Platform Release</i> . Input to <i>Component Development</i> task	Results of observations from Section 4.3.2.3
Removed decision ‘ <i>Have the customer requirements change?</i> ’	Within Robert Bosch GmbH, the product requirements were contractually agreed at this point. If there was a change to the Product Requirements this was captured during the task System Testing.

Table 4-5 Summary of Stage Two Changes to Pro-PD

4.3.5 Conclusion of Stage Two: Industrial Case Study

Version two of Pro-PD was based on a literature review, expert opinion workshop series and the product derivation practices of a large automotive systems supplier. The observations from this case study research resulted in augmentations to the original version of Pro-PD. Version two of Pro-PD is included in Appendix D. This version of Pro-PD was peer reviewed [113]. This version of Pro-PD was validated through evidence collected during the Robert Bosch GmbH case study and by way of a mapping with DOPLER^{UCon}, the results of which are discussed in the following section.

4.4 Stage Three - Academic Comparative Analysis: DOPLER^{UCon}

DOPLER^{UCon} (Decision-Oriented Product Line Engineering for effective Reuse: User-centred Configuration) was developed at JKU driven by industry needs with the goal to

define a user-centred, tool-supported product derivation approach. DOPLER^{UCon} was designed to be generic, without focusing on a particular domain.

DOPLER^{UCon} brought a unique academic industry view of product derivation. In research-industry collaboration with Siemens VAI Metals Technologies, the world leader in engineering and plant building for the iron, steel, and aluminium industries, they developed the DOPLER approach to product line engineering. The goal of the collaboration was to support modelling the variability of Siemens VAI's Continuous Casting Level 2 (CC-L2) software system for the automation of continuous casting in steel plants and to support the use of this system's variability in product derivation. As a result, they were deemed a high quality case study candidate for the purposes of this research.

In the course of the study, Pro-PD was compared to the DOPLER^{UCon} approach. While both approaches have been developed with different goals, for different purposes, and in different domains, many interesting parallels were still found.

4.4.1 Preparing for Derivation Comparison

In DOPLER^{UCon} the phase is called configuration preparation. Some of the Pro-PD *Preparing for Derivation* tasks are supported by DOPLER^{UCon} as part of the application requirements engineering phase. In Table 4-6 a summary of which tasks of the Preparing for Derivation phase in Pro-PD are supported by DOPLER^{UCon} is presented.

Preparing for Derivation Task	Purpose	Supported by DOPLER^{UCon}	DOPLER^{UCon} 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. Existing configuration contains requirements for that existing configuration. Mapping to new requirements and documentation therefore is possible).	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 realisation can be defined - Customer requirements are negotiated with the customer).	Relate product-specific requirements to the available variability; negotiate product-specific requirements.
Create the Product Requirements.	Involves merging mapped and negotiated requirements.	Supported (Negotiations lead to changes in captured requirements).	Capture product-specific requirements; negotiate product-specific requirements.
Find and Outline Requirements	The functional and non-functional requirements for the system are specified	Partly supported (captured requirements can be assigned arbitrary types. This can also be used to	Capture product-specific requirements;

	and scoped. Requirements are designated for platform implementation or product specific.	define whether they are platform or product-specific).	
Create Product 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 of particular product requirements.	Partly supported (tasks in derivation models group related decisions. Requirements related with decisions in a task also allocated to a discipline. Prioritisation of implementation iterations not supported in DOPLER ^{UCon}).	Define roles and tasks.

Table 4-6 Mapping Preparing for Derivation to DOPLER^{UCon}

From the seven tasks identified within the *Preparation for Derivation* phase of Pro-PD, it can be seen that six tasks are fully or partly supported by DOPLER^{UCon}. Only one task *Translate Customer Requirements* is not supported.

The missing support for *Translate Customer Requirements* in DOPLER^{UCon} can be explained with the differences in customer management. In a collaborative environment, as assumed by DOPLER^{UCon}, customer requirements are typically delivered in a product line compatible format.

The *Allocate Requirements* task assigns requirements to specific iterations depending on the type of project planning. There are two types of project planning. Manual non-tool supported product derivation projects tended to have ‘*big bang*’ releases after substantial development periods. Automated approaches appeared to be more iterative in nature, as each new version of the product required less effort than the manual approaches. DOPLER^{UCon} does not support iterative development directly and therefore, as it has no need, does not support the *Allocate Requirements* task.

4.4.2 Product Configuration Comparison

Product configuration is focused on the derivation of a product configuration from the product line, i.e., selecting, customising, and integrating reusable assets. In both approaches this phase is called Product Configuration. In Table 4-7 a summary is presented of which tasks of the Product Configuration phase in Pro-PD are supported by DOPLER^{UCon} and how.

Product Configuration Task	Purpose	Supported by DOPLER ^{UCon} ?	DOPLER ^{UCon} Activities involved
Derive New Configuration	Derive a new product configuration from the <i>platform architecture</i> .	<i>Supported</i> (deriving a new product is done by taking decisions. This goes hand in hand with selecting platform components due to the explicit linkage of components with decisions).	Take decisions and customise assets.
Select Closest Matching Configuration	Select a base configuration from existing/	<i>Supported</i> (possible to start with an existing configuration i.e. an	Adapt variability model.

	previous configurations.	existing derivation model. Also possible to “import” decisions from earlier configurations to the current configuration)	
Select Platform Components	Components are selected from the collection of <i>Platform Components</i> for, addition to or replacement of, components in the base product configuration.	<i>Supported</i> (Performed in parallel with deriving a new configuration. The explicit linkage of components with decisions allows selection of platform components by taking decisions in the base configuration derivation model. For components which have to be replaced, the decisions have to be changed accordingly).	Take decisions and customise assets.

Table 4-7 Mapping Product Configuration to DOPLER^{UCon}

From the three Product Configuration tasks, it can be seen that all three tasks are fully or partly supported by DOPLER^{UCon}. One difference between the two approaches is the assumption in Pro-PD that integration testing will be performed in the Product Development and Testing phase. Furthermore, due to the decision-oriented, model-based approach, in DOPLER^{UCon} the base configuration is represented by a dedicated model: the derivation model. In this model a base configuration can be selected or managed by taking decisions. Taking decisions directly leads to the inclusion and/or adaptation of (platform) components which makes the integration of the base configuration and the selected platform components unnecessary. However, the correct “integration” in this case depends on the correctness and completeness of the derivation model and the variability model it is

based on. Currently, DOPLER^{UCon} only provides basic support for validating models in this regard (syntactical correctness and architectural completeness). Further support for semantic correctness and more sophisticated consistency checking is currently under development.

4.4.3 Product Development and Testing Comparison

Product development and testing is the phase where additional development and testing is performed to fulfil customer requirements that cannot be fulfilled by the product line and the variability it provides. In DOPLER^{UCon} the tasks of the Pro-PD *Product Development and Testing* phase occur in the application requirements engineering, additional development, product integration and deployment, and product line evolution phases. In Table 4-8 a summary is presented of which tasks of the Product Development and Testing phase in Pro-PD are supported by DOPLER^{UCon} and how.

Product Development and Testing Task	Purpose	Supported by DOPLER^{UCon}?	DOPLER^{UCon} Activities involved
Identify Required Product Development	Satisfy requirements which could not be satisfied through reuse of platform assets.	<i>Supported</i> (in application requirements engineering additionally required development is identified through mapping with the available variability and negotiation with	Relate product-specific requirements to the available variability; negotiate product-specific requirements.

		the customer).	
Component Development	New components are implemented to satisfy specific functionality or existing platform components are adapted	<i>Supported</i> (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.	<i>Partly Supported</i> (can happen in additional development phase but how to do this is not defined).	Additional development.
Product Integration	The Developed or Adapted Components are integrated into the Integrated Product Configuration.	<i>Supported</i> (main purpose of the product integration and deployment phase).	Product integration and deployment.
Integration Testing	Validates the configuration of the <i>Integrated Product Configuration</i> . The integration test should reuse <i>Platform Test Artefacts</i> .	<i>Partly Supported</i> (assumed to happen in additional development phase but not defined how)	Additional development
System Tests	The product has to be checked for compliance with the product specific requirements [3]. The majority of the	<i>Partly Supported</i> (assumed to happen during product integration and deployment but how to	Product integration and deployment.

	products specifications will be a subset of the product line functionality. Therefore the tests used at platform level can be reused.	do this not defined).	
--	---	-----------------------	--

Table 4-8 Mapping Product Development and Testing to DOPLER^{UCon}

From the six tasks identified within the *Product Development and Testing* phase of Pro-PD, it can be seen that all are fully or partly supported by DOPLER^{UCon}. Those Pro-PD tasks that are only partly supported by DOPLER^{UCon} could be fully supported; however, DOPLER^{UCon} currently provides only partial support for this or assumes domain-specific plug-ins to be developed for this purpose.

4.4.4 Conclusion of Stage Three: Academic Comparative Analysis

Even though the two approaches were developed separately and with different aims and purposes in mind, many interesting parallels and comparably few differences were discovered in the following areas:

- Requirements Management
- Iterative Development
- Customer Involvement
- Tool Support
- User Management

4.4.4.1 Requirements Management

Requirements management is one area where Pro-PD and DOPLER^{UCon} have different approaches. In DOPLER^{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. DOPLER^{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.

4.4.4.2 Iterative Development

The use of *iterative development* cycles is not directly supported in DOPLER^{UCon}. However, additional attributes can be defined for requirements and these can be used to allocate specific 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 prioritisation and customer negotiation.

4.4.4.3 Customer Involvement

Customer involvement in product derivation is typically portrayed as a combative relationship involving negotiation between separate parties with contrasting motivations. This is reflected in the *Customer Negotiation* task in Pro-PD. This is in contrast to customer relationship approaches observed with DOPLER^{UCon}. DOPLER^{UCon} sees customer involvement as a collaborative role, where the customer makes design decisions alongside the derivation team. Good communication where the limitations and opportunities provided by the product line feature set are clearly explained, can nurture a collaborative relationship with the customer.

4.4.4.4 Tool Support

DOPLER^{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 to take decisions. Pro-PD does not define which tasks should be supported by tools and how they can be supported.

4.4.4.5 Role Management

Product derivation *user management* is not directly supported in Pro-PD. While DOPLER^{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 role management.

4.4.5 Summary of Pro-PD Changes due to Stage Three of Research

The result of stage three of the research was Pro-PD version three. In Table 4-6 a summary of the changes made from version two to version three is presented along with the primary motivation for these changes.

Change	Motivation
Remove task <i>Identify Required Component Development</i> .	Identifying product specific development occurs during the initiation of a derivation project within the DOPLER ^{UCon} approach and not during the later <i>Product Development and Testing</i> phase as in Pro-PD
Modify task <i>Find and Outline Requirements</i>	Scoping decisions on required component development and adaptation are decided during the <i>Find and Outline Requirements</i> task in <i>Preparing for Derivation</i> phase. This is in line with both Robert Bosch GmbH and DOPLER ^{UCon} approaches
Add task <i>Create Guidance for Decision Makers to Preparing for Derivation Phase</i>	In DOPLER ^{UCon} arbitrary guidance (e.g. multimedia) can be created for open decisions and then related to product requirements. Guidance linking remaining variability in the product requirements assists in dealing with the complexity associated with

	representing product line variability.
Modify artefact <i>Product Requirements</i> , it contains a link to variability guidance to be used by the <i>Product Architect</i> during the <i>Derive the Product</i> phase	In DOPLER ^{UCon} the arbitrary guidance (e.g. multimedia) created during the task <i>Create Guidance for Decision Makers</i> is related within the <i>Product Requirements</i> .
Add task <i>Product Integration</i> . This task integrates the <i>Base Product Configuration</i> and the selected <i>Platform Components</i>	Previously this integration was performed within the <i>Select Platform Components</i> task. In DOPLER ^{UCon} , the selected base configuration is represented by a derivation model. In this model, the platform components are selected by taking decisions. A separate activity integrates the base configuration and selected platform components.
Add task <i>Integration Testing</i> to <i>Product Configuration</i> phase	Integration testing is performed by the <i>Product Architect</i> and occurs during the <i>Product Configuration</i> phase
Add task <i>Provide Feedback to Platform Team</i> to <i>Product Development and Testing</i> phase	The DOPLER ^{UCon} task <i>Product Line Evolution</i> used feedback passed from the product team to analyse newly developed assets and analyse new requirements.
Removed task <i>Identify Required Component Adaptation</i> .	Decisions on required component development and adaptation are now decided during the <i>Find and Outline Requirements</i> task in <i>Preparing for Derivation</i> phase.
Add artefact <i>Platform Feedback</i> . This is an output of the task <i>Provide Feedback to Platform Team</i> .	Artefact used to transfer platform feedback to the platform team. It is an output of the <i>Provide Feedback to Platform Team</i> task.
Modify task <i>Allocate Requirements</i> . Responsibility for	In the DOPLER ^{UCon} task <i>Define role and Task Structures</i> , variability is managed through assigning

implementation of specific requirements is allocated to team members who hold the roles of <i>Product Developer</i> and <i>Product Architect</i>	responsibility to different members of the product team
Modify task description of <i>Select Platform Components</i>	Based on the responsibility defined during <i>Allocate Requirements</i> task, responsibility for binding variability is allocated. This is similar to the DOPLER ^{UCon} approach - <i>Define Roles and Responsibilities</i> and <i>Adapt Variability Model</i> tasks.

Table 4-9 Summary of Stage Three Changes to Pro-PD

Pro-PD version three is the output of this stage of the research. Version three of Pro-PD is included in Appendix E and is an extract from [98]. In the following section it is evaluated.

4.5 Stage Four: Evaluation

4.5.1 Evaluation Method

The final stage of this research was to evaluate Pro-PD version three. To evaluate Pro-PD, support for its tasks in prominent existing approaches were systematically analysed. The evaluation was conducted using two different approaches (see Figure 4-9), namely an inter-model evaluation with the SEI Product Line Practice Framework (PLPF) and a systematic analysis with COVAMOF and PuLSE-I.

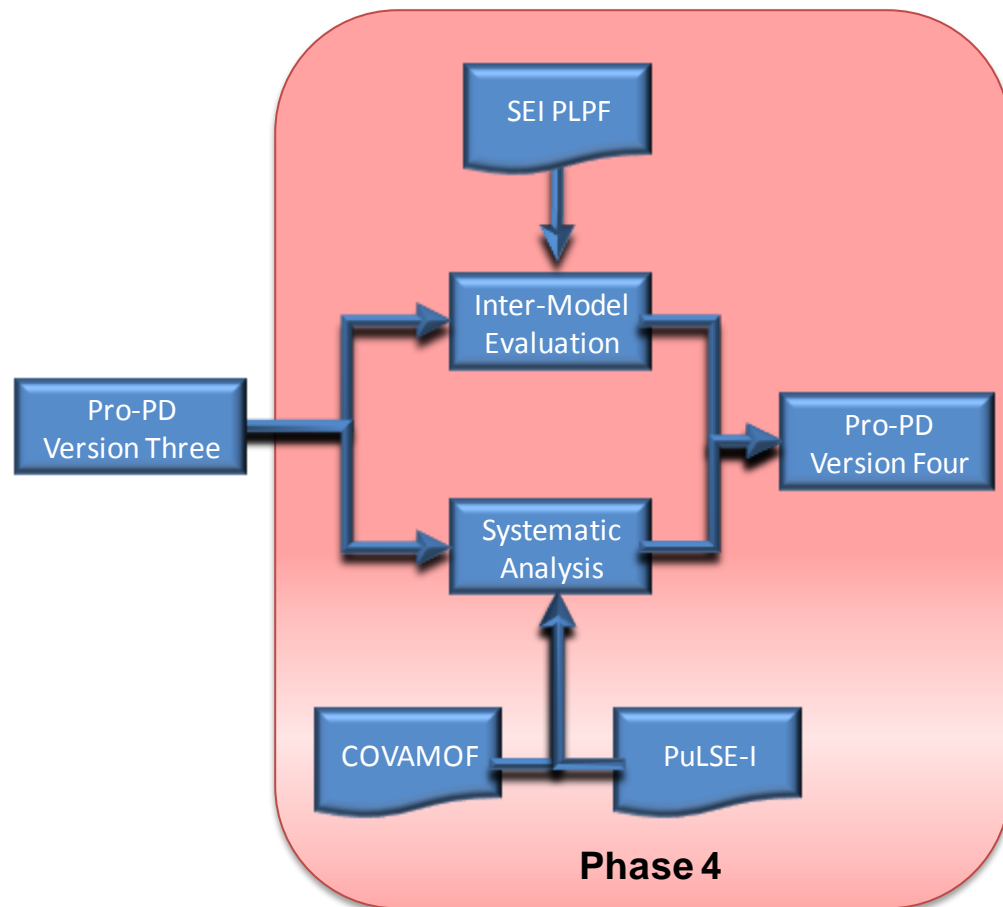


Figure 4-9 Systematic Evaluation

4.5.2 Inter-model Evaluation

To evaluate the model, Pro-PD was reverse engineered to examine how it satisfies product derivation according to the PLPF. Firstly, the PLPF addresses many aspects of SPL so practices relevant to Pro-PD must be identified. The evaluation of Pro-PD using these practices is described and the results of the evaluation are discussed.

4.5.2.1 Identifying Relevant Practices

Only a subset of the 29 practices listed in the PLPF are relevant to product derivation and within the scope of Pro-PD. The PLPF Practice Patterns will be used to identify which practices are relevant to product derivation.

Patterns are a common way of expressing common contexts and problem-solution pairs. Design patterns are a well-known example in software. In software product line engineering, patterns can be used to show how aggregations of practice areas can be orchestrated to solve recurring problems. The PLPF practice patterns can be used to help select 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: 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 that address the solution and provide the structure for the *Product Builder* pattern are

- Requirements Engineering
- Architecture Definition
- Architecture Evaluation
- Component Development
- Testing
- 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. All 29 practice areas are included in the pattern. Although every practice area is supportive of the essential activities in one way or another, some provide stronger support than others. The mapping provides a helpful foundation for anyone attempting a software product line approach and a big picture of the necessary skill sets and practice areas necessary to accomplish each of the three essential activities.

However not all of the practices in the *Essentials Coverage* pattern assigned to product development are suitable to be used as a basis for evaluation. Firstly, there are too many practice areas in each group to provide much assistance in dividing and conquering a product line approach. As Clements and Northrop [1] explain, “it is a pedagogical pattern rather than operational pattern”. Furthermore aspects of the pattern fall outside the scope of

the research. Configuration Management, Data collection, Measurement and Tracking, Technical Risk Management, Development of an Acquisition Strategy and Training practices are all considered outside the scope of Pro-PD at this stage of its development.

According to Czarnecki & Eisenecker [114] in order to successfully introduce and run, domain and application engineering in an organisation, it is necessary to address a wealth of issues. In particular they highlight that there are management and organisational issues concerning establishing both processes and the feedback loops between them. Therefore, a number of additional practices that are outside of the *Product Builder* pattern but which are within the scope of Pro-PD are examined. For instance, Pro-PD considers customer management and assigning roles and responsibilities within the team structure. For that reason, the Customer Interface Management, Operations and Structuring the Organisation practices are within the scope of the model.

Therefore, the relevant practices which will be considered in the evaluation are:

Practice
Requirements Engineering
Architecture Definition
Architecture Evaluation
Component Development
Testing
Software System Integration
Operations
Customer Interface Management
Structuring the Organisation

Table 4-10 Relevant PLPF Practices

4.5.2.2 Performing the Evaluation

A practice area is a body of work or a collection of activities that an organisation must master to successfully carry out the essential work of a product line [1]. For each practice area those sections relevant to product derivation will be emphasised. To do this, in the

practice descriptions special attention is paid to the sections: “*Aspects Peculiar to Product Lines*”, “*Application to Product Development*”, “*Specific Practices*” and “*Practice risks*”.

For each practice a short summary on how that practice relates to product derivation is provided. A set of criteria are identified within the practice summary and then labelled as a practice criterion. Pro-PD is judged as to whether it satisfies or does not satisfy each criterion. Table 4-11 shows the evaluation for the Requirements Engineering practice area. For the other practices, see Appendix A.

Practice Name	Requirements Engineering (RE)
Application to Product Development	<p>Product line requirements define the products in the product line together with the features of and the constraints on those products. Product Requirements common across the product line are written with variation points that can be filled in or exercised to create product-specific requirements (RE-1). The product line requirements guide the elicitation of the specific requirements for that product (RE-2). In product development, requirements engineering plays a key role in determining the feasibility of producing a particular product as part of the product line (RE-3).</p> <p>Requirements play a role in the production and testing of the particular product, just as they do for single-system development (RE-4).</p> <p>Product-specific requirements often "grow up" to become product line requirements if they can be slightly generalized or if they pop up in more than one product. That is the primary mechanism for the evolution of software product lines over time (RE-5).</p>
Key Criteria	<p>RE-1: The platform requirements are used as a baseline to create the product requirements</p> <p>RE-2: The product line requirements guide the elicitation of the</p>

	<p>specific requirements for that product</p> <p>RE-3: The platform requirements are used during product scoping to determine feasibility of the PL</p> <p>RE-4: The product requirements are input to production and testing</p> <p>RE-5: Specific Product Requirements are suggested for adoption by the platform</p>
Pro-PD	<p>RE-1: In the Task <i>Create the Product Requirements</i>, the organisation uses the <i>Platform Requirements</i> as a baseline to create the <i>Product Requirements</i>. This satisfies RE-1.</p> <p>RE-2: The <i>Platform Requirements</i> are used in the primary requirements elicitation tasks, <i>Coverage Analysis</i> and <i>Create the Product Requirements</i> tasks. The Platform Requirements are used as a guide for <i>Coverage Analysis</i> and taken as a baseline for the <i>Product Requirements</i> in <i>Create the Product Requirements</i>. This satisfies RE-2.</p> <p>RE-3: The task <i>Coverage Analysis</i> is at its simplest a comparison between of the <i>Translated Customer Requirements</i> and the <i>Platform Requirements</i>. Through <i>Coverage Analysis</i> which <i>Translated Customer Requirements</i> are covered by the platform is discovered. This satisfies RE-3.</p> <p>RE-4: Pro-PD uses the <i>Product Requirements</i> in <i>Product Development and Testing</i> phase.</p> <p>RE-5: In the <i>Provide Feedback to Platform Team</i> task, feedback is provided to the platform team on areas for improvement within the platform specifically product requirements which should be adopted by the platform. This satisfies RE-5.</p>

Table 4-11 Evaluation of Requirements Engineering Practice Area

The practice summary table below presents the results of the evaluation for each of the nine practices.

Practice	Key Criteria	Pro-PD Satisfies the Criteria
Requirements Engineering	RE-1	Yes
	RE-2	Yes
	RE-3	Yes
	RE-4	Yes
	RE-5	Yes
Architecture Definition	AD-1	Yes
	AD-2	Yes
	AD-3	Yes
Architecture Evaluation	AE-1	No
	AE-2	No
Component Development	CD-1	Yes
	CD-2	Yes
	CD-3	Yes
	CD-4	Yes
	CD-5	Yes
	CD-6	Yes
Testing	T-1	No
	T-2	Yes
	T-3	Yes
	T-4	Yes
	T-5	Yes
Software System Integration	SSI-1	Yes
	SSI-2	Yes
	SSI-3	Yes

	SSI-4	Yes
Operations	O-1	Yes
	O-2	No
	O-3	Yes
	O-4	Yes
Customer Interface Management	CIM-1	Yes
	CIM-2	Yes
	CIM-3	Yes
Structuring the Organisation	SO-1	Yes
	SO-2	Yes
	SO-3	Yes
	SO-4	Yes
	SO-5	Yes

Table 4-12 Practice Summary Results

Based on the above table (see Table 4-12), a classification on the extent to which each practice is satisfied by Pro-PD is formed. Classification is based on the following:

- *Satisfies Practice.* Pro-PD meets all of the practice criteria and is said to satisfy this practice
- *Partially Satisfies Practice.* Pro-PD meets some but not all of the practice criteria and so is said to somewhat satisfy the practice
- *Does not Satisfy Practice.* Pro-PD meets none of the practice criteria and so does not satisfy the practice

In the following table a summary of the practice analysis results is presented.

Practice	Result
Requirements Engineering	<i>Satisfies Practice</i>
Architecture Definition	<i>Satisfies Practice</i>
Architecture Evaluation	<i>Does not Satisfy Practice</i>
Component Development	<i>Satisfies Practice</i>
Testing	<i>Partially Satisfies Practice</i>
Software System Integration	<i>Satisfies Practice</i>
Operations	<i>Partially Satisfies Practice</i>
Customer Interface Management	<i>Satisfies Practice</i>
Structuring the Organisation	<i>Satisfies Practice</i>

Table 4-13 Practice Classification

4.5.2.3 Discussion of Results

As a result of the inter-model evaluation, Pro-PD is adapted. From the above table (see Table 4-13), it can be seen that Pro-PD completely satisfies six of the nine relevant practices.

Pro-PD only partially satisfies the Operations practice as it does not consider software maintenance which is outside of the scope of the research (see Section 1.6).

Pro-PD only partially satisfies the Testing practice as Pro-PD has no verification of artefacts between the different phases of the process. Therefore there should be verification of artefacts. In the *Preparing for Derivation* phase, a new task *Verify the Product Requirements* is included during which the new product requirements and the complete product requirements are reviewed by the *Customer*, *Platform Manager* and *Product Manager*.

Pro-PD does not satisfy the Architecture Evaluation practice as there is no separate testing of the *Product Architecture* in the *Product Configuration* phase. Therefore, an *Evaluate Product Architecture* task is included between the *Derive New Configuration* and *Select Platform Components* tasks. The task evaluates the instantiated *Product Architecture* to see if it meets the specific behavioural and quality requirements of the

product at hand. The results of the evaluation are sent to the Platform Team in the *Provide Feedback to Platform Team* task.

4.5.3 Analysis of Existing Approaches

For the second part of the evaluation support for Pro-PD tasks within existing approaches is analysed. While a framework for evaluating product derivation approaches does not exist, a framework developed for the purpose of evaluating software product line architecture design methods [99] was adapted. The objective is not to judge their usefulness or compare them in detail but to find out whether and how they support the key activities defined in Pro-PD. COVAMOF [33] and PuLSE-I [24] were chosen because of they are well-known in the community, mature enough for such an analysis and have been validated in different domains.

4.5.3.1 Adapting the Framework

The original Martinlassi [99] framework considers software product line architecture design methods focusing on method context, user, structure and validation. The purpose of the framework was to study and compare existing approaches for their design of software product line architectures. In this section, the adaption of the framework for the purposes of studying and comparing product derivation approaches is described.

Category	Elements	Questions
<i>Context</i>	Specific Goal	What is the specific goal of the product derivation approach?
	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 results of the approach?
<i>User</i>	Target Group	How are the stakeholders addressed by the approach?
<i>Contents</i>	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)?

	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	What activities/sub-activities does the approach include that are not covered by the defined key activities/sub-activities?
<i>Validation</i>	Maturity	Has the approach been validated in practical

		industrial case studies?
--	--	--------------------------

Table 4-14 Evaluation Framework (adapted from [99])

The questions regarding the category context proposed by Matinlassi were adapted from “product line architecture design method” to “product derivation approach”. One element of the framework for the category “User (target group)” was adapted, as the focus of the evaluation is on evaluating the contents (support for key activities) and not the user support. For the category contents, the first two elements structure and artefacts were adapted. Instead of focussing on product line architecture (elements defined by Matinlassi: architectural viewpoints, language, variability, tool support), one element for each key phase of product derivation was defined. For the validation category, maturity was adopted and not quality because there was no interest in the approaches’ procedures to validate the quality of the results of the product derivation approach in this context. Table 4-14 depicts the adapted evaluation framework.

4.5.3.2 Analysis

In the table below, the results of the analysis using the adapted framework are presented.

	What is the specific goal of the product derivation approach?
COVAMOF	To support the construction of a software product by selecting and configuring product family artefacts in an iterative process. The goal of this process is to allow "organisations to gain maximum benefit from COVAMOF and its associated tool support" [22].
PuLSE-I	PuLSE-I is centred on supporting the instantiation of the product line infrastructure, i.e., to use the product line infrastructure to create and maintain one member of the product line.
	What aspects of product derivation does the approach cover?
COVAMOF	The approach focuses on product configuration. It only partly covers Preparing for Derivation activities and product specific development

	activities.
PuLSE-I	PuLSE-I focuses on the whole application engineering process and covers preparing for derivation, product configuration and, additional development and testing.
	What is/are the application domain(s) the approach is focused on?
COVAMOF	The approach has been applied in different domains and is generic enough to be usable in arbitrary domains. However, adaptations to the tool support (COVAMOF-VS) are required depending on the usage context.
PuLSE-I	The approach is not focused on a specific application domain as it is defined in a fairly generic manner.
	What is the starting point for the approach?
COVAMOF	The first step in the COVAMOF Derivation Process is to create a Product entity in the COVAMOF variability model. Inputs to derivation are the customer requirements (functional and non-functional) which are assumed to be available
PuLSE-I	A customer or the management has a product request that falls under the scope of the product line. Inputs to PuLSE-I are the product request and the scope definition (created in PuLSE-Eco).
	What are the results of the approach?
COVAMOF	A product derived from the product family that meets the customer requirements
PuLSE-I	A product instantiated from the product line comprising of specification, architecture and code - partly reused from the product line and partly developed during instantiation (in an iterative manner)
	How are the stakeholders addressed by the approach?
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	Customer/Management are explicitly considered as providing the input in the form of product requests. Project management is addressed with the output of the 'plan for product line instance' step. This artefact is called the project plan. Product construction and other actual instantiation activities are assumed to be performed by dedicated application engineers.
	What activities/steps/sub-processes does the approach define to accomplish product derivation?
COVAMOF	<p>Product Definition: Creating a product entity involves defining the customer and the product name</p> <p>Product Configuration: Binding of variation points, based on customer requirements</p> <p>Product Realisation: 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</p> <p>Product Testing: Determining whether the product meets the customer requirements and deciding whether an additional product configuration/realization/testing iteration is required)</p>
PuLSE-I	<p>Plan for the Product Line Instance: Determine whether all characteristics of the required product are covered by the product line infrastructure, if not: determine the overlap between the required system and the current scope and estimate the realization effort for features beyond the product line scope.</p> <p>Create project plan: Define what is product-specific and what can be fulfilled by the product line.</p> <p>Instantiate and Validate Product Line Model: Incrementally resolve decisions in the product line model driven by the customer (represent requirements for the whole product line). When a characteristic is not</p>

	<p>covered by the product line, its requirements are directly elicited.</p> <p>Instantiate and Validate Reference Architecture: Instantiate variabilities to create an "intermediate architecture" from the product line; then modify these intermediate architecture to accommodate instance-specific requirements)</p> <p>Product Construction: Lower level design, implementation, and testing based on reusable assets as far as possible: reuse existing product line assets, implement non-existing product line assets, implement product-specifics; also perform acceptance testing (have customer requirements been fulfilled?)</p>
	What artefacts are created and managed by the approach?
COVAMOF	<p>Product Entity: Solution in Visual Studio using COVAMOF-VS Tool. Created in product definition with selected variants (selected during product configuration). Selected variants have "effectuation actions" that are executed during product realization to actually build the product, e.g., by generating binaries and configuration files.</p>
PuLSE-I	<p>Detailed Project Plan: Output of "plan for product line instance" activity</p> <p>Requirements Specification: Output of "instantiate and validate product line model"</p> <p>Product Architecture: Output of "instantiate and validate reference architecture"</p> <p>Product Ready for Delivery: Output of activity "product construction"; comprising specification, architecture, and code</p> <p>Product Configuration: Output of the all aforementioned activities; comprising domain decision model instance, architecture decision model instance, and low level configuration.</p>
	How is the Preparing for Derivation phase supported by the approach?
COVAMOF	Not Supported: Preparing for derivation tasks are not considered by

	<p>COVAMOF. Customer requirements are assumed to be available and phrased so that engineers can perform product configuration and testing based on these requirements (no translation of customer requirements). Mapping of customer requirements to the base configuration is not part of preparing derivation (i.e., product definition) but rather assumed to be done manually by engineers during product configuration. Defining role and task structures is not considered at all; COVAMOF assumes "engineers" to do the work supported by COVAMOF-VS. Creating guidance for decision-makers is not considered as part of the product derivation process but may be done in variability modelling.</p>
PuLSE-I	<p>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 inside the scope of the product line. Overlaps are evaluated and required system-specific developments are defined. The task Translate Customer Requirements is not supported whereas the task Map Customer Requirements is supported. The output of PuLSE-I is "a set of characteristics upon which the customer (or the marketing department) and the developers have agreed". Task Select 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, their roles and tasks, on a very high-level. Sub activity Define Role and Task Structures can therefore only be seen as partly supported. Guidance for decision-makers is assumed to be provided by the product line decision model. No explicit defining of such guidance is defined as part of PuLSE-I.</p>
	<p>How is the Product Configuration phase supported by the</p>

	approach?
COVAMOF	Fully Supported: The engineer creates a new Product entity in the COVAMOF variability model (this product entity can be seen as the base configuration in Pro-PD terminology). Engineers select variants by specifying values at variation points. COVAMOF-VS supports this with its 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. The first determines variation points which can be bound automatically, the latter checks entered values and made selections for their correctness. Over time (and in subsequent iterations), more and more variants are selected for the product entity. Each selected variant can have "effectuation actions" that can be executed by COVAMOF-VS to realize the product (Product Realization activity of COVAMOF), e.g., by creating make files or settings files. In Pro-PD terminology, this can be seen as a partial product configuration (task Create Partial Product Configuration).
PuLSE-I	Fully Supported: Selection of assets is part of PuLSE-I activity instantiate and validate product line model (resolve decisions driven by the customer). Create Partial Product Configuration is part of PuLSE-I activities instantiate and validate reference architecture (instantiate variabilities to create "intermediate architecture" from product line) and product construction (low-level configuration based on reusable product line assets)
	How is the Product Development and Testing supported by the approach?
COVAMOF	Partly Supported: The System Testing task is fully supported (Product testing activity of COVAMOF determine whether the product meets both the functional and the non-functional requirements). However, additional component development/testing as well as component

	integration with partial product configuration/integration testing sub-activities are not explicitly addressed.
PuLSE-I	Fully Supported: Part of PuLSE-I activity product construction is: 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.
	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 the activities defined by the COVAMOF derivation process.
PuLSE-I	Apart from activities that are considered as application engineering and not product derivation (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 PL domain engineering
	Has the approach been validated in practical industrial case studies?
COVAMOF	Sinnema and Deelstra [115] report on an industrial validation of the COVAMOF framework. They focus on showing how the use of COVAMOF (supported by the COVAMOF-VS tool) 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”. The usefulness for both stakeholder groups is shown. COVAMOF has been validated in three industrial product families [116]; two of them are reported in more detail in [3]. However, the papers merely describe the case companies and their software product families, but provide no details about how COVAMOF has been applied in the cases.
PuLSE-I	The PuLSE approach has been applied in case studies, for example

	[117]. Atkinson et al. [25] claim the approach to have been used in various contexts.
--	---

Table 4-15 Results of Analysis Using Evaluation Framework

In the following sections, the support by each approach for the three phases of Pro-PD is discussed.

4.5.3.3 Support for Preparing for Derivation

COVAMOF provides only weak support for preparing for product derivation. The approach assumes "engineers" perform all the work without clear responsibilities and does not consider the definition of role or task structures. The approach also consists of tool support (COVAMOF-VS) to guide engineers in product derivation but does not consider defining explicit guidance for decision-makers. COVAMOF assumes customer requirements to be readily available (like DOPLER).

COVAMOF provides no support for *Translating Customer Requirements, Mapping Customer Requirements to Platform* and *Defining Roles and Task Structure*.

COVAMOF provides partial support for *Create Guidance for Decision-Makers*. The COVAMOF-VS tool provides guidance by giving additional configuration information about the product at hand through variability views. 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.

PuLSE-I provides support for a number of Preparing for Derivation activities. For the task Map Customer Requirements to Platform, PuLSE-I describes the determination of whether the requested product is inside the scope of the product line; how overlaps should be evaluated and required system-specific developments identified. PuLSE-I partly supports the task *Define Role and Task Structures*. PuLSE-I defines the involved stakeholders, their roles and tasks, on a very high-level. PuLSE-I provides no support for *Translate Customer Requirements*. The *Customer Requirements* are assumed to be available and phrased so that they can be used to determine whether the requested product

in inside the scope of the product line. For the task *Create Guidance for Decision Makers*, PuLSE-I assumes that guidance is provided by the product line decision model and no explicit defining of guidance is defined as part of PuLSE-I.

4.5.3.4 Support for Product Configuration

COVAMOF supports the task *Derive New Configuration* by creating a new product entity and uses the inference engine to determine which variation points can be bound automatically; COVAMOF-VS validation engine determines whether the settings and selections are valid to improve usability also by non-experts. A *Partial Product Configuration* is iteratively created by selecting variants and through configuration created by COVAMOF-VS tool during product realization activity of COVAMOF.

PuLSE-I fully supports *Select Subset of Existing Components*. This is part of PuLSE-I activity instantiate and validate product line model where decisions are resolved through the customer. The task *Derive New Configuration* is supported by part of PuLSE-I instantiate and validate reference architecture. Variabilities are instantiated to create an intermediate architecture from the product line.

4.5.3.5 Support for Product Development and Testing

COVAMOF fully supports *System Testing* through the product testing activity in COVAMOF. This determines whether the product meets both the functional and the non-functional requirements. COVAMOF provides no support for the tasks *Implement the Solution*, *Run Developer Tests*, *Product Integration* or *Integration Testing*.

PuLSE-I fully supports *Component Development* and *Component Testing*, *Product Integration*, *Integration Testing* and *System Testing*. These are all supported as part of PuLSE-I activity product construction.

4.5.4 Discussion of Results

From the analysis, it can be seen that the COVAMOF approach is highly tool focussed, concentrating primarily on product configuration. PuLSE-I, on the other hand, has a larger scope. It includes product maintenance and release and not ‘only’ product derivation.

COVAMOF assumes ‘engineers’ to be responsible for the tasks in the derivation process and does not explicitly define roles, while PuLSE-I integrates other roles (customer/management/project manager) into the process.

PuLSE-I is not an isolated description of product derivation but has many dependencies to other parts of the PuLSE methodology. For instance, PuLSE-I adapts the scope of the product line to be able to address new customer requirements by sending change requests to other parts of the PuLSE process. It does this by modifying product line models making PuLSE highly iterative but also highly complex.

In PuLSE-I, the product team make a request to the platform activity PuLSE-Eco for assistance in product scoping. For instance PuLSE-I sends requests to adapt the scope of the product line to be able to address new customer requirements. Pro-PD by comparison is far more isolated from the domain engineering activities. It has two primary feedback loops to the platform team, In the *Coordinate with Platform Team* task, feedback is provided to the platform team on core asset usage during the project. In the *Identify and Refine Requirements* activity, the platform team receives the platform software requirements containing the required extensions to the existing platform in order to facilitate the new *product specific requirements*.

PuLSE-I with its extensive preparation for derivation description, does not explicitly consider *Create Guidance for Decision Makers*. Also roles and task structures are only generally defined in a rather high level by the PuLSE-I process and this is not seen as an activity of preparation for derivation (or i.e., plan for instantiation).

The Pro-PD task *Translate Customer Requirements* is not considered a specific product derivation task within existing approaches. The task *System Testing* should be renamed Acceptance Testing in line with existing approaches.

Both COVAMOF and PuLSE-I perceive derivation as an iterative process. COVAMOF includes explicit product testing for deciding whether to deliver or perform additional iterations. In the following chapter, the waterfall instantiation of Pro-PD is described. COVAMOF includes explicit product testing for deciding whether to deliver or perform additional iterations. PuLSE-I focuses on being an iterative approach to product derivation despite that the approach is described in a waterfall like manner. Therefore, according to these approaches product derivation is an iterative process, a point which is addressed in Chapter Six.

4.5.5 Summary of Pro-PD Changes due to Stage Four of Research

Change	Motivation
Added task <i>Verify the Product Requirements</i> to <i>Preparing for Derivation</i> phase	According to the SEI PLPF within the different activities of Pro-PD, there should be verification of artefacts. For instances, in <i>Preparing for Product Derivation</i> phase, a new task <i>Verify the Product Requirements</i> is required during which the new product requirements and the complete product requirements are reviewed by the <i>Customer, Platform Manager</i> and <i>Product Manager</i> .
Added task <i>Evaluate Product Architecture</i> to <i>Product Configuration</i> phase	Pro-PD does not satisfy the SEI PLPF Architecture Evaluation practice as there is no separate testing of the <i>Product Architecture</i> in the <i>Product Configuration</i> phase. Therefore, an <i>Evaluate Product Architecture</i> task is included between the <i>Derive New Configuration</i> and <i>Select Platform Components</i> tasks. The task evaluates the instantiated <i>Product Architecture</i> to see if it

	meets the specific behavioural and quality requirements of the product at hand.
Changed task name from <i>System Testing</i> to <i>Run Acceptance Tests</i>	The task System Testing should be renamed Acceptance Testing in line with existing approaches

Table 4-16 Summary of Stage Four Changes to Pro-PD

4.6 Formalising the Model

The researcher is using the Eclipse Process Framework (EPF) to model the product derivation process and create a formalised version of Pro-PD. By enabling inbuilt process variability within EPF process content can be selected, tailored and or removed from a process in order to strike the right balance for a particular situation. Therefore process models can be adapted and customised for a particular industry and organisation.

This has the potential for making Pro-PD 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 it was 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 4-5 the discipline mapping task within the Impact Analysis activity modelled using EPF is observed. However this particular activity may not be necessary when developing a product line in another domain it is therefore domain specific. A larger example of this situational specific process content can be seen in Chapter Six. This process flexibility allows the modelling of 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 [118].

4.6.1 Describing Pro-PD using EPF – Introducing Activities

Pro-PD versions one, two and three are described using the process building blocks of tasks, roles and work products. Pro-PD version four is described using EPF. However EPF also uses a process element called activities. Activities are a collection of related tasks, which have a specific development goal. Activities are used as the building blocks from which different development phases can be constructed. Pro-PD version four groups related tasks within phases and describes these groupings of related tasks as activities.

The introduction of activities in Pro-PD assists in the instantiation of Pro-PD for different process characteristics. The use of activities in process model instantiation is demonstrated in section 5.3 through the waterfall instantiation and section 6.4.1 in the Agile instantiation.

4.7 Summary

The objective of this chapter is to describe the development and evaluation cycles that Pro-PD has undergone. In Section 4.2 the development of version one of Pro-PD provides a snapshot into the design decisions made through an analysis of key SPL literature and discussions with SPL experts. How the development of version two of Pro-PD was influenced by observed industry practice at Robert Bosch GmbH is described in Section 4.3. Version three of Pro-PD was influenced by an academic comparative analysis with DOPLER^{UCON} and is described in Section 4.4. The final version of Pro-PD was the result of an evaluation with the SEI PLPF and prominent existing approaches to product derivation. The initial part of the evaluation is mapping the model to an existing standard. The SEI Product Line Practice Framework was analysed, practices that were relevant to product derivation and within the scope of this research were identified. Pro-PD was evaluated and judged as to whether it covered product derivation as defined by the practices of the SEI PLPF. The second part of the evaluation, support for key Pro-PD tasks within existing approaches was analysed. To evaluate Pro-PD, prominent existing approaches were systematically analysed for their support for the defined tasks.

SPL has been influenced by both academia and industry, a process reference model for product derivation should reflect this. Pro-PD has been influenced by a mixture of literature, research projects, expert opinion and industry practice. This chapter demonstrated the differences and commonalities between each of these sources. How these differences have impacted the decisions made during the development of Pro-PD was discussed, and commonalities have been used to validate Pro-PD. Furthermore, results were published and feedback was received for each stage of the research [111-113, 119].

Differences between Pro-PD and existing approaches can be an important source of creativity for identifying problems and opportunities for improvement. In general, the evaluation exercise has been a reassuring exercise. For each stage of the evaluation, there is a discussion of results. Amendments to Pro-PD based on the evaluation are described.

On a final point, many of the aspects of the comparison came down to the level of granularity which is used to define aspects of Pro-PD. As they are phrased now, they are at a high enough level to allow implementation in different organisations and low enough to get an impression of what should be done in product derivation without being an experienced product line researcher.

In the next chapter the final version of Pro-PD is described in detail.

Chapter Five: Pro-PD Description

5.1 Introduction

In this chapter the main contribution of the research, Pro-PD, is presented. Pro-PD is a minimally sufficient process reference model for product derivation. This means that only fundamental product derivation process content is included and that it is independent of the methods and techniques used to derive a product. Pro-PD focuses on the activities, roles and work artefacts used to derive products from a software product line. Pro-PD is adaptable; it can be used as a foundation from which company specific product derivation process content can be developed.

Pro-PD is a software development process that is minimal, complete, and adaptable.

- Minimal – only content that is seen as fundamental for product derivation is included
- Complete – it can be manifested as an entire process to build a system
- Adaptable – it can be adapted to fit particular process characteristics

This chapter describes Pro-PD. The chapter begins with an overview of the model concepts and how they interact (See Section 5.2). The roles (See Section 5.2.1), work products (See Section 5.2.3) and tasks are described (See Section 5.2.2). The waterfall instantiation of Pro-PD is described in Section 5.3.

5.2 Process Description

The process building blocks of Pro-PD are roles, work products, tasks and phases. Roles represent a set of related skills and responsibilities. Work products are artefacts that are produced, modified or used by tasks. Tasks are assignable units of work that usually consume or produce one or more products. Phases are collections of related tasks that share

common goals and allow the process be presented at a high level. In Figure 5-1 a conceptual model of the relationship between these model facets can be seen.

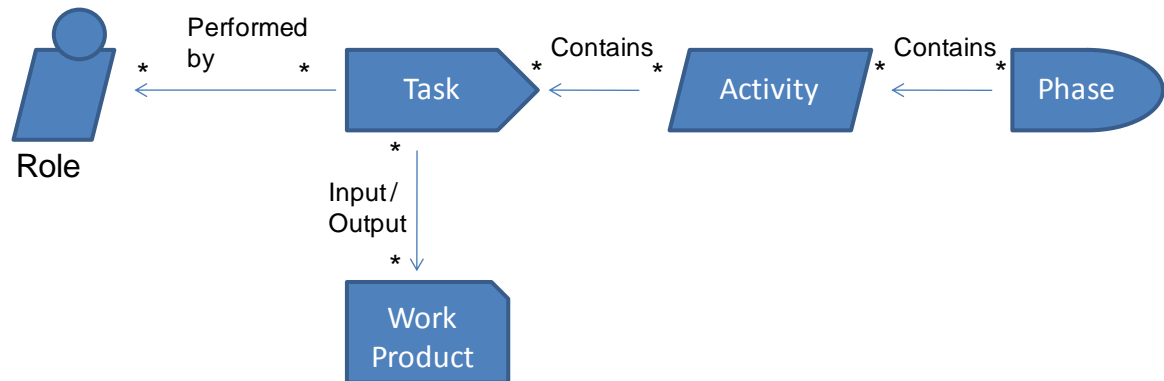


Figure 5-1 Conceptual Model of Pro-PD Structure

In the following sections the various roles, work products and tasks found within Pro-PD are described.

5.2.1 Roles

Despite attempts to automate product derivation, it remains a human activity in which tasks are performed through collaboration and the exchange of work. In Pro-PD there are several roles that represent the different responsibilities which occur during product derivation. These roles are:

- Customer
- Platform Manager
- Product Architect
- Product Developer
- Product Manager
- Product Tester

These roles are assigned to specific tasks which create and modify the different work products. The concept of roles in Pro-PD do not represent individual people but are

instead ‘hats’ that people assume responsibility for when working together. In the following sections the roles and their responsibilities are described.

5.2.1.1 Customer

The *Customer* represents the 'work' in the project. They are responsible for defining what product to build and determining the priority of features. The *Customer* can be external to the SPL organisation, a member of the client organisation who works with the SPL organisation or alternatively the *Customer* can be an internal role played by a member of the SPL organisation.

The *Customer* plays a key role in the *Customer Negotiation* and *Verify the Product Requirements* tasks. During *Customer Negotiation*, *Customer Specific Product Requirements* which cannot be satisfied by existing assets are negotiated. In *Verify the Product Requirements*, the *Product Requirements* are reviewed and verified by the key stakeholders.

The role requires good communication skills, in-depth knowledge of the problem domain and the ability to make key product specification decisions.



Figure 5-2 Customer Role

5.2.1.2 Platform Manager

The *Platform Manager* represents the interests of the platform during the derivation project. The role is responsible for requirements management within the platform, defining the platform scope, platform delivery to the product team and for managing the reusability of components (e.g. reuse degree of platform assets). The role should have a degree of understanding on the demands of the product team. The *Platform Manager* activities include *Verify the Product Requirements* and *Provide Feedback to Platform Team*.

The role requires extensive domain knowledge and experience. The *Platform Manager* should have an appreciation for the demands on the product team.



Figure 5-3 Platform Manager Role

5.2.1.3 Product Architect

The *Product Architect* is responsible for the major technical decision making within the derivation project. The role performs many of the tasks associated with a traditional requirements analyst role.

The *Product Architect* is responsible for the elicitation, creation and verification of *Product Requirements* during *Coverage Analysis*, *Create the Product Requirements* and *Verify the Product Requirements*. The *Product Architect* through the *Product Manager* makes specific platform requirement requests to the *Platform Manager*. These requests are contained in the *Product Specific Platform Requests* during *Find and Outline Requirements*. The *Product Architect* assigns product requirements to product specific component developers in *Allocate Requirements*. The *Product Architect* is responsible for defining the product architecture, ensuring the correct leveraging of platform artefacts according to the artefacts' attached documentation which are conducted in *Derive New Configuration* and *Select Closest Matching Configuration*. The *Product Architect* is responsible for the assembly of the product from the platform assets and the customer specific components during *Product Integration* and *Integration Testing*. The *Product Architect* links technical reasoning for decisions into the product requirements during *Create Guidance for Decision Makers*. The *Product Architect* is responsible for *Evaluate the Product Architecture*, the results of which are passed to the platform team during the *Provide Feedback to Platform Team* task. The *Product Architect* performs the task *Identify Required Product Development* where the additional development required to satisfy the Product Requirements is identified.

The *Product Architect* should have good knowledge of the platform and an understanding of the demands on the platform team. The role requires an expert knowledge

in requirements engineering and communication skills - verbally and in writing, knowledge of the domain and platform terminology.

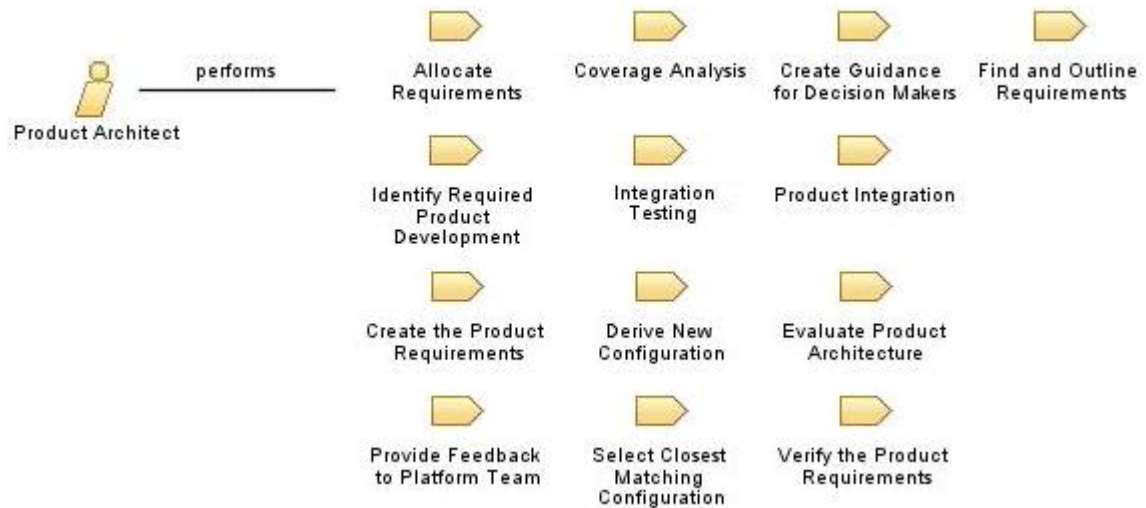


Figure 5-4 Product Architect Role

5.2.1.4 Product Developer

The *Product Developer* is responsible for *Component Development* and *Component Testing*. This includes developing new product specific or customer specific components, or adapting existing platform components. The *Product Developer* assists in analysis of the *Product Requirements* for particular components and perform *Select of Platform Components* task for selection of components to integrate with the *Base Product Configuration*. The *Product Developer* contributes to the selection of platform components to replace or add components in the *Base Product Configuration*. The *Product Developer* performs the task *Provide Feedback to Platform Team* on the usage of platform components during product development.

This role requires traditional software development skills. The *Product Developer* needs to be able to understand and conform to the product architecture. The role requires the ability to identify and build developer tests that cover implemented functionality.

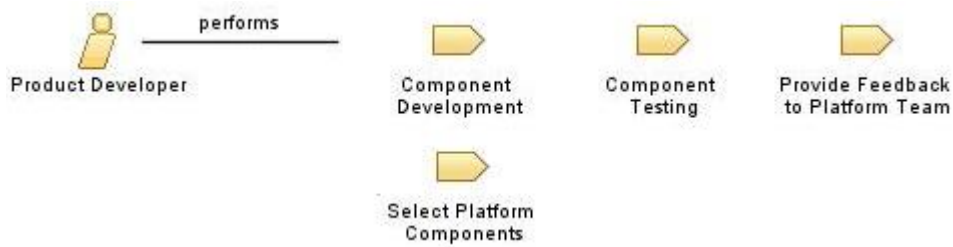


Figure 5-5 Product Developer Role

5.2.1.5 Product Manager

The *Product Manager* is responsible for customer relationship management, product sales, negotiation of product features with the customer and project planning. The *Product Manager* is responsible for *Translate Customer Requirements* where the *Customer Requirements* are translated into the internal organisational language. In the task *Verify the Product Requirements* the new *Product Requirements* are reviewed and verified. The role is responsible for *Allocate Requirements* where requirements are allocated to relevant disciplines and teams.

In *Customer Negotiation* the *Product Manager* tries to meet as many of the customer's needs as possible while retaining the profitability of the platform assets for the whole product line.

The *Product Manager*, with the help of the *Product Architect*, provides feedback on the suitability and quality of the core assets in *Provide Feedback to Platform Team* and makes specific platform requirements requests. Typically each *Product Manager* is responsible for a particular customer, for example, Airbag Control systems for BMW.

The role requires extensive domain knowledge and experience. The *Product Manager* does not need detailed technical knowledge of the platform as the *Product Architect* should provide this. The *Product Manager* should have traditional software engineering project management skills and good negotiation skills.



Figure 5-6 Product Manager Role

5.2.1.6 Product Tester

The *Product Tester* is responsible for the main testing effort within the project during *Run Acceptance Tests*. The role performs the task *Create the Product Test Cases* where the *Product Test Cases* are written for requirements in the *Product Requirements*. The *Product Tester* should co-ordinate with the platform testing team to reuse *Platform Test Artefacts*. The *Product Tester* performs the task *Provide Feedback to Platform Team* on the usage of *Platform Testing Artefacts* during product development.

The role requires knowledge of testing approaches and techniques. The *Product Tester* must be able to identify platform test artefacts for reuse.

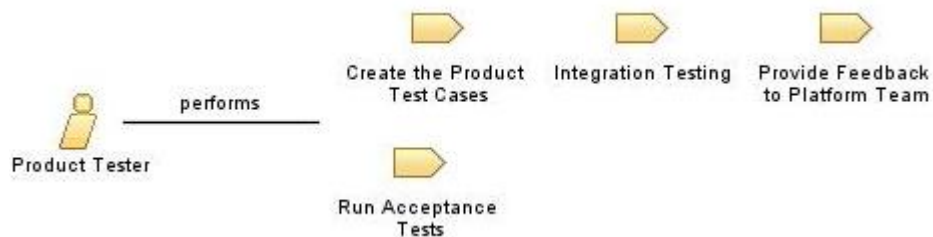


Figure 5-7 Product Tester Role

5.2.2 Units of Work: Tasks and Activities

Tasks are assignable units of work that consume or produce one or more products. To make process building and understanding easier, Pro-PD groups related tasks into Activities. Activities have specific development goals. Pro-PD uses activities as the building blocks from which different development phases (i.e. Preparing for Derivation,

Product Configuration, Product Development and Testing) (see Section 5.3) can be constructed during process instantiation.

Pro-PD contains the following activities (see Figure 5-8):

- **Initiate Project** - To conduct the preparatory tasks required to establish a product derivation project
- **Identify and Refine Requirements** - To detail the product requirements for system wide development, allocate requirements, define the role and task structures and to provide guidance for decision makers
- **Derive the Product** - To create a integrated product configuration that makes maximum use of the platform and minimises the amount of product specific development required
- **Develop the Product** - To facilitate requirements that could not be satisfied by a configuration of the existing assets through component development or adaptation
- **Test the Product** - To validate the current product build
- **Project Assessment** - To provide feedback to the platform team

Each of these is discussed in the following sections.

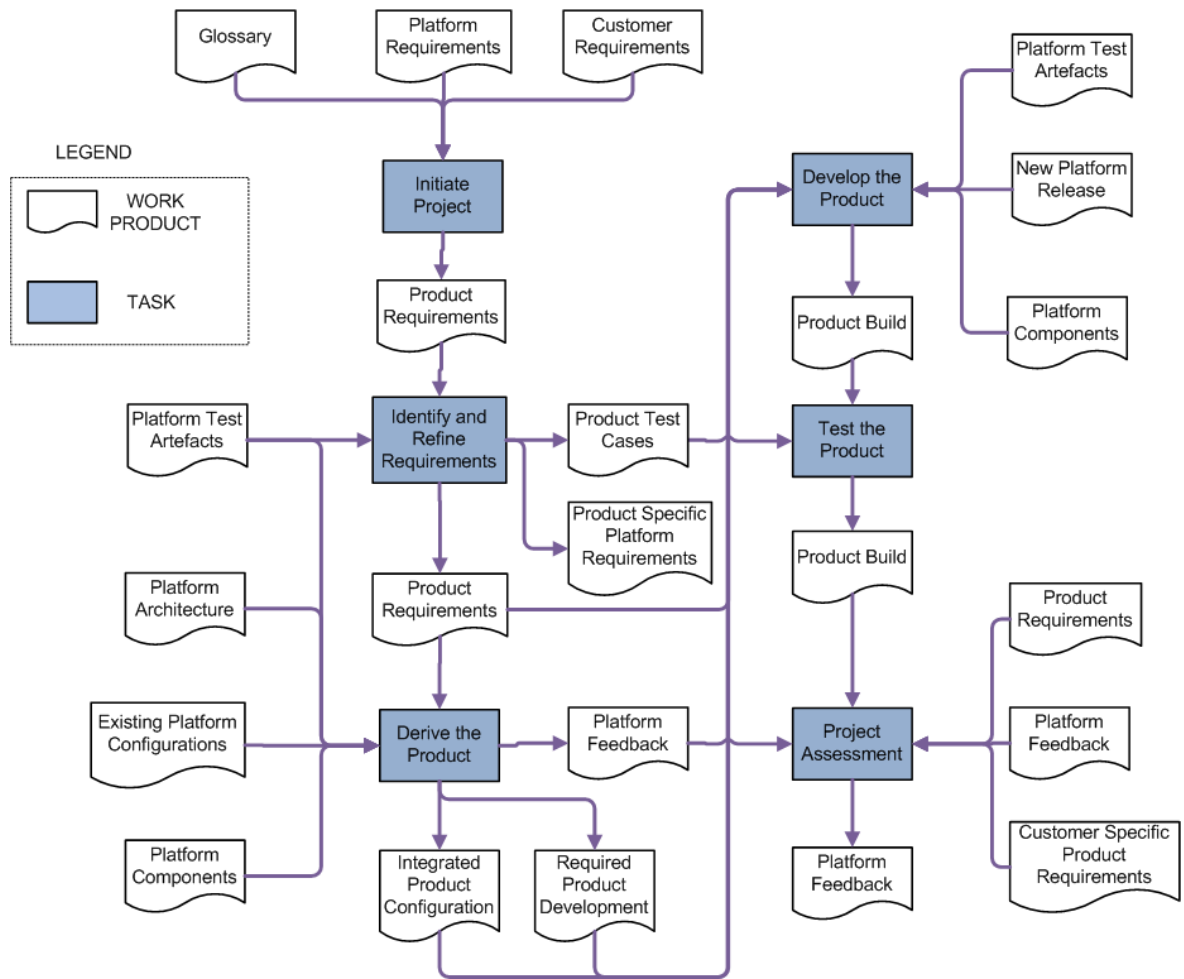


Figure 5-8 Overview of Pro-PD Activities

5.2.2.1 Initiate Project

Heading	Content
Name	Initiate Project
Goal	To conduct the preparatory tasks required to establish a product derivation project
Entry Criteria	The product is requested by a customer
Primary Roles	Product Manager Product Analyst Customer Platform Manager Product Architect
Tasks	Translate Customer Requirements Coverage Analysis Customer Negotiation Create the Product Requirements Verify the Product Requirements
Input	Customer Requirements Platform Requirements Glossary
Output	Product Requirements
Exit Criteria	The created Product Requirements are verified by the key product stakeholders. The scoping of the product may need to be re-examined if a sufficient amount of customer requirements need to be re-negotiated

Table 5-1 Initiate Project Activity Summary

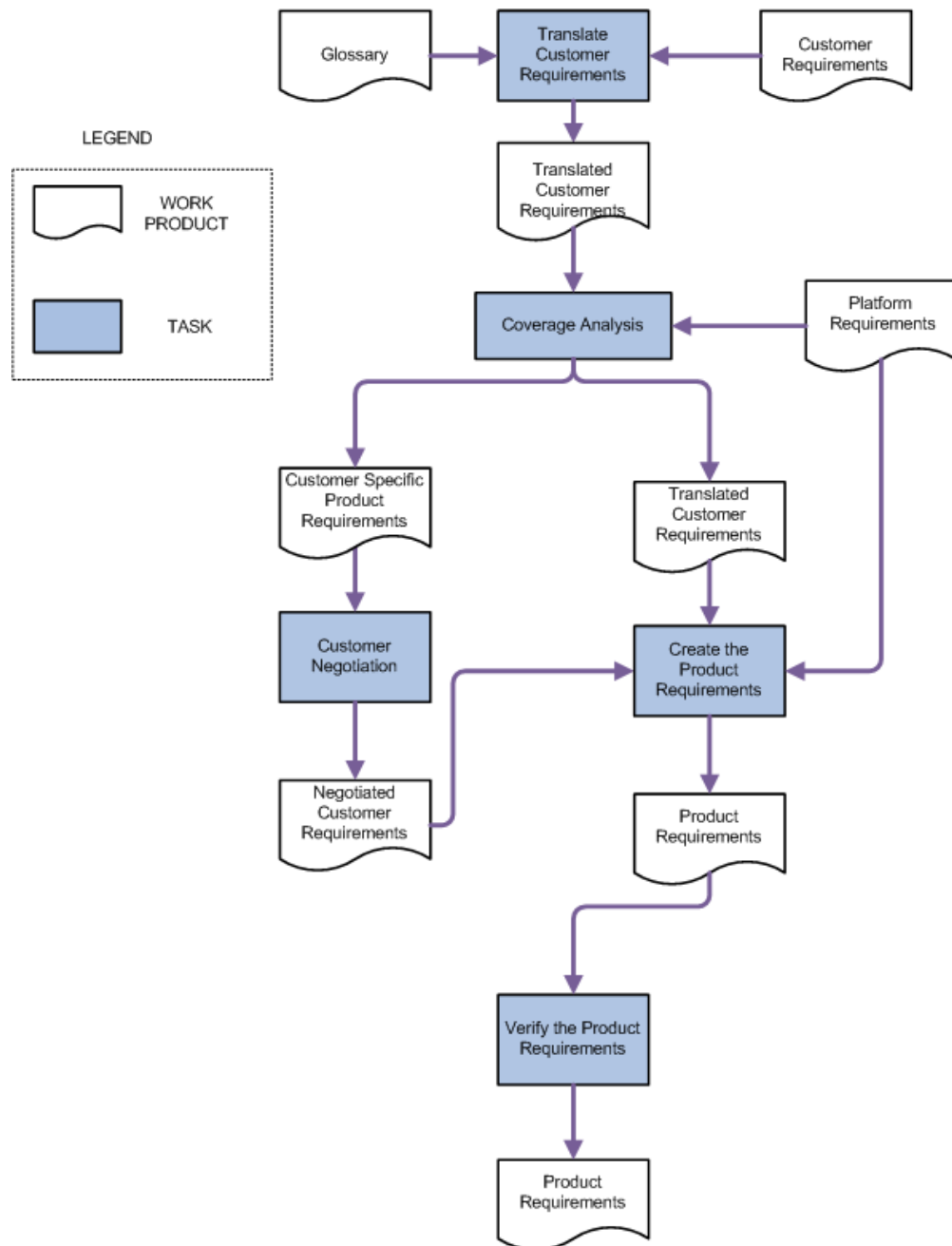


Figure 5-9 Initiate Project Activity

Derivation does not start “from scratch”, i.e., by just selecting features or taking decisions described in a variability model. The *Initiate Project* activity (see Figure 5-9 and Table

5-1) contains the preparatory tasks required to establish a product derivation project. It is typically only conducted once at the beginning of a product derivation project.

In the Task *Translate Customer Requirements*, the *Customer Requirements* are translated into the internal organisational language. To translate the customer document the *Product Manager* must adapt the language within the *Customer Requirements* to the platform language and change the structure to fit the internal organisational documentation structure. Typically a product glossary is used as a reference during the translation process. The output of this task is the *Translated Customer Requirements*. The *Translated Customer Requirements* contains cross-discipline requirements. The translation process prevents terminology confusion and customer-specific description of assets.

Coverage Analysis is at its simplest a comparison between the *Translated Customer Requirements* and the *Platform Requirements*. The *Translated Customer Requirements* which are within the scope of the platform are identified. Those requirements which are outside the scope of the platform are contained in the *Customer Specific Product Requirements*. Extensive domain experience is required for this task. If specific requirements cannot be completely accurately scoped, they are broken into smaller requirements and then mapped to specific components if possible. *Coverage Analysis* can prove a complicated task for the *Product Architect*. Firstly, the *Platform Requirements* contain requirements at the system level, whereas the *Product Requirements* may contain requirements at a different design level. Secondly, requirements from the *Translated Customer Requirements* and the *Platform Requirements* may have different granularity. As a consequence, it may be difficult to compare such requirements. Finally, the *Translated Customer Requirements* may contain requirements which are not contained in the platform; frequently it is not clear how such requirements should be handled. The output of this task is *Customer Specific Product Requirements*.

In *Customer Negotiation* the objective is to meet as many of the customer's needs as possible while retaining the profitability of the platform assets for the whole product line. *Customer Negotiation* is performed by the *Product Manager* and the *Customer* with the assistance of the *Platform Manager* if required. Effort estimation issues can make

Customer Negotiation difficult. Time-to-market requirements can cause the Product Team to make their own product-specific modifications to platform assets – modifications that might lead to uncontrolled variability in the product line [120]. Involving the platform team in customer negotiation can solve many of the problems that arise from these conflicting requirements [121].

When applied within an iterative development environment, *Customer Negotiation* involves the *Customer* specifying the priority of individual requirements. This assists in the process of allocating requirements to specific iterations. The output of this task is a set of *Negotiated Customer Requirements*.

In *Create the Product Requirements*, using the *Negotiated Customer Requirements* and the *Translated Customer Requirements* which were within the scope of the platform the *Product Requirements* are formed. To form the *Product Requirements*, the *Platform Requirements* are taken as a baseline. The *Product Requirements* are expressed as a delta or variation of the *Platform Requirements* with some additions for the *Negotiated Customer Requirements*. The role is performed by the *Product Architect* and the output of this task is the *Product Requirements*.

In the task *Verify the Product Requirements* the new *Product Requirements* are reviewed and verified by the main stakeholders within the project, the *Customer*, *Platform Manager*, *Product Architect* and *Product Manager*.

Various types of tools can be used to support and automate the *Initiate Project* activity. 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 [122]. For example, in the Industrial Case Study (See Section 4.2.1) the product team used the tool DOORS to assist in requirements traceability.

5.2.2.2 Identify and Refine Requirements

Heading	Content
Name	Identify and Refine Requirements
Goal	To detail the product requirements for system wide development, allocate requirements, define the role and task structures and to provide guidance for decision makers
Entry Criteria	The requirements set is sufficiently complete to begin next stage and verified by key stakeholders
Primary Roles	Product Manager Product Architect Product Tester
Tasks	Find and Outline Requirements Create the Product Test Cases Allocate Requirements Create Guidance for Decision Makers
Input	Product Requirements Platform Test Artefacts
Output	Product Requirements Product Test Cases Product Specific Platform Requirements
Exit Criteria	The requirements have been elicited and allocated

Table 5-2 Identify and Refine Requirements Activity Summary

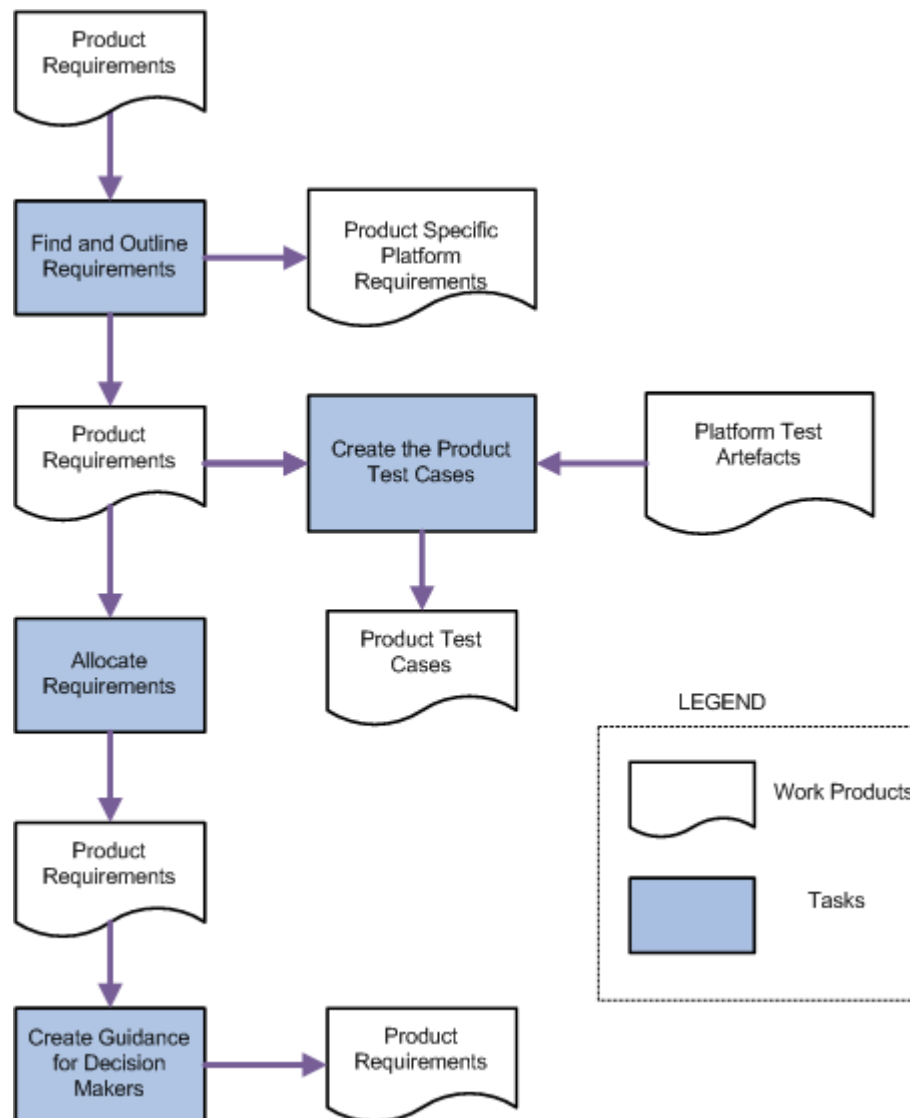


Figure 5-10 Identify and Refine Requirements Activity

The *Identify and Refine Requirements* activity (see Figure 5-10 and Table 5-2) describes the tasks performed to detail the product requirements for system wide development, allocate requirements to specific disciplines and iterations, and to define the role and task structures to be used for the implementation. This activity is typically performed at the beginning of each new development iteration.

In the *Find and Outline Requirements* task, the functional and non-functional requirements for the system are specified and scoped by the *Product Architect*. The *Product Architect* scopes the *Product Requirements*. It should be noted however that only requirements not covered by the platform are relevant input here. The decision of whether the required component development or adaptation will result in product-specific code or in adaptation of the entire product line (platform) is determined through a Change Control Board (CCB). The CCB is comprised of key 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 cost frequently cause scoping solutions to be selected that are neither optimal for the product itself nor for the product line as a whole [3]. Market forecasts, resources, implementation costs and intellectual property issues all influence the *Product Architect's* scoping decision. Intellectual property becomes an issue when a customer wants a customer-only-solution and does not want their features to become part of the general platform features. The *Product Architect* uses impact analysis models, decisions models and effort estimation models to assist in the decision making progress. While platform development must provide a consistently high-quality platform, product development must meet delivery dates and customer requirements. So with every requirement, it must be decided whether to integrate into the platform or into an individual product only [121].

The *Product Manager* based on recommendations by CCB separates the requirements for implementation in the *Product Requirements* between the Platform and Product teams. The requirements destined for platform implementation are added to the *Product Specific Platform Requirements*. The implementation of the *Product Specific Platform Requirements* occurs at platform level and is deemed outside the scope of this research. In addition to these platform requests, additional interface requirements must be added to the *Product Requirements* to allow communication between the product and platform components.

In *Create the Product Test Cases* task, the *Product Test Cases* are written for requirements in the *Product Requirements*. The product test cases are used during the *Run*

Acceptance Tests in the *Test the Product* activity and assist the product team in the verification of requirements [123]. The *Product Tester* uses the *Platform Test Artefacts* as a basis for the creation of the *Product Test Cases*.

In *Allocate Requirements*, the *Product Requirements* are allocated to the relevant organisational disciplines such as hardware, algorithms or software disciplines by the *Product Architect*. 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. During iterative instances of Pro-PD, requirements are allocated to specific development iterations. This is based on the priority of the *Product Requirements*, as specified by the customer. Output of this step is an amended *Product Requirements* with requirements allocated to specific disciplines.

Preparing for derivation also means to *Create Guidance for Decision Makers*. Guidance is essential, especially for developers who are confronted with many – often technical – decisions or features. Guidance can be linked into the *Product Requirements*, often to external sources to provide information on the background to the decision. Remaining variability must be explained to deal with complexity issues in representing product line variability.

A similar toolset to that used in the *Initiate Project* activity can be used here. Additionally tools that can automatically generate and manage test cases would be useful. For example, in the industrial case study research, the *Product Manager* utilises the DOORS tool to map each test case to a specific customer requirement.

5.2.2.3 Derive the Product

Heading	Content
Name	Derive the Product
Goal	To create a integrated product configuration that makes maximum use of the platform and minimise the amount of product specific development required
Entry Criteria	The Product Requirements have been elicited and allocated, and guidance and planning has been completed
Primary Roles	Product Developer Product Architect
Tasks	Select Closest Matching Configuration Derive New Configuration Evaluate Product Architecture Select Platform Components Product Integration Integration Testing Identify Required Product Development
Input	Platform Components Product Requirements Platform Architecture Existing Platform Configurations Platform Test Artefacts
Output	Integrated Product Configuration Platform Feedback Required Product Development
Exit Criteria	Optimum use has been made of the platform assets

Table 5-3 Derive the Product Activity Summary

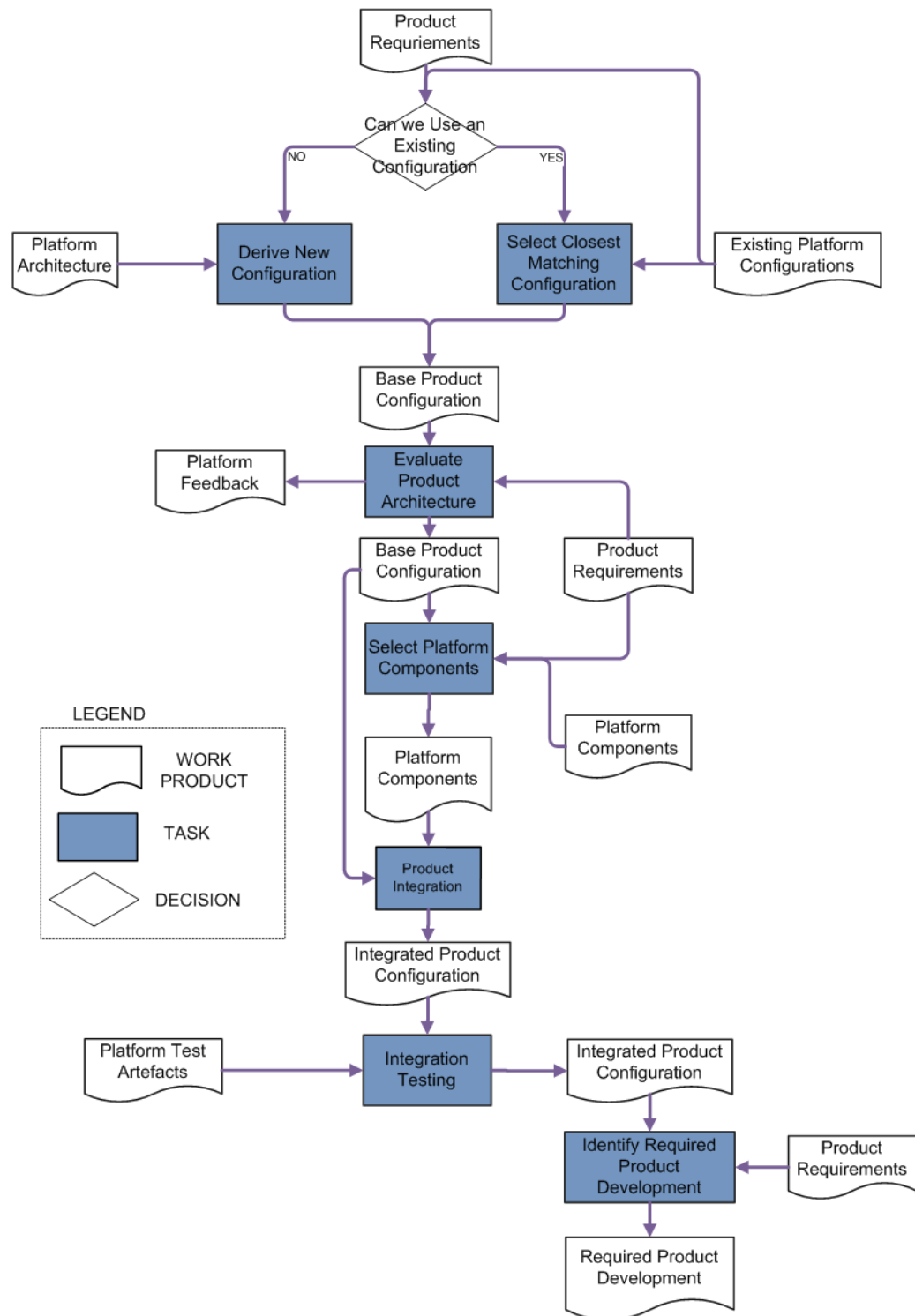


Figure 5-11 Derive the Product Activity

The main goal of the *Derive the Product* activity (see Figure 5-11 and Table 5-3) is to create an *Integrated Product Configuration* that maximises the benefits of the platform artefacts and minimises the amount of product specific development required.

The *Base Product Configuration* is constructed by the *Product Architect* either by choosing from the *Existing Platform Configurations* in the task *Select Closest Matching Configuration* or derived from the platform architecture in the *Derive New Configuration* task.

In the first case a previous product configuration is found in the platform that respects the majority of the *Product Requirements*. 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 [3]. Configuration selection can also help speed up the development process by choosing a previously tested solution especially in instances when two or more configurations can be used.

In the second case, if no *Existing Platform Configuration* can be found, a new one is derived from the *Platform Architecture*. Here the *Product Architect* makes a copy of the latest *Platform Architecture*. The *Product Architect* tailors the *Platform Architecture* to form the product architecture which is contained in the *Base Product Configuration* and resolves variation points according to the *Product Requirements*.

In some situations, the *Base Product Configuration* may be the same as the *Platform Architecture*, or it may be the result of pre-planned tailoring or binding. This often means specialising the platform architecture for use in a particular product.

In the *Evaluate Product Architecture* task the *Base Product Configuration*, which contains the instantiated product architecture, is evaluated to see if it meets the specific behavioural and quality requirements of the product at hand. The results of the evaluation are collected in the *Platform Feedback* work product and sent to the platform team in the

Provide Feedback to Platform Team task. Further development iterations may be required to ensure the architectural behavioural and quality requirements are met.

In the *Select Platform Components* task, components are selected from the collection of *Platform Components* for addition to or replacement of components in the *Base Product Configuration*. The selection of *Platform Components* involves binding any built-in variations (excluding runtime/dynamic variability which is bound onsite). The attached component documentation is used to guide the binding process, this documentation is also known as the production plan. Whether a component fits in the configuration depends on the fact whether the component correctly interacts with the other components in the configuration and no dependencies or constraints are violated. The selection of components allows the product team meet requirements which could not be met through configuration selection.

The task *Product Integration* is performed. The *Base Product Configuration* and the *Selected Platform Components* are integrated. The output is the *Integrated Product Configuration*.

The task *Integration Testing* 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 platform assets with product specific assets [110]. Due to the variability defined in the platform assets, completely testing the platform assets is impossible except for trivial cases.

The *Product Architect* performs the task *Identify Required Product Development*. Theoretically at this stage the *Integrated Product Configuration* could satisfy customer requirements and testing should begin. However, this is the ideal case and assumes all the *Product Requirements* are covered by the platform. In most cases some additional development will be required. This additional developed is captured in the *Required Product Development* work product.

Various types of tools can be used to support and automate this activity. Some of these tools have already been 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) could also be useful.

5.2.2.4 Develop the Product

Heading	Content
Name	Develop the Product
Goal	To facilitate requirements that could not be satisfied by a configuration of the existing assets through component development or adaptation
Entry Criteria	The Product Architect has identified the set of components to be developed and/or adapted
Primary Roles	Product Developer Product Architect Product Tester
Tasks	Component Development Component Testing Product Integration Integration Testing
Input	Required Product Development Product Requirements Platform Test Artefacts Integrated Product Configuration Platform Components New Platform Release
Output	Product Build
Exit Criteria	The required product specific adaptation has occurred

Table 5-4 Develop the Product Activity Summary

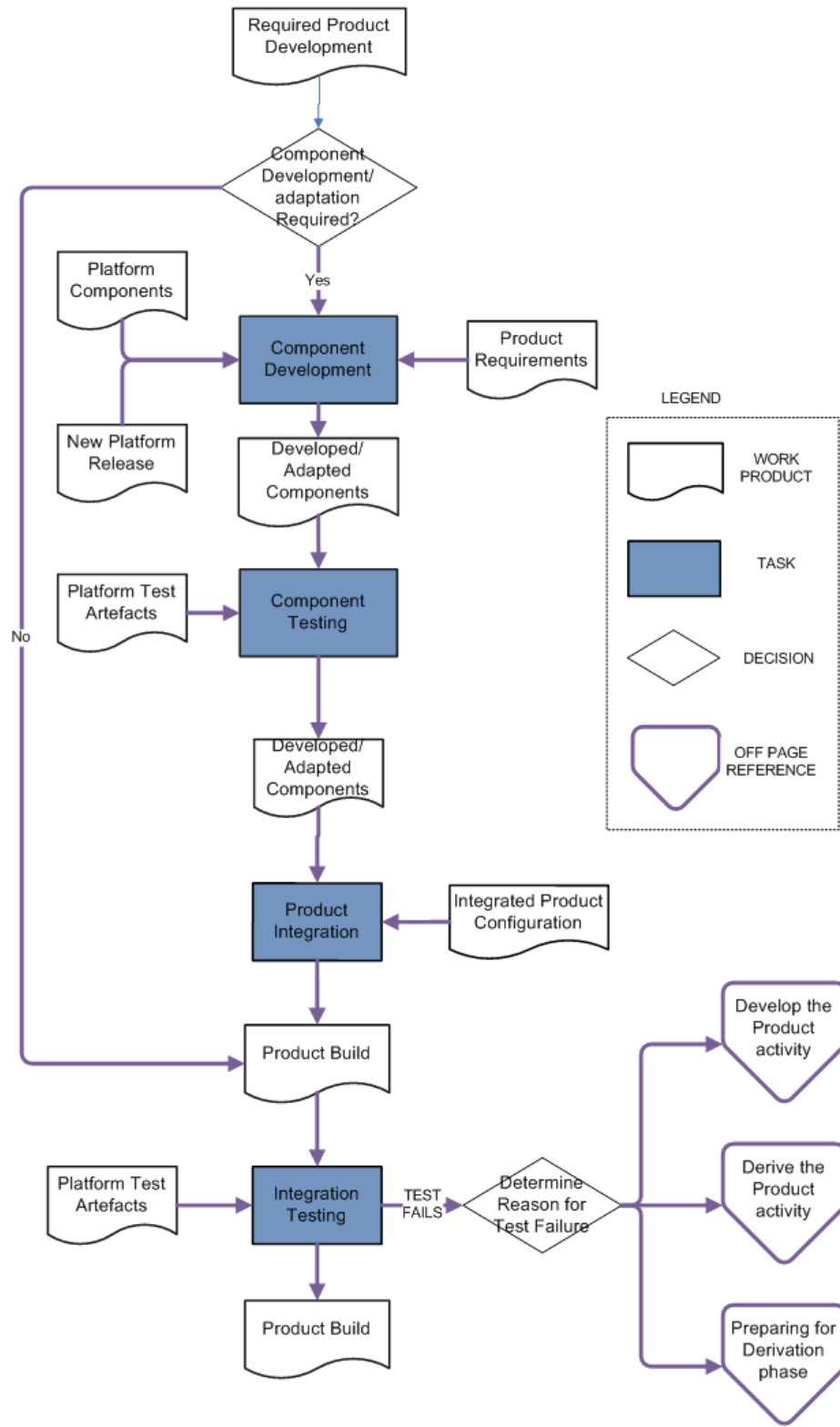


Figure 5-12 Develop the Product Activity

The purpose of the *Develop the Product* activity (see Figure 5-12 and Table 5-4) is to satisfy requirements which could not be satisfied through reuse of platform assets in product specific development.

Specific requirements in the *Product Requirements* which were not satisfied during the *Derive the Product* phase are captured in the *Required Product Development* work product. It is the responsibility of the *Product Developer* to implement the required component changes at the product level. In the *Component Development* task 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 intention of being 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 [58].

When a component is built or adapted, initial or tailored versions of a component will need to be tested rigorously through *Component Testing*. For adapted platform components *Platform Test Artefacts* can be used as a basis for the component unit tests.

In the case of required platform extensions which were identified in the *Find and Outline Requirements* activity, the platform team receives the *Product Specific Platform Requirements* containing the required extensions to the existing platform in order to facilitate the new *Product Requirements*. Both the customer-specific and platform development occurs in parallel.

Previously the synchronisation process (See Section 4.3.2.3) pattern used by the product team to handle development dependencies is described. The product team can decide on an implementation strategy based on their development needs:

Option one. The product team waits for the new platform release and then proceeds to design, implement and test customer specific components.

Option two. The product team bases new component development on the existing *Platform Architecture*. The product team first negotiates a platform interface with the platform team before proceeding to develop in parallel. Alternatively, the product team

will make assumptions on platform interface changes. If conflicts are detected when the *New Platform Release* is released, these are identified during integration testing.

In *Product Integration*, the *Developed or Adapted Components* are integrated into the *Integrated Product Configuration*. The goal is to identify integration issues as soon as possible. This can for example require writing sufficient “glue” code for interfaces [120] or implementing architectural changes to facilitate the developed/adapted assets.

Integration Testing then validates the *Integrated Product Configuration*. The integration tests should reuse *Platform Test Artefacts*. This also ensures that no new errors appear due to the integration of *Developed or Adapted Components* into the *Integrated Product Configuration*. The output is the validated *Product Build*. If the product fail *Integration Testing* then the current configuration may not provide the required functionality, or some of the selected components simply do not work together. In this case based on the *Determine Reason for Test Failure* the product returns to earlier phase of the process.

Various types of tools are required to support and automate Component Development. Tools to support new development and adaptation of existing components and support determination of effort and cost for such tasks are also required. Tools to generate unit test cases and automate the testing process are also useful. In the case study company, it was observed how the Product Manager uses Ameos, MisRA-C and different configuration management tools to facilitate component design. Tools for architecture conformance, verification and validation when combining new or adapted components into a configuration are also useful.

5.2.2.5 Test the Product

Heading	Content
Name	Test the Product
Goal	To validate current product build
Entry Criteria	The Product Build is ready for acceptance testing
Primary Roles	Product Tester
Tasks	Run Acceptance Tests
Input	Product Test Cases Product Build
Output	Product Build
Exit Criteria	The Product satisfies the Product Requirements

Table 5-5 Test the Product Activity Summary

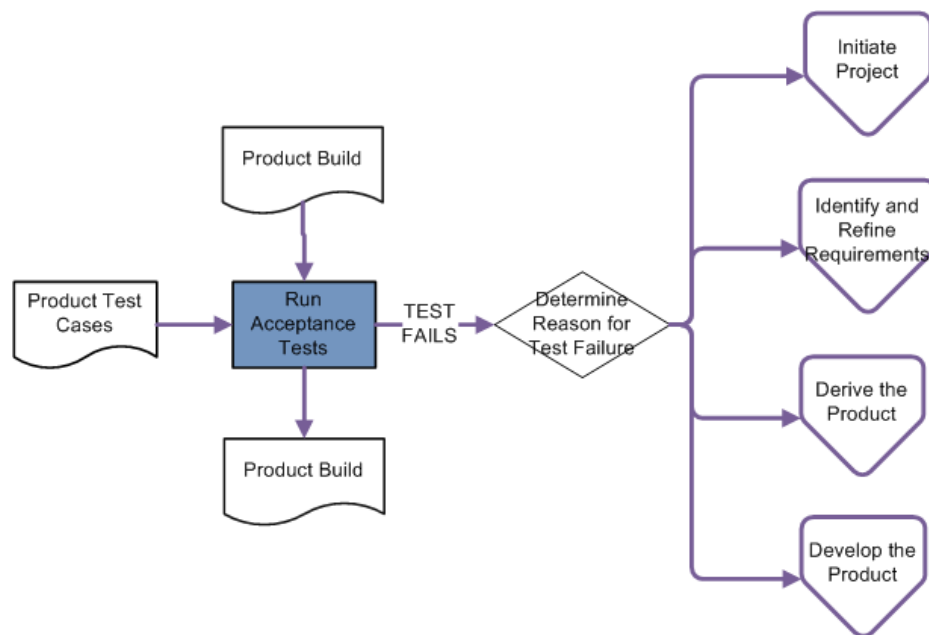


Figure 5-13 Test the Product Activity

The main goal of the *Test the Product* activity (see Figure 5-13 and Table 5-5) is to validate that the product satisfies the *Product Requirements*.

With respect to the testing process, there is no defined and proven methodology [124]. The existing proposals [110, 125] are guidelines for testing activities within a SPL.

In *Run Acceptance Tests*, the *Product Tester* checks the *Product Build* for compliance with the *Product Requirements* [3]. The *Product Tester* does this through using the *Product Test Cases* creating in the *Create the Product Test Cases* task. The majority of the *Product Requirements* will be a subset of the product line functionality. Therefore the *Product Test Cases* contain many reused *Platform Test Artefacts*. However, even if platform components have been tested at platform level, when the final *Product Build* is tested it can lead to component failures.

There are two main reasons why a *Product Build* may fail acceptance testing. Firstly the requirements set may change or expand during product derivation, for example, if the organisation uses a subset of the customer requirements to derive the *Integrated Product Configuration*, or if the customer has new wishes for the product. Secondly, if the *Product Build* does not completely provide the required functionality, or some of the selected components simply do not work together at all [3]. If the *Product Build* fails acceptance testing then the product team must return to earlier phase of derivation process based on reason for test failure.

If the product is validated through acceptance testing the process is complete and the customer product has been derived. The output is the *Product Build*.

Various types of tools are required to support and automate the *Test the Product* activity. Tools to determine the causes of test failure would also support this step.

5.2.2.6 Project Assessment

Heading	Content
Name	Project Assessment
Goal	To provide feedback to the platform team
Entry Criteria	The product has been released to the customer
Primary Roles	Product Architect Product Developer Product Manager Product Tester Platform Manager
Tasks	Provide Feedback to Platform Team
Input	Product Requirements Customer Specific Product Requirements Platform Feedback
Output	Platform Feedback
Exit Criteria	The product team have captured lessons learnt

Table 5-6 Project Assessment Activity Summary

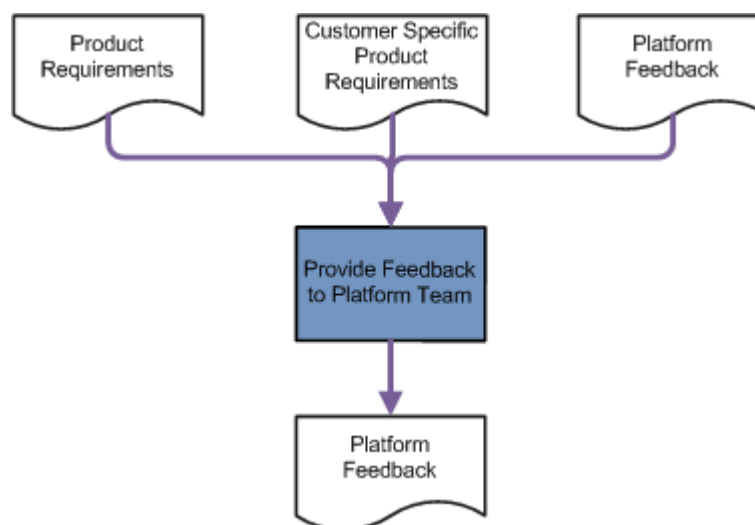


Figure 5-14 Project Assessment Activity

The goal of the *Project Assessment* activity (see Figure 5-14 and Table 5-6) is to collate the product teams experience of using the platform to derive a product.

In the *Provide Feedback to Platform Team* task, feedback is provided to the *Platform Manager* on core asset usage during the project, how user friendly the platform assets were and areas for improvement within the platform specifically product requirements which should be adopted by the platform. In addition, the product team identify product specific components that the platform could potentially benefit from through adoption. The *Platform Feedback* work product is an input to product line evolution.

5.2.3 Work Products

In Pro-PD, a work product is an artefact which is produced, modified or used by a task within the derivation process. Work products should be subjected to version control throughout the lifecycle of the derivation project. Work products may be composed of other work products. Work products are abstract generalisation for various project artefacts such as software components, software architecture, requirements, guidance material and platform/product assets. Pro-PD work products are described in Table 5-7.

Work Product Name	Description
<i>Base Product Configuration</i>	A product specific instantiation of the platform architecture or reuse of old product configuration.
<i>Customer Requirements</i>	Specification of the customer expectations for the system. It may contain an entire collection of requirements specification documents.
<i>Customer Specific Product Requirements</i>	Customer requirements which are not provided for by the platform.
<i>Developed or Adapted Components</i>	Development of new components or adaptations of platform components to implement required product functionality.
<i>Existing Platform Configurations</i>	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.
<i>Glossary</i>	Used as a customer terminology reference. It assists in translating specific customer terms to platform specific terminology with requirements specification. A product glossary is used as a reference during the translation process.
<i>Integrated Product Configuration</i>	Integrated <i>Base Product Configuration</i> and selection of platform components.
<i>Negotiated Customer Requirements</i>	Initially unsatisfied <i>Customer Requirements</i> which were renegotiated with the customer.
<i>New Platform Release</i>	A new release of the platform with requested Product Specific Platform Requirements incorporated.
<i>Platform Architecture</i>	The blueprint for how each product in the product

	line is assembled from the platform components. The platform architecture is used to create a <i>Base Product Configuration</i> when no previous configuration can be reused.
<i>Platform Components</i>	A software module that captures a set of related product line functions. Contains attached documentation that define the variability mechanism in order to meet the requirements for a particular product.
<i>Platform Feedback</i>	Feedback to the platform team regarding reusable assets. This is essentially the input for product line evolution.
<i>Platform Requirements</i>	Platform requirements define the products and the features of the products in the product line. Requirements common to the entire product line are written with variation points that can be filled in or exercised to create the <i>Product Requirements</i> .
<i>Platform Test Artefacts</i>	Platform test cases typically associated with a particular platform component.
<i>Product Build</i>	A stable release of the product.
<i>Product Specific Platform Requirements</i>	Requested platform extensions by product team.
<i>Product Requirements</i>	This artefact is based on the platform requirements and negotiation with the customer. Typically, product teams take the <i>Platform Requirements</i> as a baseline before amending them to form the <i>Product Requirements</i> .
<i>Product Test Cases</i>	Product test cases developed through reuse of

		platform test artefacts.
<i>Required Development</i>	<i>Product</i>	Requirements in the <i>Product Requirements</i> which were not satisfied during the <i>Derive the Product</i> phase are captured in the <i>Required Product Development</i> work product.
<i>Translated Requirements</i>	<i>Customer</i>	The <i>Customer Requirements</i> adapted to the platform language and the structure changed to fit the internal organisational documentation structure.

Table 5-7 Work Product Table

5.3 Waterfall Instantiation

The waterfall instantiation of Pro-PD is based on the discussion in Chapter Four (See Section 4.2.2.1). The waterfall instantiation is created using the roles (See Section 5.2.1), tasks (see Section 5.2.2), activities (see Section 5.2.2) and work products (see Section 5.2.3) defined as the elements of Version four of Pro-PD (see Figure 5-15).

The waterfall instantiation of Pro-PD is defined at a high level and not to be used ‘as is’ but through specialisation (see Section 2.4.2.1.1). In order to create a working company specific model this waterfall process needs to be specialised and a lower level of model abstraction needs to be constructed. In terms of a working industry model, it should not be considered a complete end to end process model.

The advantage of the waterfall instantiation is that it provides a high level of abstraction, an easy to follow working example of the derivation process and serves as a basis for industry specific model generation. It is a generic example that is usable across multiple domains.

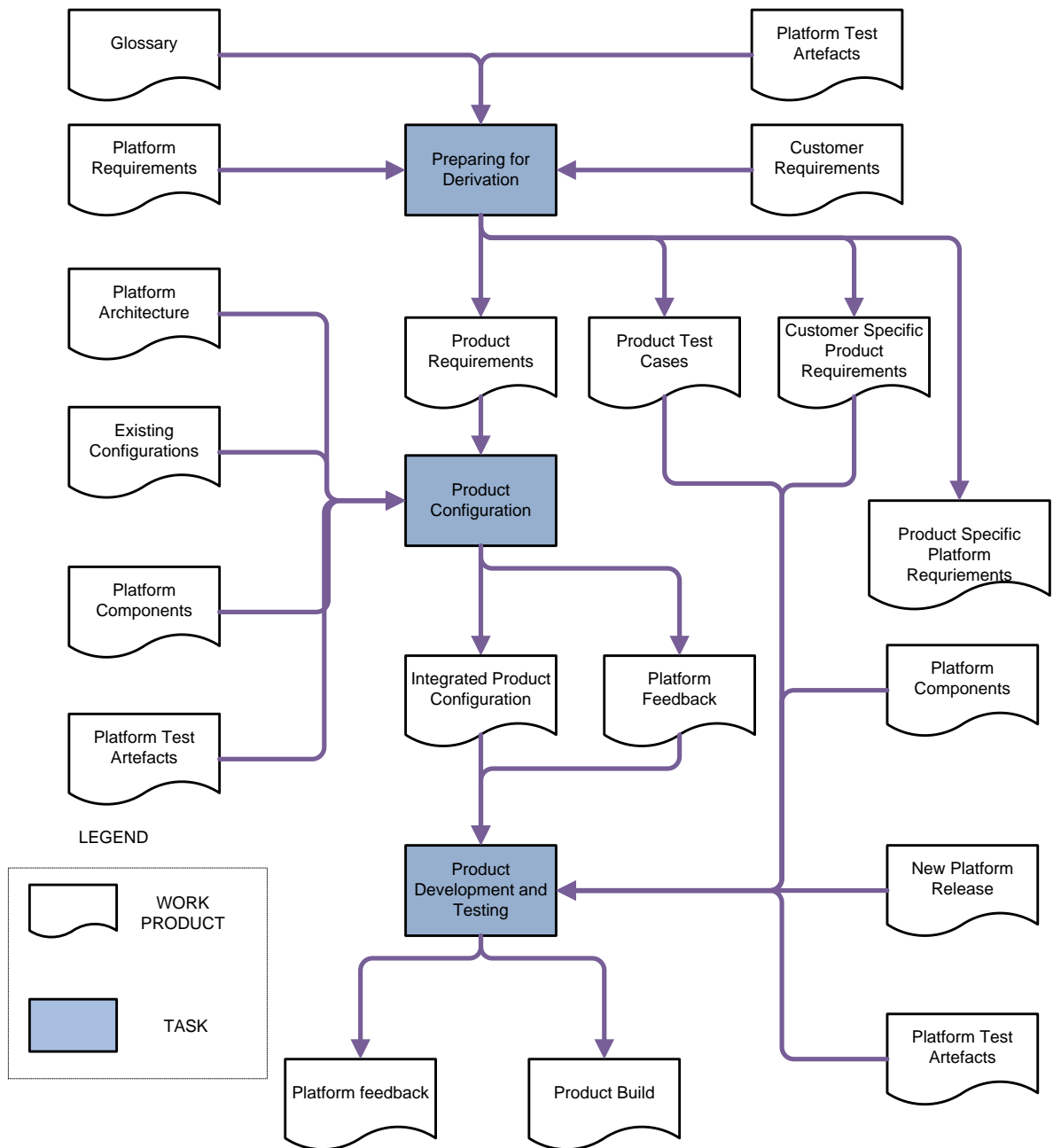


Figure 5-15 Waterfall phases and work products

5.3.1 Preparing for Derivation

Preparing for Derivation Phase (see Figure 5-16) determines the objectives, project plan and requirements for the project. The phase forms the *Product Requirements* based on the *Customer Requirements* and negotiation with the platform team. Requirements are prioritized, elicited and assigned.

The main goals of this phase are as follows:

- Define the product requirements
- Scope the Product
- Define guidance for decision makers
- Plan the project
- Identify role and task structures

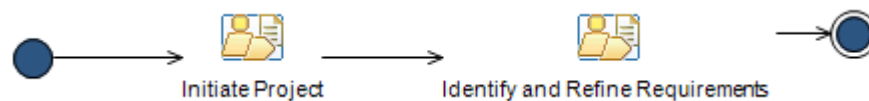


Figure 5-16 Preparing for Derivation Phase

At the end of the *Preparing for Derivation* phase is the *Product Requirements* have been elicited, specified and verified with the various project stakeholders.

5.3.2 Product Configuration

The *Product Configuration* phase (see Figure 5-17) describes the activities performed to create the *Integrated Product Configuration*. An *Integrated Product Configuration* is created through the reuse of platform artefacts.

The main goals of this phase are as follows:

- Create an integrated product configuration through reuse of platform artefacts
- Make maximum use of the platform artefacts

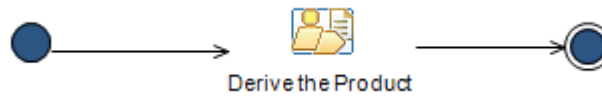


Figure 5-17 Product Configuration Phase

At this point, the *Product Build* is ready for acceptance testing.

5.3.3 Product Development and Testing

The aim of the *Product Development and Testing* phase is to deliver the software to the customer and capture the project lessons.

The main goals of this phase are as follows:

- Ensure the product completely satisfies all the *Product Requirements*
- Provide feedback to the platform team on use of platform artefacts



Figure 5-18 Product Development and Testing

At the end of the *Product Development and Testing* phase the *Product Build* is released if it passes product testing and feedback is provided to the platform team in *Project Assessment*.

5.4 Pro-PD Matrix

In Appendix I the Pro-PD process matrix for the waterfall instantiation is included. The matrix shows the relationship between the various process elements within Pro-PD and can be used as a quick reference point for the model.

5.5 Conclusion

This chapter describes Pro-PD, a process reference model for product derivation. The model describes the roles, work products and activities essential for product derivation. The chapter describes a waterfall instantiation of the model with three phases; Preparing for Derivation, Product Configuration and, Product Development and Testing. Finally a process matrix for Pro-PD has been attached, which demonstrates the relationship between the various model facets.

In the next chapter, the adaptability of Pro-PD will be demonstrated.

Chapter Six: Adapting Pro-PD to Agile

6.1 Introduction

The process structure of Pro-PD described in the previous chapter is based on the waterfall process model. In this chapter it is adapted to fit the characteristics of an Agile process model. This serves three purposes:

- 1) It serves to meet the second objective of this research, to demonstrate the adaptability of Pro-PD.
- 2) It seeks to explore the integration of Agile practices in product derivation.
- 3) There has been an iterative theme through the research stages. In the expert opinion workshops (see Section 4.2.2.1) some experts felt an iterative design was more reflective of product derivation practice. Some [22, 24] product derivation approaches supported this view (see Section 4.5.4). An Agile version of Pro-PD serves to incorporate these views on iterative product derivation.

With these purposes in mind, the layout of this chapter is as follows: the integration of the Agile and SPL development paradigms is discussed (See Section 6.2). A high level comparison of the twelve principles of the Agile manifesto and product derivation practice is presented (See Section 6.3). An Agile instantiation of Pro-PD, called A-Pro-PD, is created through Use by Specialisation (see Section 6.4). Finally Agile elements which could be adopted within A-Pro-PD are described (see Section 6.5).

6.2 Agile Practices and SPL

Recently there has been growing interest in exploring the possibilities of integrating Agile approaches into the SPL development process. In conjunction with the 2006 Software Product Line Conference (SPLC) a new workshop was arranged called Agile Product Line Engineering (APLE). From the workshop it was recognized that both Agile approaches and SPL share several common goals [126, 127] such as:

- Increasing the productivity of teams;
- Reducing the product's time to market;
- Reducing the development costs;
- Improving customer satisfaction.

Both approaches assume that requirement changes during the development will occur and Agile based testing could benefit SPL development [126]. Furthermore both development paradigms are being promoted as a means of reducing time to market, increasing productivity, and gaining cost effectiveness and efficiency of software development efforts [126]. These goals (shared by Agile and SPL) open the possibilities of introducing Agile practices into SPL activities.

While a motivation for combining Agile and SPL approaches can be found there is also recognition of the potential challenges of any integration effort due to certain differences that exist in the philosophies of both approaches such as design and change management strategies [126, 128].

Both approaches have different design strategies. Agile focuses on a simple design while SPL has an emphasis on rigorously designing and systematically maintaining SPL architectures [128]. Agile approaches do not develop flexible artefacts for reuse [128, 129]. SPL tends toward long-lived life cycles which means that the maintaining of software is necessary [129]. SPL targets satisfying the needs of several customers rather than satisfying individual needs [129]. SPL and Agile approaches include different strategies for change management: in Agile it is incremental development and in SPL the focus is on platform artefacts [126]. SPL architecture needs to be flexible to handle requirements of several customers [1], Agile does not.

Thus, despite having similar goals, both Agile and SPL have different mechanisms and logistics for achieving their respective goals. SPL aims to fulfil the requirements of several customers at the same time rather than concentrating on meeting the individual needs of a particular customer [128]. Customer involvement is much more intensive in Agile approaches where developers work closely with customers on a daily basis. Agile and SPL paradigms also differ in terms of role and importance of software design. Agile

approaches do not emphasise the importance of rigorous design and documentation; rather Agile approaches advocate implementing required functionality and then documenting the prepared code by reconstruction and refactoring. On the other hand, design and documentation are very vital activities in SPL as platform assets need to be appropriately documented to support their reusability [126].

As a result of these challenges, there are a number of risks associated with an Agile SPL approach. If an SPL based architecture is tailored to be more Agile there is a danger that valuable architecture that supports other products in the family may be damaged [126]. Traceability management, as well as maintaining of components, in Agile approaches can be difficult without explicit knowledge [127] and no tool support for Agile SPL approach exists [128].

Despite the documented challenges and risks mentioned, there are some suggested approaches for integrating Agile practices in SPL. Some suggestions include the Planning Game [129] from the XP methodology can increase agility in SPL organisations, particularly for gathering and negotiating the product requirements. The use of Agile practices during application engineering activities is suggested [128], in particular to perform product tailoring [130] and to support the collaborative working of stakeholders [126].

An Agile SPL approach may have the potential of supporting the effective and efficient development of much larger families of products than a traditional SPL approach. Moreover, the Agile project management approach Scrum can be used to manage the product derivation process and development practices like XP or test-driven development can be used for developing platform and/or product components.

6.3 The Agile Manifesto and Product Derivation

SPL and Agile practices have significant differences due to a fundamental difference in focus. SPL is aimed at longer-term strategic objectives whereas Agile practices are focused on short-term tactical objectives. Despite these differences, the potential remains for both

strategic and tactical advantages; therefore it is logical to investigate if these two approaches can complement each other.

As a first step to this comparison, the twelve principles of the Agile Manifesto [46] are presented in Table 6-1.

Agile Principles
Principle 1: Priority on early and continuous delivery of valuable software.
Principle 2: Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
Principle 3: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
Principle 4: Business people and developers must work together daily throughout the project.
Principle 5: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
Principle 6: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
Principle 7: Working software is the primary measure of progress.
Principle 8: Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Principle 9: Continuous attention to technical excellence and good design enhances agility.
Principle 10: Simplicity – the art of maximizing the amount of work not done – is – essential.
Principle 11: The best architectures, requirements, and designs emerge from self-organising teams.
Principle 12: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Table 6-1 The Agile Manifesto [46]

Exploring the twelve principles of the Agile Manifesto and product derivation leads to interesting results. First, product derivation does not interfere with the following Agile principles.

Agile methods put a priority on early and continuous delivery of valuable software (Principle 1). With the introduction of product derivation iterations, an organisation is capable of producing a set of products for a specific customer, including the core functionality, quicker than traditional software development methods.

Agile methods are designed to accept changing requirements, even late in development (Principle 2). In product derivation, late requirements can be implemented directly at product level to speed implementation. At a later stage, the Platform Team can mine those product features with reuse value and implement them in the platform. This is in line with the reactive evolution strategy as described in [131].

Agile methods deliver working software frequently and early in the development cycle (Principle 3). Product derivation teams do so as well, by assembling and configuring core assets. In fact the reason for producing working software early is so that users can get direct experience with a product and give feedback sooner than later. Product line organisations often have complete example products very early in the life of the product line, before some specific products are even started [131].

Both Agile methods and product derivation projects see collaboration as being vital. Agile methods are based on collaboration between business people and the developers (Principle 4). Product line methods expect collaboration between the Platform Team and the Product Team. These collaborations serve the purpose of a feedback circuit.

Agile methods encourage building projects around motivated individuals, giving them the environment and support they need, and trust them to get the job done (Principle 5). Product derivation neither rejects nor inhibits this practice. However, due to the nature of the communication between the Platform and Product Teams, a more formal approach to communication is required.

Agile methods believe working software is the primary measure of progress (Principle 7). Product derivation approaches through assembly and configure can produce working software early so that users can get direct experience.

Agile methods promote sustainable development (Principle 8). Product derivation supports sustainable development in a larger scale, through the use of platform assets.

Agile methods emphasis continuous attention to technical excellence and good design (Principle 9). Technical excellence and good design is encouraged in product derivation. This also, facilitates the Platform Team when mining features from derived products.

For Agile methods, simplicity – the art of maximizing the amount of work not done – is essential (Principle 10). The motivation for product derivation is the same but by using the platform assets as vehicle for eliminating work.

Agile methods regularly reflect on how to become more effective, then tunes and adjusts their behaviour accordingly (Principle 12). Product derivation does not inhibit this principle.

However product derivation does inhibit some Agile principles. Agile methods put an emphasis on face to face communication (Principle 6). The increased technical and/or organisational complexity within product derivation, introduces a need for more explicit, formal and disciplined communication.

In Agile methods the best architectures, requirements, and designs emerge from self-organising teams (Principle 11). SPL organisations typically have a complex organisational structure due to the division of labour between domain and application engineering. Pre-paring for derivation involves allocating resources and specifying roles. Teams are not self-organised.

The comparison shows that Agile practices and product derivation have few conflicts; in fact in most counts they appear complementary to each other. Thus, as SPL tries to deliver large-scale benefits through reuse, Agile practice focuses on delivering high quality single products. Thus, there appears to be adequate motivation to the combination of these two approaches.

6.4 The Model in Action – An Iterative Instantiation

In this section the Agile instantiation of Pro-PD (A-Pro-PD) is demonstrated. The demonstration shows the adaptability of the approach and the integrated Agile Pro-PD could solve many of the problems associated with the complex and cumbersome nature of product derivation.

Both Agile practices and SPL have become established software development approaches with research showing promising results and industry showing keen interest. Yet these approaches are often considered to be conflicting approaches to the development of high quality software. A-Pro-PD is meeting calls from industry [107] for research into this area while solving many of the problems associated with product derivations often complex and cumbersome nature. It is a lightweight approach to product derivation, minimising the amount of up-front investment required making SPL more accessible to small organisations with limited resources. For larger organisations, A-Pro-PD could bring a balance between formalism and agility, helping individual product teams deliver products with the best possible quality. A combination of Agile and SPL is expected to create a leaner but more disciplined product derivation process [108].

However, as seen in Section 6.2, while there has been some research on identifying the opportunities and challenges of Agile practices in SPL, no Agile frameworks for product derivation have been developed. This is a first attempt at bridging this gap.

As a first step towards Agile practices in product derivation, Pro-PD was adapted through *Use by Specialisation* (see Section 2.4.2.1) and instantiated to create an Agile Pro-PD (A-Pro-PD). A-Pro-PD is intended to offer process guidance to small, co-located teams and support the adoption of Agile practices in product derivation.

The instantiation of Pro-PD is intended to:

- Encourage the use of iterative development cycles in product derivation
- Minimise the risk associated with ‘big bang’ releases where large products are assembled just before product delivery
- Increase customer involvement in the derivation process.

The adaptation of Pro-PD involves adding two new activities and amending an existing activity.

6.4.1 Use by Specialisation - Agile Pro-PD

For Pro-PD to support Agile approaches in product derivation, the following modifications to the model are required.

- A new activity *Plan and Manage Iteration* is included
- In the activity *Identify and Refine Requirements*, an additional task *Allocate Requirements to Specific Iterations* is included
- Changes to *Test the Product* activity.

A new *Plan and Manage Iteration* activity is included where iterative development cycles are managed. This activity is inspired by the RUP [132] activity of the same name. The activity consists of two tasks, *Plan Iteration* and *Manage Iteration/Assess Results*.

In *Plan Iteration*, the scope and responsibility for a single iteration is set. The goal is to define who is responsible for resolving the remaining variability to fulfil the product requirements. This helps to provide different views on variability for the 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.

In *Manage Iteration/Assess Results*, the project status is assessed. The value of the product increment and the lessons learnt from the iteration are both assessed. Any improvements to the process are discussed. Feedback is provided on core asset use in particular how new product specific components have been developed with the possibility to become a core asset in mind. Feedback is provided on the usage of core assets, how user friendly they were and how they were applied. Feedback is provided to the platform team regarding the adoption of product specific components which would be beneficial to promote to core assets.

In the activity *Identify and Refine Requirements*, an additional task is included. Requirements are allocated to specific development iterations in the *Allocate Requirements*

to *Specific Iterations* task. This is based on the priority of the *Product Requirements*, as specified by the customer during *Customer Negotiation*.

The *Test the Product* activity is performed differently in iterative development. The goal of the activity is to validate that the current build of the system satisfies the requirements allocated to it. Throughout the iterations this activity validates that the implemented requirements are consistent with the customer expectations and that the requirements are correctly implemented on top of the product architecture. Then, a tester conducts system-level testing in parallel with development to make sure that the solution, which is continuously being integrated, satisfies the intent specified in the test cases. The tester defines what techniques to use, what the data input is, and what test suites to create. As tests run, defects are identified and added to the work items list, so that they can be prioritized as part of the work that you will do during iterations.

6.4.2 Model Phases

There are two layers to A-Pro-PD (see Figure 6-1). These are the phase increments layer and iteration lifecycle layer. Phase increments are short units of work on a particular aspect of the derivation process e.g. configuring platform components. The iterative lifecycle layer structures these phase increments to deliver stable builds of the product that incrementally progress towards the iteration objectives. These iterations result in regular product releases.

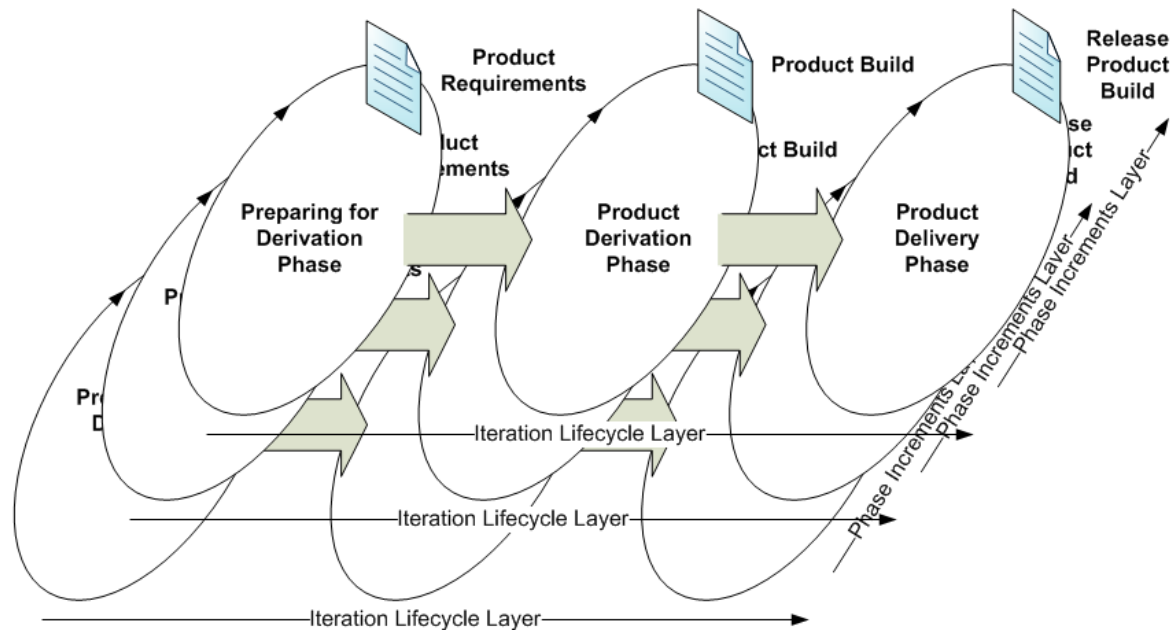


Figure 6-1 Agile Pro-PD

6.4.2.1 Preparing for Derivation Phase

Preparing for Derivation Phase (see Figure 6-2) determines the product requirements and performs the project management tasks for a specific iteration. The phase forms the product-specific requirements based on customer requirements and negotiation with the platform team. Requirements are prioritized and assigned to development iterations. The purpose of this phase is to achieve agreement among all the stakeholders on the *Product Requirements*. On subsequent iterations, this phase is used to plan and manage product iterations using the *Plan and Manage* activity. The main objectives of this phase are as follows:

- Define the product requirements (*Create the Product Requirements* task)
- Scope the Product (*Find and Outline Requirements* task)
- Define guidance for decision makers (*Create Guidance for Decision Makers* task)
- Allocate requirements for implementation in a specific iteration (*Allocate Requirements to Specific Iteration* task)
- Plan the project (*Plan Iteration* and *Manage Iteration/Assess Results* tasks)
- Identify role and task structures (*Allocate Requirements* task)

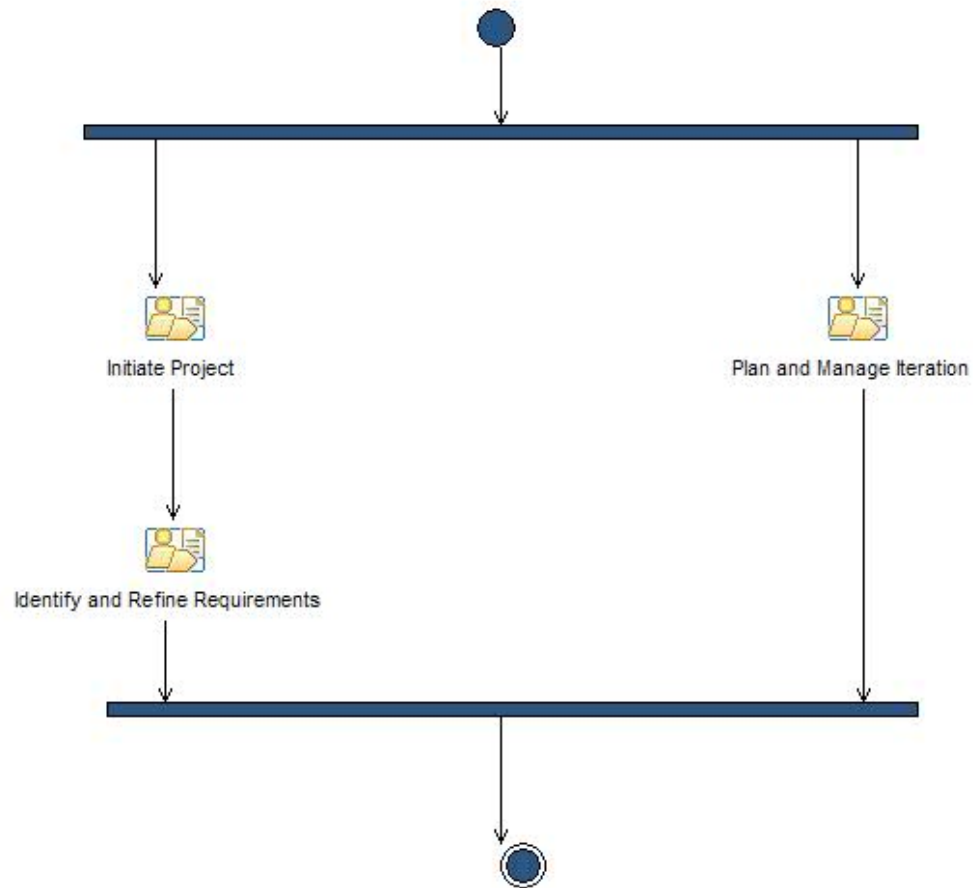


Figure 6-2 Preparing for Derivation Phase

6.4.2.2 Product Configuring Phase

This phase describes the activities performed during the *Product Configuration* phase (see Figure 6-3). Often the activities occur in parallel, creating the product base configuration and performing product specific development. The majority of the product development occurs during this phase.

The main objectives of this phase are as follows:

- Create the Product Build (*Derive the Product* activity)
- Make maximum use of the platform artefacts (*Derive the Product* activity)

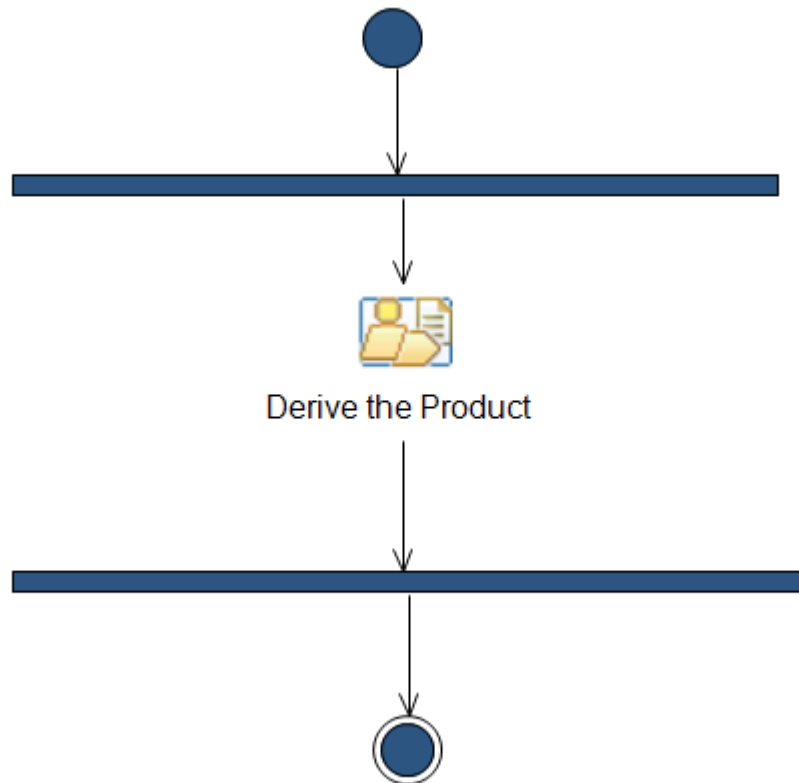


Figure 6-3 Product Derivation Phase

6.4.2.3 Product Development and Testing Phase

This phase focuses on transitioning the software into the customers environment and achieving customer agreement that the product is complete. The purpose of this phase is to ensure the product is ready for delivery to users.

The number of iterations in the phase varies from one iteration (for a simple system requiring primarily minor bug fixing) to many iterations (for a complex system involving adding features and performing activities) to make the business transition from using the old system to using the new system.

When the *Product Requirements* have been met, the project is in a position to be closed. For some products, the end of the current project lifecycle may coincide with the beginning of the next lifecycle, leading to the next generation of the same product. The main objectives of this phase are as follows:

- Test to product to ensure it meets the product requirements

- Correct defects
- Modify the software if unforeseen problems arise
- Provide feedback to platform team on usage of platform

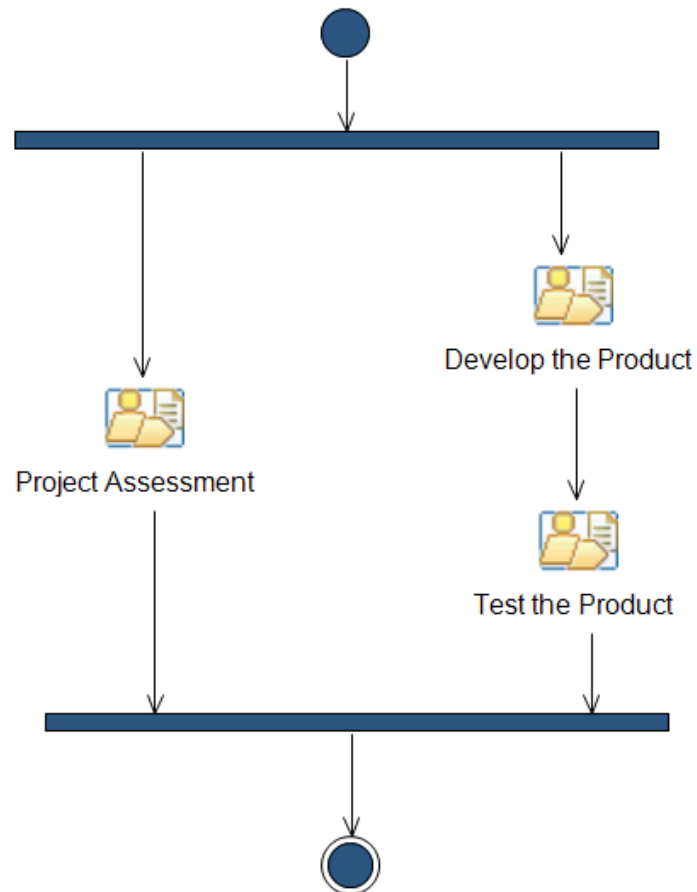


Figure 6-4 Product Development and Testing Phase

6.4.3 Phase Milestones

Phase milestones are used as gateways for phase endings. If a milestone is reached the project is ready to move to the next phase. Each iteration of the project provides an increment in functionality or design.

Each phase has a specific focus and objective. At the end of the Preparing for Derivation phase is the first milestone, the **Product Requirements Milestone**. At this point the *Product Requirements* have been defined and allocated to an iteration.

At the end of the Product Derivation Phase is the **Integrated Product Configuration Milestone**. At this point, a *Integrated Product Configuration* is constructed.

At the end of the *Product Development and Testing* Phase is the **Product Release Milestone**. At this point, a decision is made on whether the Product Requirements allocated to this iteration have been met.

6.5 Increasing Agility in Product Derivation

A-Pro-PD contains the following Agile elements:

- Adoption of Early and Continuous Delivery Strategy;
- Automation of Product Derivation;
- Product Derivation Iterations;
- Agile Testing Techniques.

6.5.1 Adoption of Early and Continuous Delivery Strategy

Typically, implementing product specific features can be time consuming. Firstly, product construction can be substantially delayed due to the Change Control Board (CCB). The CCB scopes new development to gauge the reusability of a requested feature within the product line. Secondly, development is further delayed if the Product Team defers implementing a feature until the Platform Team implements the requested platform changes at the product level.

In A-Pro-PD, the Agile principle of “early and continuous delivery of valuable software” is adopted. The Product Team implement changes at product level. The Platform Team subsequently mine any changes from the product if there is reuse potential.

In Robert Bosch GmbH this Agile principle was observed in action. To facilitate early and continuous delivery of software, the product team would not wait for scoping decisions from the CCB. Rather, the product team would negotiate a new platform interface containing required extensions to facilitate new product components before

proceeding to develop in parallel against the platform team. When the platform extensions had been implemented and the new platform was released, the product team would check for compatibility issues with newly developed components.

The researcher recommended the adoption of the Agile practice of pair programming for customer specific components. Pair programming is suitable for implementing and reviewing any changes at the product level [108]. This helps to produce better quality product code and consequently, improved code for any features that are mined for the platform.

6.5.2 Automation of Product Derivation

Automated support for product derivation is a necessity for managing the complexity and variability inherent in software product lines and according to Kurmann [108], automation is the most important aspect of an Agile software product line. Automated development approaches facilitate the Agile Principle (Principle 2) “*Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.*” [128]. Automated development techniques allow product teams to implement changing customer requirements late in the development lifecycle and automation enables these changes to be implemented quickly.

However current process models and tools for automation do not integrate well. All the stakeholders involved in product derivation are supported in their tasks by different approaches and different automation tools. Because of the difficulty of integrating these different approaches and tools, product derivation can quickly become an error-prone and tedious task.

In the research collaboration with the DOPLER^{UCon} [34], the developed tool was investigated on how it could be used within A-Pro-PD.

Of particular interest is its ability to facilitate Agile approaches. For instance, it was observed that while the DOPLER^{UCon} tool does not directly support *iterative development* cycles by defining additional attributes for requirements it could be used to allocate specific requirements to specific iterations.

6.5.3 Product Derivation Iterations

The identification of product derivation iterations is a key aspect of deriving high quality, customer satisfying products. According to Carbon et al. [128] when adopting a SPL approach, an organisation is capable of producing a first version of a product for a specific customer, including the core functionality, quicker than other software development methods. In further iterations, new functionality can be added to the scope of the product line or product specific features can be implemented [133].

In a technical report to Robert Bosch GmbH (see Appendix G), the researcher recommended that they could benefit from applying the planning game practice from the XP methodology for the management of their product iterations during the *Preparing for Derivation* phase. This would assist them in gathering and negotiating product specific requirements. During customer negotiation requirements are prioritised and allocated to specific iterations based on priority.

6.5.4 Agile Testing Techniques

Agile methods propose that testing is carried out frequently, as this helps Agile developers keep their code as error free as possible. A phased testing approach was recommended in A-Pro-PD. Based on the principles of integration testing suggested by Muccini [133], the structure and nature of the elements in a product line are leveraged. Firstly, integrate the partial configuration and use a traditional approach to integration testing. Then, based on the observation that at least the partial product configuration works properly, the other product elements can be incorporated. Product construction continues in a phased assembly test approach. For systems testing of partial or fully assembled products traditional system testing techniques can be utilized as no SPL specific methods exist.

6.6 Conclusion and Future Work

This chapter demonstrated the adaption of Pro-PD to Agile practices, this satisfies objective two of this research. Furthermore, it explores the integration of Agile practices in product derivation and thereby, contributes to the meeting of the third objective of this research - to add to the established knowledge on product derivation.

In this chapter, the results of research into the adoption of Agile practices in product derivation was presented. The research into Agile practices in product derivation is motivated by the fact that despite the widespread adoption of SPL within industry, product derivation remains an expensive and error-prone activity [2]. The adoption of Agile practices could improve the product derivation process. The Agile instantiation of Pro-PD provides a means of supporting this adoption.

The development of A-Pro-PD is a response to calls from industry for research into this area [107]. An integrated Agile approach could solve many of the problems associated with product derivation's complex and cumbersome nature.

A-Pro-PD is a lightweight approach to product derivation. A-Pro-PD may benefit smaller organisations with limited resources by minimising the amount of up-front investment required making SPL more accessible. A-Pro-PD may benefit larger organisations by bringing a balance between formalism and agility, helping individual product teams deliver products with the best possible quality. A combination of Agile and SPL is expected to create a leaner but more disciplined product derivation process.

Future work includes an ongoing investigation into the benefits of combining Agile and SPL approaches and industry validation of A-Pro-PD particularly with respect to the expected return on investment.

Finally, a criticism of current approaches to product derivation was the lack of adaptability (see Section 2.3.2.4). This chapter demonstrates how Pro-PD can be adapted to fit the characteristics of an Agile process model. This serves to meet the second objective of this research, to demonstrate the adaptability of Pro-PD.

Chapter Seven: Conclusion

7.1 Introduction

Product derivation is the process of constructing a product from a product line of software assets [4]. An effective product derivation process can help to ensure that the benefits delivered through using these shared artefacts across the products within a product line is greater than the effort required to develop the shared assets. In fact, 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" [3] might not hold if inefficient derivation practices diminishes the expected gains.

Authors in the area have stressed the importance of a defined process for product derivation [3, 6, 22, 32], yet despite this the required process support has not been provided.

Thus the objective of the work described in this thesis can be stated as:

To define a systematic process which will provide a structured approach to the derivation of products from a software product line, based on a set of tasks, roles and work products, and to demonstrate its adaptability to different process models.

This objective can be divided into a number of sub-objectives.

1. To define a structured process model for product derivation;
2. To demonstrate the adaptability of the process model;
3. To add to the established knowledge on product derivation.

7.2 Research Design

Pro-PD was iteratively developed with an evaluation at each stage of the research. Each version of Pro-PD was developed using this development – evaluation cycle (see Figure

7-1). The first phase 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 constructed based on the literature review. This preliminary version was iteratively developed and assessed through a series of workshops with SPL experts. The output of the first phase of the research was version one of Pro-PD.

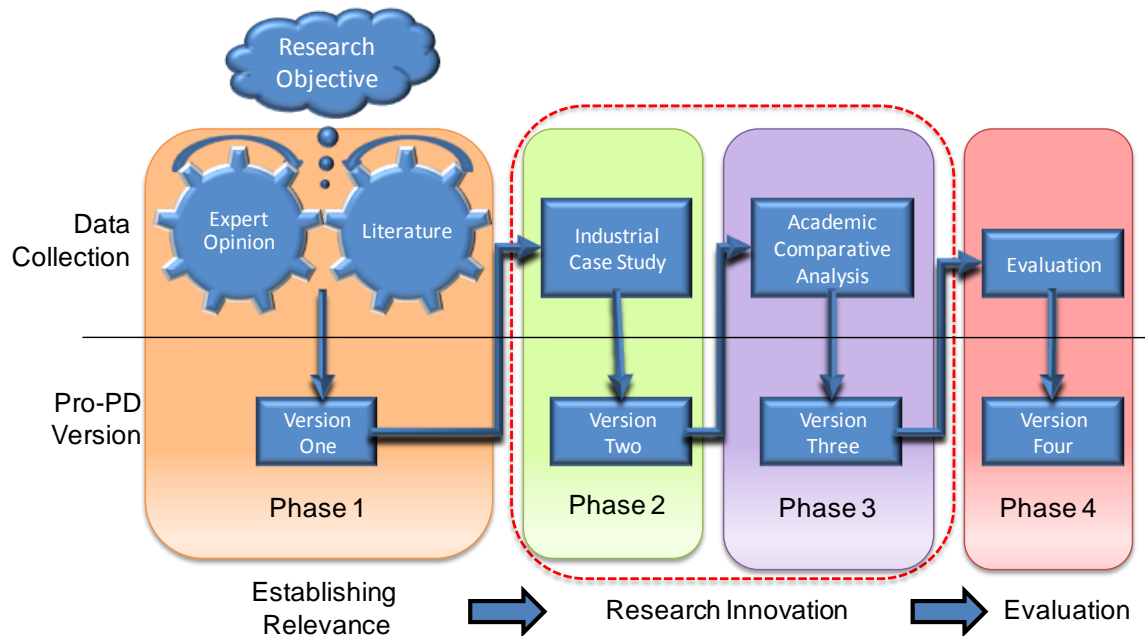


Figure 7-1 Overview of Research Design

The second phase of the research was a case study with Robert Bosch GmbH where product derivation practices within an automotive systems sub-unit were investigated. The systems produced consisted of 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 two-day workshop with key employees. Pro-PD was extended by generalizing and discussion of the observations. The output of the second phase of the research was version two of Pro-PD.

In the third phase of this research a comparative analysis between Pro-PD and another product derivation approach was conducted. DOPLER^{UCon} (**D**ecision-**O**riented **P**roduct **L**ine **E**ngineering for effective **R**euse: **U**ser-centred **C**onfiguration) [34] was developed at the Christian Doppler Laboratory for Automated Software Engineering. The

DOPLER^{UCon} approach was driven by industry needs with the overall goal to define a user-centred, tool-supported product derivation approach. During the research collaboration Pro-PD was mapped to DOPLER^{UCon}. The experience gained was used to develop Pro-PD further. The output of this third phase of the research was Pro-PD version three.

In the fourth phase of the research, the evaluation, the researcher systematically analysed the support for the model facets in prominent existing approaches. COVAMOF [22], SEI's PLPF [134], and PuLSE-I [24] were selected because of their popularity and prominence within the SPL community. To enable systematic analysis, the researcher adapted a software product line architecture evaluation framework [99]. The goal of the evaluation was not to provide a detailed survey on product derivation approaches but to validate the key activities by studying how they are supported by prominent existing approaches. The output of this final phase four of the research was the final version of Pro-PD.

7.3 Meeting the Research Objectives

Through the above research design, the following research objectives were satisfied.

7.3.1 Objective One – To define a structured process model for product derivation

Pro-PD is a process reference model for product derivation. Pro-PD describes the tasks, roles and work artefacts used to derive products from a software product line. Pro-PD is an output of the applied research design described in Section 3.4. The research meets the first objective through the development of Pro-PD. Pro-PD is described in Chapter Five of this thesis.

7.3.2 Objective Two - To demonstrate the adaptability of the process model

In Chapter Six, the research demonstrates the adaptability of Pro-PD by proposing an Agile Pro-PD (A-Pro-PD). A-Pro-PD is adapted through Use-by-Specialisation to fit the characteristics of an Agile process model.

Furthermore, the researcher of this study believes that Agile integration could strengthen Pro-PD's ability to support product derivation, particularly when dealing with small organisations. An Agile Pro-PD is meeting calls from industry for research into this area [107]. A combination of Agile and SPL is expected to create a leaner but more disciplined product derivation process [108].

The research meets the first objective through the development of A-Pro-PD. Pro-PD is described in Chapter Six of this thesis.

7.3.3 Objective Three - To add to the established knowledge on product derivation

The research proposes an Agile approach to product derivation (see Chapter Six) and identifies Agile practices that could be adopted within an Agile product derivation project (see Section 6.5).

7.4 Overview of Solution

The research collated disparate information from literature, industry practice, research projects and documented best practice to create a process reference model for Product Derivation (Pro-PD). Pro-PD is a software process for product derivation that is minimal, complete, and adaptable:

- Minimal – only content that is essential for product derivation is included
- Complete – it can be manifested as an entire process to build a system
- Adaptable – it can be adapted to different process types.

Pro-PD is a minimally sufficient process reference model for product derivation. This means that only fundamental product derivation process content is included and that it is independent of the methods and techniques used to derive a product. Pro-PD focuses on the activities, roles and work artefacts used to derive products from a software product line. Pro-PD is to be used as a foundation from which company specific product derivation process content can be developed.

The basic process building blocks of Pro-PD revolve around roles, work products and activities. Roles represent a set of related skills and responsibilities. Work products are

artefacts that are produced, modified or used by tasks. Activities are a grouping of related tasks. Tasks are assignable units of work that usually consume or produce one or more work products. Pro-PD is defined as a waterfall instantiation in Chapter Five. In Chapter Six the approach is adapted to an Agile approach.

7.5 Summary of Contribution

A literature review is presented that highlights issues within current approaches to product derivation. Observations on product derivation practice from both academia and industry are described. The potential integration of Agile practices in product derivation is demonstrated through the adaptation of the process model.

For academia, the research results provide structure to the area under concern. The work acts like a roadmap and points to areas of uncertainty, helping to identify remaining challenges. Such a roadmap encourages the insertion of those pieces that may be missing, or the extra detail that may be needed. The roadmap can offer clarity to the field. It is not claimed that the key activities presented in this research are complete. However it is hoped that other researchers can use the work described as a starting point for presenting their experiences with product derivation or as a starting point for evaluating their approaches.

For industry despite the growing adoption of SPL, product derivation remains an expensive and error-prone activity which is hard to automate and support by tools. The research in this thesis can support the adoption of automated product derivation approaches through the structuring and identification of the involved activities into a larger process model. The researcher sees the contribution of Pro-PD to the automation of product derivation as twofold: (i) it allows us to put automated approaches which tackle one particular activity into a bigger context and, (ii) it lays the foundation for tools which support the overall process (see Figure 7-2). Furthermore Pro-PD is adaptable, it can be used as a foundation from which company specific product derivation process content can be developed. Therefore providing companies with a solid starting point for the definition of an product derivation approach.

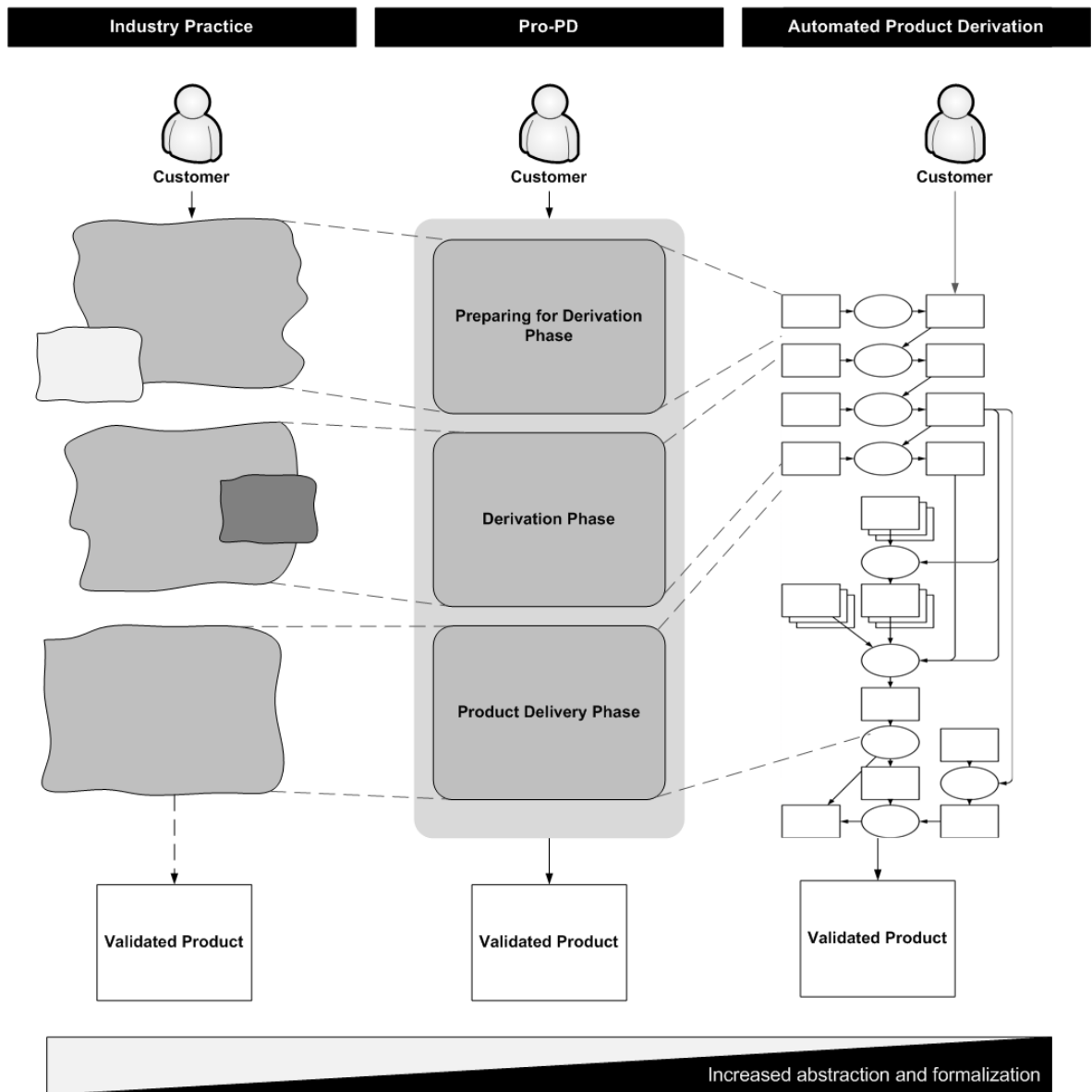


Figure 7-2 Pro-PD as a Foundation for Automated Approaches

Finally, this research also contributes to the learning through the publication of the knowledge gathered during the research (see Appendix H) and the presentation of the research to SPL groups, both academic and industrial, in Luxembourg, Germany, Ireland and Brazil.

7.6 Limitations

Pro-PD is minimally sufficient, meaning that only fundamental product derivation content is included. Factors such as the need for more formal or agile processes, team size and location, architecture complexity, conformance to standards all dictate the demands on the process however these are outside the scope of the research presented in this thesis.

The reference model described in this thesis is not the definitive guide to product derivation, and almost everybody with an interest in product derivation will find some aspect that they do not agree with. Nonetheless, it is presented as a step towards a consensus on product derivation practices and forms the anchor on which further improvements can be built.

Other limitations of the research are:

- Product Testing
- Adaptation Guidelines
- Case Study Bias

7.6.1 Product Testing

As noted in [124], testing software product lines is different from testing “normal” software since all its constituents do not have the same importance. However, there is very little evidence of product line specific testing approaches, in fact Lamancha says there is no defined and proven methodology for SPL testing [124]. There are few available details on how the process of product line testing should work other than considering reuse of platform test artefacts. The existing proposals [110, 125] are guidelines for testing activities within a SPL.

In the Robert Bosch GmbH and DOPLER^{UCon} studies, both implied the use of traditional software engineering test practices with opportunistic reuse of test artefacts where possible. The researcher envisages that a weakness in Pro-PD is its lack of support for product derivation testing.

7.6.2 Adaptation Guidelines

It is known that a well defined process can bring benefits for the organisations who adopt it. Existing organisations will have a defined and co-ordinated development process, in which roles and activities are defined and coordinated. It is important therefore for any proposed process to define how it should be tailored to an existing organisational process. Although some empirical studies show this adaptation as a critical point [135], this research does not describe rules or guidelines for the adaptation of Pro-PD.

7.6.3 Case Study Bias

As discussed in Section 3.5, all qualitative research suffers from the risk of bias and multiple interpretations of data. According to Patton [70] the small size involved in qualitative methods make it difficult to generalise the results. This is especially true in case study research where the focus on a particular case makes it unable to produce a general conclusion.

When examining the research data collected during the various research stages the researcher did his best to maintain an objective view in order to ensure minimise the bias of any conclusions. Despite this results taken from the data will be heavily influenced by the case studies conducted particularly those during the early stages of the research. Thus a limitation of the research could be a Robert Bosch GmbH bias.

7.7 Future Work

Potential future research directions for this work are as follows:

7.7.1 Industrial Application of Pro-PD

Acceptance of a reference information model can be improved by practical application. Furthermore, insights from this application can be used to further refine and improve the model.

In this instance, Pro-PD would be used to solve a practical problem within an organisation. This means that the model forms the basis of the design of an information or organisational system. In such a project, the model would not necessarily need to be used as a whole even use of selected parts of the model could be of relevance.

Following the completion of the project, the practical benefit of Pro-PD should be evaluated by both its users and the original researcher. Limitations and weaknesses in Pro-PD could be defined jointly. Pro-PD users should be asked how the weaknesses can be overcome in order to provide initial directions for the next version of Pro-PD.

7.7.2 Regulatory Conformance

The majority of regulatory requirements such as AutoSPICE [136] are primarily focused on single system development. The extension of regulatory standards to consider the most important aspects related to product line development is an important move towards the standardisation of the product line approach within regulated environments. The researcher envisages that the results of this research could be used to adapt standards such as AutoSPICE or similar standards for appropriate assessment of the product derivation process.

7.7.3 Agile Product Derivation

The researcher of this study believes that Agile integration could improve product derivation, particularly when dealing with small organisations. An investigation into Agile practices in product derivation would be meeting calls from industry for research into this area [107]. Agile practices could solve many of the problems associated with product derivation's complex and cumbersome nature. An Agile approach could be a lightweight approach to product derivation, minimising the amount of up-front investment required making SPL more accessible to small organisations with limited resources. An Agile approach to product derivation could benefit larger organisations by bringing a balance between formalism and agility, and help individual product teams deliver products of the

best possible quality. Finally, a combination of Agile and SPL is expected to create a leaner but more disciplined product derivation process.

7.8 Concluding Remarks

Sin é (Gaelic – that’s it). The model constructed is based on an extensive review of information on product derivation from a variety of sources, both academic and industrial. The process can be seen as a systematic way to perform product derivation through a well-defined sequence of activities, tasks, roles and work products.

Even it being an important contribution to the field, this work into a comprehensive process for product derivation is not complete. Further work is required for the establishment of process support for product derivation. The researcher believes this thesis is a first step in this long and complex road.

References

1. Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*. 2001, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
2. Hotz, L., Gunter, A., and Krebs, T., *A Knowledge-based Product Derivation Process and some Ideas how to Integrate Product Development*, in *Proc. of Software Variability Management Workshop*. 2003: Groningen, The Netherlands.
3. 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.
4. Deelstra, S., Sinnema, M., and Bosch, J., *Experiences in Software Product Families: Problems and Issues During Product Derivation*, in *Software Product Lines, Third International Conference*. 2004, Springer: Boston, MA, USA.
5. Griss, M.L., *Implementing Product-Line Features with Component Reuse*. ICSR-6: Proceedings of the 6th International Conference on Software Reuse. 2000, London, UK: Springer-Verlag. 137--152.
6. 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.
7. Royce, W.W., *Managing the Development of Large Software Systems: Concepts and Techniques*, in *Western Electronic Show and Convention*. 1970, WESCON Technical Papers: Los Angeles.
8. Bosch, J., *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. 1st ed. ACM Press. 2000: Addison-Wesley Professional. 368.
9. Bayer, J., Flege, O., Fettke, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J.-M., *PuLSE: a Methodology to Develop Software Product Lines*, in *Proceedings of the 1999 symposium on Software reusability*. 1999, ACM: Los Angeles, California, United States.
10. Parnas, D.L., *On the Design and Development of Program Families*, in *Programming methodology. A collection of articles by members of IFIP WG2.3*. 1978, Springer-Verlag. p. 343-361.
11. Jazayeri, M., Ran, A., and van der Linden, F., *Software Architecture for Product Families: Principles and Practice*. 2000: Addison-Wesley.
12. Ardis, M., Daley, N., Hoffman, D., Siy, H., and Weiss, D., Software Product Lines: a Case Study. *Software: Practice and Experience*, 2000. **30**(7): p. 825 - 847.
13. Northrop, L., *A Framework for Software Product Line Practice*. 2001, Software Engineering Institute: Carnegie Mellon University.
14. McGregor, J.D., Northrop, L.M., Jarrad, S., and Pohl, K., *Initiating Software Product Lines*, in *Software, IEEE*. 2002. p. 24-27.
15. Coallier, F. and Champagne, R., A Product Line Engineering Practices Model. *Science of Computer Programming*, 2005. **57**(1): p. 73-87.

16. *The Eli Whitney Museum, The Inventor: Whitney Changed the Face of the North.* [cited 12/4/2009]; Available from: <http://www.eliwhitney.org/museum/about-eli-whitney/inventor>.
17. Northrop, L., SEI's Software Product Line Tenets. *Software, IEEE*, 2002. **19**(4): p. 32-40.
18. van der Linden, F., Software product families in Europe: the Esaps & Cafe projects. *IEEE Software*, 2002. **19**(4): p. 41-49.
19. Pohl, K., Böckle, G., and v. d. Linden, F., *Software Product Line Engineering: Foundations, Principles, and Techniques*. 2005, Heidelberg: Springer.
20. 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.
21. Krebs, T., Wolter, K., and Hotz, L., *Model-based Configuration Support for Product Derivation in Software Product Families*. 2005: Koblenz, Germany.
22. Sinnema, M., Deelstra, S., and Hoekstra, P., *The COVAMOF Derivation Process*. Vol. 4039 LNCS. 2006: Springer Berlin / Heidelberg. 101-114.
23. Rabiser, R., Dhungana, D., Grünbacher, P., Lehner, K., and Federspiel, C., Product configuration support for nontechnicians: Customer-centered software product-line engineering. *IEEE Intelligent Systems*, 2007. **22**(1): p. 85-87.
24. Bayer, J., Gacek, C., Muthig, D., and Widen, T. PuLSE-I: Deriving Instances from a Product Line Infrastructure. in *7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. 2000. Edinburgh, UK.
25. Atkinson, C., Bayer, J., and Muthig, D., *Component-based product line development: the Kobra approach*, in *Proceedings of the first conference on Software product lines : experience and research directions*. 2000, Kluwer Academic Publishers: Denver, Colorado, United States.
26. Guelfi, N. and Perrouin, G., *A Flexible Requirements Analysis Approach for Software Product Lines*, in *Requirements Engineering: Foundation for Software Quality*. 2007, Springer Berlin / Heidelberg. p. 78-92.
27. Perrouin, G., Klein, J., Guelfi, N., and Jezequel, J.M. Reconciling Automation and Flexibility in Product Derivation. in *12th International Software Product Line Conference (SPLC)*. 2008.
28. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M., FORM: A Feature Oriented Reuse Method with Domain Specific Reference Architectures. *Annals of Software Engineering*, 1998. **5**(1): p. 143-168.
29. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., and Peterson, A.S. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, CMU/SEI-90TR-21. Pittsburgh, PA, USA Carnegie Mellon Software Engineering Institute.
30. Weiss, D.M. and Lai, C.T.R., *Software product line engineering: A Family-based Software Development Process*. 1st ed. 1999: Addison-Wesley Professional.
31. Kim, S.D., Min, H.G., Her, J.S., and Chang, S.H., *DREAM: A Practical Product Line Engineering using Model Driven Architecture*, in *Proceedings of the Third*

- International Conference on Information Technology and Applications (ICITA'05)*. 2005, IEEE Computer Society: Washington, DC, USA. p. 70-75.
32. 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.
 33. Sinnema, M., Deelstra, S., Nijhuis, J., and Bosch, J., *Modeling Dependencies in Product Families with COVAMOF*, in *13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006)*. 2006: Potsdam, Germany.
 34. Rabiser, R., *A User-Centered Approach to Product Configuration in Software Product Line Engineering*, in *Christian Doppler Laboratory for Automated Software Engineering*. 2009, Institute for Systems Engineering and Automation, Johannes Kepler University: Linz.
 35. Rabiser, R. and Dhungana, D. *Integrated Support for Product Configuration and Requirements Engineering in Product Derivation*. in *33rd EUROMICRO Conference on Software Engineering and Advanced Applications*. 2007.
 36. Matinlassi, M., Niemelá, E., and Dobrica, L. (2002). *Quality-Driven Architecture Design and Quality Analysis Method: A Revolutionary Initiation Approach to a Product Line Architecture*. Espoo: Technical Research Centre of Finland.
 37. Rombach, D., *Fraunhofer: the German model for Applied Research and Technology Transfer*, in *Proceedings of the 22nd international conference on Software engineering (ICSE2000)*. 2000, ACM: Limerick, Ireland.
 38. Böckle, G., Muñoz, J.B., Knauber, P., Krueger, C., Leite, J.C.S.d.P., Linden, F.v.d., M. Northrop, L., Stark, M., and M. Weiss, D., *Adopting and Institutionalizing a Product Line Culture*, in *Proceedings of the Second International Conference on Software Product Lines (SPLC 2002)*. 2002, Springer-Verlag: San Diego, CA, USA.
 39. Czarnecki, K., Helson, S., and Eisenecker, U.W. *Staged Configuration using Feature Models*. in *Proc. of the 3rd International Software Product Line Conference (SPLC 2004)*. 2004. Boston, MA, USA: Springer Berlin Heidelberg.
 40. Halmans, G. and Pohl, K., *Communicating the Variability of a Software-Product Family to Customers*. *Informatik - Forschung und Entwicklung*, 2003. **18**(3-4): p. 113-131.
 41. Ezran, M., Morisio, M., and Tully, C., *Practical Software Reuse*. 2002: Springer-Verlag. 222.
 42. Boehm, B.W., *A Spiral Model of Software Development and Enhancement*. *IEEE Computer*, 1988. **21**(5): p. 61-72.
 43. Beck, K., *Embracing Change with Extreme Programming*, in *IEEE Computer*. 1999. p. 70 - 77.
 44. Schwaber, K. and Beedle, M., *Agile Software Development with Scrum*. 2001: Prentice Hall PTR.

45. Humphrey, W.S. and Kellner, M.I. (1989). *Software Process Modeling: Principles of Entity Process Models*, CMU/SEI-89-TR-002 Carnegie Mellon Software Engineering Institute.
46. Beck, K., Beedle, M., Bennekum, A.v., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. *Manifesto for Agile Software Development*. 2001 1/3/2006 [cited 10/09/2008]; Available from: <http://agilemanifesto.org/>.
47. Beck, K. and Andres, C., *Extreme Programming Explained : Embrace Change (2nd Edition)*. 2004: Addison-Wesley Professional.
48. Beck, K., *Extreme Programming Explained: Embrace Change*. 1999, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. 190.
49. Highsmith, J. and Cockburn, A., *Agile Software Development: The Business of Innovation*, in *IEEE Computer*. 2001. p. 120-122.
50. Rakitin, S., Manifesto Elicits Cynicism. *IEEE Computer*, 2001. **34**(12): p. 4.
51. Highsmith, J., Cockburn, A., and Orr, K., Extreme Programming. *Cutter Consortium e-Business Application Delivery*, 2000: p. 4-17.
52. Reinhartz-Berger, I., Soffer, P., and Sturm, A. A Domain Engineering Approach to Specifying and Applying Reference Models. in *Proceedings of the Workshop Enterprise Modelling and Information Systems Architectures*. 2005. Klagenfurt, Austria: German Informatics Society.
53. Winter, R. and Schelp, J., *Reference Modeling and Method Construction: a Design Science Perspective*, in *Proceedings of the 2006 ACM symposium on Applied computing*. 2006, ACM: Dijon, France.
54. Esswein, W., Zumpe, D.K.S., and Sunke, N., *Identifying the Quality of e-commerce Reference Models*, in *Proceedings of the 6th international conference on Electronic commerce*. 2004, ACM: Delft, The Netherlands.
55. Bernus, P. (1999). *GERAM: Generalised Enterprise Reference Architecture and Methodology*: IFIP-IFAC Task Force on Architectures for Enterprise Integration.
56. Braun, R., Esswein, W., Gehlert, A., and Weller, J., *Configuration Management for Reference Models*, in *Reference Modeling for Business Systems Analysis*, P. Fettke and P. Loos, Editors. 2006, IGI. p. 331-357.
57. *Capability Maturity Model Integration (CMMI)*, Carnegie Mellon Software Engineering Institute.
58. Ahmed, F., Capretz, L.F., and Campbell, P., *Software Product Lines: A Process Assessment Methodology, A Practitioner's Approach*. 2009: VDM Verlag.
59. van der Linden, F., Schmid, K., and Rommes, E., *The Family Evaluation Framework*, in *Software Product Lines in Action*. 2007. p. 79-108.
60. *Product Line Technical Probe*. [cited 02/08/2009]; Available from: <http://www.sei.cmu.edu/productlines/consulting/techprobe/index.cfm>.
61. Jenkins, A.M., *Research Methodologies and MIS Research*, in *Research Methods in Information Systems*, E. Mumford, et al., Editors. 1984, North-Holland: Amsterdam, The Netherlands. p. 103-117.

62. Klein, H.K. and Myers, M.D., A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, 1999. **23**(1): p. 67-93.
63. Evered, R. and Louis, M.R., Alternative Perspectives in the Organizational Sciences: "Inquiry from the inside" and "Inquiry from the outside". *The Academy of Management Review*, 1981. **6**(3): p. 385-395.
64. Orlikowski, W. and Baroudi, J., Studying Information Technology in Organizations: Research Approaches and Assumptions. *Information Systems Research*, 1991.
65. Lee, A.S., A scientific methodology for MIS case studies. *MIS Quarterly*, 1989. **13**(1): p. 33-50.
66. Yin, R., *Case Study Research : Design and Methods*. 2003: SAGE Publications.
67. Patton, M.Q., *How to Use Qualitative Methods in Evaluation*. 1987, California: Sage Publications Inc.
68. Blaxter, L., Hughes, C., and Tight, M., *How to Research*. Second ed. 2001: Open University Press. 286.
69. Silverman, D., *Interpreting Qualitative Data*. 2nd Edition ed. 2001, London: London: Sage. 325 p.
70. Patton, M.Q., *Qualitative Evaluation & Research Methods*. 2nd ed. 1990, Newbury Park, CA: Sage Publications.
71. Dancy, J., *Introduction to Contemporary Epistemology*, ed. B. Publishers. 1985, Oxford: Wiley-Blackwell. 272
72. Cooper, D. and Schindler, P., *Business Research Methods*. 8th ed ed. 2003, New York: McGraw-Hill.
73. Strauss, A. and Corbin, J., *Basics of Qualitative Research*. 1998, Thousand Oaks, California: Sage Publications, Inc.
74. Eisenhardt, K.M., Control: Organizational and Economic Approaches. *Management Science*, 1985. **31**(2): p. 134-149.
75. Järvinen, P., *On Research Methods*. 2001, Tampere, Finland: Juvenes-Print.
76. Gill, J. and Johnson, P., *Research Methods for Managers*. 1991, London: Paul Chapman.
77. Galliers, R.D., *Choosing Information Systems Research Approaches*, in *Information Systems Research: issues, methods and practical guidelines*, R.D. Galliers, Editor. 1992, Blackwell Scientific: Oxford.
78. Franz, C.R., Robey, D., and Koeblitz, R.R., User Response to an Online Information System: A Field Experiment. *MIS Quarterly*, 1986. **10**(1): p. 29-42.
79. Morse, J.M., *Principles of Mixed Methods and Multimethod Research Design*, in *Handbook of Mixed Methods in Social and Behavioral Research*, I.T. Teddlie, Editor. 2003, Sage Publications: Thousand Oaks, CA.: p. 189-208.
80. Woods, M., Daly, J., Miller, J., and Roper, M., Multi-Method Research: an Empirical Investigation of Object-oriented Technology. *Journal of Systems and Software*, 1999. **48**(1): p. 13-26.

81. Ahlemann, F. and Gastl, H., *Process Model for an Empirically Grounded Reference Model Construction*, in *Reference Modeling for Business Systems Analysis*, P. Fettke and P. Loos, Editors. 2006, IGI Publishing.
82. Rosemann, M. and Schütte, R., *Multi-Perspective Reference Modelling*, in *Referenzmodellierung. State-of-the-art und entwicklungsperpektiven*, J. Becker, M. Rosemann, and R. Schütte, Editors. 1999, Physica-Verlag: Heidelberg. p. 22-44.
83. Schlagheck, B., *Object-oriented Reference Models for Process and Project Controlling - Foundation Construction Fields of Application*. 2000: Wiesbaden: Deutscher Univ. Verlag.
84. Walsham, G., *Interpreting Information Systems in Organizations*. 1993, New York, NY, USA: John Wiley & Sons, Inc. 286.
85. Perry, D., Sim, S.E., and Easterbrook, S., *Case Studies for Software Engineers*, in *Proceedings of the 26th International Conference on Software Engineering*. 2004.
86. Hammersley, M., Gomm, R., and Foster, P., *Case Study Method: Key Issues, Key Texts*. 2000, London: Sage Publications.
87. Fettke, P. and Loos, P., *Reference Modeling for Business Systems Analysis*. 2006: IGI Publishing.
88. Krallmann, H., Frank, H., and Gronau, N., *Systems Analysis in Enterprises. Process model, Modelling Techniques and Design Options*. 4th ed. 2002, München: Oldenbourg.
89. Patton, M.Q., *Qualitative Research & Evaluation Methods*. 3rd ed. 2002, Thousand Oaks, California: Sage Publications, Inc.
90. Cooper, H.M., *The Integrative Research Review: A Systematic Approach*. 1984, Beverly Hills: Sage Publications, Inc. 139.
91. Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., and Rosenberg, J., Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 2002. **28**(8): p. 721 - 734.
92. Becker, J., Niehaves, B., and Knackstedt, R., *Framework for the Epistemological Positioning of Reference Modelling*. , in *Referenz-modellierung. Grundlagen, techniken und domänenbezogene Anwendung*, J. Becker and P. Delfmann, Editors. 2004, Physica-Verlag.: Münster. p. 1-17.
93. Schütte, R., *Principles of Methodical Reference Modelling - Construction of Conigurative and Adaptable Models*. 1998, Wiesbaden: Gabler.
94. *The SPLC Product Line Hall of Fame*. [cited 3/02/2009]; Available from: <http://www.splc.net/fame.html>.
95. Rabiser, R. Flexible and User-Centered Visualization Support for Product Derivation. in *2nd International Workshop on Visualisation in Software Product Line Engineering (ViSPLE 2008)*. 2008. Limerick, Ireland.
96. Rabiser, R., Dhungana, D., Grünbacher, P., Lehner, K., and Federspiel, C. Involving Non-Technicians in Product Derivation and Requirements Engineering: A Tool Suite for Product Line Engineering. in *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*. 2007.

97. Rabiser, R., Wolfinger, R., and Grünbacher, P., *Three-level Customization of Software Products Using a Product Line Approach*, in *Proc. of the 42nd Hawai'i International Conference on System Sciences*. 2009, IEEE CS: Waikoloa, Big Island, HI, USA.
98. O'Leary, P., Rabiser, R., Richardson, I., and Thiel, S. Important Issues and Key Activities in Product Derivation: Experiences from Two Independent Research Projects. in *Software Product Line Conference*. 2009. San Francisco, CA, USA: Proc. of the 13th International Software Product Line Conference (SPLC 2009).
99. Matinlassi, M. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*. 2004. EICC, Scotland, UK.
100. Creswell, J.W., *Research Design – Qualitative & Quantitative Approaches*. 1994, Thousand Oaks, CA: Sage Publications.
101. Marshall, C., *Goodness Criteria: Are they Objective or Judgement Calls?*, in *The Paradigm Dialog*, E.G. Guba, Editor. 1990, Sage Publications: Newbury Park, CA. p. 188–197.
102. Seale, C., *The Quality of Qualitative Research*. 2000, London: SAGE Publications. 224.
103. Lincoln, Y.S. and Guba, E.G., *Naturalistic Enquiry*. 1985, Beverly Hills: Sage Publications. 416.
104. Liamputtong, P. and Ezzy, D., *Qualitative Research Methods*. 2005, USA: Oxford University Press, 2 edition. 424.
105. Reason, P. and Rowan, J., *Issues of Validity in New Paradigm Research*, in *Human Inquiry: A Sourcebook of New Paradigm Research*, J. Rowan, Editor. 1981: Chichester: Wiley.
106. O'Leary, P., McCaffery, F., Richardson, I., and Thiel, S. Towards Agile Product Derivation in Software Product Line Engineering. in *16th European Conference on Software Process Improvement (EuroSPI 2009)*. 2009. Madrid, Spain.
107. Bosch (2006). *Presentation: Product Derivation - Research Interests within BOSCH*, Lero,Lero, University of Limerick,Dec, 2006.
108. Kurmann, R. Agile SPL-SCM Agile Software Product Line Configuration and Release Management. in *1st International Workshop on Agile Product Line Engineering (APLE'06)*. 2006. Maryland, USA.
109. Sommerville, I., *Software Engineering*. 7th ed. 2004: Pearson Addison Wesley.
110. Pohl, K. and Metzger, A., *Software Product Line Testing*. 2006, Communications of the ACM. p. 78-81.
111. O'Leary, P., Ali Babar, M., Thiel, S., and Richardson, I. 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. Poznań, Poland: Wydawnictwo NAKOM.
112. O'Leary, P., Ali Babar, M., Thiel, S., and Richardson, I. Towards Agile Product Derivation in Software Product Line Engineering. in *4th International Workshop*

- on Rapid Integration of Software Engineering techniques (RISE)*. 2007. Luxembourg.
113. 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).
 114. Czarnecki, K. and Eisenecker, U.W., *Generative and Component-Based Software Engineering, First International Symposium, GCSE'99, Erfurt, Germany, September 28-30, 1999, Revised Papers*. Lecture Notes in Computer Science. Vol. 1799. 2000: Springer.
 115. Sinnema, M. and Deelstra, S., Industrial Validation of COVAMOF. *Journal of Systems and Software*, 2008. **81**(4): p. 584-600.
 116. Sinnema, M., Deelstra, S., Nijhuis, J., and Bosch, J., *COVAMOF: A Framework for Modeling Variability in Software Product Families*, in *Software Product Lines*. 2004. p. 197-213.
 117. Schmid, K., John, I., Kolb, R., and Meier, G., *Introducing the PuLSE approach to an Embedded System Population at Testo AG*, in *Proceedings of the 27th international conference on Software engineering*. 2005, ACM: St. Louis, MO, USA.
 118. IBM. *Eclipse Process Framework Composer*. 2007 [cited; Available from: http://www.eclipse.org/epf/downloads/tool/tool_downloads.php].
 119. O'Leary, P., Richardson, I., and Thiel, S., *Developing a Product Derivation Process Framework for Software Product Line Organisations*, in *EuroSPI 2008 Doctoral Symposium*. 2008: Dublin, Ireland.
 120. Chastek, G., Donohoe, P., and McGregor, J.D. (2002). *Product Line Production Planning for the Home Integration System Example*, in Technical Report CMU/SEI-2002-TN-029. Pittsburgh: Carnegie Mellon Software Engineering Institute.
 121. Birk, A., Heller, G., John, I., Maßen, T.v.d., Müller, K., and Schmid, K., Product Line Engineering: The State of the Practice. *IEEE Software*, 2003. **20**(6): p. 52-60.
 122. Nestor, D., O'Malley, L., Quigley, A., Sikora, E., and Thiel, S., *Visualisation of Variability in Software Product Lines*, in *Proceedings of the 1st International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS-2007)*. 2007: Limerick, Ireland.
 123. Kuloor, C. and Eberlein, A., *Requirements Engineering for Software Product Lines*, in *Proceedings of the 15th International Conference on Software & Systems Engineering and their Applications (ICSSEA'02)*. 2002: Paris, France.
 124. Lamancha, B.P., Usaola, M.P., and Velthius, M.P., *Software Product Line Testing - A Systematic Review*, in *4th International Conference on Software and Data Technologies (ICSoft)*. 2009: Sofia, Bulgaria.
 125. McGregor, J.D. (2001). *Testing a Software Product Line*, CMU/SEI-2001-TR-022. Pittsburgh, PA: Carnegie Mellon Software Engineering Institute.

126. Tian, K. and Cooper, K., *Agile and Software Product Line Methods: Are They So Different?*, in *1st International Workshop on Agile Product Line Engineering*. 2006: Maryland, USA.
127. Trinidad, P., Benavides, D., Ruiz-Cortes, A., and Segura, S. Explanations for Agile Feature Models. in *1st International Workshop on Agile Product Line Engineering (APLE'06)*. 2006. Maryland, USA.
128. Carbon, R., Lindvall, M., Muthig, D., and Costa, P. Integrating Product Line Engineering and Agile Methods: Flexible Design Up-front vs. Incremental Design. in *1st International Workshop on Agile Product Line Engineering (APLE'06)*. 2006. Maryland, USA.
129. Navarrete, F., Botella, P., and Xavier, F. An Approach to Reconcile the Agile and CMMI Context in Product Line Development. in *1st International Workshop on Agile Product Line Engineering (APLE'06)* 2006. Maryland, USA.
130. Noor, M., Rabiser, R., and Grünbacher, P., *A Collaborative Approach for Reengineering-based Product Line Scoping*, in *1st International Workshop on Agile Product Line Engineering (APLE'06)*. 2006: Maryland, USA.
131. McGregor, J.D., Agile Software Product Lines, Deconstructed. *Journal of Object Technology*, 2008. **7**(8): p. 7 - 19.
132. Kruchten, P., *The Rational Unified Process: An Introduction*. 2003: Addison-Wesley Longman Publishing Co., Inc. 320.
133. Muccini, H. and van der Hoek, A., *Towards Testing Product Line Architectures*, in *European Joint Conferences on Theory and Practice of Software (ETAPS)*. 2003, In Electronic Notes of Theoretical Computer Science: Warsaw, Poland.
134. Northorp, L. and Clements, P. *A Framework for Software Product Line Practice, version 5.0*. 2007 30th April 2008 [cited; Available from: <http://www.sei.cmu.edu/productlines/framework.html>].
135. Morisio, M., Ezran, M., and Tully, C., Success and Failure Factors in Software Reuse. *IEEE Transactions on Software Engineering*, 2002. **28**(4): p. 340-357.
136. *Automotive SPICE™ Process Reference Model (PRM)*. 2008 May 5, 2008 [cited; v4.3:[Available from: <http://www.automotivespice.com/web/dwl.html>].

Appendix A: Product Line Practice Framework Evaluation

Practice Name	Architecture Definition (AD)
Application to Product Development	<p>Once the platform architecture has been placed in the core asset base for the product line, the platform architecture is used to create product architectures (AD-1) for each new product according to the architecture's attached process.</p> <p>If the product builders discover a variation point or a needed mode of variation that is not permitted by the architecture, they should bring it to the architect's attention (AD-2); if the variation is within the product line's scope (or deemed desirable to add to the scope), the architecture may be enhanced to accommodate it (AD-3).</p>
Key Criteria	<p>AD-1: The platform architecture is used to create instances of the product architecture</p> <p>AD-2: The product developers provide feedback to the platform team on appropriate variation points</p> <p>AD-3: There are new releases of the platform architecture in order to facilitate product requests</p>
Pro-PD	<p>AD-1: The <i>Base Product Configuration</i> is constructed in the <i>Derive New Configuration</i> task by either choosing from the <i>Existing Platform Configurations</i> or deriving a new product configuration from the <i>platform architecture</i>. This satisfies AD-1.</p> <p>AD-2: In the <i>Provide Feedback to Platform Team</i> task, feedback is provided to the platform team on architecture usage including appropriate variation points. This satisfied AD-2.</p> <p>AD-3: In the case of required platform extensions which were identified in the <i>Preparing for Derivation</i> phase, the platform</p>

	team receives platform software requirements containing the required extensions to the existing platform in order to facilitate the new <i>product specific requirements</i> . This satisfies AD-3.
--	---

Table A-1 Evaluation of Architecture Definition Practice Area

Practice Name	Architecture Evaluation (AE)
Application to Product Development	Product architectures should be evaluated to make sure they meet the specific behavioural and quality requirements of the product at hand (AE-1). In practice, the extent to which product architectures require separate evaluations depends on the extent to which they differ from the product line architecture and on the degree of automation used in creating them. The results of architecture evaluation for product architectures often provide useful feedback to the architect(s) of the product line architecture and fuel improvements in it (AE-2).
Key Criteria	AE-1: The instantiated product architecture is evaluated to see if it meets the specific behavioural and quality requirements of the product at hand AE-2: Feedback from the architecture evaluation is provided to the platform team
Pro-PD	AE-1: Once the product architecture in the <i>Base Product Configuration</i> is instantiated there is no specific evaluation of product architecture. AE-1 is not satisfied. AE-2: As stated earlier, no architecture evaluation is implied in Pro-PD therefore AE-2 can not be satisfied.

Table A-2 Evaluation of Architecture Evaluation Practice Area

Practice Name	Component Development
Application to Product	Components for a product are either used directly from the core asset base (CD-1), used directly after binding the built-in

<p>Development</p>	<p>variations (CD-2), used after modification or adaptation (CD-3), or developed anew (CD-4).</p> <p>New component development should occur only after a thorough search has been made of existing core assets (CD-5).</p> <p>Whether a product component is adapted or built from scratch, it should be reviewed ultimately for "promotion" to the core asset base (CD-6). Additionally, the team identifies changes that are required in the product in order to support the new or adapted components (CD-7).</p>
<p>Key Criteria</p>	<p>CD-1: Components can be used directly from the core asset base</p> <p>CD-2: Components can be used directly after binding the built-in variations</p> <p>CD-3: Components can be used after modification or adaptation</p> <p>CD-4: Components can be developed anew</p> <p>CD-5: Before new component development occurs a thorough search for existing assets is performed</p> <p>CD-6: Whether a product component is adapted or built from scratch, it should be reviewed ultimately for "promotion" to the core asset base</p> <p>CD-7: The team identifies changes that are required in the product in order to support the new or adapted components.</p>
<p>Pro-PD</p>	<p>CD-1: In the <i>Select Platform Components</i> task, components are selected from the collection of Platform Components. This satisfies CD-1.</p> <p>CD-2: In the <i>Select Platform Components</i> task, components are selected from the collection of Platform Components. Selection of platform components for product use involves binding any built-in variations. This satisfies CD-2.</p> <p>CD-3: In the <i>Components Development</i> task the development to adapt an existing platform component at product level is performed. This satisfies CD-3.</p>

	<p>CD-4: In the <i>Components Development</i> task the development to develop a new product component is performed. This satisfies CD-4.</p> <p>CD-5: Through <i>Coverage Analysis</i> the <i>Customer Specific Product Requirements</i> which are covered by the platform are identified. Here we see which requirements are covered by the platform. This satisfies CD-5.</p> <p>CD-6: In the <i>Provide Feedback to Platform Team</i> task, feedback is provided to the platform team on core asset usage during the project, how user friendly the platform assets were and areas for improvement within the platform. In addition, the product team identify product specific components that the platform could potentially benefit from through adoption. This satisfies CD-6.</p> <p>CD-7: In the task <i>Identify Required Product Development</i> the required product development required to satisfy <i>Product Requirements</i> which were not satisfied the <i>Derive the Product</i> activity are outlined. This product development can involve construction of new components or adaptation of existing components. This satisfies CD-7.</p>
--	---

Table A-3 Evaluation of Component Development Practice Area

Practice Name	Testing (T)
Application to Product Development	<p>Testing is used in two fundamental ways for products: Between the different phases of the development process to <i>verify</i> that what was produced in the last phase is correct and suitable as input to the next phase (T-1) and to <i>validate</i> a product against its requirements (T-2). Validation tests are intended to evaluate correctness relative to requirements.</p> <p>In a product line effort, the main product development activity is assembling products from core assets. The majority of each</p>

	<p>product's specification will be defined in documentation generic to the product line. Define a complete set of functional tests for that specification (T-3).</p> <p>Define a set of interaction tests that will ensure that the additions made by the product developers do not cause failures (T-4). The tests used at the product level can be derived from the functionality tests, or possibly test templates, created at the product line level (T-5).</p> <p>The mapping between product core assets and testing assets facilitates the reuse of these test assets. The mapping associates the test cases, as well as the test drivers and test data sets, with those requirements that are common across the product line. By taking advantage of this commonality, the amount of effort associated with testing and retesting a product is reduced.</p>
Key Criteria	<p>T-1: Testing is used between phases of the development process to <i>verify</i> that what was produced in the last phase is correct and suitable as input to the next phase (T-1)</p> <p>T-2: Testing is used to <i>validate</i> a product against its requirements</p> <p>T-3: A complete set of functional tests test cases for the product specification is defined</p> <p>T-4: Define a set of interaction tests that will ensure that the additions made by the product developers do not cause failures are defined</p> <p>T-5: Use Platform test artefacts as basis for product test artefacts</p>
Pro-PD	<p>T-1: There is no testing of artefacts between phases of the development process. T-1 is not satisfied.</p> <p>T-2: In the <i>Run System Tests</i> task, the product is checked for compliance with the product specific requirements. This satisfies T-2.</p> <p>T-3: In the <i>Find and Outline Requirements</i> task, the functional</p>

	<p>and non-functional requirements for the system are specified. In <i>Create the Product Test Cases</i> task, the <i>Product Test Cases</i> are written for requirements identified in the <i>Find and Outline Requirements</i> task. This satisfies T-3.</p> <p>T-4: In the <i>Create the Product Test Cases</i> task, the <i>Product Test Cases</i> are written for requirements identified in the <i>Find and Outline Requirements</i> task. This satisfies T-4.</p> <p>T-5: During this task the Product Team uses the <i>Platform Test Case Artefacts</i> as a basis for the creation of the product specific test cases. This satisfies T-5.</p>
--	---

Table A-4 Evaluation of Testing Practice Area

Practice Name	Software System Integration (SSI)
Application to Product Development	<p>Software system integration refers to the practice of combining individually tested software components into an integrated whole. Software system integration is, or should be, the primary activity in product development within a product line.</p> <p>The production plan guides the core asset integration process, defining how developers will select the appropriate core assets from all the available ones and construct a product (SSI-1).</p> <p>All assets such as requirements, architecture and testing assets contribute to the construction and must be integrated into a delivered product (SSI-2). Integration may involve tailoring assets according to planned variations or developing product-unique software components, requirements, tests, and so forth (SSI-3).</p> <p>The core assets are engineered to work together in accordance with the product line architecture but still require tailoring and integration to build a product (SSI-4). The attached process guides tailoring and integration at the core asset level.</p>

	<p>If the production plan for a specific product calls for the addition of components or internal changes in components, some integration may be required depending on the nature of the changes. These changes are known up front and can be planned along with core asset integration.</p>
Key Criteria	<p>SSI-1: The production plan guides the core asset integration process</p> <p>SSI-2: Requirements, architecture and testing assets all contribute to the construction and must be integrated into a delivered product</p> <p>SSI-3: Integration may involve tailoring assets according to planned variations or developing product-unique software components, requirements, tests, and so forth</p> <p>SSI-4: The core assets are engineered to work together in accordance with the product line architecture but still require tailoring and integration to build a product</p>
Pro-PD	<p>SSI-1: The selection of platform components involves binding any built-in variations. The attached component documentation is used to guide this binding process during product integration. This satisfies SSI-1.</p> <p>SSI-2: Platform assets such as requirements, architecture and testing assets are used during the product construction process. This satisfies SSI-2.</p> <p>SSI-3: In Pro-PD integration can involve the tailoring of platform assets. For instance, in the <i>Product Integration</i> task, the <i>Developed or Adapted Components</i> can require writing sufficient “glue” code to implement interfaces or implementing architectural changes to facilitate the developed/adapted assets. This satisfies SSI-3.</p> <p>SSI-4: Pro-PD facilitates the tailoring and integration of core assets. In the <i>Component Development</i> task core assets can be tailored. In the <i>Product Integration</i> task components or architecture can be tailored to allow integration. This satisfies SSI-4.</p>

Table A-5 Evaluation of Software System Integration Practice Area

Practice Name	Operations (O)
Application to Product Development	<p>This practice spells out how the organization uses the core asset base to build products. The operational concept for product development documents the organization's decisions regarding</p> <ul style="list-style-type: none"> • the organizational elements and the role each of them plays in product development (O-1) • the strategy for maintaining the products and coordinating that maintenance with the evolution of the core assets, including configuration management policies and processes (O-2) • specific guidance for feeding the results of using core assets in product development back to those responsible for core assets in order to support the continued improvement of those assets (O-3) • specific guidance for recommending new core assets or changes to existing core assets (O-4)
Key Milestones	<p>O-1: The organisational elements and roles each of them plays in product development are defined</p> <p>O-2: The strategy for maintaining the products and coordinating that maintenance with the evolution of the core assets is defined</p> <p>O-3: Specific procedures have been set in place for ensuring feedback on the use of core assets from product development teams to the core asset teams</p> <p>O-4: Feedback is provided to the platform team on problems experienced using the core assets</p>
Pro-PD	O-1: In the <i>Allocate Requirements</i> task, the <i>Product Requirements</i> are allocated to the relevant organisational disciplines such as hardware, algorithms or software

	<p>disciplines. The goal is to define who is responsible for the different product requirements. This satisfies O-1.</p> <p>O-2: Pro-PD does not consider product maintenance. O-2 is not satisfied.</p> <p>O-3: In the <i>Provide Feedback to Platform Team</i> task, feedback is provided to the platform team on core asset usage during the project. This satisfies O-3.</p> <p>O-4: In the <i>Coordinate with Platform Team</i> task, feedback is provided to the platform team on problems experienced with core asset usage. This satisfies O-4.</p>
--	---

Table A-6 Evaluation of Operations Practice Area

Practice Name	Customer Interface Management (CIM)
Application to Product Development	<p>Those responsible for interacting with a particular customer about a product must act on behalf of both the customer and the product line organization at large (CIM-1).</p> <p>Typically the product manager role is responsible for representing the product developer to the customer, liaison with the customer, finalising negotiations and overseeing the programmatic aspects of the development (CIM-2).</p> <p>The organization's customer interface comes heavily into play in requirements negotiation for a product or set of products. A healthy customer interface will allow a customer to negotiate in the context of the benefits and limitations that a product line approach can provide (CIM-3). From the organization's side, it should be able to offer high-quality products of demonstrated capabilities and performance with predictable delivery dates and predictable costs.</p>

Key Criteria	<p>CIM-1: Those responsible for interacting with a particular customer about a product must act on behalf of both the customer and the product line organization at large</p> <p>CIM-2: The product manager role is responsible for representing the product developer to the customer, liaison with the customer, finalising negotiations and overseeing the programmatic aspects of the development</p> <p>CIM-3: There is a facility for customer requirements negotiation</p>
Pro-PD	<p>CIM-1: Both the <i>Product Manager</i> and the <i>Platform Architect</i> are assigned to the <i>Customer Negotiation</i> task. They are responsible for meeting as many of the customer's needs as possible while retaining the profitability of the platform. This satisfies CIM-1.</p> <p>CIM-2: The product manager responsibilities include customer relationship management, negotiation of customer requirements and liaising with the product team regarding product requirements and delivery. This satisfies CIM-2</p> <p>CIM-3: The <i>Customer Negotiation</i> task allows for requirements negotiation with the customer. This satisfies CIM-3.</p>

Table A-7 Evaluation of Customer Interface Management Practice Area

Practice Name	Structuring the Organisation (SO)
Application to Product Development	<p>This practice area deals with placing roles and responsibilities into the appropriate organizational units (SO-1) to most effectively support the product line approach. For the team assigned to product development, responsibilities include the following:</p>

	<ul style="list-style-type: none"> • making sure that each new product uses the core asset base according to the production plan (SO-2) • working with the core asset owners to evolve new capabilities if the core assets are deficient in some way for a new product (SO-3) • providing feedback to the core asset developers concerning the suitability and quality of the core assets (SO-4) <p>The product team may also negotiate customer requirements to situate new products within the scope of the product line to the greatest degree possible (SO-5)</p>
Practice Risks	<p>SO-1: The roles and responsibilities for the derivation project have been assigned</p> <p>SO-2: There is a role responsible for making sure that each new product uses the core asset base according to the production plan</p> <p>SO-3: There is a role responsible for working with the core asset owners to evolve new capabilities if the core assets are deficient in some way for a new product</p> <p>SO-4: There is a role responsible for providing feedback to the core asset developers concerning the suitability and quality of the core assets</p> <p>SO-5: The product unit(s) may negotiate customer requirements to situate new products within the scope of the product line to the greatest degree possible</p>
Pro-PD	<p>SO-1: In the <i>Allocate Requirements</i> task, the relevant organisational disciplines such as hardware, algorithms or software disciplines are assigned responsibility to requirements in the <i>Product Requirements</i>. In Pro-PD roles are pre-assigned responsibility for certain tasks. SO-1 is satisfied.</p>

	<p>SO-2: The <i>Product Architect</i> is responsible for defining the product architecture, ensuring the correct leveraging of platform artefacts and the overall design of the product. SO-2 is satisfied.</p> <p>SO-3: The <i>Product Manager</i> and <i>Product Architect</i> are responsible for negotiating changes to platform architecture to support new product development and requesting platform amendments to support product requests. SO-3 is satisfied.</p> <p>SO-4: The <i>Product Manager</i> is responsible for providing feedback to the platform team on the suitability and quality of the core assets. This satisfies SO-4.</p> <p>SO-5: In the task <i>Customer Negotiation</i>, the <i>Product Manager</i> negotiates with the customer regarding requirements outside the scope of the product line. This satisfies SO-5.</p>
--	---

Table A-8 Evaluation of Structuring the Organisation Practice Area

Appendix B: Phase Tables for Pro-PD Version One

Attribute	Content
Phase Name	Reusability Analysis
Goal	To create a partial product configuration using existing configurations and components in order to minimise the amount of product specific development required.
Input	Product Specific Requirements, Existing Platform Configurations, Platform Components, Platform Architecture
Entry Criteria	The set of Product Specific Requirements have been defined.
Tasks	If possible select closest matching existing configuration or alternatively derive a new configuration. Select components for addition to or replacement of components in the base product configuration. Create a partial product configuration in which the components do not need to be adapted Identify required component development and adaptation
Output	Integrated Product Configuration. List of Required Component Development.
Stakeholders	Product Team
Tools	Tools from Step 1.0 (requirements management and mapping between requirements, features and components). Tools that can automatically generate and manage configurations (which may include an inference engine) Tools to identify where changes are required within software components and the impacts of those changes are also beneficial.
Exit Criteria	Maximum use has been made of the platform assets.

Table B-2 Reusability Analysis Phase

Attribute	Content
Phase Name	Component Development and Adaptation
Goal	To facilitate requirements that could not be satisfied by a configuration of the existing assets through component development or adaptation
Input	Platform Components, Required Component Development
Entry Criteria	The Product Team have identified the set of components to be developed and/or adapted
Tasks	<p>The Change Control Board (CCB) identifies scope of required component development or adaptation.</p> <p>If changes are deemed product specific, the Product Team implements changes and performs unit testing on the developed or adapted components.</p> <p>If the requested changes are deemed platform specific then the Product Team makes a request for required platform changes and repeats the Reusability Analysis step. The requested component development occurs at platform level (Outside scope of Pro-PD) and is performed by the Platform Team.</p>
Output	Platform Change Request. Developed or Adapted Components.
Stakeholders	Product Team. Platform Team.
Tools	<p>Tools to identify where changes are required within software components and the impacts of those changes are of assistance.</p> <p>Tools to support new development and adaptation of existing components and support determination of effort and cost for such tasks are also beneficial.</p> <p>Tools to generate unit test cases and automate the testing process are also of use.</p>
Exit Criteria	Developed or adapted components have been validated through unit

	testing.
--	----------

Table B-3 Component Development and Adaptation Phase

Attribute	Content
Phase Name	Product Integration and Validation
Goal	To build and validate a fully integrated product
Input	Integrated Product Configuration, Developed or Adapted Components, Product Specific Test Cases, Platform Test Assets, Customer Requirements.
Entry Criteria	The Partial Product Configuration and the developed or adapted components are ready for integration.
Tasks	The developed components are integrated into the Partial Product Configuration. The product is validated through integration testing and system testing.
Output	Validated Product.
Stakeholders	Product Team. Customer.
Tools	Tools to build products from a configuration and carry out both integration and system testing of the product are required. Tools for architecture conformance, verification and validation when combining new or adapted components into a configuration would also be useful. Tools to determine the causes of test failure would also support this step.
Exit Criteria	The Product satisfies the Product Specific Requirements and passes the Product Specific Test Cases.

Table B-4 Product Integration and Validation Phase

Appendix C: Pro-PD Version One

We have developed this framework based on the results of an extensive literature review, lengthy discussions with SPL practitioners and researchers, and reviewing industrial product derivation practices. This framework consists of four main phases, which are:

- Impact Analysis;
- Reusability Analysis;
- Component Development and Adaptation;
- Product Integration and Validation.

Figure C-1 provides a diagrammatic representation of Pro-PD. Impact analysis (phase 1) is aimed at gathering product specific requirements based on customer requirements and negotiation with the Platform Team. Reusability Analysis (phase 2) purports to create a partial product configuration based on the product specific requirements and by using the available core assets. During Component Development and Adaptation (phase 3), new components are developed (if required) and existing components are adapted to satisfy requirements which could not be satisfied by configuring existing core assets. Finally, Product Integration and Validation (phase 4) aims to integrate the core asset configuration and newly developed components and to validate the integration by performing appropriate testing procedures.

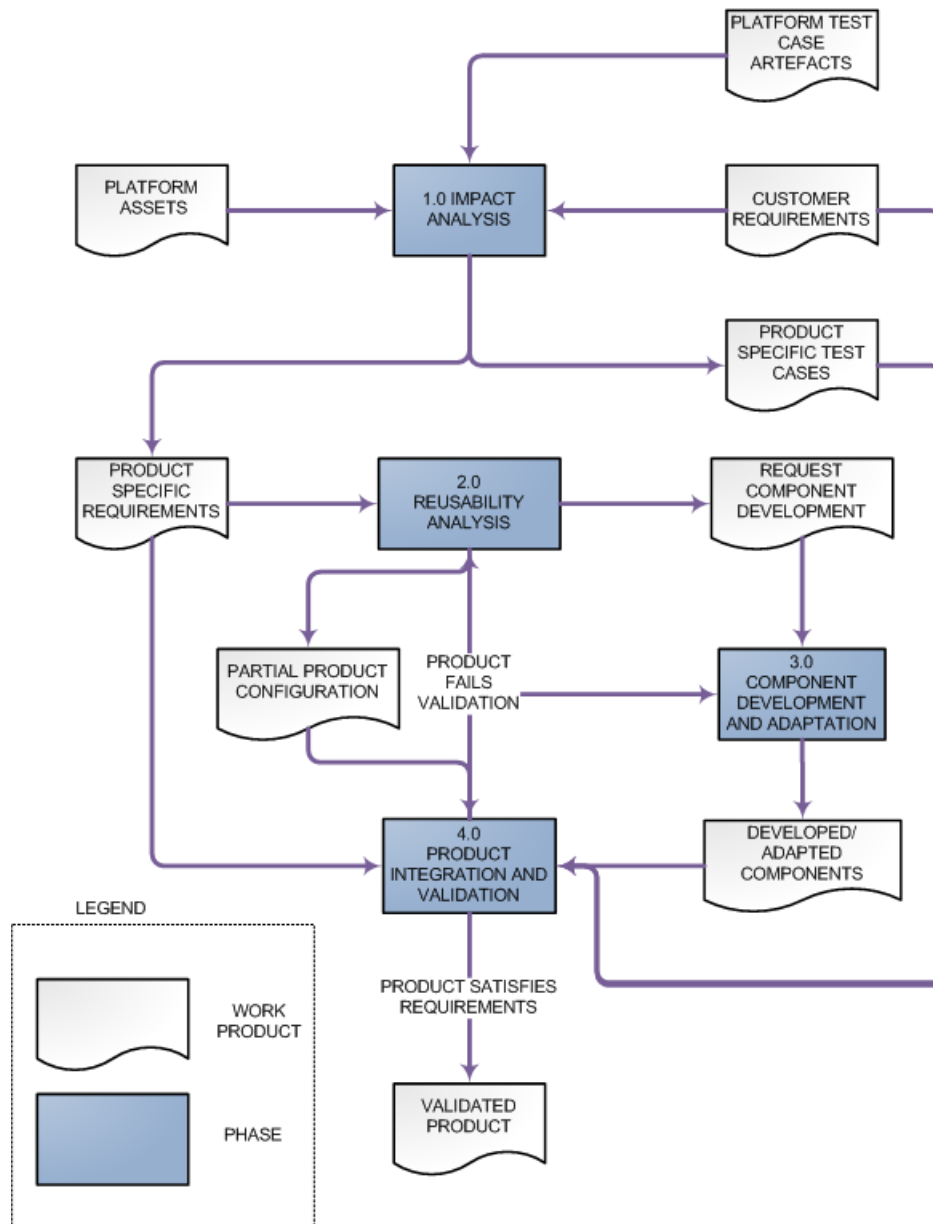


Figure C-1 An Overview of the Pro-PD

Below, we provide a more detailed description of each phase in the PD Framework.

C. 1. Impact Analysis

In this phase the customer requirements are mapped to platform features (see Figure C-2).

In the *Map Customer Requirements to Platform Features* task, the Product Team

determines the list of the customer requirements which can be satisfied by a configuration of the platform assets. Customer requirements which cannot be satisfied by existing assets are negotiated with the customer and platform architect in the *Customer Negotiation* task. Customer negotiation is an important and critical aspect of PD. The trade-off here is to meet (ideally) all of the customer’s needs while retaining the profitability of the platform assets for the whole product line. Time-to-market requirements 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 [1]. Involving the platform architects in customer negotiation can solve many of the problems that arise from these conflicting requirements [2].

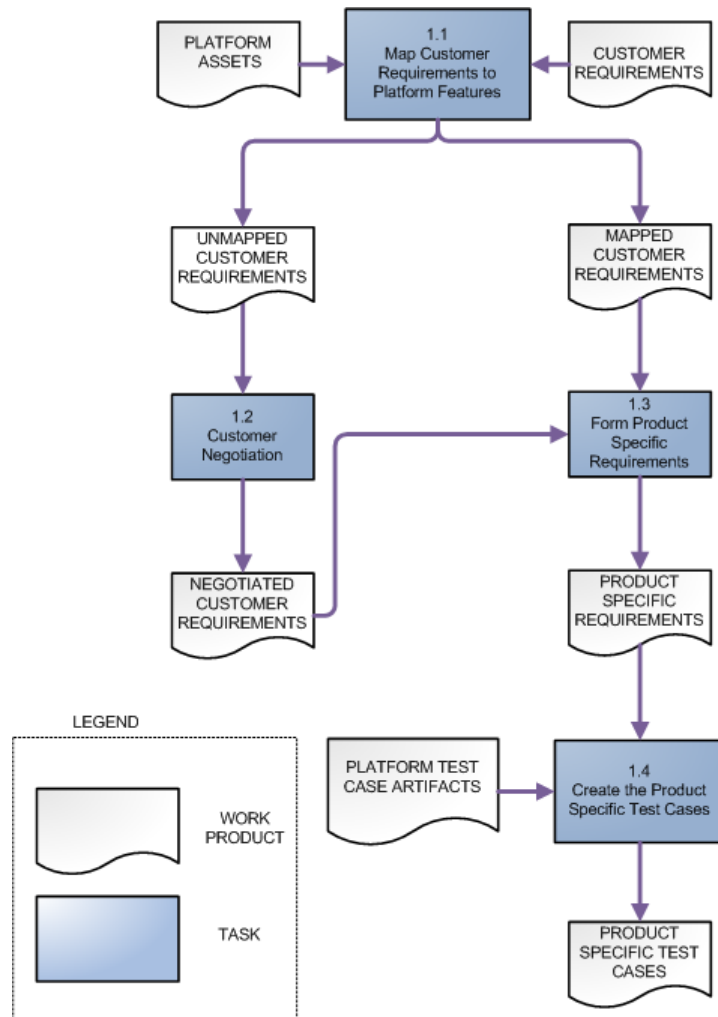


Figure C-2 Impact Analysis Phase

In *Form Product Specific Requirements*, 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 in *Create the Product Specific Test Cases*. The product specific test cases are used during system testing in Product Integration and Validation and assist the Product Team in the verification of requirements [3]. The Product Team uses the platform test cases artefacts as a basis for the creation of the product specific test cases.

C. 2. Reusability Analysis

The main goal of the Reusability Analysis phase (see Figure C-3) is to create a partial product configuration that maximises the benefits of the platform artefacts and minimises the amount of product specific development required. The Product Team selects from both the existing configurations and components within the platform and creates a partial product configuration that satisfies as much as possible the product specific requirements.

In *Select Closest Matching Configuration* task, 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 PD 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 [4]. Configuration selection can also help speed up the development process by choosing a previously tested solution especially in instances when two or more configurations can be used.

In situations where no appropriate existing configuration can be selected, the *Derive New Configuration* task is performed. Here, the Product Team must derive a new configuration from a subset of the overall Platform Architecture.

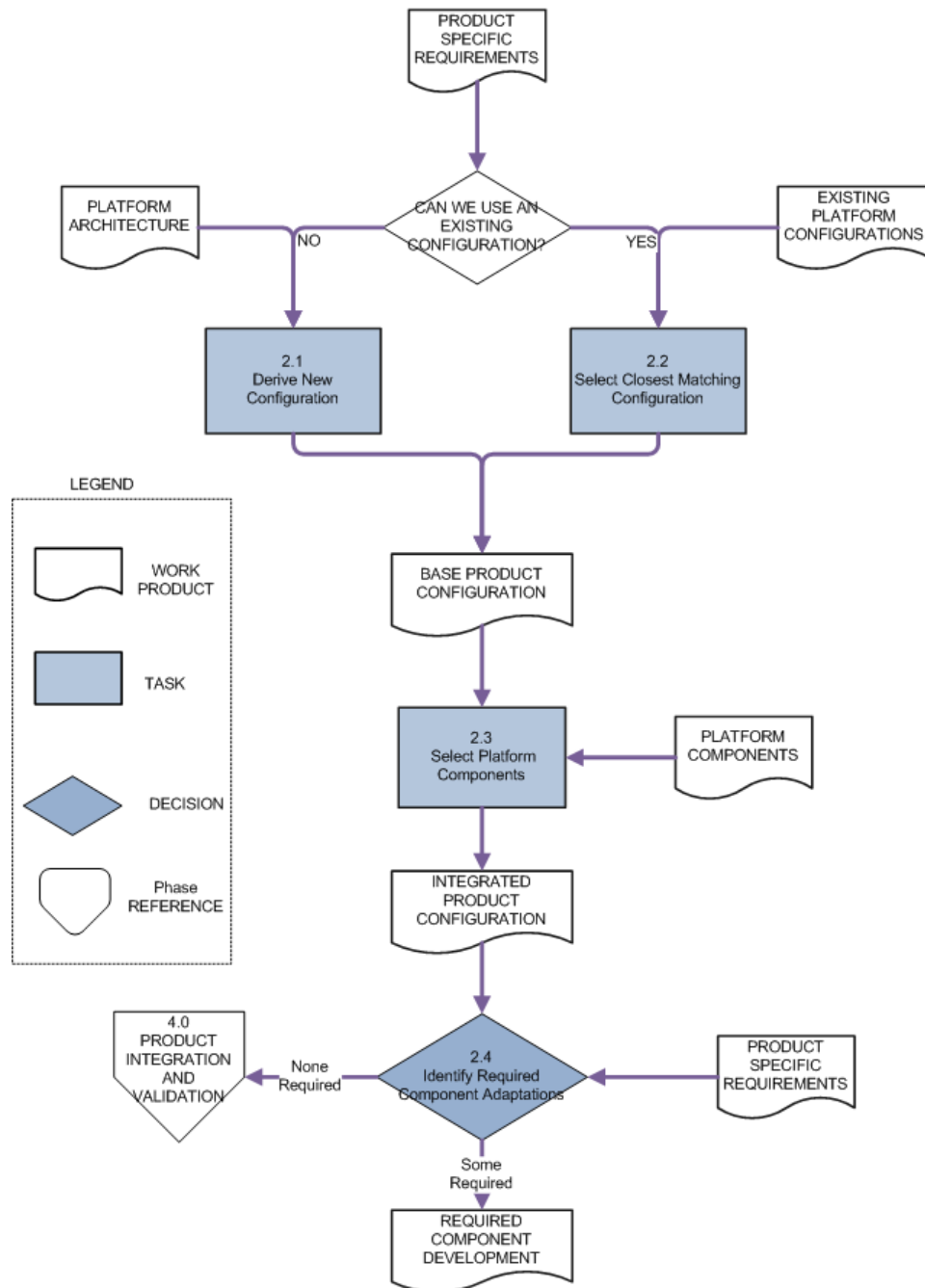


Figure C-3 Reusability Analysis Phase

In *Select Platform Components*, components are selected from the collection of Platform components for, addition to or replacement of, components in the base product configuration. The partial product configuration can be either a selection of existing configurations or a newly derived configuration. The selection of components allows the

Product Team meet 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 product configuration is now created.

At this stage, the Product Team decide if Component Development or Adaptation is required. If no further development is required the product team begin *Product Integration and Validation* phase. However, if further development is required, the requirements which could not be accommodated by the partial product configuration are handled by the Product Team in *Identify Required Component Adaptation*. The Product Team is responsible for the development of new components and adaptation of existing components.

C. 3. Component Development and Adaptation

In the Component Development and Adaptation phase (see Figure C-4) 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) during the *Scoping of Component Changes* task. 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 cost 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 [2].

If the CCB decides that the component development should occur at the platform level in the task *Request Component Development at 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 the task *Component Development at Product Level* occurs. It is the responsibility of the product development team to implement the required component changes at the product level.

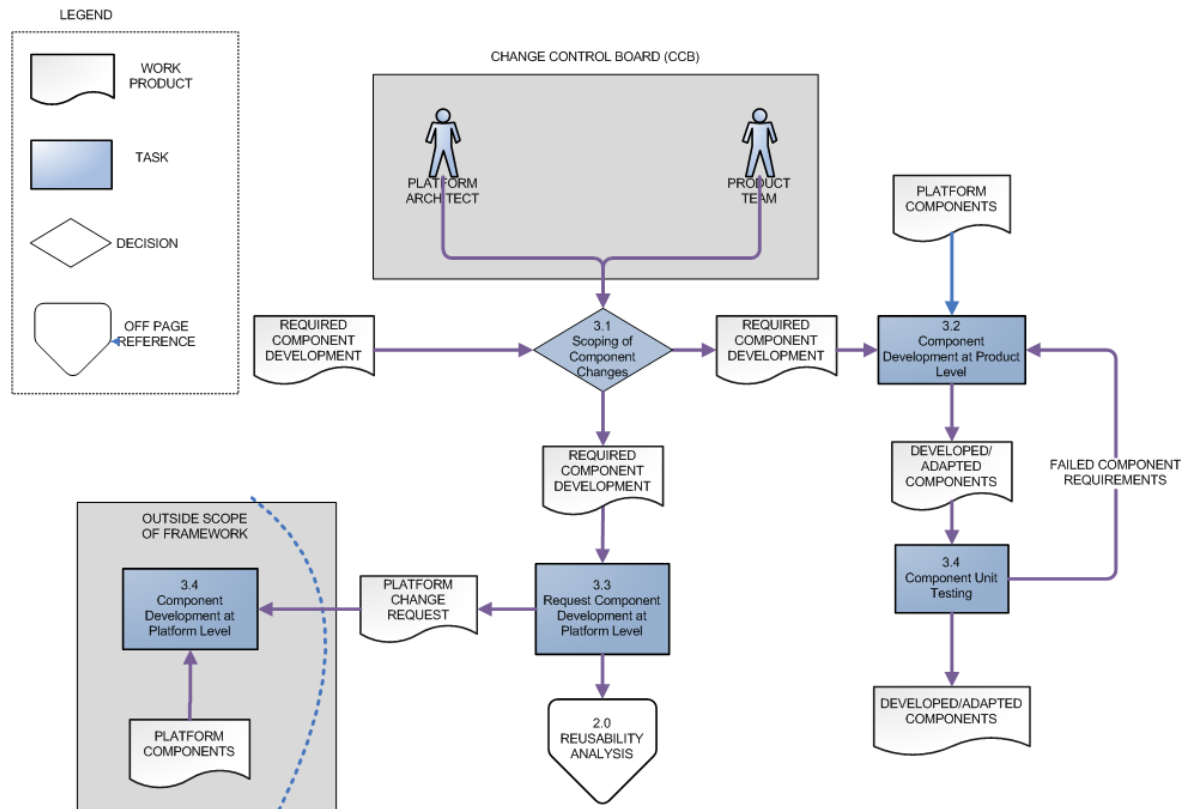


Figure C-4 Component Development and Adaptation Phase

When a component is built or adapted, initial or tailored versions of a component will need to be tested rigorously through unit testing in the task *Component Unit Testing*. According to Kauppinen [5], conventional unit test methods must be utilized as no product line specific methods have been developed so far. The output is validated developed/adapted components.

C. 4. Product Integration and Validation

In Product Integration and Validation phase (see Figure C-5), a final integrated product is created from the partial product configuration and the developed or adapted components. The product is validated through integration and system testing.

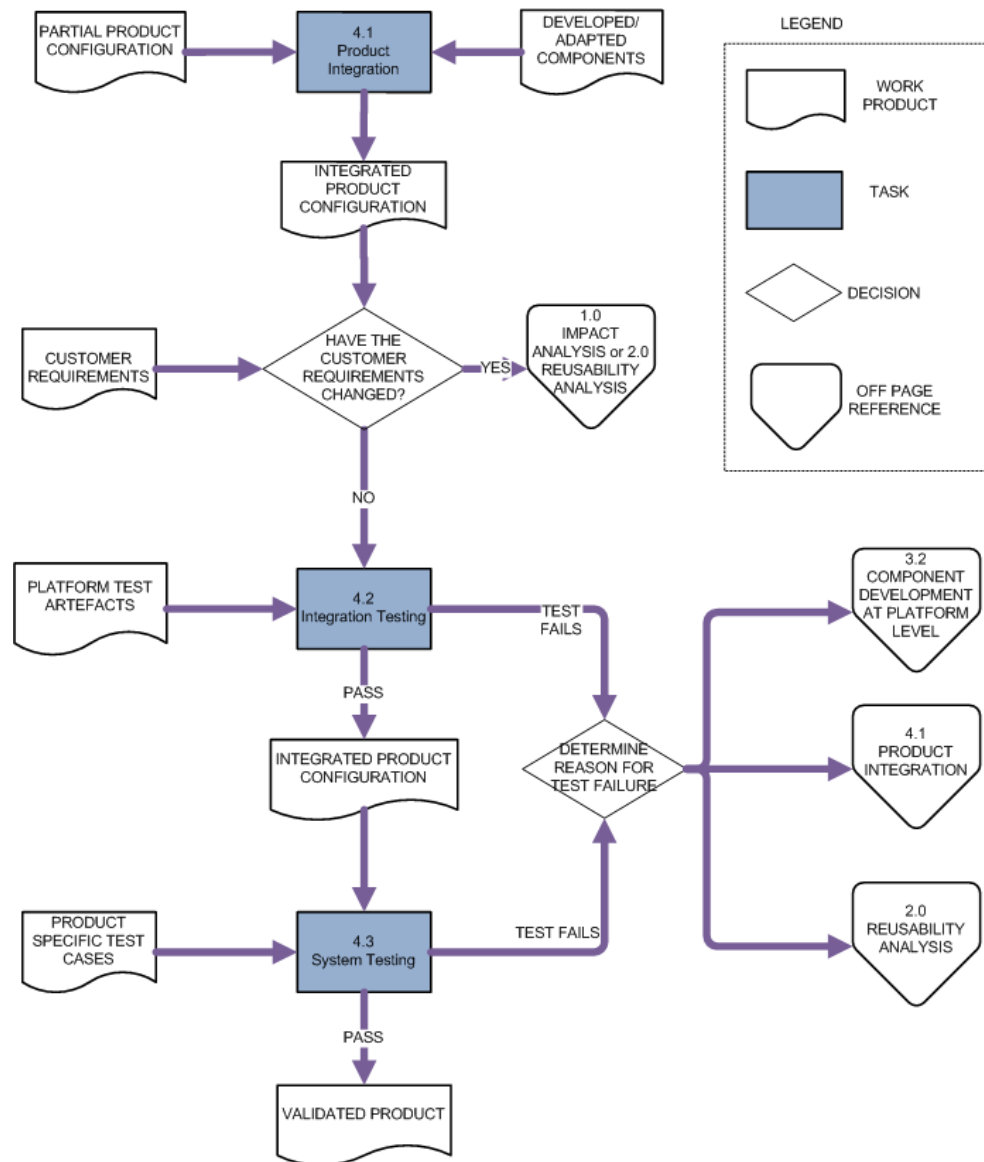


Figure C-5 Product Integration and Validation

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 [1]. 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 (*Have the Customer Requirements Changed?*). If the customer requirements have changed, the product team must return to a previous phase. Major customer requirement changes can be handled by returning to the Impact Analysis phase. Minor changes or changes required under tight time constraints are handled at the product level through returning to *Component Development and Adaptation* phase.

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*.

Due to the variability defined in the platform assets, completely testing the platform assets is impossible except for trivial cases. *Integration Testing* 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 [6].

After *Integration Testing*, *System Testing* is performed. *System Testing* verifies if a product conforms to the product specific requirements. System test artefacts such as the product specific test cases are already derived from the product specific requirements [6] in 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. In this case the Product Team should repeat either the Reusability Analysis phase or the Components Development and Adaptation phase depending on the required changes.

There are two main reasons why a product may fail *Integration Testing* or *System Testing*. Firstly the requirements set may change or expand during PD, 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.

1. Chastek, G., Donohoe, P., and McGregor, J.D. (2002). *Product Line Production Planning for the Home Integration System Example*, in Technical Report CMU/SEI-2002-TN-029. Pittsburgh: Carnegie Mellon Software Engineering Institute.
2. Birk, A., Heller, G., John, I., Maßen, T.v.d., Müller, K., and Schmid, K., Product Line Engineering: The State of the Practice. *IEEE Software*, 2003. **20**(6): p. 52-60.
3. Kuloor, C. and Eberlein, A., *Requirements Engineering for Software Product Lines*, in *Proceedings of the 15th International Conference on Software & Systems Engineering and their Applications (ICSSEA'02)*. 2002: Paris, France.
4. 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.
5. Kauppinen, R., *Testing Framework-Based Software Product Lines*, in *Technical Report C-2003-20*,. 2003, University of Helsinki, Department of Computer Science.
6. Pohl, K. and Metzger, A., *Software Product Line Testing*. 2006, Communications of the ACM. p. 78-81.

Appendix D: Pro-PD Version Two

D.1 Overview of Pro-PD

The model is structured into three main phases. Figure D-1 provides an overview of the model, showing the interactions between the main phases:

1. The goal of **Preparing for Derivation** is to perform the preparatory steps required before actual derivation can begin.
2. The goal of **Product Configuration** is to build the product by reusing as much as possible the platform artefacts and minimizing the amount of product-specific development required.
3. The goal of **Product Development and Testing** is to implement any changes required at the product level, and to test both the changes and the final product.

Each of these product derivation phases will now be discuss in more detail.

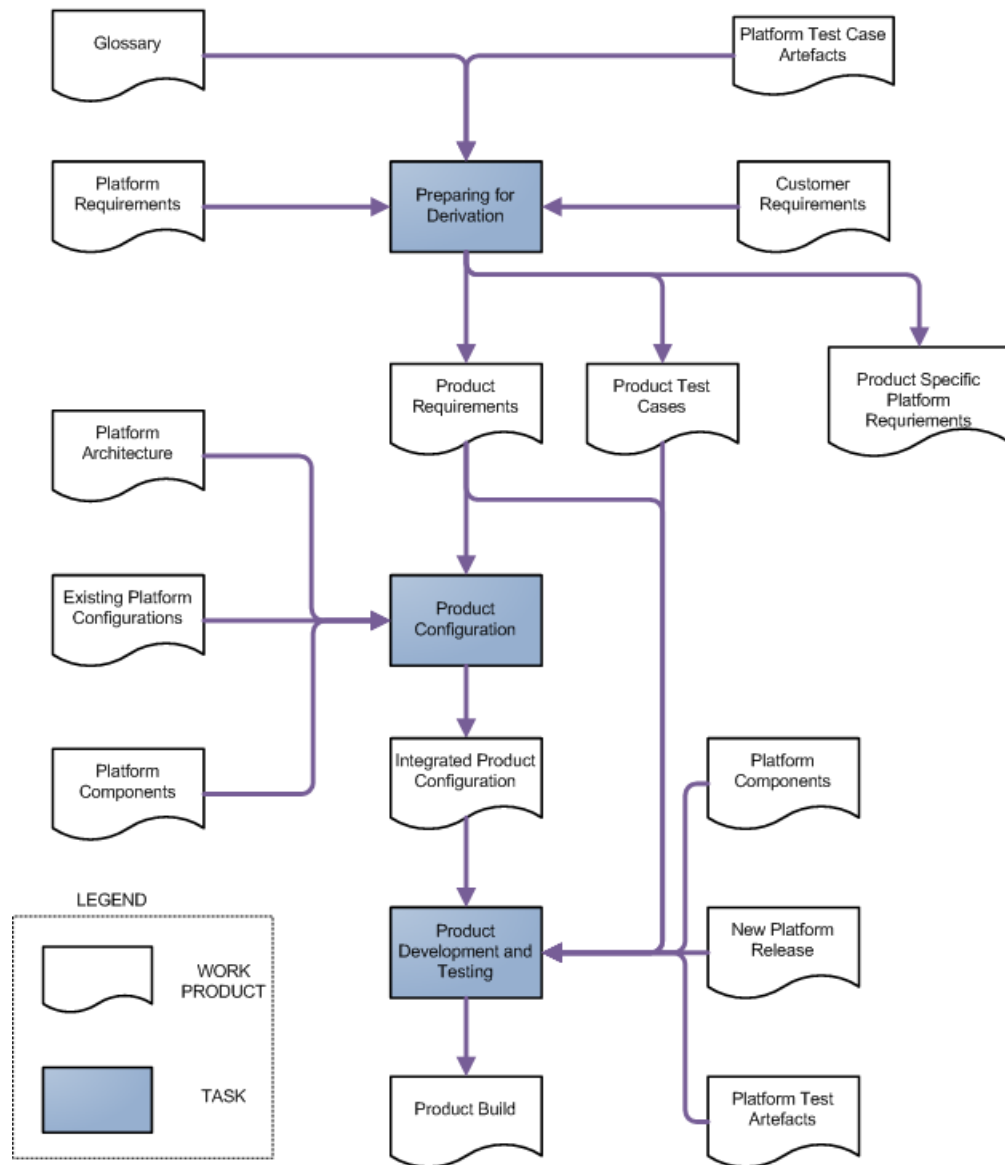


Figure D-1 Overview of Pro-PD Version Two

D.2 Preparing for Derivation

The goal of Preparing for Derivation phase is to create the Product Requirements based on the customer requirements and negotiation with the platform team. In Figure D-2, the Preparing for Derivation phase is shown.

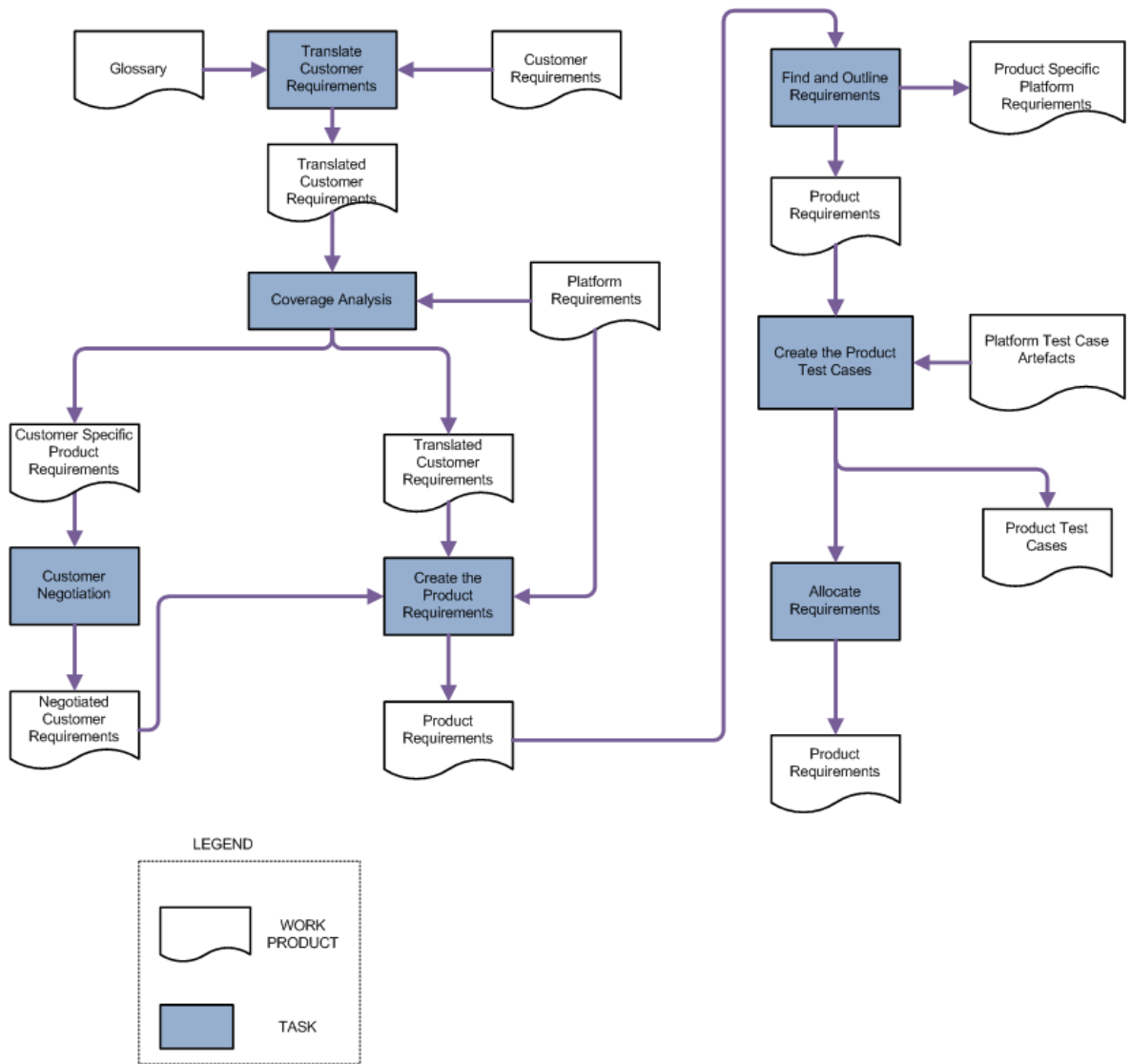


Figure D-2 Preparing for Derivation Phase

The task *Translate Customer Requirements* translates the customer requirements from a customer-specific document into an internal company-specific document. The product team uses the ‘Glossary’ document which contains customer terms and their Product Line equivalent as a guide during this task.

In *Coverage Analysis* the product team determines those customer requirements which can be satisfied through a configuration of the platform assets. The results of this task are used during the ‘Customer Negotiation’ task.

In *Customer Negotiation* the objective is to meet as many of the customer's needs as possible while retaining the profitability of the platform assets for the whole product line. The output of this task is a set of *Negotiated Customer Requirements*.

In *Create the Product Requirements*, using the *Negotiated Customer Requirements* and the *Translated Customer Requirements* which were within the scope of the platform the Product Requirements are formed. To form the *Product Requirements*, the *Platform Requirements* are taken as a baseline. The *Product Requirements* are expressed as a delta or variation of the *Platform Requirements* with some additions for the *Negotiated Customer Requirements*. The role is performed by the Product Architect and the output of this task is the Product Requirements.

In the *Find and Outline Requirements* task, the functional and non-functional requirements for the system are specified. The Product Architect scopes the Product Requirements. It should be noted however that only requirements not covered by the platform are relevant input here. The decision of whether the required component development or adaptation will result in product-specific code or in adaptation of the entire product line (platform) is determined.

The *Product Manager* based on recommendations by CCB separates the requirements for implementation in the *Product Requirements* between the Platform and Product teams. The requirements destined for platform implementation are added to the *Product Specific Platform Requirements*. The implementation of the *Product Specific Platform Requirements* occurs at platform level and is deemed outside the scope of this research. In addition to these platform requests, additional interface requirements must be added to the *Product Requirements* to allow communication between the product and platform components.

In *Create the Product Test Cases* task, the *Product Test Cases* are written for requirements in the *Product Requirements*. The product test cases are used during the *System Testing*. The *Product Tester* uses the *Platform Test Artefacts* as a basis for the creation of the *Product Test Cases*.

In *Allocate Requirements*, the *Product Requirements* are allocated to the relevant organisational disciplines such as hardware, algorithms or software disciplines by the Product Architect.

D.3 Product Configuration

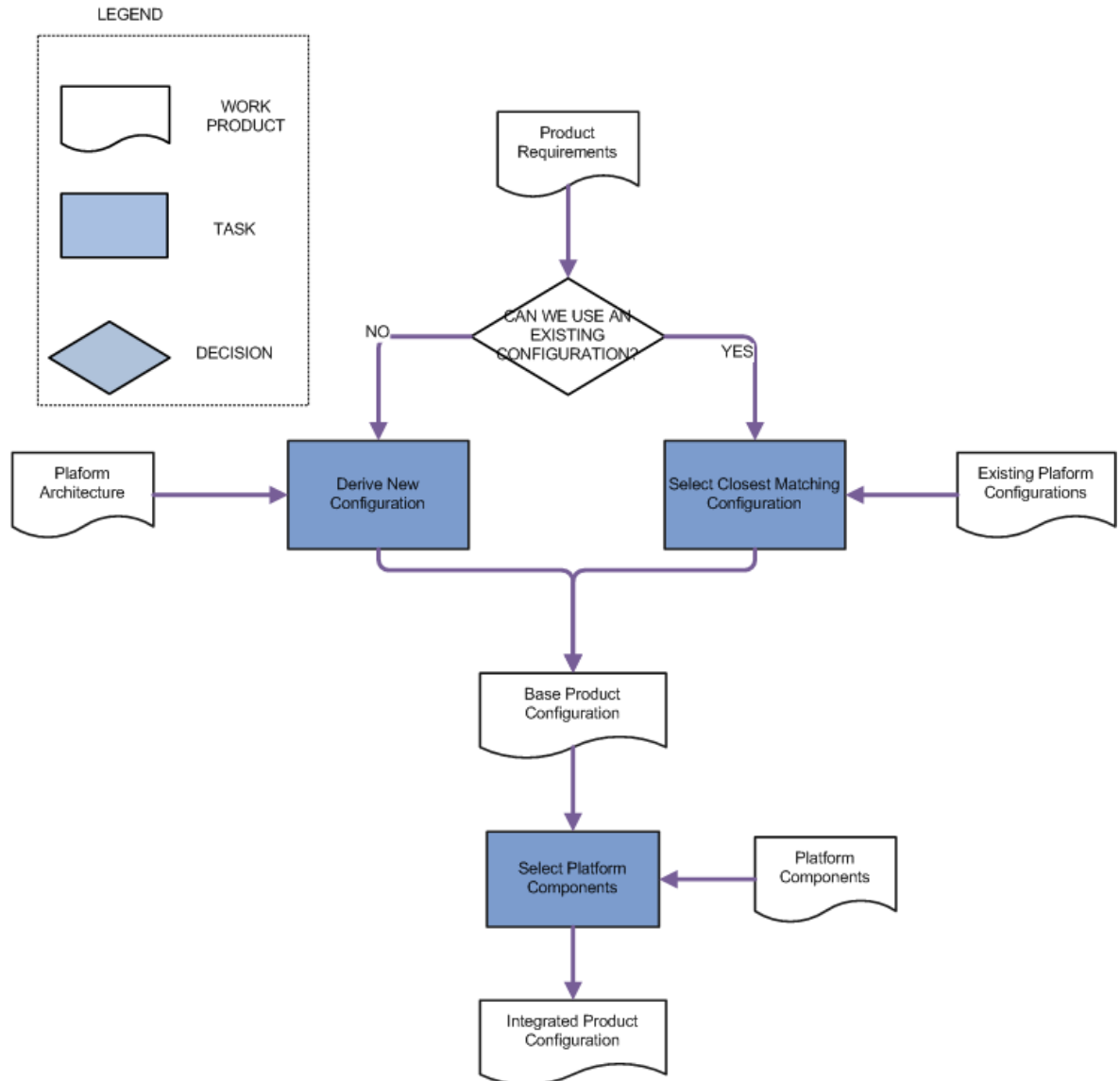


Figure D-3 Product Configuration Phase

The goal of **Product Configuration** (see Figure D-3) is to build the product by reusing as much as possible the platform artefacts and minimizing the amount of product-specific development required.

The *Base Product Configuration* is constructed by the *Product Architect* either by choosing from the *Existing Platform Configurations* in the task *Select Closest Matching Configuration* or derived from the *Platform Architecture* in the *Derive New Configuration* task.

In the first case a previous product configuration is found in the platform that respects the most of the *Product Requirements*. 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 speed up the development process by choosing a previously tested solution especially in instances when two or more configurations can be used.

In the second case, if no *Existing Platform Configuration* can be found, a new one is derived from the *Platform Architecture*. Here the *Product Architect* makes a copy of the latest *Platform Architecture*. This new version contains the *Product Specific Platform Requirements* requested in *Find and Outline Requirements* task. The *Product Architect* tailors the *Platform Architecture* to form the product architecture which is contained in the *Base Product Configuration* and resolves variation points according to the *Product Requirements*.

In *Select Platform Components* task, components are selected from the collection of *Platform Components* for, addition to or replacement of, components in the *Base Product Configuration*. The selection of *Platform Components* involves binding any built-in variations (excluding runtime/dynamic variability which is bound onsite). The output is the *Integrated Product Configuration*.

Theoretically at this stage the *Integrated Product Configuration* could satisfy customer requirements and testing should begin. However, this is the ideal case and

assumes all the *Product Requirements* are covered by the platform. In most cases some additional development will be required.

D.4 Product Development and Testing

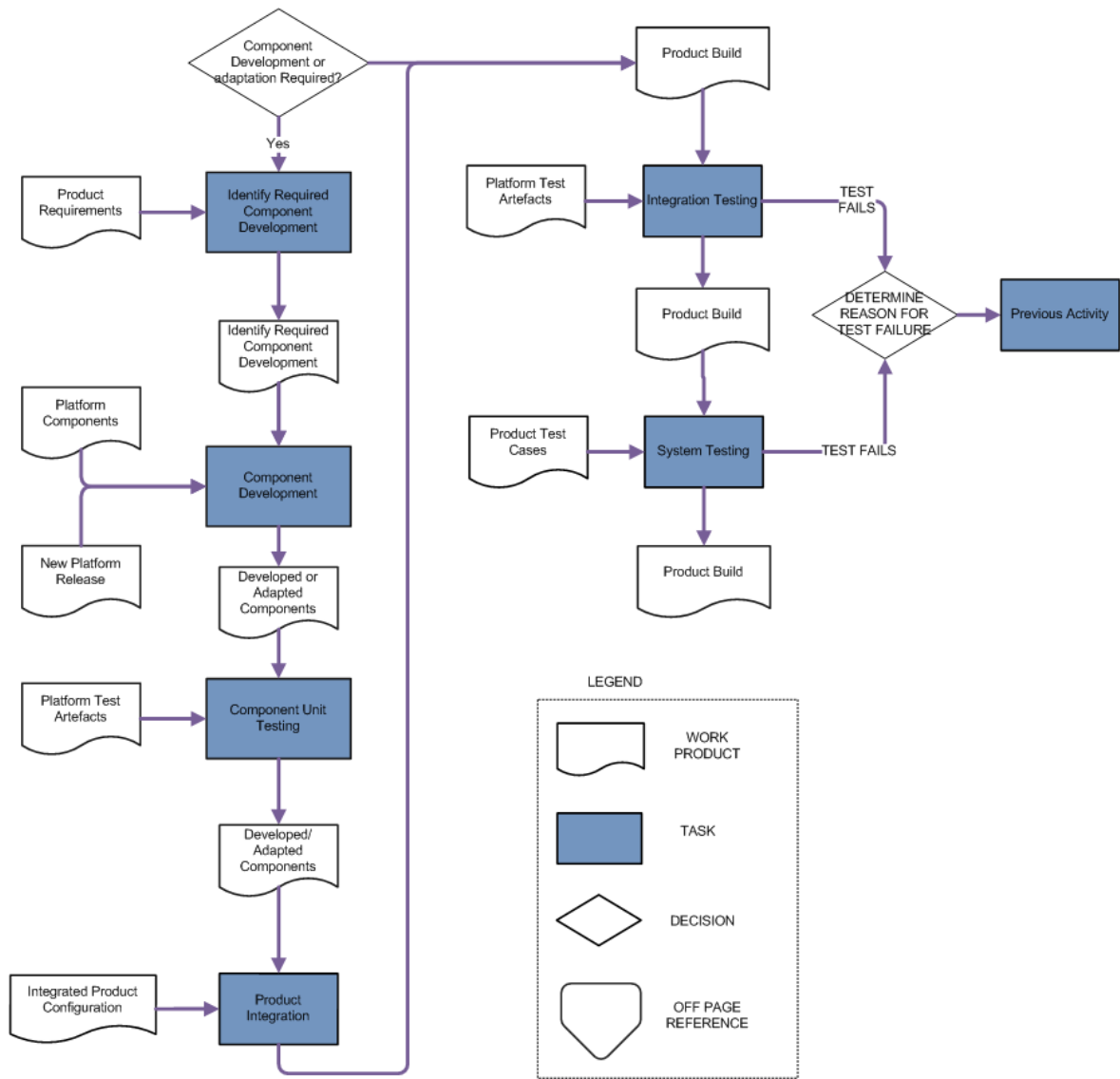


Figure D-4 Product Development and Testing Phase

The goal of Product Development and Testing (see Figure D-4) is to implement any changes required at the product level, and to test both the changes and the final product.

If a specific requirement in the *Product Requirements* is designated to be product-specific then it is the responsibility of the *Product Developer* to implement the required component changes at the product level.

In the *Identify Required Component Development* task, the *Product Developer* outlines what product specific development needs to occur. This is contained in the *Identify Required Component Development* artefact.

In the *Component Development* task the source code to implement new functionality or to adapt an existing platform component at product level is performed. New components should be developed with the intention of being promoted to a platform asset. If a platform component is considerably adapted and consider 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.

In the case of required platform extensions which were identified in the *Find and Outline Requirements* task, the platform team receives the *Product Specific Platform Requirements* containing the required extensions to the existing platform in order to facilitate the new *Product Requirements*. Both the customer-specific and platform development is occurring in parallel.

Previously the synchronisation process pattern used by the product team to handle development dependencies is described. The product team can decide on an implementation strategy based on their development needs.

Option one. The product team waits for the new platform release and then proceeds to design, implement and test customer specific components.

Option two. The product team bases new component development on the existing *Platform Architecture*. The product team first negotiates a platform interface with the platform team before proceeding to develop in parallel. Alternatively, the product team will make assumptions on platform interface changes. If conflicts are detected when the *New Platform Release* is released, these are identified during Integration Testing.

Finally, when a component is built or adapted, initial or tailored versions of a component will need to be tested rigorously through *Component Unit Testing*. For adapted

platform components *Platform Test Artefacts* can be used as a basis for the component unit tests.

In *Product Integration*, the *Developed or Adapted Components* are integrated into the *Integrated Product Configuration*. The goal is to identify integration issues as soon as possible. Newly developed or adapted assets need to be integrated with the *Integrated Product Configuration*.

Integration Testing then validates the *Integrated Product Configuration*. The integration tests should reuse *Platform Test Artefacts*. This also ensures that no new errors appear due to the integration of *Developed or Adapted Components* into the *Integrated Product Configuration*. The output is the *Product Build*.

In *System Testing* the *Product Tester* checks the *Product Build* for compliance with the *Product Requirements*. There are two main reasons why a *Product Build* may fail acceptance 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 *Integrated Product Configuration*, or if the customer has new wishes for the product. Secondly, if the *Product Build* does not completely provide the required functionality or some of the selected components simply do not work together at all. If the product is validated through acceptance testing the process is complete and the customer product has been derived. If not the product team will need to return to a previous stage of the process depending on the reason for failure.

Appendix E: Pro-PD Version Three

E.1 Overview of the PDPF

The model is structured into three main phases. Figure E-1 provides an overview of the model, showing the interactions between the main phases:

1. The goal of **Preparing for Derivation** is to perform the preparatory steps required before actual derivation can begin.
2. The goal of **Product Configuration** is to build the product by reusing as much as possible the platform artefacts and minimizing the amount of product-specific development required.
3. The goal of **Product Development and Testing** is to implement any changes required at the product level, and to test both the changes and the final product.

Each of these product derivation phases will now be discuss in more detail.

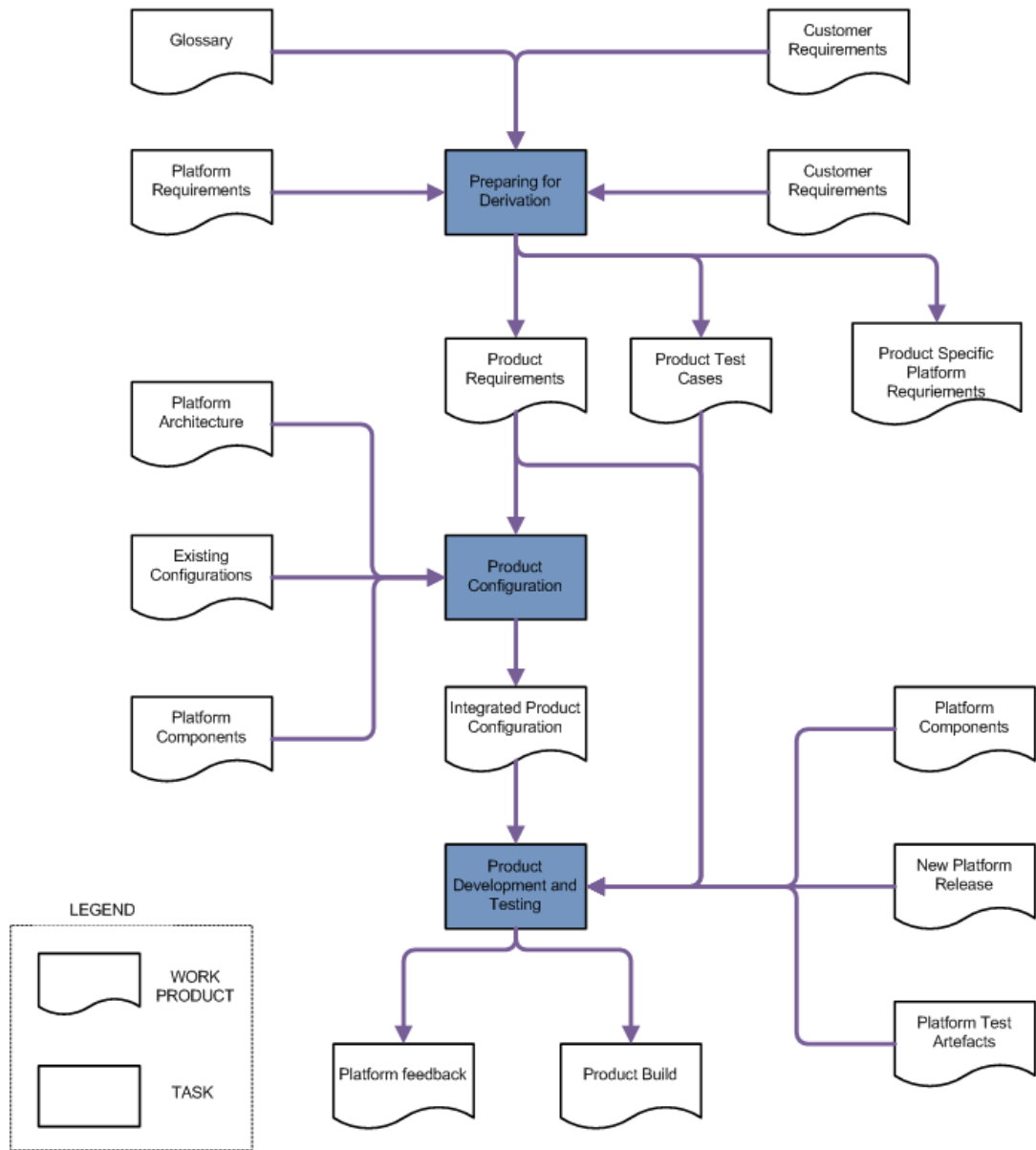


Figure E-1 Overview of Pro-PD Version Three

E.2 Preparing for Derivation

The goal of **Preparing for Derivation** (see Figure E-2) is to create the product-specific requirements based on customer requirements and negotiation with the platform team.

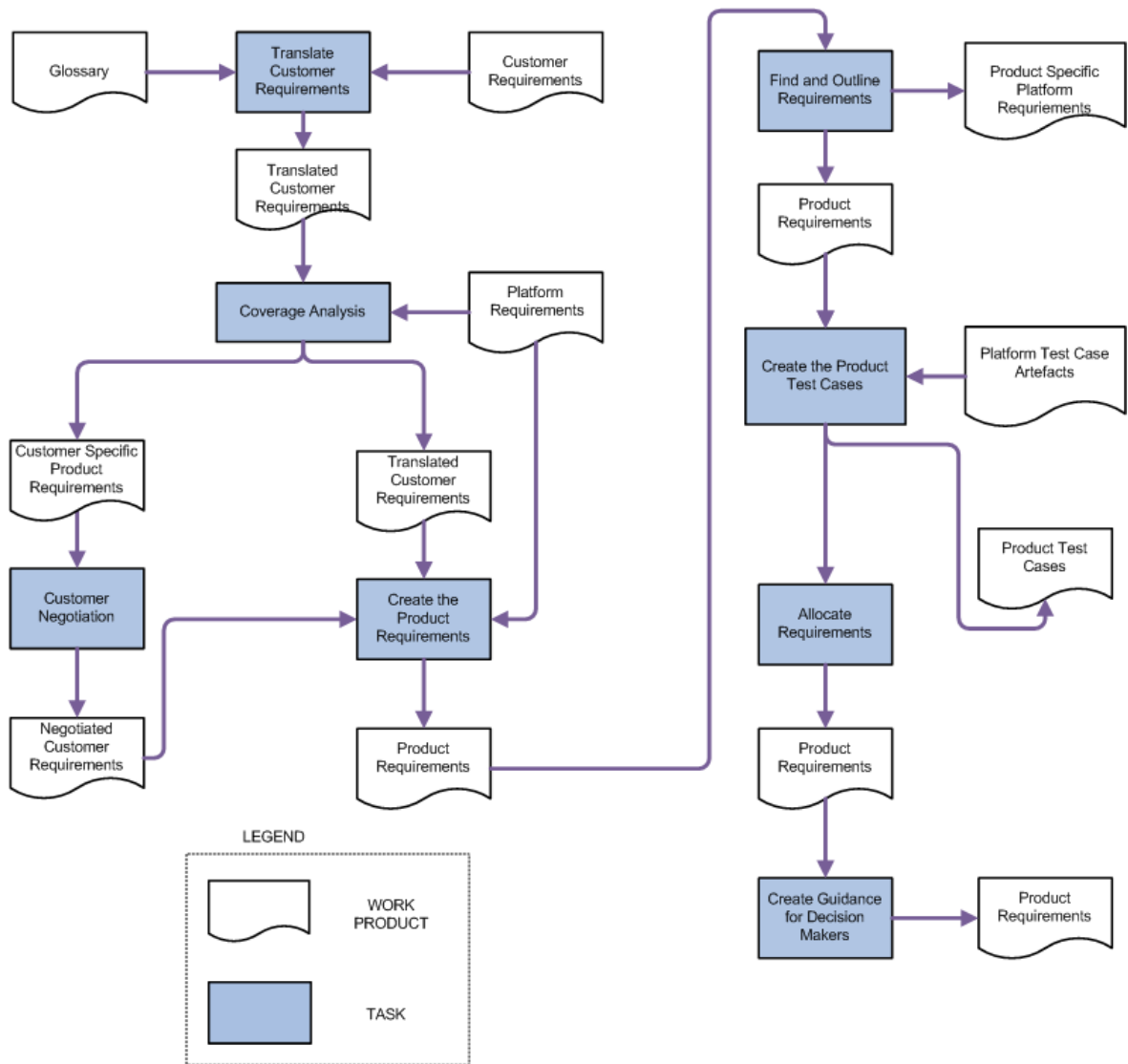


Figure E-2 Preparing for Derivation Phase

The task *Translate Customer Requirements* translates the customer requirements from a customer-specific document into an internal company-specific document. The product team uses the ‘Glossary’ document which contains customer terms and their Product Line equivalent as a guide during this task.

In *Coverage Analysis* the product team determines those customer requirements which can be satisfied through a configuration of the platform assets. The results of this task are used during the ‘Customer Negotiation’ task.

In *Customer Negotiation* the objective is to meet as many of the customer's needs as possible while retaining the profitability of the platform assets for the whole product line. The output of this task is a set of *Negotiated Customer Requirements*.

In *Create the Product Requirements*, using the *Negotiated Customer Requirements* and the *Translated Customer Requirements* which were within the scope of the platform the Product Requirements are formed. To form the *Product Requirements*, the *Platform Requirements* are taken as a baseline. The *Product Requirements* are expressed as a delta or variation of the *Platform Requirements* with some additions for the *Negotiated Customer Requirements*. The role is performed by the Product Architect and the output of this task is the Product Requirements.

In the *Find and Outline Requirements* task, the functional and non-functional requirements for the system are specified. The Product Architect scopes the Product Requirements. It should be noted however that only requirements not covered by the platform are relevant input here. The decision of whether the required component development or adaptation will result in product-specific code or in adaptation of the entire product line (platform) is determined.

The *Product Manager* based on recommendations by CCB separates the requirements for implementation in the *Product Requirements* between the Platform and Product teams. The requirements destined for platform implementation are added to the *Product Specific Platform Requirements*. The implementation of the *Product Specific Platform Requirements* occurs at platform level and is deemed outside the scope of this research. In addition to these platform requests, additional interface requirements must be added to the *Product Requirements* to allow communication between the product and platform components.

In *Create the Product Test Cases* task, the *Product Test Cases* are written for requirements in the *Product Requirements*. The product test cases are used during the *System Testing*. The *Product Tester* uses the *Platform Test Artefacts* as a basis for the creation of the *Product Test Cases*.

In *Allocate Requirements*, the *Product Requirements* are allocated to the relevant organisational disciplines such as hardware, algorithms or software disciplines by the

Product Architect. The role and task structures for the product derivation project are defined. For example, discipline mapping is performed where product requirements are allocated to relevant disciplines. The goal is to define who is responsible for resolving which parts of the remaining variability 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.

Preparing for derivation also involves *Creating Guidance for Decision-Makers*; remaining variability must be explained to help people resolving it in derivation.

E.3 Product Configuration

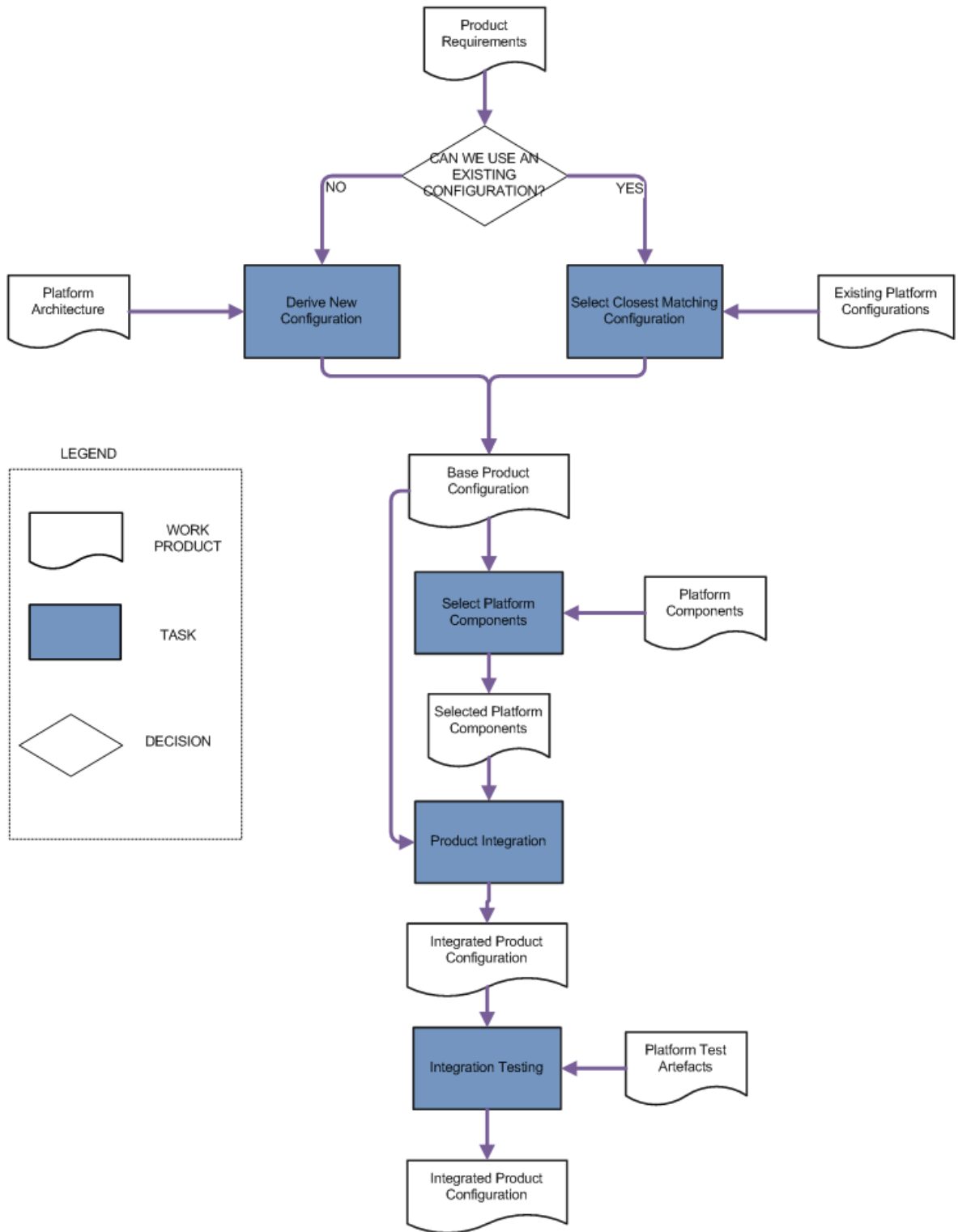


Figure E-3 Product Configuration Phase

The goal of **Product Configuration** (see Figure E-3) is to build the product by reusing as much as possible the platform artefacts and minimizing the amount of product-specific development required.

The *Base Product Configuration* is constructed by the *Product Architect* either by choosing from the *Existing Platform Configurations* in the task *Select Closest Matching Configuration* or derived from the *Platform Architecture* in the *Derive New Configuration* task.

In the first case a previous product configuration is found in the platform that respects the most of the *Product Requirements*. 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 speed up the development process by choosing a previously tested solution especially in instances when two or more configurations can be used.

In the second case, if no *Existing Platform Configuration* can be found, a new one is derived from the *Platform Architecture*. Here the *Product Architect* makes a copy of the latest *Platform Architecture*. This new version contains the *Product Specific Platform Requirements* requested in *Find and Outline Requirements* task. The *Product Architect* tailors the *Platform Architecture* to form the product architecture which is contained in the *Base Product Configuration* and resolves variation points according to the *Product Requirements*.

In *Select Platform Components* task, components are selected from the collection of *Platform Components* for, addition to or replacement of, components in the *Base Product Configuration*. The selection of *Platform Components* involves binding any built-in variations (excluding runtime/dynamic variability which is bound onsite). The responsibility for resolving particularly variability has been defined according to the *Allocate Requirements* task earlier. The output is the *Selected Platform Components*.

In *Product Integration* the *Selected Platform Components* and the *Base Product Configuration* are integrated.

Integration Testing 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 platform assets with product specific assets

Theoretically at this stage the *Integrated Product Configuration* could satisfy customer requirements and testing should begin. However, this is the ideal case and assumes all the *Product Requirements* are covered by the platform. In most cases some additional development will be required.

E.4 Product Development and Testing

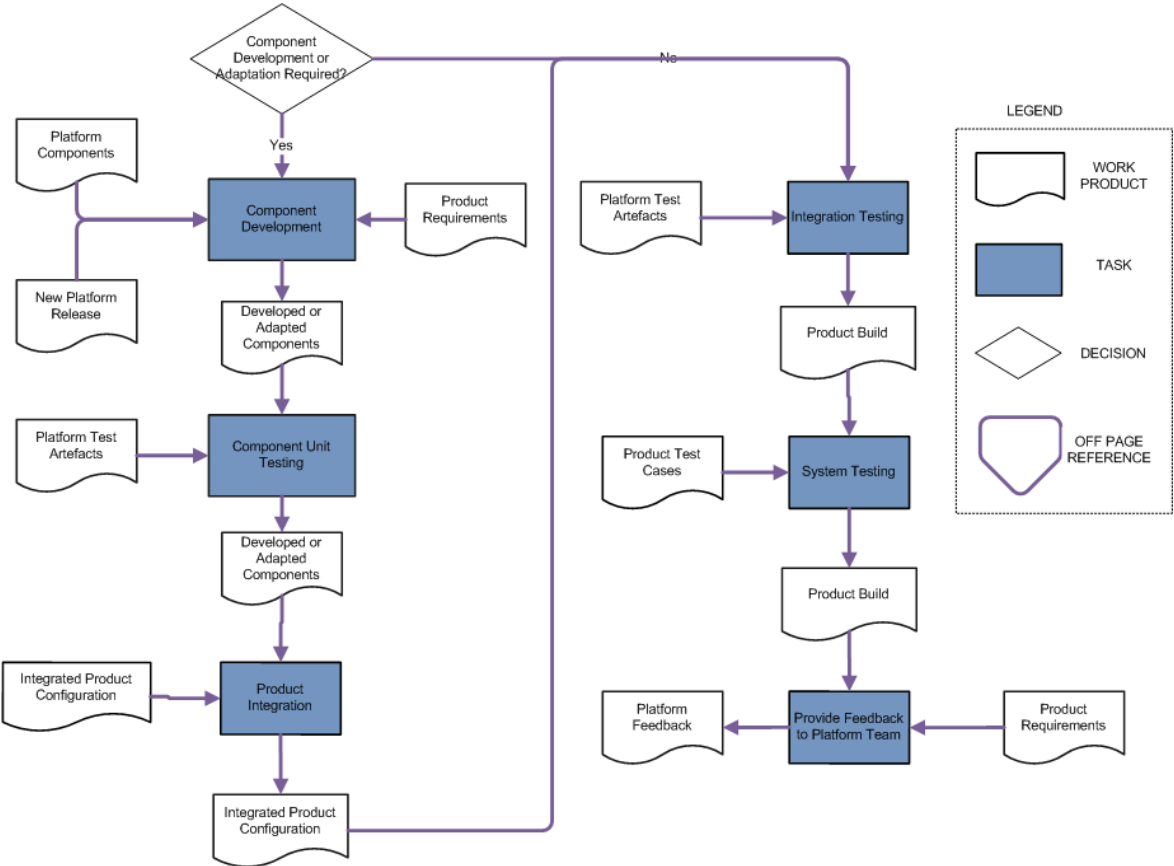


Figure E-4 Product Development and Testing Phase

The goal of *Product Development and Testing* (see Figure E-4) is to implement any changes required at the product level, and to test both the changes and the final product.

If a specific requirement in the *Product Requirements* is designated to be product-specific then it is the responsibility of the *Product Developer* to implement the required component changes at the product level. In the *Component Development* task the source code to implement new functionality or to adapt an existing platform component at product level is performed. New components should be developed with the intention of being 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.

In the case of required platform extensions which were identified in the *Find and Outline Requirements* task, the platform team receives the *Product Specific Platform Requirements* containing the required extensions to the existing platform in order to facilitate the new *Product Requirements*. Both the customer-specific and platform development is occurring in parallel.

Previously the synchronisation process pattern used by the product team to handle development dependencies is described. The product team can decide on an implementation strategy based on their development needs.

Option one. The product team waits for the new platform release and then proceeds to design, implement and test customer specific components.

Option two. The product team bases new component development on the existing *Platform Architecture*. The product team first negotiates a platform interface with the platform team before proceeding to develop in parallel. Alternatively, the product team will make assumptions on platform interface changes. If conflicts are detected when the *New Platform Release* is released, these are identified during Integration Testing.

Finally, when a component is built or adapted, initial or tailored versions of a component will need to be tested rigorously through *Component Unit Testing*. For adapted platform components *Platform Test Artefacts* can be used as a basis for the component unit tests.

In *Product Integration*, the *Developed or Adapted Components* are integrated into the *Integrated Product Configuration*. The goal is to identify integration issues as soon as possible.

In *Integration Testing*, the *Developed or Adapted Components* are integrated into the *Integrated Product Configuration*. The goal is to identify integration issues as soon as possible. Newly developed or adapted assets need to be integrated with the *Integrated Product Configuration*. *Integration Testing* then validates the *Integrated Product Configuration*. The integration tests should reuse *Platform Test Artefacts*. This also ensures that no new errors appear due to the integration of *Developed or Adapted Components* into the *Integrated Product Configuration*. The output is the *Product Build*.

In *System Testing*, the Product Tester checks the Product Build for compliance with the Product Requirements through running the *Product Test Cases* defined in *Preparing for Derivation* phase.

In the *Provide Feedback to Platform Team* task, feedback is provided to the *Platform Manager* on core asset usage during the project, how user friendly the platform assets were and areas for improvement within the platform specifically product requirements which should be adopted by the platform. In addition, the product team identify product specific components that the platform could potentially benefit from through adoption. The *Provide Feedback to Platform Team* task is essentially the input for product line evolution.

Appendix F: Systematic Literature Review Protocol

1 Executive Summary

This document details the protocol for conducting a Systematic Literature Review on product derivation approaches in software product lines. A systematic literature review (often referred as a systematic review) is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, topic area or phenomenon of interest. The protocol defined below specifies how the systematic literature review will be conducted to answer certain questions about the product derivation approaches in software product lines. The particular research questions of interest are specified in the following section.

2 Background

Software Product Line Engineering is an approach to developing software applications (software-intensive systems and software products) using platforms and mass customisation. This is achieved through the identification and management of commonalities and variations in a set of systems' artefacts such as requirements, architectures, components, and test cases.

A software product line is a set of software-intensive systems that share a common and managed set of features that satisfy the specific needs of a particular market segment and that are developed from a common set of *core assets* in a prescribed way. That is, systems engineering is focused on the development of multiple products at the same time which are based on a common core rather than on the development of multiple individual products.

The development of a product line comprises two main activities: the development of the reusable assets (core asset development) and the use of those assets for the realization of the products (product derivation). A single product is *derived* through (re)use of the core assets and exploitation of various realization techniques which thus allow feature variation to be manipulated in order to produce the required end result.

An effective product derivation process within an organisation should 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. In fact, 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" might not hold if inefficient derivation practices diminishes the expected gains. Hence, there is a strong need for the identification of a structured approach to product derivation in order to achieve maximum return on investment within SPL development.

The objective of this research is to outline a product derivation process framework which could lead to a 'best practice' approach to product derivation. The framework would assist organisations in developing a structured approach for product derivation activities and thus for achieving maximum return on investment (ROI) for using an SPL development approach.

This protocol describes the details about the execution of a systematic literature review to investigate the existing approaches to product derivation, the techniques applied and the practices in use for deriving products from a product line.

The protocol for conducting systematic literature review of variability management techniques have been developed according to Kitchenham's *Guidelines for Performing Systematic Literature Reviews in Software Engineering (2007)*.

3 Research Questions

Below is the research questions that this systematic literature review will attempt to answer.

1. What approaches are currently used to derive products from a Software Product Line?
 - What tools are used in the derivation process?
 - What artefacts, roles, tasks can be identified in product derivation?

4 Search Strategy

The following search strategy has been developed to retrieve all relevant literature that would be subjected to a selection criteria outlined in the next section.

4.1 Resources & Search Terms

The following resources and corresponding search terms will be used to execute the literature search. The search terms were developed based on the research questions above bearing in mind Kitchenham's guidelines (2007). Each term was considered for alternative spellings and synonyms and all relevant words or phrases were included.

IEEEExplore: (relevant PD results saved)

```
(software <AND>
(
(product line<in>ab) <OR>
(product line<in>ti) <OR>
(product family<in>ab)<OR>
(product family<in>ti)
) <AND>
(
(derivation<in>ab)<OR>
(derivation<in>ti)<OR>
(production<in>ab)<OR>
(production<in>ti)
))
```

Found 43

ACM Digital library: (relevant PD results saved)

abstract:"software product line" abstract:"software product family"

Found 200

Citeseer library: (didn't search)

site:citeseer.ist.psu.edu software ("product line" OR "product lines" OR "product family" OR "product families") (Derivation OR derive OR production)

Found 160 from google

ScienceDirect: (0 relevant PD results found)

TITLE-ABSTR-KEY(software AND ("product line*" OR "product famil*")) and
TITLE-ABSTR-KEY("derive*" OR production)

[9 Articles Found](#)

EI Compendex / Inspec:

((software AND ("product line*" OR "product famil*")) WN KY) AND ("derive*" OR production) WN KY))

[122 records in Compendex, Inspec & Referex for 1969-2008](#)

SpringerLink: (0 relevant PD results found)

su:(("product line" OR "product family"))

[5 Results](#)

Web of Science:

topic=((software AND ("product line*" OR "product famil*")) AND (deriv* OR production))

[100 results found](#)

4.2 Search Results

The results of the search will be downloaded and imported into a final Endnote library where all duplicates will have been removed. Each result will have the resource searched, publication title, abstract, author(s), source and date stored.

5 Publication Selection

5.1 Selection Criteria

The selection criteria to be applied to each paper determines whether the paper is to be included or excluded from the systematic literature review.

The paper will be **included** if:

- It introduces an approach to the area of product derivation in software product lines.
- It describes an existing derivation approach or validation of an approach.

The paper will be **excluded** if:

- It does not explicitly focus on product derivation in software product lines.

The following specific subjects are intended when the protocol addresses “product derivation” in general: customer negotiation, feature mapping, requirements translation, test case creation, reusable asset identification, product configuration, feature scoping, component development, product integration, integration testing and system testing.

If there is any doubt as to whether the paper should be included or excluded, it will be flagged for review by a second researcher. If subsequently, there is still doubt, it will be included in the review.

5.2 Selection Process

A single researcher will read the title, abstract and conclusion of each paper to ascertain whether it can be removed from the review process due to irrelevance or should be included as the selection criteria have been met. It may be evident that the paper is irrelevant following title and abstract examination at which time the paper may be removed from the review without examination of the conclusion. A second researcher will perform a consistency check by repeating the process on a random sample of the search results. Following this, the decisions of selection from both researchers will be compared and any disagreements will be resolved through a joint investigation.

6 Publication Data Extraction

6.1 Data to be Extracted

Along with the questions defined above for quality assessment purposes, the following data will be extracted from each paper included in the systematic literature review.

Title	Text
Authors	Text
Source	Text
Date of Publication	Text
Found by search	Text
Date of Extraction	Text
Name of approach (if any)	Text
Addresses problem space?	Yes/No

Addresses solution space?	Yes/No
Addresses customer negotiation?	Yes/No
Addresses feature mapping techniques/approaches?	Yes/No
Addresses requirements translation?	Yes/No
Addresses test case creation?	Yes/No
Addresses reusable asset identification?	Yes/No
Addresses product configuration?	Yes/No
Addresses feature scoping?	Yes/No
Addresses component development?	Yes/No
Addresses product integration?	Yes/No
Addresses product testing?	Yes/No
What method of validation was used?	Case Study / Field Study / Action Research / Experiment / Argument
Comments on validation method	Text
Is there tool support for the approach?	Yes/No
What are the limitations of the approach?	Text
Do the authors purport to address the limitations in future research?	Yes/No
Contextual comment wrt the approach	Text

Appendix G: Bosch Technical Report

Bosch Technical Report

(Due to confidentially reasons this report cannot be published in full)

Abstract

This technical report describes the Bosch derivation process from the perspective of a user i.e. the engineering that makes use of the product line in order to create a customer product.

The technical report describes the process of deriving an individual customer product from the product line platform.

- What is contained in the platform and
 - What has to be done (steps, activities) to make a product out of the platform
 - What tools are involved in the derivation process
 - Organisational structure and roles in a derivation project
-

Appendix H: List of Publications

O’Leary, P., Richardson, I., McCaffery, F., and Thiel, S. Preparing for Product Derivation: Activities and Issues. in 4th International Conference on Software and Data Technologies - ICSOFT 2009. 2009. Sofia, Bulgaria: INSTICC - Institute for Systems and Technologies of Information, Control and Communication.

O’Leary, P., Rabiser, R., Richardson, I., and Thiel, S. Important Issues and Key Activities in Product Derivation: Experiences from Two Independent Research Projects. in Software Product Line Conference. 2009. San Francisco, CA, USA: Proc. of the 13th International Software Product Line Conference (SPLC 2009).

O’Leary, P., McCaffery, F., Richardson, I., and Thiel, S. Towards Agile Product Derivation in Software Product Line Engineering. in 16th European Conference on Software Process Improvement (EuroSPI 2009). 2009. Madrid, Spain.

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).

O’Leary, P., Richardson, I., and Thiel, S., Developing a Product Derivation Process Framework for Software Product Line Organisations, in EuroSPI 2008 Doctoral Symposium. 2008: Dublin, Ireland.

O’Leary, P., Thiel, S., Botterweck, G., and Richardson, I. (2008). Bosch Technical Report: LERO - The Irish Software Engineering Research Centre.

O’Leary, P., Ali Babar, M., Thiel, S., and Richardson, I. Towards Agile Product Derivation in Software Product Line Engineering. in 4th International Workshop on Rapid Integration of Software Engineering techniques (RISE). 2007. Luxembourg.

O'Leary, P., Ali Babar, M., Thiel, S., and Richardson, I. 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. Poznań, Poland: Wydawnictwo NAKOM.

Appendix I: Pro-PD Matrix

Symbol	Description
P	Primary Performer of a task
a	Secondary Performer (or assists) the task
<	An input to a task
>	An output from a task
</>	Both an input and an output from the task

Figure I-1 Pro-PD Matrix Legend

Appendix J: Task Attribute Tables for Pro-PD

Version One

Attribute	Attribute Description
Name	1.1 Map Customer Requirements to Platform Features
Purpose	Determine the list of the customer requirements which can be satisfied by a configuration of the platform assets
Inputs	Platform Assets Customer Requirements
Outputs	Unmapped Customer Requirements Mapped Customer Requirements
Main Description	The product team compare the customer requirements to the platform assets. The customer requirements are checked to see if they fall under the scope of the platform and can therefore be mapped to platform assets.
Roles	Product Manager

Table J-1 Map Customer Requirements to Platform Features Task

Attribute	Attribute Description
Name	1.3 Form Product Specific Requirements
Purpose	Create the Product Specific Requirements
Inputs	Mapped Customer Requirements Negotiated Requirements
Outputs	Product Specific Requirements
Main Description	The Mapped Customer Requirements and the Negotiated Requirements are merged to form the Product Specific Requirements.
Roles	Product Manager

Table J-2 Form Product Specific Requirements Task

Attribute	Attribute Description
Name	1.4 Create the Product Specific Test Cases
Purpose	Create the Product Specific Test Cases
Inputs	Product Specific Requirements Platform Test Case Artefacts
Outputs	Product Specific Test Cases
Main Description	The Product Specific Requirements are used to create the product specific test cases. The Platform Test Case Artefacts can be used as a basis for creation of the Product Specific Test Cases
Roles	Product Tester

Table J-3 Create the Product Specific Test Cases Task

Attribute	Attribute Description
Name	2.1 Derive New Configuration
Purpose	A new configuration from a subset of the overall Platform Architecture is derived
Inputs	Platform Architecture
Outputs	Base Product Configuration
Main Description	In situations where no appropriate existing configuration can be selected, the Product Team derive a new configuration from a subset of the overall Platform Architecture.
Roles	Product Architect

Table J-4 Derive New Configuration Task

Attribute	Attribute Description
Name	2.2 Select Closest Matching Configuration
Purpose	An Initial Product Configuration is selected from the set of existing platform configurations.
Inputs	Existing Platform Configurations
Outputs	Base Product Configuration
Main Description	A base configuration is selected from the set of existing platform configurations. 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.
Roles	Product Architect

Table J-5 Select Closest Matching Configuration Task

Attribute	Attribute Description
Name	2.3 Select Platform Components
Purpose	Satisfy Product Specific Requirements which could not be met through configuration selection.
Inputs	Platform Components Base Product Configuration
Outputs	Integrated Product Configuration
Main Description	In Select Subset of Existing Components, components are selected from the collection of Platform components for, addition to or replacement of, components in the base product configuration. The integrated product configuration can be either a selection of existing configurations or a newly derived configuration. The selection of components allows the Product Team meet requirements which could not be met through configuration selection.
Roles	Product Architect

Table J-6 Select Subset of Existing Components Task

Attribute	Attribute Description
Name	2.4 Identify Required Component Adaptations
Purpose	Identify the required component adaptations to satisfy Product Specific Requirements which could not be satisfied by configuration of platform assets
Inputs	Product Specific Requirements
Outputs	Required Component Development
Main Description	The Product Team decide if Component Development/Adaptation is required. If further development is required, the Product Specific Requirements which could not be satisfied by the Integrated Product Configuration are identified.
Roles	Product Manager

Table J-7 Identify Required Component Adaptations Task

Attribute	Attribute Description
Name	3.1 Scoping of Component Changes
Purpose	Decide whether to integrate a given requirement into the platform or into an individual product only.
Inputs	Required Component Development
Outputs	Required Component Development
Main Description	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) during the Scoping of Component Changes task. With any required development the CCB must decide whether to integrate a given requirement into the platform or into an individual product only.
Roles	Product Manager Product Architect Platform Manager

Table J-8 Scoping of Component Changes Task

Attribute	Attribute Description
Name	3.2 Component Development at Product Level
Purpose	Implement required component development/adaptation
Inputs	Required Component Development Platform Components
Outputs	Developed/Adapted Components
Main Description	If the development or adaptation is designated to be product-specific then component development occurs. It is the responsibility of the product development team to implement the required component changes at the product level. This can also consist of adapting existing platform components at product level
Roles	Product Developer

Table J-9 Component Development at Product Level Task

Attribute	Attribute Description
Name	3.3 Request Component Adaptation at Platform Level
Purpose	Request changes to platform in order to meet certain Product Specific Requirements
Inputs	Required Component Development
Outputs	Platform Change Request
Main Description	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.
Roles	Product Manager

Table J-10 Request Component Adaptations at Platform Level Task

Attribute	Attribute Description
Name	3.4 Component Unit Testing
Purpose	Test new components or adapted existing components
Inputs	Developed/Adapted Components
Outputs	Developed/Adapted Components
Main Description	When a component is built or adapted, initial or tailored versions of a component will need to be tested rigorously through unit testing.
Roles	Product Tester

Table J-11 Component Unit Testing Task

Attribute	Attribute Description
Name	4.1 Product Integration
Purpose	Integrated Partial Product Configuration and developed/adapted components
Inputs	Developed/Adapted Components Partial Product Configuration
Outputs	Integrated Product Configuration
Main Description	The newly developed/adapted components are integrated with the partial product configuration. The Product Team integrates the developed/adapted components and the partial product configuration by writing sufficient “glue” code to interface with the components. This includes implementing any required architectural changes to facilitate the developed/adapted components.
Roles	Product Architect

Table J-12 Product Integration Task

Attribute	Attribute Description
Name	4.2 Integration Testing
Purpose	Validates the Platform assets for this particular configuration
Inputs	Platform Test Artefacts Integrated Product Configuration
Outputs	Integrated Product Configuration
Main Description	Due to the variability defined in the platform assets, completely testing the platform assets is impossible except for trivial cases. Integration Testing 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.
Roles	Product Tester

Table J-13 Integration Testing Task

Attribute	Attribute Description
Name	4.3 System Testing
Purpose	Verifies product conforms to the Product Specific Requirements
Inputs	Product Specific Test Cases Integrated Product Configuration
Outputs	Validated Product
Main Description	Verifies if a product conforms to the product specific requirements. System test artefacts such as the product specific test cases are already derived from the product specific requirements [6] in Impact Analysis.
Roles	Product Tester

Table J-14 System Testing Task